HONEYWELL

COMMUNICATIONS SNA6 APPLICATION PROGRAMMER'S GUIDE

SOFTWARE

COMMUNICATIONS SNA6 APPLICATION PROGRAMMER'S GUIDE

SUBJECT

Application Interface Facility LU Type 0 Session Calls and LU Type 6.2 Conversation Verbs for Use in DPS 6 or DPS 6 PLUS COBOL and Assembly Language Programs

SPECIAL INSTRUCTIONS

This manual supersedes GR11-01, dated March 1986. This manual has been extensively revised; therefore, change indicators have been omitted.

SOFTWARE SUPPORTED

This publication supports Release 4.0 of the SNA6 program products.

ORDER NUMBER

GR11-02

October 1986

Honeywell

PREFACE

The purpose of this manual is to describe the SNA6 Application Interface Facility (AIF). It is intended for the COBOL or Assembly language programmer at a DPS 6 or DPS 6 PLUS system. It provides the information necessary to write application programs to communicate with transaction programs running under Customer Information Control System (CICS) or Information Management System (IMS).

The major topics presented in this manual are:

- Section 1 provides an introduction of AIF and the manual
- Section 2 describes the LU Type 0 session calls that are used in Assembly language, the session call control block, and Assembly language programming considerations
- Section 3 describes the LU Type 0 COBOL session calls, the SNA work area in the WORKING-STORAGE-SECTION, and COBOL programming considerations

USER COMMENTS FORMS are included at the back of this manual. These forms are to be used to record any corrections, changes, or additions that will make this manual more useful.

©Honeywell Information Systems Inc., 1986 File No.: 1163, 1R63, 1763 GR11-02

- Section 4 describes the LU Type 6.2 conversation verbs that are used in an Assembly language program, the verb parameter block, and Assembly language programming considerations

- Section 5 describes the LU Type 6.2 COBOL conversation verbs, the SNA work area in the WORKING-STORAGE-SECTION, and COBOL programming considerations

- Section 6 describes LU Type 0 restart logic and message resynchronization

- Section 7 describes Communications Network Management, SNA Operator Control (SOPR) services, and maintainability through various operating system utilities

- Appendix A presents a detailed description of the AIF product architecture

- Appendix B contains sample Assembly language programs demonstrating both LU Type 0 and LU Type 6.2 conversations

- Appendix C provides sample COBOL programs for both LU Type session calls

- Appendix D lists the LU Type 0 session call return codes

- Appendix E contains a listing of the Session Call Control Block (SCCB) template with offsets

- Appendix F lists the LU Type 6.2 conversation return codes

- Appendix G includes a listing of the Verb Parameter Block (VPB) template with offsets.

- A glossary is provided to explain the meaning of terms used within the text of this manual.

This manual assumes the reader is familiar with the SNA networking system and COBOL or Assembly language programming. The reader needs to be familiar with the following Honeywell manuals:

- <u>ONE PLUS COBOL 74 Language Reference Manual</u> (Order No. HE34)
- <u>One PLUS Advanced COBOL Compiler User's Guide</u> (Order No. HE31)
- <u>ONE PLUS Assembly Language (MAP) Reference</u> (Order No. HE38)

In addition, the reader needs to be familiar with the SNA transaction program protocols used by your IBM distributed processing application.

The following syntax conventions are used in this manual:

<u>Syntax Conventions</u>	<u>Meaning</u>
UPPERCASE CHARACTERS	Items in capital letters must be input as shown, for example, SCNOER.
Angle brackets <>	Items in lowercase letters enclosed in angle brackets < > describe what you need to supply; for example, <node name>.
Brackets []	Items in square brackets are optional; for example, [sccb address].
Braces {}	Braces indicate that the user has a choice between two or more entries. At least one of the entries enclosed in braces must be chosen (unless the entries are also enclosed in square brackets); for example, {NORMAL|ABNORMAL}.
Vertical line |	Vertical bars separate the choices within braces. At least one of the entries separated by bars must be chosen (unless the entries are enclosed in square brackets); for example [{SYNC|ASYNC}].

The following conventions are used to indicate the relative levels of topic headings used in this manual:

<u>Level</u>	<u>Heading Format</u>
1 (Highest)	<u>ALL CAPITAL LETTERS, UNDERLINED</u>
2	<u>Initial Capital Letters, Underlined</u>
3	ALL CAPITAL LETTERS, NOT UNDERLINED
4	Initial Capital Letters, Not Underlined

CONTENTS

CONTENTS

CONTENTS

ILLUSTRATIONS

ILLUSTRATIONS

TABLES

Section 1
INTRODUCTION

The Systems Network Architecture (SNA) Application Interface Facility (AIF) allows a programmer to write COBOL or Assembly language programs to communicate with Customer Information Control System (CICS) or Information Management System (IMS) transaction programs at an IBM host. The applications at the DPS 6 or DPS 6 PLUS can be user-written, LU Type 0 or LU Type 6.2 applications, or transaction processing routines. AIF offers SNA6 users access to information residing on an IBM host.

This facility allows applications to interface to an SNA network at a high level. AIF manages data structures on behalf of the application program. However, some knowledge of SNA protocols is necessary.

To provide this level of interface, AIF supports session calls for Session Type 0 users and basic conversation verbs for Type 6.2 users. These session and verb callss are interface with a user's control block through monitor calls to SNA. Macrocalls are provided for the applications.

This section discusses the following:

- The SNA Application Interface Facility
- LU TYPE 0 sessions with Host Programming Considerations
- LU TYPE 6.2 conversations with Host Programming Considerations.

HOW TO USE THIS MANUAL

This manual is provides an application programmer at either a DPS 6 or DPS 6 PLUS system with the information necessary to write COBOL or Assembly language application programs to communicate with the IBM host CICS/IMS transaction processing programs for Type 0 and CICS transaction programs for Type 6.2.

Section 2 describes the LU Type 0 session calls that are used in Assembly language, the session call control block, and special considerations in writing an Assembly language program using AIF.

Section 3 describes the LU Type 0 COBOL session calls, the SNA work area in the WORKING-STORAGE-SECTION, and special considerations in writing a COBOL program using AIF.

Section 4 describes the LU Type 6.2 conversation verbs that are used in an Assembly language program, the verb parameter block, and special considerations in writing an Assembly language program using AIF.

Section 5 describes the LU Type 6.2 COBOL conversation verbs, the SNA work area in the WORKING-STORAGE-SECTION, and special considerations in writing a COBOL program using AIF.

Section 6 describes LU Type 0 restart logic and message resynchronization.

Section 7 describes Communications Network Management, SNA Operator Control (SOPR) services, and maintainability through Data Base Augmented Real-Time Tracing System (DARTS).

Appendix A presents a detailed description of the AIF product architecture. Appendix B contains sample Assembly language programs demonstrating both LU Type 0 and LU Type 6.2 conversations. Appendix C provides sample COBOL programs for both LU Type session calls. Appenix D lists the LU Type 0 session call return codes. Appendix E contains a listing of the Session Call Control Block (SCCB) template with offsets. Appendix F lists the LU Type 6.2 conversation return codes. Appendix G includes a listing of the Verb Parameter Block (VPB) template with offsets.

A glossary is provided to explain the meaning of terms used within the text of this manual.

APPLICATION INTERFACE FACILITY

Applications on a DPS 6 executing under the MOD 400 Executive or applications on a DPS 6 PLUS executing under the HVS 6 PLUS Operating System can communicate with IBM hosts that use SNA network products.

AIF supports the application-to-application communications facilities available with CICS and IMS systems. AIF provides the communications support necessary to implement the following:

- User Assembly language and COBOL communication with CICS or IMS via LU Type 0 macrocalls
- User Assembly language and COBOL advanced program to program communication with CICS via LU Type 6.2 conversation verbs.

LU TYPE 0 SESSIONS

AIF supports an application-to-application interface over a Session Type 0. The Session Type 0 is an interprogram Logical Unit (LU) defined within SNA. It is supported by both CICS and IMS and is used for communications between these subsystems and applications on several IBM processors.

The Session Type 0 can use any feature of SNA that is defined by Session Type 0 FM profile 4 and TS profile 4. The SNA features that these sessions can use are further defined by how CICS treats a full function LU or how IMS treats a secondary LU programmable.

The constraints on this type of session are imposed by CICS or IMS, each of which has a slightly different set of rules governing the exchange of information. Since there are slight differences in implementing the macrocalls for CICS and IMS, it is important for the application programmer to know with which the application program is communicating and how the LU is defined.

AIF transactions are allowed to perform any function through CICS or IMS; e.g., inquiry, update, etc. These IBM subsystems specify the order in which SNA requests and responses can be sent but impose no restrictions on what can be done over the session.

Host Programming Considerations

A Session Type 0 requires that both applications expect the same format and protocol. These applications must be written as two complementary halves of a transaction. They must agree on application protocols, transaction processor protocols, and the host GEN environment.

Although host considerations are described in detail in the SNA6 Host System Programmer's Guide (GB88), the AIF programmer should be aware of the host terminal definition of the application with which he will be communicating. These definitions influence how the AIF session calls are issued and which parameters must be supplied with them.

The host views the application (LU) as a terminal, and defines it within tables. The following subsections list the host terminal definitions with which you will be concerned. Check with the host system programmer to determine the definition of the terminal macro(s) that the IBM system uses.

CICS TERMINAL CONTROL TABLE PARAMETERS

This subsection describes the Terminal Control Table (TCT) parameters which are of interest to you if your application is to communicate with a CICS transaction program.

BRACKET=YES

This parameter indicates that bracket protocol is to be enforced for the LU/LU session. This parameter is required for a full function terminal.

BUFFER=buffer_size

This parameter indicates the size of the receive buffer for the LU. This is the maximum data length the DPS 6 or DPS 6 PLUS application can receive. The buffer size specified to CICS indicates how CICS does chaining.

RELREQ={YES|NO,YES|NO}

This parameter instructs CICS whether to release the LU if it is requested by another application and whether disconnect requests are to be honored. If LUs are to be released to another VTAM application, the DPS 6 or DPS 6 PLUS application may have to re-issue the INIT.

RUSIZE=ru_size

This parameter specifies the maximum size of the request unit (RU) that the LU can receive. The size of the RU with relation to the buffer size determines how much chaining is done and how many receives one must do when not using the message completion option.

TRMSTAT=term_state

This parameter indicates the type of activity that can occur at this LU. The terminal state determines whether the application can send to or receive from the host.

IMS TERMINAL DEFINITION PARAMETERS

The IMS terminal definitions control the protocol conversation in the LU-LU session to an even greater extent than the CICS terminal definitions. This subsection describes the parameters that are of particular interest to you if your application is to communicate with an IMS program.

COMPT$_n$=(x[,y,z])

This parameter specifies the component types and the processing associated with that node. A node can have up to four components (n=1-4) and three subparameters for each component. For the purpose of writing AIF transaction programs, you only need to know the value of the first of these subparameters (x). The value of x can be either:

Program1 - IMS does not assume program protection and can send consecutive messages without waiting for intervening input requests.

Program2 - IMS assumes component protection and does not send consecutive messages without intervening input requests.

OPTIONS=(termresp,acknowl,relreq)

This parameter specifies certain communications associated with the LU. These options dictate some of the basic communication design of the DPS 6 or DPS 6 PLUS application.

1. Terminal Response Mode Options (termresp).

   When an application operates in terminal response mode, all operations between the terminal (or application) and IMS stop when IMS receives a transaction and do not resume until IMS receives an acknowledgment that the application received IMS's reply.

   This option can be defined as follows:

   a. TRANRESP: The transaction being executed can select terminal response mode.

   b. NORESP: Terminal response mode is not used for any transaction.

   c. FORCRESP: Terminal response mode is forced for all transactions.

2. Acknowledgment (acknowl)

   This specifies the mode of acknowledgment between the terminal (application) and IMS. This option can be defined as follows:

   a. ACK: This option indicates that transactions are recoverable and must be acknowledged. If this option is specified, the AIF application must request definite response on all input messages.

b. OPTACK: This option indicates that only input messages containing a Begin Bracket (BB) indicator are acknowledged with an outbound message containing an End Bracket (EB) indicator. If this option is specified a request by AIF for definite response is optional.

3. Release Request (relreq)

   This parameter indicates whether IMS should release an LU if requested by another VTAM subsystem. This option can be defined as follows:

   a. RELRQ: This option specifies that IMS must honor requests from other VTAM subsystems and release the LU.

   b. NORELRQ: This option specifies that IMS not release an LU when it is requested by another subsystem.

Refer to the IBM manual IMS/VS Programming Guide for Remote SNA Systems for further information about programming secondary LU Type 0 sessions to connect to IMS applications.

LU TYPE 6.2 CONVERSATIONS

AIF supports an advanced program to program communication interface over an LU Type 6.2 Conversation. The LU Type 6.2 is an interprogram Logical Unit (LU) defined within SNA. It is supported by CICS and is used for communications between transaction programs and network resources.

The LU Type 6.2 can use any feature of SNA that is defined by LU Type 6.2 FM Profile 19 and TS Profile 7. The SNA features that these sessions can use are further defined by how CICS treats an LU 6.2 and the extent to which it has been implemented in AIF.

The SNA features that these sessions can use are constrained only by the level of LU Type 6.2 functions that are incorporated in the program products. AIF supports the basic conversation implementation of LU Type 6.2. Applications must conform to the rules for basic conversations. For more information on LU Type 6.2 programming considerations, refer to the appropriate IBM manuals listed in the front of this book.

The AIF transactions are allowed to perform any service or application function through CICS; e.g., inquiry, update, etc. An LU Type 6.2 application expects the same format and protocol on both sides of the conversation. These applications must be written as two complementary halves of a transaction. They must agree on application protocols, transaction protocols, and conversation states.

Section 2
PROGRAMMING LU TYPE 0 SESSIONS IN ASSEMBLY LANGUAGE

This section describes the Assembly language session calls that are used to converse over a Session Type 0 with host transaction programs. Topics include:

- Session call format
- Programming considerations
  - Getting started
  - Synchronous vs. Asynchronous Processing
  - Creating a session call control block
  - Checking the return code
- Individual session calls
  - Format
  - Descriptions
  - Return codes

SESSION CALL FORMAT

The session calls used by AIF are macrocalls provided by the DPS 6 or DPS 6 PLUS system. These session calls have a list of arguments that can be specified by the programmer or accepted in their existing form. If no arguments are specified, then all that is provided is the monitor call. AIF session calls follow the conventions for Assembly language as described in the ONE PLUS Assembly Language (MAP) Reference (HE38) manual. The session call can have an optional label if no label is used at least one blank space must precede the session call.

User-selected items are known as arguments. These arguments are positional within the session call macros. The order of positional arguments indicates the variables to which data is applied. Thus, the order of your arguments must be the same as the order of the positional arguments within the session call macro.

The following rules govern the use of positional arguments:

- Omitted arguments that precede an included argument must be indicated by the presence of a delimiting comma for each omission.
- One or more spaces must separate the macrocall name from its arguments, with a comma between each argument. (The horizontal tab character is equivalent to a space.)
- A semicolon at the end of a line indicates that the next line is a continuation line.

In the following example, the first and third arguments have been omitted; their positions have been held by delimiting commas. Spaces separate the session call name from its arguments.

```
        $SINIT  ,'AIFNODE1',,'AA',SYNC
```

The arguments for these session calls are found in the SCCB. An SCCB must be provided for each session call. These fields can be altered either during initialization or by including the appropriate arguments in the session call itself.

At the completion of each session call, when control is returned to the application, a return code is placed in register $R1. This return code indicates whether a session call has been completed error free. The application should check this return code after each session call to verify the return status of the session call. Additional information, if desired, can be found in the output control word, found at the offset SC_OCT of the SCCB.

When AIF is activated, it defines one or more pools of logical units (LU) according to the configuration file for that node. Each pool of LUs is reserved for a specific host CICS or IMS system. AIF can either start a session to the host system at initiation or it can wait for an application to request a session. The time of session initiation is a configuration option.

When an application requests to initiate a session with an LU, AIF checks the availability of that LU and assigns it if it is available. If the specified LU is unavailable, AIF checks first for an available reserved LU, second for an available preestablished LU, and then for any available LU to assign to the session. AIF either returns the address of the LU with which the session is started, or rejects the request if there is no LU available.

An application gains access to a host-initiated session by executing a $SACPT session call. Executing the accept session call causes the application to be connected to a host-initiated session and causes the LU to send a positive response to the host, accepting the session.

PROGRAMMING CONSIDERATIONS

Many of the programs that use AIF session calls are written in Assembly language. These applications may be reentrant and may not require more than one occurrence of a given macrocall.

Special considerations that the programmer must bear in mind are discussed in this section:

- Getting started
- Synchronous vs. Asynchronous processing
- Creating a session call control block
- Host-initiated sessions
- Checking the return code.

Getting Started

When using AIF session calls in an Assembly language program, remember the folowing steps:

1. In order to use the session calls and utility macros included with AIF, you must first make them available to your program. When beginning your program, include the following statement:

   ```
   LIBM    '>>LDD>MACROS>MAC_USER'
   ```

2. Then issue the macrocalls $SSCCB and $SAIRC to define the SCCB and return codes in memory.

3. You must also set aside a workspace with room for the stack, the SCCB, and your send/receive buffer, as in the following example:

```
*
*   WORK LOCATIONS: STACK, SCCB, & SEND/RECEIVE BUFFER
*
WKSP        EQU    0                 BEGINNING OF WORKSPACE
MYSTACK     EQU    WKSP+50           REGISTER STACK
CNTLWD      EQU    MYSTACK           FOR PROGRAM CONTROL
MYSCCB      EQU    CNTLWD+1          BEGINNING OF SCCB
BUFFER      EQU    MYSCCB+SC_SIZ     SEND/RECEIVE BUFFER
BUFSZ       EQU    2000              BUFFER SIZE
WKSPSZ      EQU    BUFFER+BUF_SZ     WORKSPACE SIZE
```

Synchronous vs. Asynchronous Processing

AIF session calls can be processed either synchronously or asynchronously.

SYNCHRONOUS PROCESSING

Synchronous processing implies that when the application passes an instruction to AIF for processing, it waits for the application to complete that instruction before continuing.

In Figure 2-1, a $SINIT session call has been issued synchronously. The application completes its segment of processing and passes the request to AIF. AIF executes the $SINIT completely and passes the return code to the application. The application does not process other instructions while AIF is executing the $SINIT session call.

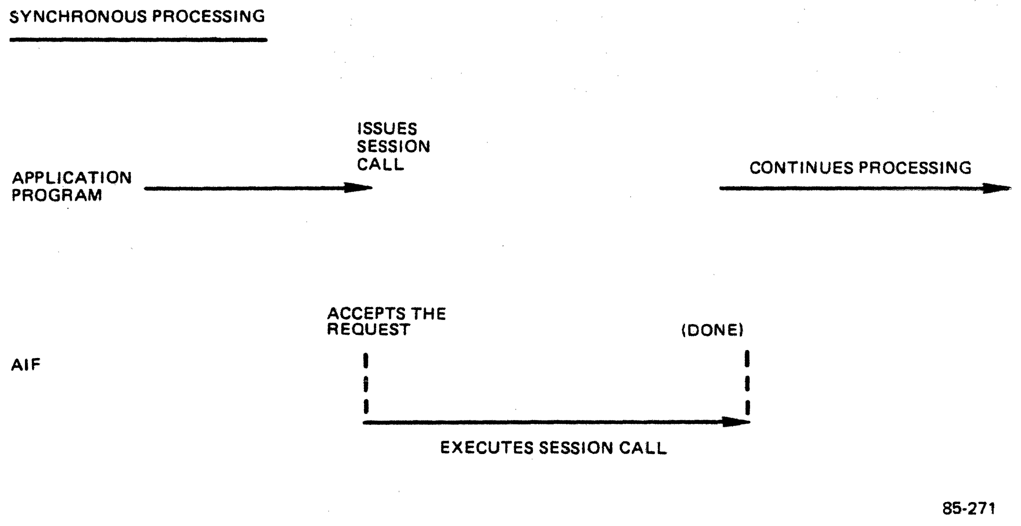
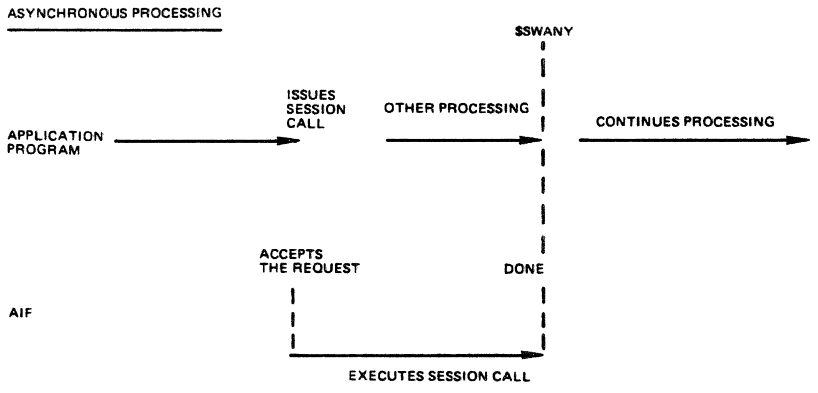


Figure 2-1. Synchronous Processing

ASYNCHRONOUS PROCESSING

Asynchronous processing implies that when the application passes an instruction to AIF for processing, the application continues to process other instructions while it waits for AIF to complete that instruction.

In Figure 2-2, a $SINIT session call has been issued asynchronously. The application completes its segment of processing and passes the request to AIF. While AIF executes the $SINIT session call, the application is processing other instructions. In order for the application to find out that AIF has finished executing the $SINIT session call, the application must execute a $SWANY or a $STEST session call.



85-272

Figure 2-2. Asynchronous Processing

Each time you issue an asynchronous order, you must check the receive buffer before you can receive information. You can do this by either the $STEST or the $SWANY session call. These two session calls differ as follows:

1. The $STEST session call checks to see if there is information in the queue to be received and immediately reports back to the application. This call can be executed any time you wish to check for an outstanding order, and as often as you wish to check, because the application regains control immediately after the test is completed.

2. The $SWANY session call checks for information on the queue and waits until there is information waiting before it returns control to the application.

Session Call Control Block

Communication between the application program and AIF is through the application-provided SCCB. Following a $SINIT or a $SCACPT, the same SCCB is used for all subsequent session calls until a particular session is terminated. If a program is to run multiple sessions, you must provide a separate SCCB for each session.

When the application provides parameters with a given macrocall, the macrocode updates the appropriate SCCB fields before executing an AIF monitor call. If any of the fields have been changed, the new values are in the SCCB when you reexamine it.

The first parameter of each macro is the location of the SCCB, except in the case of $SWANY. If not specified as the first parameter of the macro, this pointer must be in register $B4. Allowable formats for this parameter and all address pointers are the same as found in the "Addressing Parameters" section of the System Programmer's Guide, Vol. 2.

Where a value rather than an address is provided in a parameter, allowable formats are:

1. (*)$B1(.$R)
2. LABEL
3. =$R1
4. =literal
5. !LABEL

When you establish a session through a $SINIT or a $SACPT, you must supply an SCCB. This SCCB is used for all session calls for this session. The application can move the session call parameters to the SCCB before executing the session call (see example 1 below). The programmer can also provide the parameters for the session call in the macro itself (see example 2 below).

The following examples show both methods of creating an SCCB for the $SINIT session call. Which convention you choose to follow depends upon the requirements of your program.

Example 1:

The following example shows the parameters in the SCCB being loaded before issuing the session call. Offsets to the SCCB are provided in the displacement macro $SSCCB. (Refer to the SCCB template in Appendix E for appropriate offsets.)

```
NODENM  DC   'AIF505  '
HLU_NM  DC   'CICS    '
STD_NM  DC   'AB'
        .
        .
        .
        LDB  $B4, $B6.SCCB       Load SCCB address to $B4
        LDI  NODENM              Get first 4 bytes of nodename
        SDI  $B4.SC_NOD          Store first 4 bytes of
                                 nodename in SCCB
        LDI  NODENM+2            Get second 4 bytes of nodename
        SDI  $B4.SC_NOD+2        Store second 4 bytes of
                                 nodename in SCCB
        LDI  HLU_NM              Get first 4 bytes of Remote LU
                                 name
        SDI  $B4.SC_RLN          Store first bytes of Remote LU
                                 name
        LDI  HLU_NM+2            Get second 4 bytes of Remote
                                 LU name
        SDI  $B4.SC_RLN+2       Store second byres of Remote
                                LU name
        LDR  $R2, STD_NM        Get STD name
        STR  $R2.SC_STD         Store STD name in SCCB
        LBT  $B4.SC_ICT,SCRTNS  Set bit for synchronous
                                execution
        $SINIT
```

Example 2:

The following example shows the $SINIT session call with the same parameters specified within the macrocall.

```
$SINIT ,'AIF505','CICS','AB',SYNC
```

Host-Initiated Sessions

AIF supports host-initiated sessions; that is, it accepts unsolicited binds. In order to accept an unsolicited bind, an LU must be reserved with the HOST_INIT_SESS parameter specified as Y (YES) in the LU entry of the configuration file.

The program name, node name, STD name, and base level are provided to the application program by AIF via the standard operating system parameter list (refer to the System Programmer's Guide, Vol. 2). When the application program begins execution, it must issue a $SACPT session call as the first session call, providing the STD name and the node name for the LU to be used. The node name and the STD name provided with the $SACPT call must be the same as the parameters passed by AIF.

After the $SACPT call is executed, the application is in receive state. The $SACPT session call allows AIF access to a host-initiated session. The application must execute a receive to have access to the bind. AIF associates the first unsolicited bind (host-initiated session request) to the first $SACPT session call from the task group that AIF spawned.

An unsolicited bind can be for a program designated in the AUTO_ATTACH entry of the AIF configuration or it can be any other unsloicited bind sent from the host.

When AIF receives an unsolicited bind for a specific LU, AIF checks the LU entry for an AUTO_ATTACH program. If it finds one, AIF spawns a group with the program_name as the lead task, and passes to the lead task the STD name, node_name, and base_level used in the spawn group. If AIF does not find an AUTO_ATTACH program in the LU entry, it accepts the session and looks for the program name in the first four bytes of the first record received, then spawns a group based on the ATTACH_PROGRAM entry. If none is provided, default values are used to spawn the group.

The application can issue multiple $SACPTs to check for additional host-initiated sessions intended for this application. For an application to accept more than one session, all LUs that can receive binds for that application must be reserved LUs with HOST_INIT_SESS=Y. Each of these LUs must have the same group_id specified in the LU entry in the configuration file. Note, if multiple $SACPTs are used, multiple SCCB should also be used.

NOTE

> In order to execute a START_UP.EC instead of an attached program, you must create an attach program table entry with a dummy name (e.g., ATTACH_PROG=ABC), specifying the appropriate spawn group parameters, and include an ALIAS for ABC (eg., ALIAS=>>SYSLIB2>EC?EXECL) to execute the START_UP.EC specified in the home directory. Refer to SNA6 Network Configuration for further information.

Checking the Return Code

After a session call is executed, AIF returns a return code to the Session Call Control Block (SCCB) to indicate how the call was completed. The application should examine this return code at the completion of each session call to determine if the call has been completed error free.

The return code has 16 bits and is placed in register $R1 by AIF before control is returned to the application program. The return code can also be found in SC_RCD.

Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any session call. These bits should be examined individually, then "masked out" so that the application can examine the remaining bits. If the bit is on, then the return code indicated is true. The following masks are provided in the $SAIRC macrocall for checking each of the first five bits as follows.

Bit 0 RCABRT

> The session has been terminated. An SOPR command has been entered that caused the session to terminate, or the session has been unbound by the host. The reason for this termination can be found in the "abort reason" code in the SCCB (SCCB.SC_ABT).

Bit 1   RCSTOP

An SOPR STOP command has been received. If the session is still active (bit 0 = 0), then check the SC_TIM field in the SCCB to determine the time at which the session ends. During this time the application can continue to process, but should normally terminate.

The time found in the TIME argument (SCCB.SC_TIM) is the wall clock time in standard 48-bit format, at which the session terminates.

Bit 2   RCRINT

An interrupt has been received. The interrupt type is found in SC_INT in the SCCB.

There are three categories of interrupt:

1. Expedited or normal flow data flow control commands
2. Communications Network Management data
3. Control information passed to application by AIF.

If sense data is present, it is found in SCCB.SC_ESD.

Bit 3   RCSCNL

The call has been cancelled; it is not processed. If the application desires the order to be processed, the call must be reexecuted.

Bit 4   RCSCMP

The call has been completed.

A return code can indicate more than one condition occurring at the same time. For example, it can indicate both an interrupt and a completed call, a session abort and a completed call, or no session abort and a cancelled call.

The masks RCABRT, RCSTOP, RCRINT, RCSCNL, and RCSCMP are provided for your convenience in checking bits 0 through 4. After you have checked these bits, null them out and examine bits 5 through 15. If you choose to null these bits by using RCMASK, which is provided in the software (RCMASK=07FF), use the following statement:

```
AND   $R1,=RCMASK
```

Bits 5 through 15 contain the return code for a completed or cancelled call. One way of doing this part of the return code is to issue a "compare" instruction as follows:

```
CMR     $R1,=RMNOER   Checks for "No error" code
BE      CONT_1
```

If the Return code contains a "no error" message, branch to the next segment of the program. If the return code contains an error condition, you might decide to record it to an error-out file, branch to another segment of the program, or shut down completely.

If during asynchronous processing an error is detected immediately (e.g., a parameter is incorrectly specified), the return code provides the error and the call is cancelled. However, if during asynchronous processing AIF issues the monitor call before an error is detected, the return code returns with a zero indicating no error has been detected. However, an error could occur elsewhere (e.g., at the host) and AIF would not be aware of it. In order to determine if an error occurs with the session after AIF has performed all of its error detection, issue either a $STEST or a $SWANY for the return code.

Appendix F contains a complete list of return codes. These labels and their hexadecimal values can be found in the macro: $SAIRC (AIF Return Codes).

SESSION CALLS

Table 2-1 contains a list and description of the session calls used by AIF in an Assembly language program. The format of these session calls is detailed on the following pages along with a discussion of the input arguments and an output description.

Table 2-1. AIF Session Calls

Session Call	Description
$SACPT	Accept Session
$SCASR	Cancel Outstanding Asynchronous Request
$SGTAT	Get Session Attributes
$SINIT	Initiate or Restart a Session
$SPOLL	Test for LU associated with task group
$SRECV	Receive message in application's buffer
$SRI	Read Interrupt
$SSEND	Request AIF to send a message or message segment
$SSI	Send Interrupt
$SSRSP	Caller instructs AIF to send a response
$STERM	Terminate session
$STEST	Test conditions
$SWANY	Wait on any event
$SACEB	Converts ASCII to EBCDIC
$SEBAC	Converts EBCDIC to ASCII

$SACPT - Accept Session Call

The $SACPT session call causes AIF to connect the local application to a host initiated session.

FORMAT:

```
[label] $SACPT        [sccb pointer]       P1: $B4
                      [,node name]         P2: SC_NOD
                      [,std name]          P3: SC_STD
```

ARGUMENT:

sccb pointer

This parameter contains a pointer to the address of the SCCB. If this parameter is missing, the address is assumed to be contained in register $B4.

node name (SC_NOD)

Identifies the AIF node to which the application is directing this session call. This field contains eight alphanumeric characters. If you are loading the SCCB yourself and your node name has fewer than eight characters, this field must be left-justified and space-filled.

std name (SC_STD)

The configured Session Type Descriptor (STD) which lists the attributes of the session to be established. This field consists of two alphanumeric characters.

DESCRIPTION:

The $SACPT session call causes AIF to connect the local application to a host-initiated session if there is one available. If no session is available, AIF returns and continues processing. The LU to which this bind refers most be a reserved LU.

If your application is part of a host-initiated session, the $SACPT session call should be the first call executed. When the $SACPT call is completed, the session is in receive state.

NOTE

This call is always made synchronously.

RETURN CODES:

The application should check the return code after each execution of a session call. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any session call. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15.

In addition to the general return codes, the following values are possible:

Value	Label	Description
0000	RMNOER	No error
0019	RMACTO	ACCEPT timed out
0040	RMINOD	Invalid node name
0099	RMISTD	Invalid STD name
009A	RMILUT	Invalid LU type in STD
009B	RMNOAT	No LU attached

session id (SC_SID)

This two-word field is supplied by AIF after it accepts the session request. The first word is the session group name, which is assigned by AIF to each of the sessions running in this session group. This value is used by AIF to return a unique one-word session identifier for this session. This value is stored in the second word. This field is reserved for system use and must never be altered by the application.

maximum ru size (SC_MRU)

This field shows the RU size that is returned.

$SCASR - Cancel Asynchronous Request

The $SCASR session call causes AIF to cancel an outstanding asynchronous request, if possible.

FORMAT:

```
[label]     $SCASR    [sccb pointer]      Pl: $B4
```

ARGUMENT:

sccb pointer

This parameter contains a pointer to the address of the SCCB. If this parameter is missing, the address is assumed to be contained in register $B4.

DESCRIPTION:

The $SCASR session call cancels an asynchronous request, if there is one outstanding. If the previously executed asynchronous request were completed when the $SCASR session call was executed, then the return code from the $SCASR session call is the return code for the completed asynchronous session call. If the previously executed asynchronous session call was not completed when the $SCASR session call was executed and AIF succeeded in cancelling the request, the return code from the $SCASR session call indicates that the session call has been cancelled.

If there is no asynchronous session call outstanding when the $SCASR session call is executed, then the return code is RCNOUT (no outstanding session call).

NOTE

The $SCASR session call cannot be used to cancel a $SINIT session call, even if it has been executed asynchronously.

RETURN CODES:

The application should check the return code after each execution of a session call. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any session call. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15.

In addition to the general return codes, the following values are possible:

Value	Label	Description
0017	RMNOUT	No outstanding asynchronous call

NOTE

If the previously executed asynchronous call were already completed, the return code is for that call.

Example:

In the following example, the application requests that AIF cancel an outstanding asynchronous request. AIF assumes that register $B4 is pointing to the SCCB of the session call to be cancelled.

```
    ENDIT                 $SCASR
```

$SGTAT - Get A Session Attribute

The $SGTAT session call provides the application with attribute information for the session specified in the SCCB pointer.

FORMAT:

```
[label]  $SGTAT  [sccb pointer]          P1: $B4
                 [,attribute buffer]     P2: SC_BUF
                 [,attribute length]     P3: SC_DLG
                 [,{R|L}]                P4: SC_ICT.SCRHBI
                 [,type]                 P5: SC_SIN
```

ARGUMENTS:

sccb pointer

This parameter contains the address of the SCCB of the session for which you are requesting attributes. If not declared, the address is assumed to be in register $B4.

attribute buffer (SC_BUF)

A pointer to the application's attribute buffer. This buffer will receive the data returned by this call.

attribute buffer length (SC_DLG)

The length of the receive buffer in bytes. The maximum allowable length of this buffer is 32,747 bytes.

{R|L} (SC_ICT.SCRHBI)

Specifies whether data starts on the left (L) or right (R) byte of the buffer address word.

type (SC_SIN)

Specifies the type of attribute you are requesting. The attribute information available is BINDIM, which has a value of 1. You can specify either the attribute type or its value.

DESCRIPTION:

The $SGTAT session call provides the application with attribute information , one attribute at a time, for the session whose SCCB pointer is specified when issuing the call. If you plan to ask for the bind image, the STD entry in the AIF configuration must include the parameter SAVE_BIND=Y.

Special notice should be given to the situation where an interrupt is received either prior to or during the execution of the $SGTAT session call.

1. When an interrupt is received before the execution of the $SGTAT, the application is given the data that was in the receive queue and informed of the interrupt.

2. If an interrupt is received during the execution of a $SGTAT, the order is not completed, control is returned to the application, and the return code indicates that an interrupt has been received.

NOTE

This call is always made synchronously.

RETURN CODES

The application should check the return code after each execution of a session call. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any session call. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15.

In addition to the general return codes, the following values are possible:

Value	Label	Description
0000	RMNOER	No error
0010	RMIMPS	Improper State
0013	RMRB2S	Receive buffer too small
0015	RMIINT	Invalid attribute type
0018	RMNBDS	No BIND_IMAGE saved for $SGTAT
0032	RMDTCL	Send/receive rejected; data traffic cleared or inactive.

Received Interrupt Type (SC_INT)

This field contains the interrupt type if one is received during the execution of this session call.

Error Code or Sense Data Received (SC_ESD)

This field can contain either detailed information about an error condition or sense data from a remote LU, if a negative response has been received.

Received Buffer Data Length (SC_ADL)

This field contains the actual length of the received data in bytes.

$SINIT - Establish A Session

The $SINIT session call is used to establish or restart a session. In issuing the session call, you must indicate for which purpose it is to be executed, by specifying RESTART or NO RESTART. If you are using $SINIT session call to establish a session, you must use the following format:

FORMAT:

```
[label]  $SINIT  [sccb pointer]         P1: $B4
                 [,node name]           P2: SC_NOD
                 [,remote lu name]      P3: SC_RLN
                 [,std name]            P4: SC_STD
                 [,{SYNC|ASYNC}]        P5: SC_ICT.SCRTNS
                 [,NO_RESTART]          P6: SC_ICT.SCRSTR
```

ARGUMENTS:

sccb pointer

This parameter contains a pointer to the address of the SCCB to be used for this session. If not declared, the address is assumed to be in register $B4.

node name (SC_NOD)

Identifies the AIF node to which the application is directing this session call. This field contains eight alphanumeric characters. If you are loading the SCCB yourself and your node name has fewer than eight characters, this field must be left-justified and space-filled.

remote lu name (SC_RLN)

The name by which the remote LU is known to this application. This field contains eight alphanumeric characters. If you are loading the SCCB yourself and your remote lu name has fewer than eight characters, this field must be left-justified and space-filled.

std name (SC_STD)

The configured Session Type Descriptor (STD) which lists the attributes of the session to be established. This field consists of two alphanumeric characters.

SYNC|ASYNC (SC_ICT.SCRTNS)

This parameter indicates whether execution of this call is synchronous or asynchronous.

NO_RESTART (SC_ITC.SCRSTR)

NO_RESTART is used to indicate that this is a newly established session; including NO_RESTART causes this bit to be reset.

DESCRIPTION:

The initiate session call requests that AIF establish a session between an LU at the DPS 6 or DPS 6 PLUS and an LU at the host, and that the local LU be assigned exclusively to the application. In the event that AIF assigns a preestablished session to the application, the application should store the send/receive sequence numbers in case a RESTART of this session ever becomes necessary. These sequence numbers are not reset to zero after each use. To the host, this appears as one session. On the DPS 6 or DPS 6 PLUS side, the session is a serially reusable resource. After the $SINIT is executed, the session enters send state.

NOTE

A $SINIT session call, executed asynchronously, cannot be cancelled by using the $SCASR session call macro.

RETURN CODES:

The application should check the return code after each execution of a session call. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any session call. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15.

In addition to the general return codes, the following values are possible:

Value	Label	Description
0000	RMNOER	No error
0003	RMRNEG	-RSP returned by host
0004	RMNBIF	Bind negotiation failed
0040	RMINOD	Invalid node name
0096	RMNNAC	Node not yet active
0097	RMNLAC	Node active but no active LUs yet
0098	RMNOAV	LUs active, but none available for this session
0099	RMISTD	Invalid STD name
009A	RMILUT	Invalid LU type in STD

If the $SINIT session call is successful (RMNOER), SC_SQN and SC_RSQ have the send/receive sequence numbers for the session.

session id (SC_SID)

This two-word field is supplied by AIF after it accepts the session request. The first word is the session group name, which is assigned by AIF to each of the sessions running in this session group. This value is used by AIF to return a unique one-word session identifier for this session. This value is stored in the second word. This field is reserved for system use and must never be altered by the application.

maximum ru size (SC_MRU)

This field shows the RU size that is returned.

Example:

The following session call requests to establish a synchronous session between the node named AIF501 and the remote LU named CICS. AIF assumes that the address of the SCCB is in register $B4.

```
$SINIT  ,'AIF501','CICS','AA',SYNC,NO_RESTART
```

$SINIT - Restart Session

If you are using $SINIT to restart a session, you must include the following parameters:

```
[label] $SINIT  [sccb pointer]                 P1: $B4
                [,{SYNC|ASYNC}]                P5: SC_ICT.SCRTNS
                [,RESTART]                     P6: SC_ICT.SCRSTR
                [,session id                   P7: SC_SID
                 ,msg resync send sequence     P8: SC_MRS
                 ,msg resync rec sequence]     P9: SC_MRR
```

sccb pointer

This parameter contains a pointer to the address of the SCCB to be used for this session. If not declared, the address is assumed to be in register $B4.

{SYNC|ASYNC} (SC_ICT.SCRTNS)

This parameter indicates whether execution of this call is synchronous or asynchronous.

RESTART (SC_ITC.SCRSTR)

RESTART is indicated only when the user wishes to restart an abnormally terminated session; including RESTART causes this bit to be set.

session id (SC_SID)

This two-word field is supplied by AIF after each $SINIT session call if RESTART is specified. The first word is the session group name, which is assigned by AIF to each of the sessions running in this session group. This value is used by AIF after the first $SINIT session call to return a unique one-word session identifier for this session. This value is stored in the second word. This field is reserved for system use and must never be altered by the application.

message resynchronization send sequence number (SC_MRS)

If RESTART is specified, AIF places the sequence number of the last sent message that the application program has sent in this field. This number should be stored after each send, so that it can be retrieved if a RESTART is necessary.

message resynchronization receive sequence number (SC_MRR)

If RESTART is specified, AIF places the sequence number of the last received message in this field. This number should be stored after each receive, so that it can be retrieved if a RESTART is necessary.

DESCRIPTION:

The $SINIT session call is used to restart a session in the event that it has been abnormally terminated. Restart logic and restart rules are described in detail in Section 6.

RETURN CODES

The application should check the return code after each execution of a session call. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any session call. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15.

In addition to the general return codes, the following values are possible:

Value	Label	Description
0000	RMNOER	No error
0003	RMRNEG	-RSP returned by host
0004	RMNBIF	Bind negotiation failed
0020	RMRSRF	Restart not possible
0040	RMINOD	Invalid node name
0096	RMNNAC	Node not yet active
0097	RMNLAC	Node active, but no active LUs yet
0098	RMNOAV	LUs active, but none available for this session
0099	RMISTD	Invalid STD name
009A	RMILUT	Invalid LU type in STD

If the $SINIT session call is successful (RMNOER), SC_SQN and SC_RSQ have the send/receive sequence numbers for the session.

The following AIF sense data are associated with RMRSRF:

Value	Label	Description
0001	SD0001	Restart timed out or LU released by SOPR
0002	SD0002	Session not restartable type
0004	SD0004	Restart mismatch; synchronous point records do not match

If RESTART is successful, the application should examine the output control word (SCCB.SC_OCT) for the following indicators. If the bit is on, the condition described is true.

SCRSTS: STSN received for message resynchronization; application should store current value of send and receive sequence numbers

SCL6RX: DPS 6 or DPS 6 PLUS application must retransmit last full message

SCHORX: Host application must retransmit last full message; receive required of DPS 6 or DPS 6 PLUS application.

Example:

The following session call requests AIF to restart the above session after it has been abnormally terminated. AIF assumes that the address of the SCCB is in $B4 and uses the send/receive sequence numbers from the SCCB.

```
    $SINIT  ,,,,,RESTART
```

$SPOLL - Poll Session

The $SPOLL session call checks to see if any LU associated with the application program's task group has been attached by the remote program.

FORMAT:

```
[label]  $SPOLL  [sccb pointer]    P1: $B4
                 [,node name]      P2: SC_NOD
                 [,std name]       P3: SC_STD
```

ARGUMENTS:

sccb pointer

This parameter contains the address of the SCCB to be used for this session. The sccb pointer used for a $SPOLL must be unique and should not be currently used by an active session. If not declared, the address is assumed to be in register $B4.

node name (SC_NOD)

Identifies the AIF node to which the application is directing this session call. This field contains eight alphanumeric characters. If you are loading the SCCB yourself and your node name has fewer than eight characters, this field must be left-justified and space-filled.

std name (SC_STD)

The configured Session Type Descriptor (STD) which lists the attributes of the session to be established. This field consists of two alphanumeric characters.

DESCRIPTION:

The $SPOLL session call causes AIF to test to see if any LU associated with the application programmer's task group has been attached (bound) by the remote program. The $SPOLL session call is similar to the $SACPT session call, except that the $SPOLL does not cause a connection between AIF and the application program if a bound LU is found.

NOTE

This call is always made synchronously.

RETURN CODES:

The application should check the return code after each execution of a session call. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any session call. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15.

In addition to the general return codes, the following values are possible:

Value	Label	Description
0005	RMLUAT	Indicates that there is an LU being bound
0040	RMINOD	Invalid node name
0099	RMISTD	Invalid STD name
009B	RMNOAT	No LU attached for $SPOLL

$SRECV - Receive Message

The $SRECV session call causes AIF to deliver to the application's buffer a message or message segment from the session partner.

FORMAT:

```
  [label] $SRECV  [sccb pointer]              P1: $B4
                  [,receive data buffer]      P2: SC_BUF
                  [,rec'v buffer length]      P3: SC_DLG
                  [,{R|L}]                    P4: SC_ICT.SCRHBI
                  [,{SYNC|ASYNC}]             P5: SC_ICT.SCRTNS
                  [,{MSG|M_SEG}]              P6: SC_ICT.SCRMSG
```

ARGUMENTS:

sccb pointer

This parameter contains the address of the SCCB to be used for this session. If not declared, the address is assumed to be in register $B4.

receive data buffer (SC_BUF)

A pointer to the application's receive buffer.

receive data buffer length (SC_DLG)

The length of the receive buffer in bytes. The maximum allowable length of this buffer is 32,767 bytes.

{R|L} (SC_ICT.SCRHBI)

Specifies whether data starts on the left (L) or right (R) byte of the buffer address word.

{SYNC|ASYNC} (SC_ICT.SCRTNS)

This parameter indicates whether the execution of this call is synchronous or asynchronous.

{MSG|M_SEG} (SC_ICT.SCRMSG)

Specifying MSG indicates that a complete message (whole chain of request units) is to be delivered to the application's buffer. If M_SEG is specified, single request units are delivered to the application's buffer. When the last message segment is delivered, AIF sets the end of message bit in the output control word (SCREOM).

DESCRIPTION:

The $SRECV session call causes AIF to deliver a message or message segment (request unit) to the application's buffer from the session partner.

If the user specifies MSG, then AIF assembles the chain before delivery. If the user's buffer is not large enough, the message is not delivered; the actual length of the message or message segment is returned to the application. The application can either re-execute the receive with an adequate buffer, or re-execute the receive specifying M_SEG.

NOTE

If a RESTART of this session is a possibility, then the receive sequence number should be stored by the application executing this $SRECV session call.

RETURN CODES:

The application should check the return code after each execution of a session call. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any session call. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15.

In addition to the general return codes, the following values are possible:

Value	Label	Description
0000	RMNOER	No error
0010	RMIMPS	Improper State
0013	RMRB2S	Receive buffer too small
0032	RMDTCL	Send/receive rejected; data traffic cleared or inactive

Received Interrupt Type (SC_INT)

This field contains the interrupt type if one is received during the execution of this session call.

Error Code or Sense Data Received (SC_ESD)

This field can contain either detailed information about an error condition or sense data from a remote LU, if a negative response has been received.

Receive Data Buffer Length (SC_ADL)

This field contains the actual length of the received data in bytes.

Output control word (SC_OCT)

This field contains certain indicators that are of interest after a successful $SRECV session call. When one or more of these bits is set, the condition described is true.

Value	Label	Description
8000	SCRWRP	Reply requested (CD)
4000	SCRRQD	Definite response required (RQD)
2000	SCRLST	LAST message received (EB)
1000	SCRFMH	Function management header (FMH)
0200	SCREOM	End of message (EC)
0400	SCRBOM	Beginning of message (BC)

Special notice should be given to the situation where an interrupt was received prior to or during the execution of a $SRECV session call. Two situations are possible:

1. An interrupt was received before the execution of the $SRECV session call. In this case, the application is given the data if it was in the receive queue and the application is also informed of the interrupt. The return code is either RCRINT+RCSCNL (X'3000') or RCRINT+RCSCMP (X'2800'), depending on whether or not there was data in the receive queue.

2. An interrupt is received during the execution of a $SRECV session call. In this case, the order is not completed and return is made to the application with a return code RCRINT+RCSCNL (X'3000').

Example:

The following example causes AIF to deliver an assembled asynchronous message to the application's buffer, which is 256 bytes long, left-byte aligned. The values for parameters 1 and 2 remain as they were prior to issuing this session call.

```
$SRECV ,,=256,L,ASYNC,MSG
```

$SRI - Read Interrupt

The $SRI session call reads interrupt information from the host or control information from the AIF LU when there is no other AIF session call outstanding.

FORMAT:

```
[label]  $SRI  [sccb pointer]   P1: $B4
```

ARGUMENT:

sccb pointer

This parameter contains the address of the SCCB to be used for this session. If not declared, it is assumed to be in register $B4.

DESCRIPTION:

The $SRI session call enables the application to read interrupt information from the host or control information from AIF when there is no other AIF session call outstanding.

If either of the following situations occurs, the condition is reported to the application, the SCCB is updated the same way as for the $STEST or $SWANY session call and a return is made to the application.

As with any asynchronous call, the application must execute a $SWANY or $STEST session call to determine when the $SRI session call is complete and regain control.

1. When an interrupt is received, the Received Interrupt Type and the Error Code Or Sense Data Received fields in the SCCB contains the appropriate information.

2. If data has been received for which there is no outstanding order, the user must issue a $SRECV session call to gain access to this data. The length of the received data is in SC_ADL.

NOTE

The $SRI session call is always made asynchronously.

RETURN CODES

The application should check the return code after each execution of a session call. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any session call. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15.

In addition to the general return codes, the following values are possible:

Value	Label	Description
0002	RMDRNR	Data received but no read
0010	RMIMPS	Improper state
0032	RMDTCL	Send/receive reject; data traffic cleared/inactive

Received Interrupt Type

This field contains the interrupt type if one is received during the execution of this session call.

Error Code or Sense Data Received

This field contains either detailed information about an error condition or sense data if received from a remote LU.

Example:

This session call allows the application to read interrupt information from the host when there is no other session call outstanding. This example assumes that register $B4 has previously been loaded with the address of the SCCB.

```
RDINT   $SRI
```

$SSEND - Send Message

The $SSEND session call sends a message (chain) or message segment (RU) to a session partner.

FORMAT:

```
[label]  $SSEND  [sccb pointer]           P1: $B4
                 [,send data buffer]      P2: SC_BUF
                 [,send buffer length]    P3: SC_DLG
                 [,{R|L}]                 P4: SC_ICT.SCRHBI
                 [,{SYNC|ASYNC}]          P5: SC_ICT.SCRTNS
                 [,{REPLY|RLCLR|          P6: SC_ICT.SCSWRP
                    LAST}]                  : SC_ICT.SCSLST
                 [,{MNTCMP|MCMP}]         P7: SC_ICT.SCSMNC
                 [,{FMH|NOFMH             P8: SC_ICT.SCSFMH
                 [,{RQD|RQE}]             P9: SC_ICT.SCSRQD
```

ARGUMENTS:

sccb pointer

This parameter contains the address of the SCCB to be used for this session. If not declared, the address is assumed to be in register $B4.

send data buffer (SC_BUF)

A pointer to the application's data buffer.

send data buffer length SC_DLG)

The length of the data in bytes. The maximum buffer size is 32,767 bytes.

{R|L} (SC_ICT.SCRHBI)

This argument specifies whether data starts on the left or right byte of the buffer address word. The user specifies R|L.

{SYNC|ASYNC} (SC_ICT.SCRTNS)

This parameter indicates whether execution of the call is synchronous or asynchronous.

{REPLY|RLCLR} (SC_ICT.SCSWRP)
LAST (SC_ICT.SCSLST)

REPLY indicates to the application to send with reply requested (set change direction indicator in request header). This parameter is meaningful only when you are sending the last message segment or a chain.

The LAST parameter causes AIF to flag the last message (set end bracket indicator in request header). This parameter is meaningful only at the beginning of a message (chain). This option is only valid with IMS applications.

RLCLR clears both the REPLY and the LAST bits in the input control word.

{MNTCMP|MCMP} (SC_ICT.SCSMNC)

MNTCMP indicates that the message chain is not complete. MCMP resets this indicator in the input control word.

{FMH|NOFMH} (SC_ICT.SCSFMH)

This parameter bit indicates that data is to be sent with Function Management Header in Request/Response Unit.

{RQD|RQE} (SC_ICT.SCSRQD)

RQD sends a messaage and requests a definite response. RQE sends a message and requests an exception response.

DESCRIPTION:

The $SSEND session call instructs the sending of a message (chain) or message segment (RU) to the session partner. Special notice should be given to the situation where the application is executing a $SSEND session call but an interrupt is received before or during the execution of the session call.

When you are sending an entire message (chain), use the MCMP parameter. When sending message segments, use MCTCMP, except for the last segment, with which you use MCMP.

If an interrupt has already been received when the $SSEND session call is executed, the application is informed of the interrupt. If an interrupt is received during the execution of the $SSEND session call, the $SSEND session call completes, and when the application executes the $SWANY or $STEST session call, return is made to the application. The return code indicates the interrupt received and the result of the $SSEND session call.

NOTE

If RESTART of this session is a possibility, then the send sequence number and the entire message must be saved by the application executing this $SSEND session call.

RETURN CODES

The application should check the return code after each execution of a session call. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any session call. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15. In addition to the general return codes, the following values are possible:

Value	Label	Description
0000	RMNOER	No error
0003	RMRNEG	-RSP returned by host; application should examine sense data
0010	RMIMPS	Improper State
0012	RMIRHI	Invalid input control indicators; application should examine sense data
0032	RMDTCL	Send/receive rejected; data traffic cleared or inactive

The following AIF sense data are associated with RMIRHI:

Value	Label	Description
0828	SD0828	Reply not possible, session partner quiesced
4004	SD4004	LAST not allowed for this session
4040	SD4040	REPLY or LAST required

Received Interrupt Type

This field contains the interrupt type if one is received for the application during the execution of this session call.

Error Code or Sense Data Received

This field can contain detailed information about an error condition or sense data from a remote LU.

Example:

The following session call sends a whole message of 256 bytes with left byte alignment with FM header. This $SSEND session call is the first and only $SSEND session call for this message. This $SSEND session call is executed asynchronously and requests a definite response.

```
$SSEND ,,256,L,ASYNC,RLCLR,MCMP,FMH,RQD
```

$SSI - Send Interrupt

The $SSI session call is used to send Data Flow Control commands to the session partner or to pass control information to the System Service Control Point or to AIF.

FORMAT:

```
[label] $SSI  [sccb pointer]           P1: $B4
              [,send data buffer]      P2: SC_BUF
              [,send buffer length]    P3: SC_DLG
              [,{R|L}]                 P4: SC_ICT.SCRHBI
              [,type]                  P5: SC_SIN
              [,{REPLY                 P6: SC_ICT.SCSWRP
                 LAST}]                    SC_ICT.SCSLST
              [,sense data]            P7: SC_SSD
```

ARGUMENTS:

sccb pointer

This parameter contains the address of the SCCB to be used for this session. If not declared, the address is assumed to be in register $B4.

send data buffer (SC_BUF)

A pointer to the application's send data buffer. This parameter is required only if you are sending CNM data.

send data buffer length (SC_DLG)

The length in bytes of the send data in the buffer. The maximum allowable size is the MAXIMUM RU SIZE which has been configured minus three. This parameter is required only if you are sending CNM data.

{R|L} (SC_ICT.SCRHBT)

This argument specifies whether data starts on the left (L) or right (R) byte of the buffer address word. This parameter is required only if you are sending CNM data.

type (SC_SIN)

This field contains the interrupt type for this send. Refer to the $SCCB template (Appendix E) for possible values for this field.

REPLY (SC_ICT.SCSWRP)
LAST (SC_ICT.SCSLST)

If the application specifies LAST, the end bracket indicator is set.

If the application specifies REPLY, the change direction indicator is set.

sense data (SC_SSD)

This field contains the sense data if the specific interrupt type calls for it. If the application places the sense data in registers $R6 and $R7, then this parameter is specified as register $R7 or =$R7. If the literal sense data value is included for this parameter, then it must be in a form acceptable as the operand of an LDI instruction, such as, =Z'08240000'.

DESCRIPTION:

The $SSI session call is used to send the following three types of information:

1. Send data flow control commands to the session partner

2. Pass control information to AIF (e.g., enable/disable restart).

3. Pass statistical information to SSCP.

The format of the buffers that you create to send CNM alerts and maintenance statistics are detailed in Section 6.

NOTE

The $SSI session call is always made synchronously.

RETURN CODES:

The application should check the return code after each execution of a session call. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any session call. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15.

In addition to the general return codes, the following values are possible:

Value	Label	Description
0000	RMNOER	No error
0003	RMRNEG	-RSP returned by host; application should examine sense data
0010	RMIMPS	Improper State
0012	RMIRHI	Invalid input control indicators; application should examine sense data.
0015	RMIINT	Invalid Interrupt Type
0016	RMICOD	Invalid status word or user code
0032	RMDTCL	Send/receive rejected; data traffic cleared or inactive

The following sense data are associated with RMIRHI:

Value	Label	Description
0828	SD0828	Reply not possible, session partner quiesced
4004	SD4004	LAST not allowed for this session
4040	SD4040	REPLY or LAST required

Received Interrupt Type (SC_INT)

This field contains an interrupt type if one is received during the execution of this session call.

Error Code or Sense Data Received (SC_ESD)

This field contains either detailed information about an error condition or sense data if received from a remote LU.

Example:

The following session call sends a data flow control command, LUSTAT, with change direction indicator and the sense data 0824 to the session partner. (LUSTAT is a label whose value is found in the SCCB.)

```
    SNDINT  $SSI  ,,,,=LUSTAT,REPLY,=Z'08240000'
```

$SSRSP - Send Response

The $SSRSP session call requests that AIF send a response to a previous message which requires a response.

FORMAT:

```
[label]  $SSRSP  [sccb pointer]              P1: $B4
                 [,{SYNC|ASYNC}]             P2: SC_ICT.SCRTNS
                 [,{PRSP                     P3: SC_ICT.SCSRSP
                    NRSP                       : SC_ICT.SCSNEG
                    WAIT_FOR_RTR|NO_RTR}]
                 [,sense]                    P4: SC_SSD·
```

ARGUMENTS:

sccb pointer

This parameter contains the address of the SCCB to be used for this session. If not declared, the address is assumed to be in register $B4.

{SYNC|ASYNC} (SC_ICT.SCRTNS)

This parameter indicates whether execution of this call is synchronous or asynchronous.

NRSP (SC_ICT.SCSNEG)
PRSP (SC_ICT.SCSRSP)

This argument indicates whether to send a positive response or a negative response.

If a negative response is indicated (NRSP), the LU sends a negative response accompanied by whatever sense data is found in the SCCB. If the user wishes no sense data to be sent, he must provide a sense data of nulls.

{WAIT_FOR_RTR|NO_RTR}

If the data flow control command BID is rejected by the application program, this parameter indicates whether the session partner should wait for the Ready to Receive (RTR) or if none is to be sent.

If WAIT_FOR_RTR is indicated, AIF sends a negative response with sense data Z'0814'; if NO_RTR is indicated, AIF sends a negative response with sense data Z'0813'.

sense (SC_SSD)

This four-byte field provides sense data if NRSP is specified. If no sense data is to be sent, this field should be set to nulls by the application.

DESCRIPTION:

The $SSRSP session call sends either a negative or a positive response to a previous message on behalf of the application. If the response is negative, the application also has the option of sending sense data.

RETURN CODES:

The application should check the return code after each execution of a session call. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any session call. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15.

In addition to the general return codes, the following values are possible:

Value	Label	Description
0000	RMNOER	No error
0010	RMIMPS	Improper State
0012	RMIRHI	Invalid input control indicators; application should examine sense data
0032	RMDTCL	Send/receive rejected; data traffic cleared

The following AIF sense data are associated with RMIRHI:

Value	Label	Description
4041	SD4041	Response type improperly indicated

Example:

The following session call sends a negative response on behalf of the application and sets the sense data to nulls.

```
$SSRSP ,,NRSP,=0000
```

$STERM - Terminate Session

The $STERM session call terminates the AIF session.

FORMAT:

```
[label] $STERM   [sccb pointer]        P1: $B4
                 [,{NORM|ABNORM}]      P2: SC_ICT.SCATRM
```

ARGUMENTS:

sccb pointer

This parameter contains the address of the SCCB to be used for this session. If not declared, the address is assumed to be in register $B4.

{NORM|ABNORM} (SC_ICT.SCATRM)

NORM or ABNORM indicates to the host the reasons for which this session is being terminated

DESCRIPTION

The $STERM session call terminates the AIF session. Termination can be either normal or abnormal. Whether it is normal or abnormal is indicated by a parameter within the $STERM session call.

- If the $STERM indicates normal termination, an orderly termination message is sent to the session partner's LU.
- If the $STERM indicates abnormal termination, the following events occur:
  - The AIF LU terminates the session.
  - AIF sends an abnormal termination message to inform the host LU.

  After the session is terminated, the LU task is again available for other users.

Abnormal termination can be issued at any time; the last session call is cancelled if it is not completed.

NOTE

The $STERM session call is always made synchronously.

RETURN CODES:

The application should check the return code after each execution of a session call. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any session call. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15.

In addition to the general return codes, the following values are possible:

Value	Label	Description
0000	RMNOER	No error
0010	RMIMPS	Improper State (only applies to normal termination)

The following sense data are associated with RMIMPS:

Value	Label	Description
200D	SD200D	Response required
2040	SD2040	Normal termination rejected; data on receive queue
2041	SD2041	Transaction not completed yet

Received Interrupt Type

This field contains the interrupt type, if one is received during the execution of a normal termination.

Error Code or Sense Data Received

This field contains either detailed information about an error condition or sense data if received from a remote LU.

Example:

The following session call causes the AIF session to terminate normally.

```
DONE    $STERM   ,NORM
```

$STEST - Test for Events

The $STEST session call tests conditions for the session whose SCCB is pointed to by register $B4.

FORMAT:

```
[label]   $STEST  [sccb pointer]     P1: $B4
```

ARGUMENT:

sccb pointer

This parameter contains the address of the SCCB to be used for this session. If not declared, this address is assumed to be in register $B4.

DESCRIPTION:

This session call tests conditions for the session whose SCCB is pointed to by register $B4. Executing this session call causes AIF to immediately report to the application one of the following conditions in register $R1 and SCCB:SC_RCD:

1. No event
2. Interrupt received
3. Asynchronous order completed or cancelled
4. Permission to send after a send was rejected due to data traffic inactive or pacing
5. Data has been received for which there is no outstanding order.

Conditions 2 and 3 can coexist.

If an interrupt was received, the Received Interrupt Type and the Error Code Or Sense Data Received fields in the SCCB contain information pertaining to the type of interrupt.

If an asynchronous order were completed or cancelled, then AIF delivers the return code of the completed order immediately and the application must examine all pertinent fields in the SCCB.

If data has been received for which there is no outstanding order, the user must issue a $SRECV session call to gain access to this data. Nothing is delivered to the user as a result of the $STEST session call, but the length of the received data is found in the SC_ADL of the SCCB.

NOTE

The $STEST session call can be executed while an asynchronous call is outstanding. This session call is always made synchronously. If there were an asynchronous order outstanding, the condition is tested, reported, and the order remains outstanding. Once the test determines that the order has been completed, the call is no longer outstanding.

RETURN CODES:

The application should check the return code after each execution of a session call. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any session call. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15.

In addition to the general return codes, the following values are possible:

Value	Label	Description
0000	RMNOEV	No Event
0001	RMPTSN	Permission to send
0002	RMDRNR	Data received but no read

Received Interrupt Type (SC_INT)

This field contains the interrupt type, if there is one during the execution of this session call.

Error Code or Sense Data Received (SC_ESD)

This field contains either detailed information about an error condition or sense data if received from a remote LU.

Receive Data Buffer Length (SC_ADL)

This field contains the actual length of the received data in bytes.

Example:

This session call tests the status of the session indicated by the SCCB to which register $B4 is pointing.

```
CHECK   $STEST
```

$SWANY - Wait on Events

The $SWANY session call causes AIF to issue a system "wait any" on behalf of the application. The application remains dormant until one of the requests is complete.

FORMAT:

```
[label]     $SWANY
```

ARGUMENT:

This session call has no arguments.

DESCRIPTION:

The $SWANY session call causes execution of the application program to be suspended until any asynchronous request terminates. Asynchronous requests other than AIF requests also cause control to return to the $SWANY session call executor providing that the P-bit in the request block was set by the executor prior to the execution of the $SWANY macrocall.

Unless you have an outstanding call, you should not issue a $SWANY session call. If you do issue a $SWANY session call with no outstanding asynchronous call, AIF returns an RCNOUT return code to indicate that there are no orders outstanding.

If an application had more than one session established, with outstanding asynchronous orders on multiple sessions, executing a $SWANY session call returns control to the application with register $B4 containing the SCCB address of the session whose request has completed.

RETURN CODES:

The application should check the return code after each execution of a session call. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any session call. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15.

In addition to the general return codes, the $SWANY session call can return return codes according to the following conventions:

1. If, after a $SWANY session call is executed, register $B4 contains the address of the SCCB, then register $R1 contains the AIF session call return code.

2. If, after a $SWANY session call is executed, register $B4 contains the address of the terminated request block, then register $R1 contains the completion status for that request block.

Upon return, registers $R1 and $B4 contain the following information:

	AIF Call Complete	Other Asynchronous Call Complete
$B4	ADDRESS OF SCCB	ADDRESS OF TERMINATED REQUEST BLOCK
$R1	AIF CALL RETURN CODE	POSTED COMPLETION STATUS OF COMPLETED REQUEST BLOCK

NOTE

This session call is always made synchronously. When this call is executed, AIF issues a "wait any" on behalf of the application. The application program remains dormant until one of the requests is complete. If an application does any asynchronous AIF processing, the application should never execute a $WAITA. This command causes unspecified results.

Example:

This session call causes the application program to remain dormant until an asynchronous request terminates.

```
    WAIT    $SWANY
```

$SACEB - ASCII-To-EBCDIC Conversion Routine

Converts data from ASCII to EBCDIC.

FORMAT:

```
label    $SACEB
```

ARGUMENT:

There are no arguments associated with this macro.

DESCRIPTION:

These session calls convert data from ASCII to EBCDIC. Since IBM handles data in EBCDIC and DPS 6 or DPS 6 PLUS handles it in ASCII, you may sometimes wish to convert data from one to the other, either before sending or before receiving.

The Application Interface Facility software provides the following macros to perform these conversions.

$SACEB	ASCII-To-EBCDIC Conversion

When this macro is activated, you must initialize registers $B2, $B4, $R2, $R4, and $R6 to contain the values listed in Table 2-2. If you wish to convert in place, $B2=$B4.

Table 2-2. Register Contents at Conversion

Register	Contents
$B2	Pointer to buffer to be converted
$B4	Pointer to buffer to contain converted data
$R2	Index for buffer to be converted
$R3	Function code ($SACEB=1; $SEBAC=2)
$R4	Index for buffer to contain converted data
$R6	Length of data in bytes

NOTE

The maximum length of data that can be converted by a single call is 32,767 bytes.

$SEBAC - EBCDIC-To-ASCII Conversion Routine

Converts data from EBCDIC to ASCII.

FORMAT:

```
    label   $SEBAC
```

ARGUMENT:

There are no arguments associated with this macro.

DESCRIPTION:

These session calls convert data from EBCDIC to ASCII. Since IBM handles data in EBCDIC and DPS 6 or DPS 6 PLUS handles it in ASCII, you may sometimes wish to convert data from one to the other, either before sending or before receiving.

The Application Interface Facility software provides the following macros to perform these conversions.

$SEBAC	EBCDIC-To-ASCII Conversion

When this macro is activated, you must initialize registers $B2, $B4, $R2, $R4, and $R6 to contain the values listed in Table 2-2. If you wish to convert in place, $B2=$B4.

Table 2-3. Register Contents at Conversion

Register	Contents
$B2	Pointer to buffer to be converted
$B4	Pointer to buffer to contain converted data
$R2	Index for buffer to be converted
$R3	Function code ($SACEB=1; $SEBAC=2)
$R4	Index for buffer to contain converted data
$R6	Length of data in bytes

NOTE

The maximum length of data that can be converted by a single call is 32,767 bytes.

Section 3 *PROGRAMMING LU TYPE 0 SESSIONS IN COBOL*

This section describes the session calls that the COBOL programmer uses to converse over a Session Type 0 with host transaction programs. Topics include:

- COBOL session calls
- Session call format
- Programming Considerations
  - Synchronous vs. Asynchronous Processing
  - WORKING-STORAGE SECTION
  - Checking the Return Code
- Session Calls.

COBOL SESSION CALLS

The session calls used by the Application Interface Facility (AIF) in a COBOL application program call corresponding Assembly language subroutines using the "CALL...USING..." verb. These calls are listed in Table 3-2.

The parameters that the session calls use are positional. They are defined in the WORKING-STORAGE SECTION of the COBOL program. In this manual, these parameters are defined in the discussion of the WORKING-STORAGE SECTION and are listed without redefinition in the format description of each session call.

At the completion of each session call, when control is returned to the application, a return code is placed in the RETURNS field. This return code indicates whether a call has been completed error free. The application should check the return code after each session call to verify that the call was completed error-free.

A sample COBOL program is provided in Appendix C to demonstrate the use of the AIF session calls in a COBOL program.

SESSION CALL FORMAT

The session calls used by AIF in a COBOL program reference Assembly language subroutines which include system-provided macrocalls. The COBOL session calls have a list of arguments that must be specified each time a session call is executed. These arguments, which you have defined in the WORKING-STORAGE SECTION, correspond to parameters in the SCCB that are used by the Assembly language subroutine. The AIF COBOL session calls follow the conventions for COBOL (described in detail in the ONE PLUS COBOL 74 Language Reference Manual (HE34).

When an AIF session call is activated, it defines one or more pools of logical units. Each pool of logical units is reserved for a specific host CICS or IMS system. AIF can either start a session to the host system at initiation or it can wait for an application to request a session. The time of session initiation is a configuration option.

An application requests to initiate a session with a reserved LU by executing the CSINIT session call. AIF checks the availability of that LU and assigns it if it is available. If the specified LU is unavailable, AIF checks first for an available reserved LU, second for an available preestablished LU, and then for any available LU to assign to the session. AIF either returns the address of the LU with which the session is started, or rejects the request if there is no LU available.

An application gains access to a host-initiated session by executing a CSACPT session call. Executing the accept session call causes the application to be connected to the host-initiated session and causes the LU to send a positive response to the host, accepting the session.

PROGRAMMING CONSIDERATIONS

The special considerations that the COBOL programmer must bear in mind fall into the following categories:

- Synchronous vs. Asynchronous Processing
- WORKING-STORAGE SECTION
- Host-initiated sessions
- Linking the program
- Checking the return code.

Synchronous vs. Asynchronous Processing

AIF session calls can be processed either synchronously or asynchronously.

SYNCHRONOUS PROCESSING

Synchronous processing implies that when the application passes an instruction to AIF for processing, it waits for the application to complete that instruction before continuing.

In Figure 3-1, a CSINIT session call has been issued synchronously. The application completes its segment of processing and passes the request to AIF. AIF executes the CSINIT session call completely and passes the return code to the application. The application does not process other instructions while AIF is executing the CSINIT session call.

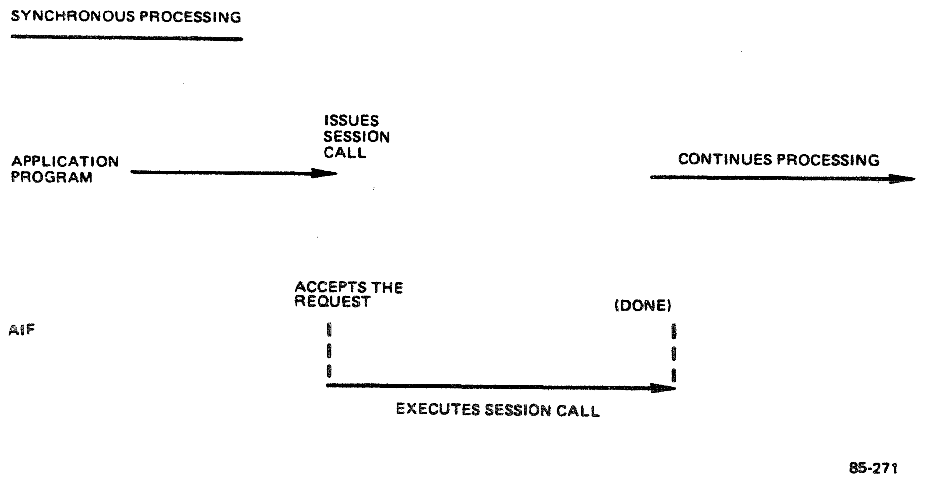


Figure 3-1. Synchronous Processing

ASYNCHRONOUS PROCESSING

Asynchronous processing implies that when the application passes an instruction to AIF for processing, the application continues to process other instructions while it waits for AIF to complete that instruction.

In Figure 3-2, a CSINIT session call has been issued asynchronously. The application completes its segment of processing and passes the request to AIF. While AIF executes the CSINIT session call, the application is processing other instructions. In order for the application to find out that AIF has finished executing the CSINIT session call, the application must execute a CSWANY or a CSTEST session call.

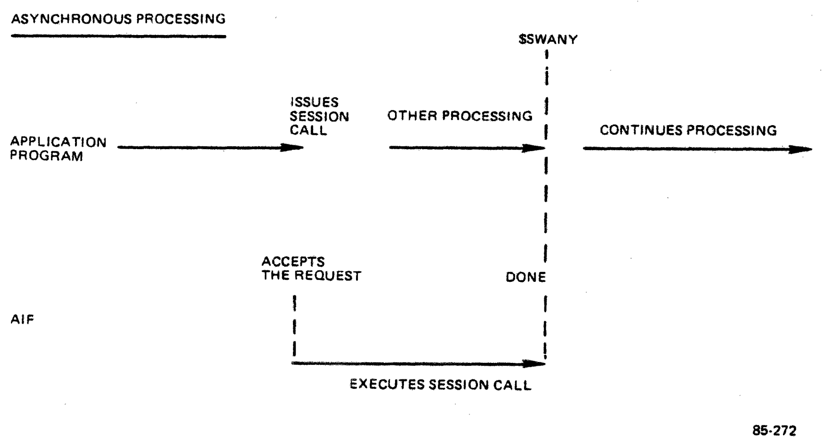


Figure 3-2. Asynchronous Processing

Each time you issue an asynchronous order, you must check the receive queue before you can receive information. You can do this by either the CSTEST or the CSWANY session call. These two session calls differ as follows:

1. The CSTEST session call checks to see if there is information in the queue to be received and immediately reports back to the application. This call can be executed any time you wish to check for an outstanding order, and as often as you wish to check, because the application regains control immediately after the test is completed.

2. The CSWANY session call checks for information on the queue and waits until there is information waiting before it returns control to the application.

WORKING-STORAGE SECTION

The WORKING-STORAGE SECTION defines the area to be used as the SNA work area. The parameters specified in these fields are passed to the SCCB when the session calls are executed.

The following parameters must be defined in the WORKING-STORAGE SECTION. These parameters are used to create the session call control block which is used by the Assembly language subroutines you are calling.

Figure 3-3 shows a sample WORKING-STORAGE SECTION in which the SNA work area has been defined. The data-names that are used here are examples; you can name them according to your own naming conventions.

```
DATA DIVISION.
WORKING-STORAGE SECTION.
77   SNA-WORK-AREA                 PIC X(200).
77   NODE-NAME                     PIC X(8) VALUE "AIF501".
77   REMOTE-LU-NAME                PIC X(8) VALUE "A06CICS".
77   STD-NAME                      PIC XX VALUE "BB".
77   SYNC-CALL                     PIC X VALUE "S".
77   ASYNC-CALL                    PIC X VALUE "A".
77   RESTART                       PIC X VALUE "R".
77   NO-RESTART                    PIC X VALUE "N".
77   SESSION-ID                    PIC X(4).
77   MSG-RESYNC-SEND-SQN           PIC 9(5) VALUE 0.
77   MSG-RESYNC-RCV-SQN            PIC 9(5) VALUE 0.
77   SEND-BUFFER                   PIC X(80).
77   SEND-BUFFER-SIZE              PIC 9(5) VALUE 80.
77   DATA-BUFFER-ALIGNMENT         PIC X VALUE "L".
77   REPLY-REQUEST                 PIC X VALUE "R".
77   MSG-COMPLETE                  PIC X VALUE "Y".
77   FMH                           PIC X VALUE "N".
77   RQD                           PIC X VALUE "N".
77   RECEIVE-BUFFER                PIC X(80).
77   RECEIVE-BUFFER-SIZE           PIC 9(5) VALUE 80.
77   MSG                           PIC X VALUE "Y".
77   RECEIVED-DATA-LENGTH          PIC 9(5) VALUE 0.
77   INTERRUPT-DATA-LENGTH         PIC 9(5) VALUE 0.
77   WORK-AREA-ID                  PIC X(4).
77   SEND-RESPONSE-TYPE            PIC X VALUE "-".
77   SENSE-DATA                    PIC X(8).
01   RETURNS.
     02 RETURN-A.
        03   SESSION-ABORT         PIC X VALUE "N".
        03   STOP-RCVD             PIC X VALUE "N".
        03   INTRPT-RCVD           PIC X VALUE "N".
        03   SERV-REQ-CANC         PIC X VALUE "N".
        03   SERV-REQ-COMP         PIC X VALUE "N".
        03   COBOL-ERROR           PIC X VALUE "N".
     02 RETURN-B                   PIC 9(4) VALUE 0.
77   INTERRUPT-TYPE                PIC 99 VALUE 0.
77   RCVD-SENSE                    PIC X(8).
01   TIMEOUT.
     02 DATE1.
        03   YY                    PIC 99 VALUE 0.
        03   MM                    PIC 99 VALUE 0.
        03   DD                    PIC 99 VALUE 0.
     02 TIME1.
        03   HH                    PIC 99 VALUE 0.
        03   MN                    PIC 99 VALUE 0.
        03   SSSS                  PIC 9(4) VALUE 0.
77   TERMINATE-TYPE                PIC X VALUE "N".
77   GET-ATTR-TYPE                 PIC 99 VALUE "01".
```

Figure 3-3. WORKING-STORAGE SECTION for AIF

```
01  OUTPUT-CONTROL-WORD.
    02 REPLY-REQUESTED-CD      PIC X VALUE "N".
    02 DEFINITE-RESP-REQ       PIC X VALUE "N".
    02 LAST-MSG-RCVD-EB        PIC X VALUE "N".
    02 FMH-IN-RCVD-DATA        PIC X VALUE "N".
    02 BEGIN-MSG-RCVD-BC       PIC X VALUE "N".
    02 END-MSG-RCVD-EC         PIC X VALUE "N".
    02 SET-SEND-RECV-SEQ       PIC X VALUE "N".
    02 APPL-RESEND-REQUIRED    PIC X VALUE "N".
    02 HOST-RESEND-REQUIRED    PIC X VALUE "N".
77  CONVERT-FROM-FIELD         PIC X(20).
01  CONVERT-FROM-LEFT-POSIT    COMP-1 VALUE 1.
77  CONVERT-TO-FIELD           PIC X(20).
01  CONVERT-TO-LEFT-POSIT      COMP-1 VALUE 6.
01  CONVERSION-LENGTH          COMP-1 VALUE 10.
```

Figure 3-3 (cont.). WORKING-STORAGE SECTION for AIF

These fields are defined as follows:

SNA-WORK-AREA

This input parameter is the name of a contiguous memory area that is at least 200 bytes long. This corresponds to the "sccb pointer" argument of the Assembly language session calls.

If your program will be running multiple sessions, you must define a unique SNA-WORK-AREA for each session.

Example:

```
77  SNA-WORK-AREA      PIC X(200).
```

NODE-NAME

This input parameter contains the name of the AIF node on the DPS 6 or DPS 6 PLUS system with which the session is being established. This field contains up to eight alphanumeric characters.

Example:

```
77  NODE-NAME          PIC X(8)  VALUE "SNANODE1".
```

REMOTE-LU-NAME

This input parameter contains the name by which the remote LU is known to this application. This field contains up to eight alphanumeric characters. The REMOTE-LU-NAME equates to the APPL VTAM macro on the host.

Example:

```
77  REMOTE-LU-NAME    PIC X(8)  VALUE "A06CICS ".
```

STD-NAME

This input parameter contains the two alphanumeric character field which is the session type descriptor name. The STD is defined in the AIF configuration file.

Example:

```
77  STD-NAME          PIC X(2) VALUE "BB".
```

SYNC-CALL
ASYNC-CALL

These input parameters indicate whether execution of the call is to be synchronous or asynchronous. Each field contains one character, either S or A. Both parameters must be included.

Example:

```
77  SYNC-CALL         PIC X VALUE "S".
77  ASYNC-CALL        PIC X VALUE "A".
```

RESTART
NO-RESTART

These input parameters indicate whether or not the application wishes to restart an abnormally terminated session. Each field contains one character, either R or N. Both parameters must be included.

Example:

```
77  RESTART           PIC X VALUE "R".
77  NO-RESTART        PIC X VALUE "N".
```

SESSION-ID

If RESTART is specified, AIF places a unique session identifier in this field before returning control to the application. This field contains four system-supplied, alphanumeric characters that can be used to restart an abnormally terminated session. This field should be stored if restart is a possibility or if you plan to execute multiple sessions.

Example:

```
77  SESSION-ID        PIC X(4).
```

MSG-RSYNC-SEND-SQN

If RESTART is specified, AIF places the sequence number of the last message sent in this field each time the application does a send. This field contains up to five numeric characters and should be stored after each send in case a RESTART is necessary.

Example:

```
77  MSG-RSYNC-SEND-SQN   PIC 9(5) VALUE 0.
```

MSG-RSYNC-RCV-SQN

If RESTART is specified, AIF places the sequence number of the last message that the application has received in this field each time the application issues a receive. This field can be up to five numeric characters and should be stored after each receive so that it can be retrieved if a RESTART is necessary.

Example:

```
77  MSG-RSYNC-RCV-SQN    PIC 9(5) VALUE 0.
```

SEND-BUFFER

This input parameter sets up the buffer for the messages to be sent. It can contain up to 32,767 characters. If the data in the send/receive buffers must be converted between ASCII and EBCDIC, the application must take care of the conversion. Two macros are provided for this purpose, CSACEB and CSEBAC.

Example:

```
77  SEND-BUFFER       PIC X(80).
```

SEND-BUFFER-SIZE

This input parameter contains the length of the send data buffer. The maximum buffer size is 32,767 bytes.

Example:

```
77  SEND-BUFFER-SIZE            PIC 9(5) VALUE 80.
```

DATA-BUFFER-ALIGN

This input parameter specifies whether data starts in the left (L) or right (R) byte of the buffer address word.

Example:

```
77  DATA-BUFFER-ALIGN           PIC X VALUE "L".
```

REPLY-REQUEST

This input parameter indicates whether the message being sent is now complete (L), and if the application expects a reply to this message (R). LAST sets the end bracket indicator (not valid with CICS applications); REPLY sets the change direction indicator.

Example:

```
77  REPLY-REQUEST     PIC X VALUE "R".
```

MSG-COMPLETE

This input parameter indicates whether a complete message is to be sent or single response units which must be assembled into a chain. Possible values are "Y" or "N".

Example:

```
77  MSG-COMPLETE      PIC X VALUE "Y".
```

FMH

This input parameter indicates whether the function management header (FMH) is part of the data to be sent. Possible values are Y or N.

Example:

```
77  FMH               PIC X VALUE "N".
```

RQD

This input parameter indicates whether a definite response is to be sent. Possible values are "Y" or "N".

Example:

```
77  RQD               PIC X VALUE "N".
```

RECEIVE-BUFFER

This input parameter sets up the buffer that receives the data during the session. The length of the data cannot exceed the size specified in the RECEIVE-BUFFER-SIZE (80 characters in this example).

Example:

```
77  RECEIVE-BUFFER    PIC X(80).
```

RECEIVE-BUFFER-SIZE

This input parameter designates the size of the RECEIVE-BUFFER in characters.

Example:

```
77  RECEIVE-BUFFER-SIZE          PIC 9(5) VALUE 80.
```

MSG

This input parameter designates whether the message being delivered to the application's buffer is a complete message or a message segment. If a whole message is being delivered, AIF must wait for the entire message and determine whether or not it fits into the RECEIVE-BUFFER. If the message is too large for the receive buffer, AIF delivers only the size of the message so that a new buffer can be assigned. Possible values for this parameter are "Y" (complete message) or "N" (message segment).

```
77  MSG                          PIC X VALUE "Y".
```

RECEIVED-DATA-LENGTH

This output parameter is to contain the length of the data received.

Example:

```
77  RECEIVED-DATA-LENGTH         PIC 9(5) VALUE 0.
```

INTERRUPT-DATA-LENGTH

This output parameter contains the length of any interrupt data that has been received.

Example:

```
77  INTERRUPT-DATA-LENGTH        PIC 9(5) VALUE 0.
```

WORK-AREA-ID

This output parameter, which is used by the CSWANY session call, contains the SNA-WORK-AREA value of the last COBOL session call that was executed. The WORK-AREA-ID does not have to correspond to the SNA-WORK-AREA unless there are no other active sessions.

Example:

```
77  WORK-AREA-ID                 PIC X(4).
```

SEND-RESPONSE-TYPE

This parameter indicates the type of response that is being sent. The following response types are possible:

Type	Value
Negative Response	"-"
Positive Response	"+"
Wait for Ready-to-receive	"R"
Not Ready-to-receive	"N"
None	" "

Example:

```
77  SEND-RESPONSE-TYPE            PIC X VALUE "-".
```

SENSE-DATA

This input parameter is required when a negative response is being sent. The parameter is specified using hexadecimal-coded ASCII characters.

Example:

```
77  SENSE-DATA        PIC X(8).
```

RETURNS

This output parameter defines the field into which the return code from all AIF session calls is placed. The RETURNS field is divided into RETURN-A, which consists of six yes/no conditions, and RETURN-B, which contains a four character decimal status code to provide further detail about the conditions indicated in RETURN-A.

The two subfields of RETURNS are presented below and are described in Table 3-1. Refer to "Checking the Return Code" for more information about RETURNS.

```
02  RETURN-A.
  03  SESSION-ABORT   PIC X VALUE 'N'.
  03  STOP-RCVD       PIC X VALUE 'N'.
  03  INTRPT-RCVD     PIC X VALUE 'N'.
  03  SERV-REQ-CANC   PIC X VALUE 'N'.
  03  SERV-REQ-COMP   PIC X VALUE 'N'.
  03  COBOL-ERROR     PIC X VALUE 'N'.
02  RETURN-B          PIC 9(4) VALUE 0.
```

Table 3-1. COBOL Session Call RETURNS Fields

Field	Meaning
SESSION-ABORT	LU-LU session or node has been aborted and no longer exists.
STOP-RCVD	SOPR STOP command received. If the TIME argument is supplied with the STOP command, check the TIME field for the time at which the session ends. This field indicates how much time you have to complete the session.
INTRPT-RCVD	Interrupt received. See INTERRUPT output parameter.
SERV-REQ-CANC	This request has been cancelled. The application must issue it again if necessary.
SERV-REQ-COMPLETE	This request has been completed.
COBOL-ERROR	Error in using COBOL interface to the AIF. See RETURN-B for return code.

INTERRUPT-TYPE

This parameter shows the reason for interrupt when one is sent or received.

Example:

```
77   INTERRUPT-TYPE              PIC 99 VALUE 0.
```

A complete list of interrupt types is provided in Appendix D.

RCVD-SENSE

This output parameter contains the hexadecimal representation of the sense data from the host if sense data is present. This field corresponds to SC_ESD in the SCCB.

Example:

```
77   RCVD-SENSE         PIC X(8).
```

TIMEOUT

This output parameter provides a formatted data area for the date and time that a session must be stopped when a STOP command is processed for the session or node. This field must be 14 decimal digits long, as follows:

Example:

```
01  TIMEOUT
    02  DATE1.
        03 YY             PIC 99 VALUE 0.
        03 MM             PIC 99 VALUE 0.
        03 DD             PIC 99 VALUE 0.
    02  TIME1.
        03 HH             PIC 99 VALUE 0.
        03 MN             PIC 99 VALUE 0.
        03 SSSS           PIC 9(4) VALUE 0.
```

TERMINATE-TYPE

This input parameter indicates whether termination is normal (N) or abnormal (A).

Example:

```
77   TERMINATE-TYPE    PIC X VALUE "N".
```

GET-ATTR-TYPE

This input parameter indicates what attribute the CSGTAT call is requesting. The only attribute available is 01 (bind image).

Example:

```
77   GET-ATTR-TYPE     PIC 99 VALUE "01".
```

OUTPUT-CONTROL-WORD

This output parameter provides information about the received data. The characteristics that can be specified are listed below. Each of these parameters must be stated. Possible values are "Y" or "N".

```
 01  OUTPUT-CONTROL-WORD.
     02 REPLY-REQUESTED-CD        PIC X.
     02 DEFINITE-RESP-REQ         PIC X.
     02 LAST-MSG-RCVD-EB          PIC X.
     02 FMH-IN-RCVD-DATA          PIC X.
     02 BEGIN-MSG-RCVD-BC         PIC X.
     02 END-MSG-RCVD-EC           PIC X.
     02 SET-SEND-RECV-SEQ         PIC X.
     02 APPL-RESEND-REQUIRED      PIC X.
     02 HOST-RESEND-REQUIRED      PIC X.
```

CONVERT-FROM-FIELD

This input parameter defines the buffer to be converted by the ASCII-to-EBCDIC conversion subroutines. The maximum size of this buffer is 32,767 bytes.

Example:

```
77   CONVERT-FROM-FIELD          PIC X(20).
```

CONVERT-FROM-LEFT-POSIT

This input parameter provides a starting index for the data in CONVERT-FROM-FIELD.

Example:

```
01   CONVERT-FROM-LEFT-POSIT     COMP-1 VALUE 1.
```

CONVERT-TO-FIELD

This input parameter defines the buffer into which the converted data will be placed by the ASCII-to-EBCDIC conversion subroutines. The maximum size of this buffer is 32,767 bytes.

Example:

```
77   CONVERT-TO-FIELD            PIC X(15).
```

CONVERT-TO-LEFT-POSIT

This input parameter provides a starting index for the data in CONVERT-TO-FIELD.

Example:

```
01   CONVERT-TO-LEFT-POSIT       COMP-1 VALUE 6.
```

CONVERSION-LENGTH

This input parameter contains the length in bytes of the data to be converted. The maximum length of this data is 32,767 bytes.

Example:

```
01   CONVERSION-LENGTH           COMP-1 VALUE 10.
```

Host-Initiated Sessions

AIF supports host-initiated sessions; that is, it accepts unsolicited binds. In order to accept an unsolicited bind, an LU must be reserved with the HOST_INIT_SESS parameter specified as Y (YES) in the LU entry of the configuration file.

When the application program begins execution, it must issue a CSACPT session call as the first session call, providing the STD name and the node name for the LU to be used. The CSACPT session call allows AIF access to a host-initiated session. AIF associates the first unsolicited bind (host-initiated session request) to the first CSACPT session call from the task group that AIF spawned.

An unsolicited bind can be for a program designated in the AUTO_ATTACH entry of the AIF configuration or it can be any other unsloicited bind sent from the host.

When AIF receives an unsolicited bind for a specific LU, AIF checks the LU entry for an AUTO_ATTACH program. If it finds one, AIF spawns a group with the program_name as the lead task, and passes to the lead task the STD name, node_name, and base_level used in the spawn group. If AIF does not find an AUTO_ATTACH program in the LU entry, it accepts the session and looks for the program name in the first four bytes of the first record received, then spawns a group based on the ATTACH_PROGRAM entry. If none is provided, default values are used to spawn the group.

The application can issue multiple CSACPTs to check for additional host-initiated sessions intended for this application. For an application to accept more than one session, all LUs that can receive binds for that application must be reserved LUs with HOST_INIT_SESS=Y. Each of these LUs must have the same group_id specified in the LU entry in the configuration file.

NOTE

> In order to execute a START_UP.EC instead of an attached program, you must create an attach program table entry with a dummy name (eg., ATTACH_PROG=ABC), specifying the appropriate spawn group parameters, and include an ALIAS for ABC (eg., ALIAS=>>SYSLIB2>EC?EXECL) to execute the START_UP.EC specified in the home directory. Refer to SNA6 Network Configuration for further information.

Linking the Program

If a COBOL application program is written as a program to be attached, that is, it includes an ACCEPT session call (CSACPT), then a LINKAGE SECTION must be included in the program. The LINKAGE SECTION must include three entries to accommodate the node name, STD name, and base level, as in the following example:

```
LINKAGE SECTION.
77  NODE     PIC X(8).
77  STD      PIC XX.
77  BASE_LVL PIC 99.
PROCEDURE DIVISION USING NODE, STD, BASE_LVL.
```

The LINKAGE SECTION is necessary whether the program is to be compiled using COBOLA or COBOLM. The programs are coded in the same way, regardless of which compiler is used, but they are linked differently.

Within the COBOL application program, the three fields in the LINKAGE SECTION must be moved to corresponding fields in WORKING-STORAGE before they can be used in any AIF calls.

Two sample LINK directive sets are presented below to demonstrate the different Linker directives you can use. The following matrix shows which set you should use, based upon LU type, whether you are writing an attached program, and the COBOL compiler you are using.

Compiler used:	COBOLA	COBOLM
ACCEPTS calls used:	3	1
No ACCEPTS calls used:	1	1

LINK DIRECTIVE SET 1

```
&N
&A
LINKER &1
LIB >LDD>ZCART/
LIB >LDD>ZCMRT*
LINK &1
LINK CSPHRZ
MAP
QT
```

* Use either LIB, where ZCART is used for COBOLA and ZCMRT is used for COBOLM.

```
LINK DIRECTIVE SET 3

&N
&A
LINKER &1
LIB >LDD>ZCART
LINKN CSLEAD
LINK &1
LINK CSPHRZ
MAP
LDEF CBLADR,&1
QT
```

NOTES

The module CSPHRA is the parameter processing routine for LU Type 0 calls.

Programs compiled by COBOLM automatically have the node name, STD name, and base level moved to the LINKAGE SECTION. Programs compiled by COBOLA use the CSLEAD Linker module to perform this function. This module must be linked into the bound unit of any program that executes a CSACPT or CSATCH and is compiled using COBOLA.

Refer to the Multiuser COBOL Compiler User's Guide (HE32) for information about linking programs compiled under COBOLA and COBOLM into a single bound unit.

Checking the Return Code

On return from AIF, a COBOL interface routine fills the output parameter fields with the SCCB results from the subroutine.

After the session call is made, a return code is placed in the RETURNS field. The RETURNS field is divided into RETURN-A, which consists of six yes/no conditions, and RETURN-B, which contains a four-character decimal return code to provide further detail about the conditions indicated in RETURN-A.

RETURN-A reports the following conditions:

- SESSION-ABORT--The session has been aborted.
- STOP-RCVD--SOPR stop command has been received.
- INTRPT-RCVD--An interrupt has been received.
- SERV-REQ-CANC--This request has been cancelled.
- SERV-REQ-COMP--This request has been completed.
- COBOL-ERROR--A COBOL interface error has occurred.

If the value of COBOL-ERROR is Y, then an error has occurred in the COBOL interface to AIF. The following are the general return codes that are in RETURN-B if you have a COBOL error. The value of XX is the number of the parameter in which there is an error:

Code	Meaning
XX01	Unrecognized parameter
XX02	Parameter must be 1 byte long
XX03	Parameter must be 5 bytes long
XX04	Default not acceptable
XX05	Node name error
XX06	Remote LU name error
XX07	Invalid session-id
XX08	Unknown interrupt type
XX09	Nondecimal digit
XX10	Nonhexadecimal digit
XX11	Error in conversion

The values of both RETURN-A and RETURN-B should be checked after the completion of each session call. Since it is possible to have more than one Y value in RETURN-A, and to have a value greater than zero after a successfully completed call, the application should check all fields in RETURN-A and RETURN-B for all possible combinations.

If the return code contains a "no error" message, go to the next segment of the program. If the return code contains an error condition, you might decide to record it to an error-out file, go to another segment of the program, or shut down completely.

Additional return codes are listed with the individual session calls to which they pertain. The return codes and their values are listed in Appendix D.

SESSION CALLS

The AIF session calls used in COBOL programs are detailed on the following pages.

Table 3-2. AIF Session Calls

Session Call	Description
CSACPT	Accept session call
CSCASR	Cancel outstanding asynchronous request
CSGTAT	Get attributes
CSINIT	Initiate or restart a session
CSPOLL	Test for LU associated with task group
CSRECV	Receive message in application's buffer
CSRI	Read interrupt
CSSEND	Request AIF to send a message or message segment
CSSI	Send interrupt
CSSRSP	Application instructs AIF to send a response
CSTERM	Terminate session
CSTEST	Test conditions
CSWANY	Wait on any event
CSACEB	ASCII-to-EBCDIC conversion
CSEBAC	EBCDIC-to-ASCII conversion

CSACPT - Accept Session Call

The CSACPT session call causes AIF to connect to a host initiated session.

FORMAT:

```
CALL "CSACPT" USING SNA-WORK-AREA
                    NODE-NAME
                    REMOTE-LU-NAME
                    STD-NAME
                    SYNC-CALL
                    SESSION-ID
                    NO-RESTART
                    MSG-RESYNC-SEND-SQN
                    MSG-RESYNC-RCV-SQN
                    RETURNS
                    INTERRUPT-TYPE
                    TIMEOUT
                    RCVD-SENSE
```

DESCRIPTION:

The CSACPT session call causes AIF to connect to a host-initiated session if there is one available. If there is no session, AIF returns and continues processing. The LU to which this bind refers is a reserved LU.

If your application is part of a host-initiated session, the CSACPT session call should be the first call executed. When this call is completed, the session is in receive state.

NOTE

This call is always made synchronously.

RETURN CODES:

The application should check the return code after each execution of a session call. In addition to the values described for RETURN-A and RETURN-B in "Checking the Return Code," the CSACPT session call can return the following values in RETURN-B:

Value	Description
0000	No error
0025	ACCEPT Timed out
0064	Invalid node name

0153	Invalid STD name
0154	Invalid LU type in STD
0155	No LU attached

SESSION-ID

This four-character field is supplied by AIF after it accepts the session request. The first word is the session group name, which is assigned by AIF to each of the sessions running in this session group. This value is used by AIF to return a unique one-word session identifier for this session. This value is stored in the second word. This field is reserved for system use and must never be altered by the application.

CSCASR - Cancel Asynchronous Request

The CSCASR session call causes AIF to cancel an outstanding asynchronous request, if possible.

FORMAT:

```
CALL "CSCASR" USING SNA-WORK-AREA
```

DESCRIPTION:

The CSCASR session call cancels an outstanding asynchronous request. If the previously executed asynchronous request was already completed when the CSCASR was executed, then the return code from CSCASR is for a completed asynchronous call. If the previously executed asynchronous call was not completed when CSCASR was executed and AIF succeeded in cancelling the request, the return code from CSCASR indicates that the call has been cancelled.

NOTE

The CSCASR session call cannot be used to cancel a CSINIT session call, even if it has been executed asynchronously.

RETURN CODES:

The application should check the return code after each execution of a session call. After the completion of the CSCASR session call, the following combinations are possible:

- If SERV-REQ-CANC=Y (all other fields in RETURN-A = N) and RETURN-B=0, you have cancelled the previously outstanding call.
- If SERV-REQ-CANC=Y and RETURN-B>0, the previous call completed with error. (RETURN-B contains the error code for the previous call.)
- If SERV-REQ-COMP=Y and RETURN-B=0, the previous outstanding call executed.

In addition to these combinations and the values described for RETURN-A and RETURN-B in "Checking the Return Code," CSCASR can return the following values in RETURN-B.

Value	Description
0023	No outstanding asynchronous call

CSGTAT - Get Session Attributes

The CSGTAT session call provides the application with an attribute for the session specified in the SNA-WORK-AREA.

FORMAT:

```
CALL "CSGTAT" USING SNA-WORK-AREA
                    RECEIVE-BUFFER
                    RECEIVE-BUFFER-SIZE
                    DATA-BUFFER-ALIGN
                    GET-ATTR-TYPE
```

DESCRIPTION:

The CSGTAT session call provides the application with an attribute for the session whose SNA-WORK-AREA is specified when issuing the call. If you plan to use this session call to request the bind image, the STD entry in the AIF configuration must include the parameter SAVE_BIND=Y.

Special notice should be given to the situation where an interrupt is received either prior to or during the execution of the CSGTAT session call.

1. When an interrupt is received before the execution of the CSGTAT, the application is given the data that was in the receive queue and informed of the interrupt.

2. If an interrupt is received during the execution of a CSGTAT, the order is not completed, control is returned to the application, and the return code indicates that an interrupt has been received.

NOTE

This call is always made synchronously.

RETURN CODES:

The application should check the return code after each execution of a session call. After the completion of the CSGTAT session call, the following combinations are possible:

- If SERV-REQ-COMP=Y and RETURN-B=0, the receive data buffer contains the attributes of the session specified.

- If the value of another field in RETURN-A is Y, the CSGTAT was not successful, and RETURN-B contains the return code to indicate the reason for the error.

In addition to these combinations and the values described for RETURN-A and RETURN-B in "Checking the Return Code," the CSGTAT session call can return the following values in RETURN-B:

Value	Description
0000	No error - session established
0016	Improper state - i.e., trying to receive, but in send state
0024	No BIND_IMAGE saved for $SGTAT
1013	Receive buffer too small
1014	Invalid attribute type
1032	Receive rejected; data traffic cleared/inactive

CSINIT - Initiate Session

The CSINIT session call can be used in two contexts:

1. To establish a session between the application and the transaction at the host

2. To restart this session if it has been abnormally terminated.

In issuing the session call, you must indicate for which purpose it is to be executed.

```
CALL "CSINIT" USING   SNA-WORK-AREA
                      NODE-NAME
                      REMOTE-LU-NAME
                      STD-NAME
                      SYNC-CALL|ASYNC-CALL
                      SESSION-ID
                      NO-RESTART
                      MSG-RESYNC-SEND-SQN
                      MSG-RESYNC-RCV-SQN
                      RETURNS
                      INTERRUPT-TYPE
                      TIMEOUT
                      RCVD-SENSE
```

CSINIT to Establish a Session

The initiate session call requests that AIF establish a session between an LU at the DPS 6 or the DPS 6 PLUS and an LU at the host, and that the local LU be assigned exclusively to the application. Always specify NO_RESTART on initial start-up.

In the event that AIF assigns a preestablished session to the application, the application should store the send/receive sequence numbers, in case a RESTART of this session ever becomes necessary. These sequence numbers are not reset to zero after each use. To the host, this appears as one session. On the local application side, the session is a serially reusable resource.

If multiple sessions are being established, a separate SNA-WORK-AREA must be provided for each session. The session ID should also be stored so that if a RESTART becomes necessary, you can specify which session to restart.

NOTE

A CSINIT session call, executed asynchronously, cannot be cancelled by using the CSCASR session call.

CSINIT to Restart a Session

The CSINIT session call is used to restart a session in the event that it has been abnormally terminated. Restart logic and restart rules are described in detail in Section 6.

RETURN CODES:

The application should check the return code after each execution of a session call. After the completion of the CSINIT session call, the following combinations are possible:

- If SERV-REQ-COMP=Y and RETURN-B=0, the session has been initiated successfully.
- If the value of another field in RETURN-A is Y, the CSINIT was not successful, and RETURN-B contains the return code to indicate the reason for the error.

In addition to these combinations and the values described for RETURN-A and RETURN-B in "Checking the Return Code," the CSINIT session call can return the following values in RETURN-B.

Value	Description
0000	No error - session established
0003	Negative response received
0004	Bind negotiation failed
0016	Improper state - i.e., trying to CSINIT with RESTART, but not in abnormally terminated state
0032	Restart not possible
0048	System error - i.e., not enough memory available to establish session
0049	Resource not available
0064	Invalid node name
0065	Invalid session-ID (Restart)
0150	AIF Node not yet active
0151	No active LU for session
0152	No LU available for session
0153	Invalid STD name
0154	Invalid LU type in STD
1809	Link failure

Each time you do a CSINIT with RESTART, you should check the OUTPUT-CONTROL-WORD to verify the send/receive sequence numbers and to find out whether it is necessary to retransmit the last message either from the DPS 6 or DPS 6 PLUS or from the host.

The RCVD-SENSE field contains sense data, if present, as listed in Appendix D.

CSPOLL - Poll Session Call

The CSPOLL session call checks to see if any LU associated with the application program's task group has received an unsolicited bind from the remote program.

FORMAT:

```
CALL "CSPOLL" USING SNA-WORK-AREA
                    NODE-NAME
                    STD-NAME
                    RETURNS
```

DESCRIPTION:

The CSPOLL session call causes AIF to test to see if any LU associated with the application program's task group has been attached (bound) by the remote program. The CSPOLL session call is similar to the CSACPT session call except that the CSPOLL does not cause a connection between AIF and the application program if a bind has be received.

- The SNA WORK-AREA used for a CSPOLL must be unique and should not be currently used by an active session.

NOTE

This call is always made synchronously.

RETURN CODES:

The application should check the return code after each execution of a session call. In addition to the values described for RETURN-A and RETURN-B in "Checking the Return Code," the CSACPT session call can return the following values in RETURN-B.

Value	Description
0064	Invalid node name
0153	Invalid STD name
0155	No LU attached
0005	There is an LU being bound

CSRECV - Receive Message

The CSRECV session call causes AIF to deliver a message or message segment from the session partner to the application's buffer.

FORMAT:

```
    CALL "CSRECV" USING SNA-WORK-AREA
                             RECEIVE-BUFFER
                             RECEIVE-BUFFER-SIZE
                             DATA-BUFFER-ALIGN
                             SYNC-CALL|ASYNC-CALL
                             MSG
                             RECEIVED-DATA-LENGTH
                             OUTPUT-CONTROL-WORD
```

DESCRIPTION:

The CSRECV session call causes AIF to deliver a message to the application's buffer from the session partner.

If the user specifies MSG, then AIF assembles the chain before delivery. If the user's buffer is not large enough, the message is not delivered; the actual length of the message or message segment is returned to the application in the RECEIVED-DATA-LENGTH. The application can either execute the receive again with an adequate buffer, or move N to the MSG field and execute the receive. If you specify N, single segments are delivered to the application's buffer. If the message segment delivered is the last segment, then AIF sets the end of message bit in the OUTPUT-CONTROL-WORD.

Special notice should be taken when an interrupt is received prior to or during the execution of a CSRECV.

If an interrupt has already been received when the CSRECV session call is executed, the application is given the data and informed of the interrupt. RETURNS shows either SERV-REQ-CANC=Y and INT-REC=Y or SERV-REQ-COMP=Y and INT-REC=Y, depending on whether or not the data was in the receive queue.

If an interrupt is received during the execution of a CSRECV, the order is not completed, and return is made to the application.

Check the OUTPUT-CONTROL-WORD before proceeding, to determine if end of message indicator has been received or if the host requires a response.

RETURN CODES:

The application should check the return code after each execution of a session call. After the completion of the CSRECV session call the following combinations are possible:

- If SERV-REQ-COMP=Y and RETURN-B=0, then the CSRECV had been completed with no error.
- If the value of another field in RETURN-A is Y, the CSRECV was not successful, and RETURN-B contains the return code to indicate the reason for the error.

If SERV-REQ-COMP=Y, check the OUTPUT-CONTROL-WORD to make sure that the beginning of message and end of message indicators have been received. If there is no end of message indicator, you must do another CSRECV to receive the next segment of the message.

In addition to these combinations and the values for RETURN-A and RETURN-B described in "Checking the Return Code," the CSRECV session call can return the following values in RETURN-B.

Value	Description
0000	No error - CSRECV successful
0016	Improper state - i.e., trying to receive while in send state
0019	Receive buffer too small
0048	System error - unable to receive
0050	Receive rejected; data traffic cleared/inactive
0065	Invalid session-ID
0066	Asynchronous service request outstanding
0256	Session unbound by host
1809	Link failure

NOTE

If a RESTART of this session is a possibility, then the receive sequence number should be stored by the application executing this CSRECV session call.

CSRI - Read Interrupt

The CSRI session call reads interrupt information from the host or control information from the AIF LU when there is no other AIF session call outstanding.

FORMAT:

```
    CALL "CSRI" USING  SNA-WORK-AREA
                       INTERRUPT-DATA-LENGTH
```

DESCRIPTION:

The CSRI session call enables the application to read interrupt information from the host or control information from AIF when there is no other AIF session call outstanding.

If either of the following situations occurs, the condition is reported to the application, the SNA-WORK-AREA is updated the same way as for CSTEST or CSWANY and a return is made to the application.

As with any asynchronous call, the application must execute a CSWANY or CSTEST session call to determine when the CSRI session call is complete and regain control.

1. When an interrupt is received, the INTERRUPT-TYPE and the SENSE-DATA fields in the SNA-WORK-AREA contains the appropriate information.

2. If data has been received for which there is no outstanding order, the user must issue a CSRECV to gain access to this data. The length of the received data is in INTERRUPT-DATA-LENGTH parameter of the SNA-WORK-AREA.

NOTE

The CSRI session call is always made asynchronously.

RETURN CODES

The application should check the return code after each execution of a session call. After the completion of the CSRI session call, the following combinations are possible:

- If SERV-REQ-COMP=Y and RETURN-B=0, the interrupt has been received with no error.

- If the value of another field in RETURN-A is Y, the CSRI was not successful, and RETURN-B contains the return code to indicate the reason for the error.

In addition to these combinations and the values for RETURN-A and RETURN-B described in "Checking the Return Code," the CSRI session call can return the following values in RETURN-B.

Value	Description
0002	Data received but no read
0016	Improper state - i.e., trying to receive while in send state
0050	Receive rejected; data traffic cleared/inactive

CSSEND - Send Message

The CSSEND session call sends a message (RU) or message segments (chain) to a session partner.

FORMAT:

```
    CALL "CSSEND" USING   SNA-WORK-AREA
                          SEND-BUFFER
                          SEND-BUFFER-SIZE
                          DATA-BUFFER-ALIGN
                          SYNC-CALL|ASYNC-CALL
                          REPLY-REQUEST
                          MSG-COMPLETE
                          FMH
                          RQD
```

DESCRIPTION:

The CSSEND session call instructs the sending of a message (RU) or message segments (chain) to a remote LU. When you are sending an entire message, the MSG-COMPLETE parameter must be Y. When sending message segments, the MSG-COMPLETE parameter must be N, except for the last segment, when MSG-COMPLETE = Y.

Special notice should be given to the situation where the application is executing a CSSEND session call but an interrupt is received before or during the execution of the call.

If an interrupt has already been received when the CSSEND session call is executed, the application is informed of the interrupt. If an interrupt is received during the execution of the CSSEND session call, the CSSEND session call completes, and when the application executes the CSWANY or CSTEST session call, return is made to the application. The return code indicates the interrupt received and the result of the CSSEND session call.

NOTE

If restart of this session is a possibility, then the send sequence number and the entire message must be saved by the application executing this CSSEND session call.

RETURN CODES:

The application should check the return code after each execution of a session call. After the completion of the CSSEND session call, the following combinations are possible.

- If SERV-REQ-COMP=Y and RETURN-B=0, the CSSEND has been completed with no error.
- If the value of another field in RETURN-A is Y, the CSSEND was not successful, and RETURN-B contains the return code to indicate the reason for the error.

In addition to these combinations and the values for RETURN-A and RETURN-B described in "Checking the Return Code," the CSSEND session call can return the following values in RETURN-B.

Value	Description
0000	No error - send successful
0003	Negative response received
0016	Improper state - i.e., trying to send in receive state
0018	Invalid input control indicator(s) - i.e., REPLY-REQUEST improperly indicated
0048	System error
0050	Send rejected
0256	Session unbound by host

CSSI - Send Interrupt

The CSSI session call is used to send Data Flow Control commands to the session partner or to pass control information to the System Service Control Point or to AIF.

FORMAT:

```
    CALL 'CSSI' USING      SNA-WORK-AREA
                           SEND-BUFFER
                           SEND-BUFFER-SIZE
                           DATA-BUFFER-ALIGNMENT
                           INTERRUPT-TYPE
                           REPLY-NAME
                           SENSE-DATA
```

DESCRIPTION:

The CSSI session call is used to send the following three types of information:

1. Send data flow control commands to the session partner
2. Pass control information to AIF.
3. Pass statistical information to SSCP.

A list of interrupt types is discussed in Appendix D.

The format of the buffers that you create to send CNM, alerts and maintenance statistics are detailed in Section 7.

NOTE

The CSSI session call is always made synchronously.

RETURN CODES:

The application should check the return code after each execution of a session call. After the completion of the CSSI session call, the following combinations are possible.

- If SERV-REQ-COMP=Y and RETURN-B=0, the interrupt has been sent with no error.
- If the value of another field in RETURN-A is Y, the CSSI was not successful, and RETURN-B contains the return code to indicate the reason for the error.

In addition to these combinations and the values for RETURN-A and RETURN-B described in "Checking the Return Code," the CSSI session call can return the following values in RETURN-B.

Value	Description
0000	No error
0003	Negative response received
0016	Improper state
0018	Invalid input control indicator(s)
0020	Invalid interrupt type
0021	Invalid status word/user code
0050	Receive rejected; data traffic cleared/inactive

CSSRSP - Send Response

The CSSRSP session call requests that AIF send a response to a previous message.

FORMAT:

```
CALL "CSSRSP" USING   SNA-WORK-AREA
                      SYNC-CALL|ASYNC-CALL
                      SEND-RESPONSE-TYPE
                      SENSE-DATA
```

DESCRIPTION:

The CSSRSP session call sends a response to a previous message on behalf of the application. The following response types are possible:

Type	Value
Negative Response	"-"
Positive Response	"+"
Wait for Ready-to-receive	"R"
No Ready-to-receive	"N"
None	" "

If this response is negative, the application also has the option of sending sense data.

RETURN CODES:

The application should check the return code after each execution of a session call. After the completion of the CSSRSP session call the following combinations are possible.

- If SERV-REQ-COMP=Y and RETURN-B=0, the response has been sent with no error.
- If the value of another field in RETURN-A is Y, the CSSRSP was not successful, and RETURN-B contains the return code to indicate the reason for the error.

In addition to these combinations and the values for RETURN-A and RETURN-B described in "Checking the Return Code," the CSSRSP session call can return the following values in RETURN-B:

Value	Description
0000	No error
0016	Improper state
0018	Invalid input control indicator(s) - SEND RESPONSE TYPE improperly indicated
0050	Send rejected; data traffic cleared/inactive

CSTERM - Terminate Session

The CSTERM session call terminates the AIF session.

FORMAT:

```
    CALL "CSTERM" USING SNA-WORK-AREA
                        TERMINATE-TYPE
```

DESCRIPTION:

The CSTERM session call terminates the AIF session. Termination can be either normal or abnormal. Whether it is normal or abnormal is indicated by a parameter within the CSTERM session call.

- If the CSTERM session call indicates normal termination, an orderly termination message is sent to the session partner's LU.
- If the CSTERM session call indicates abnormal termination, the following events occur:
  - The AIF LU terminates the session.
  - AIF sends an abnormal termination message to inform the host LU.

  After the session is terminated, the LU task is again available for other users.

Abnormal termination can be issued at any time; the last session call is cancelled if it is not completed.

NOTE

The CSTERM session call is always made synchronously.

RETURN CODES:

The application should check the return code after each execution of a session call. After the completion of the CSTERM session call the following combinations are possible:

- If SERV-REQ-COMP=Y and RETURN-B=0, the session has been terminated.

- If the value of another field in RETURN-A is Y, the session was not terminated as intended, and RETURN-B contains the return code to indicate the reason for the error.

In addition to these combinations and the values for RETURN-A and RETURN-B described in "Checking the Return Code," the CSTERM session call can return the following values in RETURN-B:

Value	Description
0000	No error
0016	Improper state - i.e., normal termination rejected because data is on receive queue

CSTEST - Test for Events

The CSTEST session call tests conditions for the session whose work area address is provided in SNA-WORK-AREA.

FORMAT:

```
CALL "CSTEST" USING SNA-WORK-AREA
                    INTERRUPT-DATA-LENGTH
```

DESCRIPTION:

This session call tests conditions for the session currently being executed. Executing this call causes AIF to immediately report to the application one of the following conditions:

1. No event
2. Interrupt received
3. Asynchronous order completed or cancelled
4. Permission to send after a send was rejected due to data traffic inactive or pacing
5. Data has been received for which there is no outstanding order.

Conditions 2 and 3 can coexist.

If an interrupt was received, the INTERRUPT-TYPE and the SENSE-DATA fields in the SNA-WORK-AREA contain information pertaining to the type of interrupt.

If an asynchronous order were completed or cancelled, then AIF delivers the return code of the completed order immediately, and the application must examine all pertinent fields in the SNA-WORK-AREA.

If data has been received for which there is no outstanding order, the user must issue a CSRECV session call to gain access to this data. Nothing is delivered to the user as a result of the CSTEST session call, but the length of the received data is found in the INTERRUPT-DATA-LENGTH parameter of the SNA-WORK-AREA.

NOTE

The CSTEST session call can be executed while an asynchronous call is outstanding. This session call is always made synchronously. If there was an asynchronous order outstanding, the condition is tested, reported, and the order remains outstanding. Once the test determines that the order has been completed, the call is no longer outstanding.

RETURN CODES:

The application should check both RETURN-A and RETURN-B after each execution of a session call. After the completion of the CSTEST call, the following combinations are possible.

- If all of the fields in RETURN-A are N and RETURN-B=0, there is an asynchronous call outstanding.
- If SERV-REQ-COMP=Y and RETURN-B=0, then the previously executed asynchronous call has been completed successfully.
- If SERV-REQ-COMP=Y and RETURN-B>0, then the previously executed asynchronous call has been completed with error.
- If SERV-REQ-CANC=Y and RETURN-B>0, then the previously executed call has been cancelled for the reason designated.

In addition to these combinations and the COBOL error codes described in "checking the Return Code," the CSTEST session call can return the following values in RETURN-B:

Value	Description
0000	No event
0001	Permission to send - i.e., a previous attempt to send was rejected
0002	Data received but no read

CSWANY - Wait on Events

The CSWANY session call causes AIF to issue a system "wait any" on behalf of the application. The application is dormant until one of the requests is complete.

FORMAT:

```
CALL "CSWANY" USING SNA-WORK-AREA
                    WORK-AREA-ID
```

DESCRIPTION:

The CSWANY session call causes execution of the application program to be suspended until any asynchronous request terminates. Asynchronous requests other than AIF requests also cause control to return to the CSWANY session call executor providing that the P-bit in the request block was set by the executor prior to the execution of the CSWANY macrocall.

You must specify an SNA-WORK-AREA when issuing a CSWANY. If an application has multiple sessions established, specifying an SNA-WORK-AREA does not imply that the CSWANY responds only to an event on that session. If an application has more than one session established, with outstanding asynchronous orders on multiple sessions, executing a CSWANY session call returns control to the application with WORK-AREA-ID containing the session ID of the session whose request has completed.

NOTE

The CSWANY session call is always made synchronously.

RETURN CODES:

The application should check both RETURN-A and RETURN-B after each execution of a session call. After the completion of the CSWANY call, the following combinations are possible.

- If SERV-REQ-COMP=Y and RETURN-B=0, then the previously executed asynchronous call has been completed successfully.
- If SERV-REQ-COMP=Y and RETURN-B>0, then the previously executed asynchronous call has been completed with error.
- If SERV-REQ-CANC=Y and RETURN-B>0, then the previously executed call has been cancelled for the reason designated.

CSACEB - ASCII-to-EBCDIC Conversion

The CSACEB session call converts data from ASCII to EBCDIC.

FORMAT:

```
CALL "CSACEB" USING SNA-WORK-AREA
                    CONVERT-FROM-FIELD
                    FROM-LEFT-MOST-POSITION
                    CONVERT-TO-FIELD
                    TO-LEFT-MOST-POSITION
                    CONVERSION-LENGTH
```

DESCRIPTION:

The CSACEB session call converts data from ASCII to EBCDIC. The parameters used with this session call provide the buffers containing the data to be converted and the converted data.

The maximum length of data that can be converted is 32,767 bytes.

If you want to convert the data in place, specify the same dataname for the CONVERT-FROM-FIELD and the CONVERT-TO-FIELD.

CSEBAC - EBCDIC-to-ASCII Conversion

The CSEBAC session call converts data from EBCDIC to ASCII.

FORMAT:

```
CALL "CSEBAC" USING SNA-WORK-AREA
                    CONVERT-FROM-FIELD
                    FROM-LEFT-MOST-POSITION
                    CONVERT-TO-FIELD
                    TO-LEFT-MOST-POSITION
                    CONVERSION-LENGTH
```

DESCRIPTION:

The CSEBAC session call converts data from EBCDIC to ASCII. The parameters used with this session call provide the buffers containing the data to be converted and the converted data.

The maximum length of data that can be converted is 32,767 bytes.

If you want to convert the data in place, specify the same dataname for the CONVERT-FROM FIELD and the CONVERT-TO-FIELD.

Section 4 PROGRAMMING LU TYPE 6.2 CONVERSATIONS IN ASSEMBLY LANGUAGE

This section describes the Assembly language verbs that are used in an LU Type 6.2 conversation with host service or transaction programs. Topics include:

- Basic Conversation Verbs
- Programming considerations
  - Getting started
  - Creating a verb parameter block
  - Conversation states
  - Checking the return code
- Individual conversation verbs
  - Format
  - Descriptions
  - Return codes.

BASIC CONVERSATION VERBS

The basic conversation verbs used by AIF are system-provided macrocalls. These verbs have a list of arguments that can be specified by the programmer or accepted in their existing form. AIF verbs follow the conventions for Assembly language, which are described in detail in the ONE PLUS Assembly Language (MAP) Reference manual (HE38). The verb can have an optional label. If no label is used, at least one blank space must precede the verb.

When AIF is activated, it defines the resources to be made available to the session while that conversation is active. AIF allocates a session for a conversation from a group of available LU sessions. AIF can either start a session to the host system at initiation or it can wait for an application to request to allocate a conversation. The time of session initiation is a configuration option.

An application requests to allocate a conversation with a remote transaction program by executing the $SALLO verb. AIF looks for an available session to allocate for that conversation. If no session is immediately available, the application can specify whether control should be returned to the program. The conversation uses a session for only the time it takes to execute the verb. After the verb is executed, the conversation retains its resources until a deallocate verb is issued or a deallocate confirmation is received from the host application.

An application gains access to a host-initiated conversation by executing a $SATCH verb. When an ATTACH command is received from the host, AIF loads the transaction program by spawning a group with the attached application as the lead task, and sends a response to the host that the program is attached. The DPS 6 PLUS programs must issue a $SATCH verb before any other verbs are issued.

User-selected items are known as arguments. These arguments are positional within the verb--the order of positional arguments indicates the variables to which data is applied. Thus, the order of your arguments must be the same as the order of the positional arguments within the verb.

The following rules govern the use of positional arguments:

- Omitted arguments that precede an included argument must be indicated by the presence of a delimiting comma for each omission.

- One or more spaces must separate the verb name from its arguments, with a comma between each argument. (The horizontal tab character is equivalent to a space.)

- A semicolon at the end of a line indicates that the next line is a continuation line.

In the following example, the first argument has been omitted; its position has been held by a delimiting comma. Spaces separate the verb name from its arguments.

```
$SALLO  ,'AIFNODE1','LU104',=Z'20F0F0F0',AVAIL,CONFIRM
```

The arguments for these conversation verbs are found in the verb parameter block (VPB). A VPB must be provided for each verb. These fields can be altered either during initialization or by including the appropriate arguments in the verb itself.

At the completion of each verb, when control is returned to the application, a return code is placed in register $R1. The return code can also be found in VP_RCD. This return code indicates whether a verb has been completed error free. The application should check this return code after each verb to verify the return status of the verb. Additional information, if desired, can be found in the output control word (VP_OCT), and other output parameters as defined for individual session calls.

PROGRAMMING CONSIDERATIONS

Many of the programs that use AIF conversation verbs are written in Assembly language. These applications may be reentrant and may not require more than one occurrence of a given verb.

Special considerations that the programmer must bear in mind fall into five categories, which are discussed in this section:

- Getting Started
- Creating a verb parameter block
- Conversation state
- Host initiated sessions
- Checking the return code.

Getting Started

When using AIF verbs in an Assembly language program, remember the following steps:

1. In order to use the verbs and utility macros included with AIF, you must first make them available to your program. When beginning your program, include the following statement:

   ```
   LIBM    '>>LDD>MACROS>MAC_USER'
   ```

2. Then issue the macrocalls $SVPB and $SAIRC to define the VPBB and return codes in memory.

3. You must also set aside a workspace with room for the stack, the VPB, and your send/receive buffer, as in the following example:

```
*
*   WORK LOCATIONS: STACK, VPB, & SEND/RECEIVE BUFFER
*
WKSP        EQU    0              BEGINNING OF WORKSPACE
MYSTACK     EQU    WKSP+50        REGISTER STACK
CNTLWD      EQU    MYSTACK        FOR PROGRAM CONTROL
MYVPB       EQU    CNTLWD+1       BEGINNING OF VPB
BUFFER      EQU    MYVPB+VP_SIZ   SEND/RECEIVE BUFFER
BUFSZ       EQU    2000           BUFFER SIZE
WKSPSZ      EQU    BUFFER+BUF_SZ  WORKSPACE SIZE
```

Verb Parameter Block

Communication between the application program and AIF is through the application-provided VPB. The programmer should note that the same VPB is used each time a particular conversation is referenced until that conversation is deallocated. If a program is to run multiple conversations, you must supply a separate VPB for each conversation.

When the application provides parameters with a given verb, the macrocode updates the appropriate VPB fields before executing an AIF monitor call. If any of the fields have been changed, the new values are in the VPB when you reexamine it.

The first parameter of each verb is the location of the VPB, with the exception of $SWAIT. If not specified as the first parameter of the verb, this pointer must be in register $B4 Allowable formats for this parameter and all address pointers are the same as found in the "Addressing Parameters" section of the System Programmer's Guide, Vol. 2.

Where a value rather than an address is provided in a parameter, allowable formats are:

1. (*)$B1(.$R)
2. LABEL
3. =$R1
4. =literal
5. !LABEL

Conversation verb users must provide a separate VPB for each conversation. The programmer can provide the parameters for the verbs by moving the parameters to the VPB before issuing the verb (Example 1) or when issuing the verb (Example 2).

The following examples show both methods of creating a VPB for the $SATCH verb. Which convention you choose to follow depends upon the requirements of your program.

Example 1:

The following example shows the parameters in the VPB being loaded before issuing the verb. Offsets to the VPB are provided in the displacement macro $SVPB. (Refer to the VPB template in Appendix I for appropriate offsets.)

```
NODENM  DC   'AIF505  '
STD_NM  DC   'BB'
SLV_VL  DC   0
        .
        .
        .
        LDB  $B4, $B6.VPB        Load VPB address to $B4
        LDI  $B6.NODENM          Get first 4 bytes of nodename
        SDI  $B4.VP_NOD          Store first 4 bytes of
                                 nodename in VPB
        LDI  $B6.NODENM+2        Get second 4 bytes of nodename
        SDI  $B4.VP_NOD+2        Store second 4 bytes of
                                 nodename in VPB
        LDR  $R2,$B6.STD_NM      Get STD name
        STR  $R2,$B4.VP_STD      Store STD name
        LDR  $R2,$B6.SLV_VL      Set sync level to none
        STR  $R2,$B4.VP_SLV      Store the sync level

        $SATCH
```

Example 2:

The following example shows the $SALLO verb with the parameters specified within the macrocall.

This sequence causes the equivalent of the following to be issued:

```
    $SATCH ,'AIF505','BB',NONE
```

Conversation States

The subset of verbs that a program can issue at a given time is determined by the state of the conversation at that time. For example, if a conversation is in receive state, it cannot issue a send verb without first issuing a verb to change the conversation to send state. The program must be aware of the state of the conversation, which can be found in the VP_CST field of the VPB. Executing many of the basic conversation verbs causes the conversation to change its state.

Table 4-1 lists the conversation states and their definition. Table 4-2 shows what verbs a conversation can issue from each state. The description of each verb includes the state of the conversation at the end of execution.

Table 4-1. Conversation States

State	Definition
Reset	The state in which the program can allocate a conversation.
Send	The state in which the program can send data or request confirmation.
Defer	The state in which the program can request confirmation or flush the LU's send buffer to prepare to change states.
Receive	The state in which the program can receive data or confirmation information.
Confirm	The state in which the program can send a confirmation reply.

Table 4-2. Conversation States From Which Verbs Can Be Issued

Verb	Conversation State				
	Reset	Send	Defer	Receive	Confirm
$SALLO	X				
$SATCH	X				
$SCONF		X	X		
$SCNFD					X
$SDEAL flush		X			
$SDEAL sync level		X			
$SDEAL abend		X	X	X	X
$SFLSH		X	X		
$SPONR				X	
$SPTOR		X			
$SRAW		X		X	
$SRTOS				X	X
$SSDAT		X			
$SSERR		X		X	X
$SWAIT				X	

Host-Initiated Conversations

AIF supports host-initiated conversations. The program name, node name, STD name, and base level are provided to the application program by AIF via the standard operating system parameter list. Refer to the System Programmer's Guide, Vol. 2. When the application program begins execution, it must execute a $SATCH verb as the first conversation verb, providing the STD name and the node name for the LU to be used. The node name and the STD name provided with the $SATCH verb must be the same as the parameters passed by AIF.

After the $SATCH verb is executed, the application is in receive state. The $SATCH verb allows AIF access to a host-initiated conversation. AIF associates the first unsolicited bind (host-initiated session request) to the first $SATCH session call from the task group that AIF spawned.

The application can issue multiple $SATCHs to check for additional host-initiated sessions intended for this application. For an application to accept more than one conversation, all LUs that can receive binds for that application must be reserved LUs. Each of these LUs must have the same group_id specified in the LU entry in the configuration file.

NOTE

> In order to execute a START_UP.EC instead of an attached program, you must create an attach program table entry with a dummy name (eg., ATTACH_PROG=ABC), specifying the appropriate spawn group parameters, and include an ALIAS for ABC (eg., ALIAS=>>SYSLIB2>EC?EXECL) to execute the START_UP.EC specified in the home directory. Refer to SNA6 Network Configuration for further information.

Checking the Return Code

After a session call is executed, AIF returns a status code known as the return code to the Verb Parameter Block (VPB) to indicate how the call was completed. The application should examine this return code at the completion of each verb to determine if the call has been completed error free.

The return code has 16 bits and is placed in register $R1 by AIF before control is returned to the application program. The value of the return code can also be found in VP_RCD.

Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any session call. If the bit is on, then the return code is set. These bits should be examined individually, then "masked out" so that the application can examine the remaining bits. The following masks are provided in the $SAIVR macrocall for checking each of the first five bits as follows.

Bit 0 VRABND

The conversation has abended or deallocated. An SOPR command has been entered that caused the conversation to abend, or the conversation was deallocated by the remote program. The specific reason for this termination can be found in the bits 5 through 15 of the return code or in VP_ABT.

Bit 1 VRSTOP

An SOPR STOP command has been received that causes the conversation to be deallocated when the specified time has elapsed. If no time is entered, the conversation is deallocated immediately. During this time the application can continue to process, but should normally terminate.

The time found in the TIME argument (VPB.VP_TIM) is the wall clock time in standard 48-bit format at which the session terminates.

Bit 2 VRRINT

This bit is reserved and should not be used by the application.

Bit 3 VRSCNL

The verb has been cancelled; it is not processed. If the application desires the order to be processed, the verb must be reexecuted. The specific reason for which the call has been cancelled can be found in the bits 5 through 15 of the return code.

Bit 4 VRSCMP

The service request (verb) has been completed.

A return code can indicate more than one condition occurring at the same time. For example, it can indicate both a deallocation and a completed call, or an SOPR STOP and a completed call.

The masks VRABND, VRSTOP, VRRINT, VRSCNL, and VRSCMP are provided for your convenience in checking bits 0 through 4. After you have checked these bits, null them out and examine bits 5 through 15. If you choose to null these bits by using VRMASK, which is provided in the software (VRMASK=07FF), use the following statement:

```
AND   $R1,=VRMASK
```

Bits 5 through 15 contain the return code for a completed or cancelled call. One way of doing this part of the return code is to issue a "compare" instruction as follows:

```
CMR     $R1,=VROKAY    (VROKAY = 0000)
BE      CONT_1
```

If the return code contains an "okay" message, branch to the next segment of the program. If the return code contains an error condition, you might decide to record it to an error-out file, branch to another segment of the program, or shut down completely.

Appendix F contains a complete list of return codes. These labels and their hexadecimal values can be found in the macro $SAIRC (AIF Return Codes).

INDIVIDUAL VERB FORMATS

Table 4-3 lists the basic conversation verbs that are supported by AIF. These verbs are described in detail on the following pages.

Table 4-3. AIF LU Type 6.2 Verbs

Verb	Description
$SALLO	Allocate verb
$SATCH	Attached verb
$SCONF	Confirm verb
$SCNFD	Confirmed verb
$SDEAL	Deallocate verb
$SFLSH	Flush verb
$SPONR	Post on Receipt verb
$SPTOR	Prepare to Receive verb
$SRAW	Receive and Wait verb
$SRTOS	Request to Send verb
$SSDAT	Send Data verb
$SSERR	Send error verb
$SWAIT	Wait verb
$SACEB	Converts ASCII to EBCDIC
$SEBAC	Converts EBCDIC to ASCII

$SALLO - Allocate Verb

The $SALLO verb is used to allocate a conversation between a local program and a remote program.

FORMAT:

```
  [label]  $SALLO  [vpb address]            P1: $B4
                   [,node name]             P2: VP_NOD
                   [,remote lu name]        P3: VP_RLN
                   [,trans program name]    P4: VP_TPN&VP_TPL
                   [,std name]              P5: VP_STD
                   [,return control]        P6: VP_ICT.VBRCTL
                   [,sync level]            P7: VP_SLV
```

ARGUMENTS:

vpb address

This parameter contains a pointer to the address of the VPB to be used for this conversation. If not declared, the address is assumed to be in register $B4.

node name (VP_NOD)

This parameter identifies the AIF node to which the application is directing this verb. This field contains eight alphanumeric characters. If you are loading the VPB yourself, and your node name contains fewer than eight characters, this field must be left-justified and space-filled.

remote lu name (VP_RLN)

The name by which the remote LU is known to this application. This field contains eight alphanumeric characters. If you are loading the VPB yourself, and the remote lu name contains fewer than eight characters, this field must be left-justified and space-filled.

trans program name (VP_TPN + VP_TPL)

This parameter contains the name of the transaction program to be attached to the host. This host program becomes the session partner of the program executing this $SALLO.

How you enter the transaction program name determines how the string is passed to the host. If you enter an ASCII string, =A'name', $SALLO translates the string to EBCDIC and puts the length of the string in VP_TPL. If you enter a hexadecimal string, =Z'hexname', where hexname contains an even number of hexadecimal digits, $SALLO puts the length of the string in VP_TPL and does not translate it.

If you are loading the VPB yourself, clear bit VP_TPL.VBTPNT to indicate that you want the transaction program name translated, or set this bit to indicate that you do not want the TPN translated. Put the length of the transaction program name into the right byte of VP_TPL.

std name (VP_STD)

The configured session type descriptor (STD) that lists the attributes of the conversation to be allocated, as defined in the configuration for this node. This field consists of two alphanumeric characters.

return control (VP_ICT.VBRCTL)

This parameter indicates whether the local LU should return control to the local program, in the event that it is unable to allocate a conversation.

The following arguments are valid for this parameter:

- AVAIL - allocates a session for the conversation before returning control to the program. If the local LU fails to obtain a session for the conversation, an allocation error is reported in $SALLO return code.

- IMMED - allocates a session for the conversation if one is immediately available and then returns control to the session.

  - If a session is immediately available, the conversation is allocated and control is returned with a return code of OKAY. The local LU must be the contention winner.

- If a session in not immediately available, the conversation is not allocated and control is returned with a return code of VRUNSU.

- If a session is immediately available and an error occurs in allocating a conversation, the error is reported in the return code for the $SALLO.

NOTE

If an LU is configured with the contention winner as non-negotiable, the LU must be both reserved and preestablished to be available for allocation with a return control of IMMED.

sync level (VP_SLV)

This parameter indicates how the local and remote programs perform confirmation processing on this conversation. The following arguments are valid for this parameter:

- NONE - do not perform confirmation processing on this conversation. Programs that specify NONE do not issue any verbs or recognize return parameters related to synchronization.

- CONFIRM - performs confirmation processing only on this conversation. Programs that specify CONFIRM issue verbs and recognize returned confirmation parameters, but do not recognize return parameters related to synchronization.

DESCRIPTION:

The $SALLO verb first allocates a session between a local LU and a remote LU, then allocates a conversation over that session, between a local program and a remote program, and puts the conversation in send state. Once you have allocated a conversation over a session, that session becomes available to other conversations until this conversation is deallocated.

The $SALLO verb is used to allocate conversations for either transaction programs or service component programs. The parameters issued with this verb identify the partners in the conversation and provide bind information about the conversation.

The $SALLO verb must be issued before any other verbs that refer to the specified conversation. At the completion of the $SALLO verb, the conversation enters send state.

RETURN CODES:

The application should check the return code after each execution of a verb. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any verb. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15.

In addition to the general return codes, the following values are possible.

Value	Label	Description
0000	VROKAY	OK
0040	VRINOD	Invalid node name
0042	VRITPN	Invalid transaction program name (null value)
0049	VRSLNS	Synchronization level not supported by LU
004B	VRIRTC	Invalid return control
0096	VRNNAC	Node not yet active
0097	VRNLAC	No active LU for session
0098	VRNOAV	No LU available for session
0099	VRISTD	Invalid STD name
009A	VRILUT	Invalid LU type in STD

In addition, if you specified a return control of IMMED, the following return code is possible.

Value	Label	Description
0001	VRUNSU	Unsuccessful

$SATCH - Attached Verb

The $SATCH verb is used by an attached program to gain access to the conversation.

FORMAT:

```
[label]  $SATCH  [vpb address]        P1: $B4
                 [,node name]         P2: VP_NOD
                 [,std name]          P3: VP_STD
                 [,sync level]        P4: VP_SLV
```

ARGUMENTS:

vpb address

This parameter contains a pointer to the address of the VPB to be used for this conversation. If not declared, the address is assumed to be in register $B4.

node name (VP_NOD)

This parameter identifies the AIF node to which the application is directing this verb. This field contains eight alphanumeric characters. If you are loading the VPB yourself, and your node name contains fewer than eight characters, this field must be left-justified and space-filled.

std name (VP_STD)

The configured session type descriptor (STD) which lists the attributes of the conversation to be allocated. This field consists of two alphanumeric characters.

sync level (VP_SLV)

This parameter indicates how the local and remote programs perform confirmation processing on this conversation.

The following arguments are valid for this parameter:

- NONE - do not perform confirmation processing on this conversation. Programs that specify NONE do not issue any verbs or recognize return parameters related to synchronization.

- CONFIRM - performs confirmation processing only on this conversation. Programs that specify CONFIRM issue verbs and recognize returned confirmation parameters, but do not recognize return parameters related to synchronization.

DESCRIPTION:

The $SATCH verb causes the program to be connected to a host-initiated conversation. When the host issues an ATTACH command to allocate a conversation, AIF loads the DPS 6 transaction by spawning a group with the program as the lead task. When the program is loaded, it must issue the $SATCH verb to tell the host that the transaction program has been attached to the session, and the node name and STD name with which it is associated.

If the application is intended for host-initiated sessions, the $SATCH should be the first verb executed. After the $SATCH verb is executed, the conversation enters receive state.

RETURN CODES:

The application should check the return code after each execution of a verb. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any verb. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15.

In addition to the general return codes, the following values are possible.

Value	Label	Description
0000	VROKAY	OK
0040	VRINOD	Invalid node name
0099	VRISTD	Invalid STD name
009B	VRNOAT	No LU attached by Remote TP
00D0	VRAESP	Synchronization level not supported by LU

$SCONF - Confirm Verb

The $SCONF verb sends a confirmation request to the remote program.

FORMAT:

```
[label]  $SCONF  [vpb address]  P1: $B4
```

ARGUMENTS:

vpb address

This parameter contains a pointer to the address of the VPB to be used for this conversation. If not declared, the address is assumed to be in register $B4.

DESCRIPTION:

The $SCONF verb requests that the remote program send an acknowledgment, and waits for a response. The $SCONF verb is used in confirmation processing, and in verifying that the conversation has been allocated or data has been received. $SCONF is not used if the conversation has been allocated with a synchronization level of NONE. This verb causes the LU to flush its send buffers.

When the $SCONF verb is issued in defer state following a $SPTOR, the conversation enters receive state. When the $SCONF verb is issued in defer state following $SDEAL, the conversation enters reset state. When the $SCONF verb is issued in send state, the state does not change.

RETURN CODES:

The application should check the return code after each execution of a verb. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any verb. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15.

In addition to the general return codes, the following values are possible for bits 5 through 15.

Value	Label	Description
0000	VROKAY	OK
0047	VRVBNS	Verb not supported (conversation was allocated with a sync level of none)
0041	VRIRID	Invalid resource ID
0011	VRNSDF	Not in send/defer state
0018	VRLRNF	Logical record not finished yet
00F1	VRDAPG	Remote deallocation--ABEND program
00F2	VRDASV	Remote deallocation--ABEND service
00F3	VRDATM	Remote deallocation--ABEND timer
0004	VRPEPR	Program error--purging
0007	VRSEPR	Service program error, purging
0103	VRPGER	Resource failure, no retry
0100	VRUNBI	Session unbound by host unexpectedly
0101	VRSSHU	Session shutdown by host orderly
0102	VRURTO	You are timed out by SOPR command
0310	VRADLU	ACTLU/DACTLU received
0711	VRLKFL	Link failure
0712	VRADPU	ACTPU/DACTPU received
0713	VRACSA	$A (SOPR) 'ABORT' AIF node
0714	VRSABT	$S abort AIF group

OUTPUT CONTROL WORD

The request to send received field in the output control word (VP_OCT.VBRRTS) indicates whether the remote program has issued a request to send notification, requesting the local program to enter receive state and placing itself in send state. If VP_OCT.VBRRTS is set, then this condition is true.

$SCNFD - Confirmed Verb

The $SCONFD verb sends a confirmation response to the remote program.

FORMAT:

```
[label]  $SCNFD  [vpb address]  Pl: $B4
```

ARGUMENTS:

vpb address

This parameter contains a pointer to the address of the VPB to be used for this conversation. If not declared, the address is assumed to be in register $B4.

DESCRIPTION:

The $SCNFD verb sends a confirmation to a remote program, always in response to a request for confirmation. The $SCNFD verb is used in confirmation processing and error detection. $SCNFD is not used if the conversation has been allocated with a synchronization level of NONE.

The what-received parameter of the previous receive and wait verb determines what state the conversation enters after the $SCNFD is executed. If the $SRAW returned a confirm indicator, the conversation enters receive state. If the $SRAW indicated confirm-send, the conversation enters send state. If the $SRAW indicated confirm-deallocate, the conversation enters reset state.

RETURN CODES:

The application should check the return code after each execution of a verb. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any verb. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15.

In addition to the general return codes, the following values are possible for bits 5 through 15:

Value	Label	Description
0000	VROKAY	OK
0047	VRVBNS	Verb not supported (conversation was allocated with a sync level of none)
0041	VRIRID	Invalid resource ID
0018	VRNCNF	Not in confirm state

$SDEAL - Deallocate Verb

The $SDEAL verb deallocates the specified conversation from the transaction program.

FORMAT:

```
[label]  $SDEAL  [vpb address]        P1: $B4
                 [,type]              P2: VP_TYP
                 [,LOG|NO_LOG]        P3: VP_ICT.VBLGDA
                 [,log data buffer]   P4: VP_BUF
                 [,log data length]   P5: VP_DLG
```

ARGUMENTS:

vpb address

This parameter contains a pointer to the address of the VPB to be used for this conversation. If not declared, the address is assumed to be in register $B4.

type (VP_TYP)

This parameter specifies whether the deallocation is to be completed as part of this verb or deferred until another verb is issued or a certain condition is met.

The following arguments are valid for this parameter:

- SYNC_L - perform deallocation according to the sync level specified when the conversation was allocated:
  - If sync level = NONE, $SDEAL flushes the local LU's send buffer and deallocates normally.
  - If sync level = CONFIRM, $SDEAL sends a confirmation request to the remote LU and, if the return code is OK, deallocates the conversation normally. If the return code is UNSUCCESSFUL, $SDEAL returns the conversation to its previous state.
- FLUSH - flushes the local LU's send buffer and deallocates the conversation normally.

The following type arguments are for error handling, and are application-dependent.

- PROG_AB - flushes the local LU's send buffer when the conversation is in send or defer state and deallocates the conversation abnormally.

- SVC_AB - flushes the local LU's send buffer when the conversation is in send or defer state and deallocates the conversation abnormally.
- TIM_AB - flushes the local LU's send buffer when the conversation is in send or defer state and deallocates the conversation abnormally.

NOTE

If ABEND deallocation occurs when the conversation is in send state, logical record truncation can occur. When the conversation is in receive state, data purging can occur.

{LOG|NO_LOG}

This parameter indicates whether or not the system error log is transferred to the transaction when the conversation is deallocated in an ABEND situation.

log data buffer

This parameter is a pointer to the product specific error data that is kept in the system error logs of the local and remote LUs. This parameter is used only with an ABEND deallocation type.

log data length

This parameter specifies the length of the log data buffer in bytes. The maximum allowable length of this buffer is 32,767 bytes.

DESCRIPTION:

The $SDEAL verb deallocates the specified conversation from the transaction program. The parameters issued with this verb identify the conversation to be deallocated and the type of deallocation to be performed.

After the $SDEAL verb is executed, the conversation enters reset state.

NOTE

AIF does not support a state that corresponds to the AIF deallocate state. If you receive a deallocate-confirm message after a $SCNFD verb, the conversation has been deallocated and its resources returned to the system. The conversation is then in reset state.

RETURN CODES:

The application should check the return code after each execution of a verb. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any verb. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15.

In addition to the general return codes, the following values are possible for any execution of the $SDEAL.

Value	Label	Description
0000	VROKAY	OK
0010	VRNSND	Not in send state
0018	VRLRNF	Logical record not finished yet
004C	VRITYP	Invalid type specified

If you executed the $SDEAL with a type of ABEND, the following return codes are possible.

Value	Label	Description
001A	VRPDEA	Improper state
004C	VRITYP	Invalid type specified

If you executed the $SDEAL with a type of SYNC_L and the conversation was allocated with synchronization level of CONFIRM, the following return codes are possible.

Value	Label	Description
0047	VRVBNS	Verb not supported (conversation was allocated with a sync level of none)
004C	VRITYP	Invalid type specified
0011	VRNSDF	Not in send/defer state
0018	VRLRNF	Logical record not finished yet
00B0	VRAETN	TPN not recognized
00C0	VRAEPI	PIP not allowed
00C1	VRAEIP	PIP not specified correctly
00C2	VRAESI	Security not valid
00C3	VRAECM	Conversation type mismatch
00D0	VRAESP	Sync level not supported by program
00D1	VRAERP	Reconnect level not supported by program
00D2	VRAENR	TP not available--no retry
00D3	VRAETR	TP not available--retry

00E0	VRAEAN	ACC not valid
00F1	VRDAPG	Remote deallocation--ABEND program
00F2	VRDASV	Remote deallocation--ABEND service
00F3	VRDATM	Remote deallocation--ABEND timer
0007	VRSEPR	Service program error, purging

$SFLSH - Flush Verb

The $SFLSH verb flushes the local LU's send buffer.

FORMAT:

```
    [label]  $SFLSH  [vpb address]  P1: $B4
```

ARGUMENTS:

vpb address

This parameter contains a pointer to the address of the VPB to be used for this conversation. If not declared, the address is assumed to be in register $B4.

DESCRIPTION:

The $SFLSH verb flushes the local LU's send buffer. Any information that was in the buffer is sent to the remote LU. The $SFLSH verb is useful for transferring incomplete buffers of data to the remote LU, thus avoiding a delay in processing.

If you execute a $SFLSH when the conversation is in defer state following a $SPTOR, the conversation enters receive state. If you execute a $SFLSH when the conversation is in send state, the state of the conversation does not change.

RETURN CODES:

The application should check the return code after each execution of a verb. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any verb. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15.

In addition to the general return codes, the following values are possible.

Value	Label	Description
0000	VROKAY	OK
0041	VRIRID	Invalid resource ID
0011	VRNSDF	Not in send/defer state
0103	VRPGER	Resource failure, no retry
0100	VRUNBI	Session unbound by host unexpectedly
0101	VRSSHU	Session shutdown by host orderly
0102	VRURTO	You are timed out by SOPR command
0310	VRADLU	ACTLU/DACTLU received
0711	VRLKFL	Link failure
0712	VRADPU	ACTPU/DACTPU received
0713	VRACSA	$A (SOPR) 'ABORT' AIF node

$SPONR - Post on Receipt Verb

The $SPONR verb causes the LU to signal the conversation when there is information to receive.

FORMAT:

```
[label]  $SPONR  [vpb address]  P1: $B4
                 [,fill]        P2: VP_ICT.VBFILL
                 [,length]      P3: VP_DLG
```

ARGUMENTS:

vpb address

This parameter contains a pointer to the address of the VPB to be used for this conversation. If not declared, the address is assumed to be in register $B4.

fill

This parameter specifies when posting should occur in terms of the length specified in the next parameter.

The following arguments are valid for this parameter.

- BUFFER - data is buffered into units of the length specified in the next parameter. Posting occurs when the buffer is full or the end of data is indicated.
- LL - posting occurs when a complete or truncated logical record is received, or when part of a logical record is received that is as long as or longer than the length specified in the next parameter.

length

This parameter specifies the maximum length of the receive buffer.

DESCRIPTION:

The $SPONR verb causes the LU to signal the conversation when there is information to receive. The information can be data, status information, or a request for confirmation. The $SPONR can be used with the $SWAIT verb or the $SRAW to allow you to continue with other program processing while waiting for data from the host.

Executing the $SPONR verb does not cause the state of the conversation to change. In order to execute the $SPONR, you must be in receive state. If you are not in receive state, you must first issue the $SPTOR verb.

RETURN CODES:

The application should check the return code after each execution of a verb. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any verb. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15.

In addition to the general return codes, the following values are possible.

Value	Label	Description
0000	VROKAY	OK
0041	VRIRID	Invalid resource ID
0016	VRNRCV	Not in receive state

If the return code indicates OKAY and the output control word indicates that the conversation has been posted, then posting has occurred and the LU has information that the program can receive. The program has the option of issuing a $SRAW at this point or it can ignore this posting by issuing a $SWAIT, and receive this data at a later time.

OUTPUT CONTROL WORD

The conversation posted field in the output control word (VP_OCT.VBPOST) indicates whether the conversation has been posted. If this bit is true, the conversation is posted and $SRAW can be used to receive data or information. If this bit is false, posting is active for this conversation and $SWAIT can be used to wait for posting to occur.

$SPTOR - Prepare to Receive Verb

The $SPTOR verb changes the state of the specified conversation from send to receive.

FORMAT:

```
[label]  $SPTOR  [vpb address]  P1: $B4
                 [,type]        P2: VP_TYP
                 [,locks]       P3: VP_ICT.VBLOCK
```

ARGUMENTS:

vpb address

This parameter contains a pointer to the address of the VPB to be used for this conversation. If not declared, the address is assumed to be in register $B4.

type

This parameter specifies whether the prepare-to-receive is to be completed as part of this verb or deferred until another verb is issued or a certain condition is met.

The following arguments are valid for this parameter:

- SYNCLVL - perform the prepare-to-receive according to the synchronization level specified when the conversation was allocated:
  - If sync level = NONE, $SPTOR flushes the local LU's send buffer and enters the receive state.
  - If sync level = CONFIRM, $SPTOR sends a confirmation request to the remote LU and, if the return code is VROKAY, enters the receive state. If the return code is VRUNSU, $SPTOR returns the conversation to its previous state.
- FLUSH - flushes the local LU's send buffer and enters the receive state.

locks

This parameter specifies whether the local program must wait for a reply when a request for confirmation is executed following a $SPTOR. This parameter is relevant only if the conversation was allocated with a sync level of CONFIRM, and the $SPTOR is executed with a type of SYNCLVL.

The following arguments are valid for this parameter.

- SHORT - Control is returned to the local program when an acknowledgment is received.
- LONG - control is returned to the local program when data is received from the remote program following an acknowledgment.

DESCRIPTION:

The $SPOTR verb changes the state of the conversation from send to receive. The parameters issued with this verb identify the conversation whose state is being changed, the type of prepare-to-receive to be performed, and when control is to be returned to the local program after the receive.

After the $SPTOR is executed, the conversation enters receive state. If the $SPTOR is unsuccessful, the conversation remains in send state.

RETURN CODES:

The application should check the return code after each execution of a verb. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any verb. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15.

The value you specify for type determines what return codes are possible. In addition to the general return codes, the following values are possible for all types.

Value	Label	Description
0000	VROKAY	OK
004C	VRITYP	Invalid type specified

In addition, If you executed the $SPTOR with a type of SNCLVL and the conversation was allocated with synchronization level of CONFIRM, the following return codes are possible.

Value	Label	Description
0007	VRSEPR	Service program error, purging
0011	VRNSND	Not in send state
0018	VRLRNF	Logical record not finished yet
0041	VRIRID	Invalid resource ID
0047	VRVBNS	Verb not supported (conversation was allocated with a sync level of none)
00B0	VRAETN	TPN not recognized
00C0	VRAEPI	PIP not allowed
00C1	VRAEIP	PIP not specified correctly
00C2	VRAESI	Security not valid
00C3	VRAECM	Conversation type mismatch
00D0	VRAESP	Sync level not supported by program
00D1	VRAERP	Reconnect level not supported by program
00D2	VRAENR	TP not available--no retry
00D3	VRAETR	TP not available--retry
00E0	VRAEAN	ACC not valid
00F1	VRDAPG	Remote deallocation--ABEND program
00F2	VRDASV	Remote deallocation--ABEND service
00F3	VRDATM	Remote deallocation--ABEND timer
0004	VRPEPR	Program error--purging
0103	VRPGER	Resource failure, no retry
0100	VRUNBI	Session unbound by host unexpectedly
0101	VRSSHU	Session shutdown by host orderly
0102	VRURTO	You are timed out by SOPR command
0310	VRADLU	ACTLU/DACTLU received
0711	VRLKFL	Link failure
0712	VRADPU	ACTPU/DACTPU received
0713	VRACSA	$A (SOPR) 'ABORT' AIF node
0714	VRSABT	$S abort AIF group

$SRAW - Receive and Wait Verb

The $SRAW verb causes the LU to wait for data and receive it.

FORMAT:

```
[label]  $SRAW   [vpb address]              P1: $B4
                 [,data buffer]             P2: VP_BUF
                 [,data buffer length]      P3: VP_DLG
                 [fill]                     P4: VP_ICT.VBFILL
```

ARGUMENTS:

vpb address

This parameter contains a pointer to the address of the VPB to be used for this conversation. If not declared, the address is assumed to be in register $B4.

data buffer

This parameter identifies the buffer set up to receive the data from the remote program.

data buffer length

This parameter specifies the maximum length of data that the program can receive.

fill

This parameter specifies whether data is received in logical record format or by buffers.

The following arguments are valid for this parameter.

- BUFFER - data is buffered into units of the length specified in the data buffer length parameter. When the buffer is full or the end of data is indicated, it is transmitted to the local program.
- LL - Each complete or truncated logical record is transmitted to the local program. when a logical record is received that is as long as or longer than the data buffer length, the logical record is broken up into units of that length.

DESCRIPTION:

The $SRAW verb causes the LU to wait for data to arrive at the specified conversation and receive it. The information can be data, status information, or a request for confirmation. If there is data in the receive queue when this verb is executed, the waiting time is eliminated. After $SRAW is executed, control is returned to the local program and the type of information received is indicated.

If the conversation is in send state when this verb is issued, the local LU flushes its send buffer and the conversation changes to receive state. A send indicator is sent to the remote LU, to notify the remote program that it can send data to the local program.

The value of the WHAT_RECEIVED parameter determines the state of the conversation after the $SRAW is executed. If WHAT_RECEIVED indicates DATA, DATA_COMPLETE, DATA_INCOMPLETE, or LL_TRUNCATED, the conversation enters (or remains in) receive state. If WHAT_RECEIVED indicates SEND, the conversation enters send state. If WHAT_RECEIVED indicates CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE, the conversation enters confirm state.

RETURN CODES:

The application should check the return code after each execution of a verb. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any verb. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15.

In addition to the general return codes, the following values are possible.

Value	Label	Description
0000	VROKAY	OK
0002	VRPENT	Program error, not truncating
0003	VRPETR	Program error, truncating
0004	VRPEPR	Program error--purging
0014	VRNSOR	Not in send/receive state
0018	VRLRNF	Logical record not finished yet
0041	VRIRID	Invalid resource ID
00B0	VRAETN	TPN not recognized
00C0	VRAEPI	PIP not allowed
00C1	VRAEIP	PIP not specified correctly

00C2	VRAESI	Security not valid
00C3	VRAECM	Conversation type mismatch
00D0	VRAESP	Sync level not supported by program
00D1	VRAERP	Reconnect level not supported by program
00D2	VRAENR	TP not available--no retry
00D3	VRAETR	TP not available--retry
00E0	VRAEAN	ACC not valid
00F0	VRDANM	Deallocate normal
00F1	VRDAPG	Remote deallocation--ABEND program
00F2	VRDASV	Remote deallocation--ABEND service
00F3	VRDATM	Remote deallocation--ABEND timer
0006	VRSETR	Service error, truncating
0005	VRSENT	Service error, not truncating
0007	VRSEPR	Service error--purging
0103	VRPGER	Resource failure, no retry
0100	VRUNBI	Session unbound by host unexpectedly
0101	VRSSHU	Session shutdown by host orderly
0102	VRURTO	You are timed out by SOPR command
0310	VRADLU	ACTLU/DACTLU received
0711	VRLKFL	Link failure
0712	VRADPU	ACTPU/DACTPU received
0713	VRACSA	$A (SOPR) 'ABORT' AIF node
0714	VRSABT	$S abort AIF group

RETURN PARAMETER

actual data length (VP_ADL)

This field contains the length of the received data. The actual data length includes the two byte binary field that specifies the logical record length and the length of the record itself. The length can range from 2 to 7FFF.

OUTPUT CONTROL WORD

The request to send received field in the output control word (VP_OCT.VBRRTS) indicates whether the remote program has issued a request to send notification, requesting the local program to enter receive state and placing itself in send state.

WHAT_RECEIVED

The WHAT_RECEIVED field (VP_WAR) defines what the transaction program has received, and should be examined when the return code is OKAY. The following values are possible within VP_WAR:

02 SEND (VBRSND)--the remote program has entered receive state causing the local program to enter send state. The local program can now issue a $SSDAT.

04 CONFIRM (VBRCNF)--the remote program has sent a confirmation request to the local program. The local program can respond by issuing a $SCNFD or another verb, such as a $SSERR.

05 CONFIRM DEALLOCATE (VBRCDA)--the remote program has issued a deallocate with type SNCLVL and a synchronization level of CONFIRM. The local program can respond by issuing a $SCNFD or another verb, such as a $SSERR.

06 CONFIRM SEND (VBRCSN)--the remote program has issued a prepare to receive with type SNCLVL and a synchronization level of CONFIRM. The local program can respond by issuing a $SCNFD or another verb, such as a $SSERR.

08 LL_TRUNCATED (VBRLLT)--The $SRAW was issued with the LL fill parameter and the length field is received truncated. The program does not receive the length of the data.

09 DATA INCOMPLETE WHEN LENGTH=0 (VBDIC0)--The $SRAW was issued with a LENGTH of zero and an incomplete logical record is being received by the program. No data is passed to the caller.

0A DATA AVAILABLE WHEN LENGTH=0 (VBDAT0)--The $SRAW was issued with a LENGTH of zero and a complete logical record is being received by the program. No data is passed to the caller.

14 DATA (VBRDAT)--The $SRAW was issued with the buffer fill parameter and data is being received by the program.

15 DATA COMPLETE (VBRDCP)--The $SRAW was issued with the LL fill parameter and a complete logical record, or the completion of a logical record, is being received by the program.

16 DATA INCOMPLETE (VBRDIC)--The $SRAW was issued with the LL fill parameter and an incomplete logical record is being received by the program. The program must issue one or more additional $SRAWs to receive the remainder of the logical record.

$SRTOS - Request to Send Verb

The $SRTOS verb indicates to the remote program that the local program is requesting to enter send state.

FORMAT:

```
[label]  $SRTOS  [vpb address]  Pl: $B4
```

ARGUMENTS:

vpb address

This parameter contains a pointer to the address of the VPB to be used for this conversation. If not declared, the address is assumed to be in register $B4.

DESCRIPTION:

The $SRTOS verb indicates to the remote program that the local program is requesting to enter send state. When the local program receives a send indicator in response, the conversation changes to send state.

If a negative response is received, the conversation remains in receive state. If a positive response is received with a send indicator, the conversation changes to send state.

RETURN CODES:

The application should check the return code after each execution of a verb. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any verb. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15.

In addition to the general return codes, the following values are possible:

Value	Label	Description
0000	VROKAY	OK
0041	VRIRID	Invalid resource ID
0015	VRNRCS	Not in receive/confirm state
0019	VRCSCD	In confirm state (received CONFIRM SEND or CONFIRM DEALLOCATE on the preceding $SRAW

$SSDAT - Send Data Verb

The $SSDAT verb sends data to the remote program.

FORMAT:

```
[label]  $SSDAT  [vpb address]        P1: $B4
                 [,data buffer]       P2: VP_BUF
                 [length]             P3: VP_DLG
```

ARGUMENTS:

vpb address

This parameter contains a pointer to the address of the VPB to be used for this conversation. If not declared, the address is assumed to be in register $B4.

data buffer

This parameter contains a pointer to the local LU's send buffer. This buffer contains the data being sent in the form of logical records. Each logical record consists of a two-byte field specifying the length of the data in that logical record, and the logical record itself. A buffer can contain any number of complete or partial records that fills the buffer.

length

This parameter specifies the length of the data in the local LU's send buffer. This value is independent of the length of data contained in any individual logical record and independent of the size of the send buffer. The maximum length is 32,876 bytes.

DESCRIPTION:

The $SSDAT verb sends data to the remote program. This data can be data, status information, or confirmation. The data is formatted into logical records, which are buffered before being transmitted. A logical record, by definition, can range from 0002 bytes, including only the LL field, to 7FFF bytes, including a two-byte LL field and 32765 bytes of data.

Executing the $SSDAT does not change the state of the conversation.

RETURN CODES:

The application should check the return code after each execution of a verb. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any verb. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15.

In addition to the general return codes, the following values are possible:

Value	Label	Description
0000	VROKAY	OK
0004	VRPEPR	Program error--purging
0010	VRNSND	Not in send state
004A	VRIVLL	Invalid logical record length
0044	VRLNER	Data length errror
0018	VRLRNF	Logical record not finished yet
0041	VRIRID	Invalid resource ID
00B0	VRAETN	TPN not recognized
00C0	VRAEPI	PIP not allowed
00C1	VRAEIP	PIP not specified correctly
00C2	VRAESI	Security not valid
00C3	VRAECM	Conversation type mismatch
00D0	VRAESP	Sync level not supported by program
00D1	VRAERP	Reconnect level not supported by program
00D2	VRAENR	TP not available--no retry
00D3	VRAETR	TP not available--retry
00E0	VRAEAN	ACC not valid
00F1	VRDAPG	Remote deallocation--ABEND program
00F2	VRDASV	Remote deallocation--ABEND service
00F3	VRDATM	Remote deallocation--ABEND timer
0100	VRUNBI	Session unbound by host unexpectedly
0101	VRSSHU	Session shutdown by host orderly
0102	VRURTO	You are timed out by SOPR command
0310	VRADLU	ACTLU/DACTLU received
0711	VRLKFL	Link failure
0712	VRADPU	ACTPU/DACTPU received
0713	VRACSA	$A (SOPR) 'ABORT' AIF node
0714	VRSABT	$S abort AIF group

OUTPUT CONTROL WORD

The request to send received field in the output control word (VP_OCT.VBRRTS) indicates whether the remote program has issued a request to send notification, requesting the local program to enter receive state and placing itself in send state. This condition is true when VP_OCT.VBRRTS is set.

$SSERR - Send Error Verb

The $SSERR verb indicates to the remote program that an error has occurred.

FORMAT:

```
[label]  $SSERR  [vpb address]        P1: $B4
                 [,type]              P2: VP_TYP
                 [,LOG|NO_LOG]        P3: VP_ICT.VBLGDA
                 [,log data buffer]   P4: VP_BUF
                 [,log data length]   P5: VP_DLG
```

ARGUMENTS:

vpb address

This parameter contains a pointer to the address of the VPB to be used for this conversation. If not declared, the address is assumed to be in register $B4.

type

This parameter specifies whether the error has occurred as a result of the application or as a result of the LU services transaction program, to identify to whom the error should be reported.

The following arguments are valid for this parameter.

- PROG - The error has occurred at the application level. The resulting error code is reported to the remote LU.
- SVC - The error has occurred at the LU services level.

{LOG|NO_LOG}

This parameter indicates whether or not the system error log is transferred to the transaction.

log data buffer

This parameter contains a pointer to the local LU's log data buffer. This buffer contains the data being sent to the remote LU's log data buffer.

log data length

This parameter specifies the length of the data in the local LU's log data buffer. This value excludes the logical record length and can be a maximum of 32,763 bytes.

DESCRIPTION:

The $SSERR verb indicates to the remote program that the local program has detected an error. The parameters issued with this verb identify the conversation on which the error has occurred and the type of error which has been detected. The local LU is in send state and the remote LU in receive state. If the conversation was in send state when this verb was issued, the local LU's send buffer is flushed and the state of the conversation does not change.

If the conversation is in receive or confirm state when the $SSERR is executed, the conversation enters send state.

RETURN CODES:

The application should check the return code after each execution of a verb. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any verb. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15.

The state of the conversation when you issue the $SSERR determines what return codes are possible. In addition to the general return codes, the following values are possible after any execution of the $SSERR.

Value	Label	Description
0000	VROKAY	OK
0017	VRNSRC	Not in send, receive, or confirm state
0041	VRIRID	Invalid resource ID
0103	VRPGER	Resource failure, no retry
0100	VRUNBI	Session unbound by host unexpectedly
0101	VRSSHU	Session shutdown by host orderly
0102	VRURTO	You are timed out by SOPR command
0310	VRADLU	ACTLU/DACTLU received
0711	VRLKFL	Link failure
0712	VRADPU	ACTPU/DACTPU received
0713	VRACSA	$A (SOPR) 'ABORT' AIF node
0714	VRSABT	$S abort AIF group

In addition, if the conversation is in send state when you execute the $SSERR, the following return codes are possible.

Value	Label	Description
00B0	VRAETN	TPN not recognized
00C0	VRAEPI	PIP not allowed
00C1	VRAEIP	PIP not specified correctly
00C2	VRAESI	Security not valid
00C3	VRAECM	Conversation type mismatch
00D0	VRAESP	Sync level not supported by program
00D1	VRAERP	Reconnect level not supported by program
00D2	VRAENR	TP not available--no retry
00D3	VRAETR	TP not available--retry
00E0	VRAEAN	ACC not valid
00F1	VRDAPG	Remote deallocation--ABEND program
00F2	VRDASV	Remote deallocation--ABEND service
00F3	VRDATM	Remote deallocation--ABEND timer
0004	VRPEPR	Program error--purging

If the conversation is in receive state when you execute the $SSERR, the following return codes are possible.

Value	Label	Description
004C	VRITYP	Invalid type specified
00B0	VRAETN	TPN not recognized
00C0	VRAEPI	PIP not allowed
00C1	VRAEIP	PIP not specified correctly
00C2	VRAESI	Security not valid
00C3	VRAECM	Conversation type mismatch
00D0	VRAESP	Sync level not supported by program
00D1	VRAERP	Reconnect level not supported by program
00D2	VRAENR	TP not available--no retry
00D3	VRAETR	TP not available--retry
00E0	VRAEAN	ACC not valid
00F0	VRDANM	Deallocate normal
0007	VRSEPR	Service program error, purging

If the conversation is in confirm state when you execute the $SSERR, the following return codes are possible.

Value	Label	Description
004C	VRITYP	Invalid type specified

OUTPUT CONTROL WORD

The REQUEST TO SEND RECEIVED field in the output control word (VP_OCT.VBRRTS) indicates whether the remote program has issued a request to send notification, requesting the local program to enter receive state and placing itself in send state. This condition is indicated when this bit is set.

$SWAIT - Wait Verb

The $SWAIT verb waits for posting to occur on any of a list of conversations.

FORMAT:

```
[label]  $SWAIT  [vpb list]     P1: $B4
```

ARGUMENTS:

vpb list

This parameter contains a pointer to the address of the list of VPBs identifying the conversations on which the $SWAIT is waiting.

The VPB list consists of a single precision unsigned integer containing the number of pointers in the list, followed by a list of the addresses of all of the VPBs used by the transaction program.

DESCRIPTION:

The $SWAIT verb causes the local program to suspend processing and wait for posting to occur on any conversation from a list of conversations. This verb is issued after the $SPONR (Post on Receipt) verb. Following the $SWAIT verb, you must execute the $SRAW verb to gain access to the data.

If you have issued the $SPONR to allow the application to continue other program processing while waiting for data from the host. The $SWAIT brings you back to the conversation that has been posted.

Executing the $SWAIT verb does not change the state of the conversation.

RETURN CODES:

The application should check the return code after each execution of a verb. Bits 0 through 4 have special meaning and represent general AIF return codes that could occur for any verb. These bits should be examined individually, then "masked out" so that the application can examine bits 5 through 15.

In addition to the general return codes, the following values are possible.

Value	Label	Description
0000	VROKAY	OK
0001	VRUNSU	Unsuccessful
0041	VRIRID	Invalid resource ID (the verb parameter list contains an invalid resource identifier. $B4 contains the pointer to this ID.
0016	VRNRCV	Not in receive state

RETURN PARAMETER

conversation posted

The address of the verb parameter block for the conversation that has been posted is returned in $B4.

If you have multiple conversations, then this parameter contains the VPB address of the conversation that has been posted.

$SACEB - ASCII-to-EBCDIC Conversion Routine

Converts data from ASCII to EBCDIC.

FORMAT:

```
    label   $SACEB
```

ARGUMENT:

There are no arguments associated with this macro.

DESCRIPTION:

These session calls convert data from ASCII to EBCDIC. The maximum length of data that can be converted by a single call is 32,767 bytes.

Since IBM handles data in EBCDIC and AIF handles it in ASCII, you may sometimes wish to convert data from one to the other, either before sending or after receiving.

The AIF software provides the following macros to perform these conversions.

$SACEB	ASCII-To-EBCDIC Conversion

When this macro is activated, you must initialize registers $B2, $B4, $R2, $R4, and $R6 to contain the values listed in Table 4-4. If you wish to convert in place, $B2 and $B4 must reference the same address.

Table 4-4. Register Contents at Conversion

Register	Contents
$B2	Pointer to buffer to be converted
$B4	Pointer to buffer to contain converted data
$R2	Index for buffer to be converted
$R3	Function code ($SACEB=1; $SEBAC=2)
$R4	Index for buffer to contain converted data
$R6	Length of data in bytes

IMPORTANT!

Do not convert the two-byte binary LL field of the logical record.

$SEBAC - EBCDIC-to-ASCII Conversion Routine

Converts data from EBCDIC to ASCII.

FORMAT:

```
label   $SEBAC
```

ARGUMENT:

There are no arguments associated with this macro.

DESCRIPTION:

These session calls convert data from EBCDIC to ASCII. The maximum length of data that can be converted by a single call is 32,767 bytes.

Since IBM handles data in EBCDIC and AIF handles it in ASCII, you may sometimes wish to convert data from one to the other, either before sending or after receiving.

The AIF software provides the following macros to perform these conversions.

$SEBAC	EBCDIC-To-ASCII Conversion

When this macro is activated, you must initialize registers $B2, $B4, $R2, $R4, and $R6 to contain the values listed in Table 4-4. If you wish to convert in place, $B2 and $B4 must reference the same address.

Table 4-5. Register Contents at Conversion

Register	Contents
$B2	Pointer to buffer to be converted
$B4	Pointer to buffer to contain converted data
$R2	Index for buffer to be converted
$R3	Function code ($SACEB=1; $SEBAC=2)
$R4	Index for buffer to contain converted data
$R6	Length of data in bytes

IMPORTANT!

Do not convert the two-byte binary LL field of the logical record.

Section 5
PROGRAMMING LU TYPE 6.2 CONVERSATIONS IN COBOL

This section describes the AIF conversation verbs that the COBOL programmer uses to converse over an LU Type 6.2 conversation with host transaction programs. Topics include:

- COBOL conversation verbs
- Conversation format
- Programming Considerations
  - WORKING-STORAGE SECTION
  - Checking the return code
  - Conversation states
  - Session calls

COBOL CONVERSATION VERBS

The basic conversation verbs used by AIF in a COBOL application program call correspond to Assembly language subroutines using the "CALL...USING..." verb. These calls are listed in Table 5-3.

The parameters that these verbs use are defined in the WORKING-STORAGE SECTION of the COBOL program. In this manual, these parameters are defined in the discussion of the WORKING-STORAGE SECTION and are listed without redefinition in the format description of each conversation verb.

At the completion of each execution of a verb, when control is returned to the application, a return code is placed in the RETURNS field. This return code indicates whether a verb has been completed error free. The application should check the return code after each execution of a verb to verify that the execution was completed error-free.

Sample COBOL programs is provided in Appendix C to demonstrate an AIF application in a COBOL program.

CONVERSATION FORMAT

The conversation verbs used by AIF in a COBOL program reference Assembly language subroutines which include system-provided macrocalls. The COBOL conversation verbs have a list of arguments that must be specified each time a verb is executed. These arguments, which you define in the WORKING-STORAGE SECTION, correspond to parameters in the verb parameter block (VPB) that are used by the Assembly language subroutine. These arguments are positional and must be included each time the verb is issued. The AIF COBOL conversation verbs follow the conventions for COBOL (described in detail in the ONE PLUS COBOL 74 Language Reference Manual (HE34).

When an AIF conversation is activated, it defines the resources to be made available to the session while that conversation is active. AIF allocates a session for a conversation from a group of available LU sessions. AIF can either request a session to the host system at initiation or it can wait for an application to request to allocate a conversation before requesting a session. The time of logon is a configuration option.

An application requests to allocate a conversation with a remote transaction program by executing the CSALLO verb. AIF looks for an available session to allocate for that conversation. If no session is immediately available, the application can specify whether control should be returned to the program. The conversation uses a session for only the time it takes to execute the verb. After the verb is executed, the conversation retains its resources until a deallocate verb is issued or a deallocate-confirmation is received from the host application.

An application gains access to a host-initiated conversation executing a CSATCH verb. When an ATTACH command is received from the host, AIF loads the transaction program by spawning a group with the attached application as the lead task, and sends a response to the host that the program is attached. If the DPS 6 or DPS 6 PLUS program is intended to be part of a host-initiated session, it must execute the CSATCH verb before any other verbs are issued.

PROGRAMMING CONSIDERATIONS

The special considerations that the COBOL programmer must bear in mind fall into the following categories.

- WORKING-STORAGE SECTION
- Conversation state
- Host-initiated sessions
- Linking the program
- Checking the return code
- Conversation format.

WORKING-STORAGE SECTION

The WORKING-STORAGE SECTION defines the area to be used as the SNA work area. The parameters specified in these fields are passed to the VPB when the conversation verbs are executed.

The following parameters must be defined in the WORKING-STORAGE SECTION. These parameters are used to create the verb parameter block which is used by the Assembly language subroutines you are calling.

Figure 5-1 shows a sample WORKING-STORAGE SECTION in which the SNA work area has been defined. The data-names that are used here are examples; you can name them according to your own naming conventions.

These fields are defined as follows:

SNA-WORK-AREA

This input parameter is the name of a contiguous memory area that is at least 200 bytes long. This corresponds to the verb parameter block (VPB) argument of the Assembly language conversation verbs.

Example:

```
SNA-WORK-AREA1          PIC X(200).
```

NODE-NAME

This parameter identifies the AIF node on the DPS 6 or DPS 6 PLUS to which the application is directing this verb. This field can contain up to eight alphanumeric characters.

Example:

```
77 NODE-NAME     PIC X(8) VALUE "AIF501".
```

```
DATA DIVISION.
WORKING-STORAGE SECTION.
77   SNA-WORK-AREA1                PIC X(200).
77   NODE-NAME                     PIC X(8) VALUE "AIF501".
77   REMOTE-LU-NAME                PIC X(8) VALUE "A06CICS".
77   CONVERSATION-ID               PIC X(4).
77   TRANS-PROGRAM-NAME            PIC X(8) VALUE "TP42".
77   TRANS-TPN                     PIC X VALUE "Y".
77   NO-TRANS-TPN                  PIC X VALUE "N".
77   STD-NAME                      PIC XX VALUE "BB".
77   RETURN-CONTROL                PIC X VALUE "A".
77   SYNC-LEVEL                    PIC X VALUE "C".
77   TYPE                          PIC X VALUE "S".
77   LOG                           PIC X VALUE "L".
77   NO-LOG                        PIC X VALUE "N".
01   LOG-DATA-RECORD               PIC 9(5).
     05 LL                         COMP-1 VALUE 84.
     05 GDS-ID                     COMP-1.
     05 LOG-DATA                   PIC X(80).
77   LOG-DATA-SIZE                 PIC 9(5) VALUE 80
77   LOCKS                         PIC X VALUE "S".
77   SEND-BUFFER                   PIC X(80).
77   SEND-BUFFER-SIZE              PIC 9(5) VALUE 80.
77   RECEIVE-BUFFER                PIC X(80).
77   RECEIVE-BUFFER-SIZE           PIC 9(5) VALUE 80.
77   RECEIVED-DATA-LENGTH          PIC 9(5) VALUE 0.
77   FILL                          PIC X VALUE "B".
01   RETURNS.
     02 RETURN-A.
        03   ABEND-DEALLOCATE      PIC X VALUE "N".
        03   STOP-RCVD        PIC X VALUE "N".
        03   SERV-REQ-CANC         PIC X VALUE "N".
        03   SERV-REQ-COMP         PIC X VALUE "N".
        03   COBOL-ERROR           PIC X VALUE "N".
     02 RETURN-B                   PIC 9(4) VALUE 0.
01   OUTPUT-CONTROL-WORD.
     02 REQ-TO-SEND-RCVD           PIC X VALUE "N".
     02 CONV-POSTED                PIC X VALUE "N".
     02 WHAT-RECEIVED              PIC 99.
77   POSTED-CONV-ID                PIC X(4).
77   RCVD-SENSE                    PIC X(8).
77   CONVERT-FROM-FIELD            PIC X(20).
01   CONVERT-FROM-LEFT-POSIT       COMP-1 VALUE 1.
77   CONVERT-TO-FIELD              PIC X(20).
01   CONVERT-TO-LEFT-POSIT         COMP-1 VALUE 6.
01   CONVERSION-LENGTH             COMP-1 VALUE 10.
```

Figure 5-1. Working-Storage Section for LU Type 6.2

REMOTE-LU-NAME

The name by which the remote LU is known to this application. This field can contain up to eight alphanumeric characters. This name equates to the application VTAM macro at the IBM host.

Example:

```
77 REMOTE-LU-NAME     PIC X(8) VALUE "A06CICS".
```

CONVERSATION-ID

This parameter returns a unique four-character conversation-id which is supplied by AIF.

Example:

```
77 CONVERSATION-ID          PIC X(4).
```

TRANS-PROGRAM-NAME

This parameter contains the name of the transaction program to be attached to the host. This host program becomes the session partner of the local program.

Example:

```
77 TRANS-PROGRAM-NAME      PIC X(8) VALUE "TP42".
```

TRANSLATE-TPN
NO-TRANSLATE-TPN

This parameter specifies whether the transaction program name specified above requires translation from ASCII to EBCDIC.

Example:

```
77 TRANS-TPN               PIC X VALUE "Y".
77 NO-TRANS-TPN            PIC X VALUE "N".
```

STD-NAME

The configured session type descriptor (STD) which lists the attributes of the conversation to be allocated. This field consists of two alphanumeric characters and is defined at AIF configuration time.

Example:

```
77 STD-NAME           PIC XX VALUE "BB".
```

RETURN-CONTROL

This parameter indicates whether the local LU should return control to the local program, in the event that it is unable to allocate a conversation.

The following arguments are valid for this parameter:

- A (AVAIL) - allocates a session for the conversation before returning control to the program.
- I (IMMEDIATE) - allocates a session for the conversation if one is immediately available and then returns control to the session.

Example:

```
77 RETURN-CONTROL     PIC X VALUE "A".
```

SYNC-LEVEL

This parameter indicates how the local and remote programs perform confirmation processing on the specified conversation.

The following arguments are valid for this parameter:

- N (NONE) - do not perform confirmation processing on this conversation. Programs that specify NONE do not issue any verbs or recognize return parameters related to synchronization.
- C (CONFIRM) - performs confirmation processing only on this conversation. Programs that specify CONFIRM issue verbs and recognize returned confirmation parameters.

Example:

```
   77 SYNC-LEVEL      PIC X VALUE "C".
```

TYPE

This parameter specifies whether the execution of the verb is to be completed as part of this verb or deferred until another verb is issued or a condition is met.

The following arguments can be used for this parameter:

- S (SYNCLVL) - executes the verb according to the synchronization level specified when the conversation was allocated:

- F (FLUSH) - flushes the local LU's send buffer and executes the verb.

The following TYPE arguments are used for error handling and are application dependent.

- P (ABEND_PROG) - with CSDEAL, flushes the local LU's send buffer when the conversation is in send or defer state and deallocates normally.

- V (ABEND_SVC) - with CSDEAL, flushes the local LU's send buffer when the conversation is in send or defer state and deallocates the conversation abnormally.

- T (ABEND_TIMER) - with CSDEAL, flushes the local LU's send buffer when the conversation is in send or defer state and deallocates the conversation abnormally.

Example:

```
77 TYPE              PIC X VALUE "S".
```

LOG

This parameter indicates whether or not the system error log is transferred to the transaction when deallocating the conversation. The value of LOG is L; the value of NO-LOG is N. If you specify that error logging should occur, the TYPE parameter must be specified as P, V, or T.

Example:

```
77 LOG               PIC X VALUE "L".
77 NO-LOG            PIC X VALUE "N".
```

LOG-DATA

This parameter is a pointer to the product specific error data that is kept in the system error logs of the local and remote LUs. Error data is declared in the General Data Stream record format as described in the IBM <u>SNA Format and Protocol Reference Manual for LU Type 6.2.</u> The record must start on a word boundary and all fields must be filled by the application. This parameter is only used with an ABEND deallocation type.

Example:

```
01 LOG-DATA-RECORD.
   05 LL                    PIC 9(4) COMP-1 VALUE 84.
   05 GDS-ID                PIC 9 COMP-1.
   05 LOG-DATA              PIC X(80).
```

LOG-DATA-SIZE

This parameter specifies the length of the LOG-DATA.

Example:

```
77 LOG-DATA-SIZE            PIC 9(5) VALUE 80
```

LOCKS

This parameter specifies whether the local program waits for a reply when a request for confirmation points is executed following a CSPTOR (prepare to receive) verb.

The following arguments are valid for this parameter.

- S (SHORT) - Control is returned to the local program when an acknowledgement is received.
- L (LONG) - control is returned to the local program when data is received from the remote program following an acknowledgment.

SEND-BUFFER

This parameter identifies the buffer which holds the data to be sent to the remote program. This buffer contains the data being sent in the form of logical records. Each logical record consists of a two-byte field specifying the length of the data in that logical record, and the logical record itself. A buffer can contain any number of complete or partial records that fills the buffer.

Example:

```
77 SEND-BUFFER              PIC X(80).
```

SEND-BUFFER-SIZE

This parameter specifies the length of the SEND-BUFFER.

Example:

```
77 SEND-BUFFER-SIZE         PIC 9(5) VALUE 80.
```

RECEIVE-BUFFER

This parameter identifies the buffer which receives the data from the remote program.

Example:

```
77 RECEIVE-BUFFER            PIC X(80).
```

RECEIVE-BUFFER-SIZE

This parameter specifies the length of the RECEIVE-BUFFER.

Example:

```
77 RECEIVE-BUFFER-SIZE       PIC 9(5) VALUE 80.
```

RECEIVED-DATA-LENGTH

This parameter specifies the actual length of the data which has been received from the remote program.

Example:

```
77 RECEIVED-DATA-LENGTH      PIC 9(5).
```

FILL

This parameter specifies how the program receives data in terms of the logical record format of the data. The following arguments are valid for this parameter.

- B (BUFFER) - data is buffered into units of the length specified in the LENGTH parameter, independent of its logical record format. The verb is executed when the buffer is full or the end of data is indicated.
- L (LL) - the verb is executed when a complete or truncated logical record is received, or when part of a logical record is received that is at least as long as the length specified in the LENGTH parameter.

Example:

```
77 FILL             PIC X VALUE "B".
```

RETURNS

This output parameter defines the field into which the return code from all AIF session calls is placed. The RETURNS field is divided into RETURN-A, which consists of five yes/no conditions, and RETURN-B, which contains a four character decimal status code to provide further detail about the conditions indicated in RETURN-A.

RETURN-A reports the following conditions:

- ABEND-DEALLOCATE--the conversation has Abended.
- STOP-RCVD--SOPR stop command has been received.
- SERV-REQ-CANC--This request has been cancelled.
- SERV-REQ-COMP--This request has been completed.
- COBOL-ERROR--A COBOL interface error has occurred.

Example:

```
01 RETURNS.
   02  RETURN-A.
       03  ABEND-DEALLOCATE        PIC X VALUE "N".
       03  STOP-RCVD               PIC X VALUE "N".
       03  SERV-REQ-CANC           PIC X VALUE "N".
       03  SERV-REQ-COMP           PIC X VALUE "N".
       03  COBOL-ERROR             PIC X VALUE "N".
       02  RETURN-B                PIC 9(4) VALUE 0.
```

TIMEOUT

This output parameter provides a formatted data area for the date and time that a session must be stopped when a STOP command is processed for the session or node. This field must be 14 decimal digits long, as in the following format:

Example:

```
01 TIMEOUT
  02 DATE1.
   03 YY          PIC 99 VALUE 0.
   03 MM          PIC 99 VALUE 0.
   03 DD          PIC 99 VALUE 0.
  02 TIME1.
   03 HH          PIC 99 VALUE 0.
   03 MN          PIC 99 VALUE 0.
   03 SSSS        PIC 9(4) VALUE 0.
```

RCVD-SENSE

This output parameter contains the hexadecimal representation of the sense data from the host if sense data is present. This field corresponds to VP_ESD in the VPB.

Example:

```
77 RCVD-SENSE          PIC X(8).
```

OUTPUT-CONTROL-WORD

This output parameter provides information about the received data. The characteristics that can be specified are listed below. Each of these parameters must be stated. The possible values for the first two parameters are "Y" or "N". For the third parameter refer to the Receive and Wait (CSRAW) verb for the possible parameters.

```
01 OUTPUT-CONTROL-WORD.
   02 REQ-TO-SEND-RCVD          PIC X VALUE "N".
   02 CONV-POSTED               PIC X VALUE "N".
   02 WHAT-RECEIVED             PIC 99.
```

CONVERT-FROM-FIELD

This input parameter defines the buffer to be converted by the ASCII-to-EBCDIC conversion subroutines. The maximum size of this buffer is 32,767 bytes.

Example:

```
77 CONVERT-FROM-FIELD PIC X(20).
```

CONVERT-FROM-LEFT-POSIT

This input parameter provides a starting index for the data in CONVERT-FROM-FIELD.

Example:

```
01 CONVERT-FROM-LEFT-POSIT COMP-1 VALUE 1.
```

CONVERT-TO-FIELD

This input parameter defines the buffer into which the converted data will be placed by the ASCII-to-EBCDIC conversion subroutines. The maximum size of this buffer is 32,767 bytes.

Example:

```
77 CONVERT-TO-FIELD       PIC X(15).
```

CONVERT-TO-LEFT-POSIT

This input parameter provides a starting index for the data in CONVERT-TO-FIELD.

Example:

```
01 CONVERT-TO-LEFT-POSIT   COMP-1 VALUE 6.
```

CONVERSION-LENGTH

This input parameter contains the length in bytes of the data to be converted. The maximum length of this data is 32,767 bytes.

Example:

```
01 CONVERSION-LENGTH       COMP-1 VALUE 10.
```

Conversation States

The subset of verbs that a program can issue at a given time is determined by the state of the conversation at that time. For example, if a conversation is in receive state, it cannot issue a send verb without first issuing a verb to change the conversation to send state. The program must be aware of the state of the conversation. Executing many of the basic conversation verbs causes the conversation to change its state.

Table 5-1 lists the conversation states and their definition. The description of each verb includes the state of the conversation at the end of execution. Table 5-2 shows what verbs a conversation can issue from each state.

Table 5-1. Conversation States

State	Definition
Reset	The state in which the program can allocate a conversation.
Send	The state in which the program can send data or request confirmation.
Defer	The state in which the program can request confirmation or flush the LU's send buffer to prepare to change states.
Receive	The state in which the program can receive data or confirmation information.
Confirm	The state in which the program can send a confirmation reply.

Table 5-2. Conversation States From Which Verbs Can Be Issued

Verb	Conversation State				
	Reset	Send	Defer	Receive	Confirm
CSALLO	X				
CSATCH	X				
CSCONF		X			
CSCNFD					X
CSDEAL flush		X			
CSDEAL sync level		X			
CSDEAl abend		X	X	X	X
CSFLSH		X	X		
CSPONR				X	
CSPTOR		X			
CSRAW		X		X	
CSRTOS				X	X
CSSDAT		X			
CSSERR		X		X	X
CSWAIT					X

Host-Initiated Sessions

AIF supports host-initiated sessions; that is, it accepts unsolicited binds.

When the application program begins execution, it must issue a CSATCH session call as the first session call, providing the STD name and the node name for the LU to be used. The CSATCH session call allows the AIF application access to a host-initiated conversation. AIF associates the first unsolicited bind (host-initiated session request) to the first CSATCH session call from the task group that AIF spawned.

AIF accepts the session and looks for the program name in the first four bytes of the first record received, then spawns a group based on the ATTACH_PROGRAM entry. If none is provided, default values are used to spawn the group.

The application can issue multiple CSATCHs to check for additional host-initiated sessions intended for this application. Each of these LUs must have the same group_id specified in the LU entry in the configuration file.

NOTE

In order to execute a START_UP.EC instead of an attached program, you must create an attach program table entry with a dummy name (eg., ATTACH_PROG=ABC), specifying the appropriate spawn group parameters, and include an ALIAS for ABC (eg., ALIAS=>>SYSLIB2>EC?EXECL) to execute the START_UP.EC specified in the home directory. Refer to <u>SNA6 Network Configuration</u> for further information.

Linking the Program

If a COBOL application program is written as a program to be attached, that is, it includes an ATTACHED verb (CSATCH), then a LINKAGE SECTION must be included in the program. The LINKAGE SECTION must include three entries to accommodate the node name, STD name, and base level, as in the following example:

```
LINKAGE SECTION.
77  NODE     PIC X(8).
77  STD      PIC XX.
77  BASE_LVL PIC 99.
PROCEDURE DIVISION USING NODE, STD, BASE_LVL.
```

The LINKAGE SECTION is necessary whether the program is to be compiled using COBOLA or COBOLM. The programs are coded in the same way, regardless of which compiler is used, but they are linked differently.

Within the COBOL application program, the three fields in the LINKAGE SECTION must be moved to corresponding fields in WORKING-STORAGE before they can be used in any AIF calls.

Two sample LINK directive sets are presented below to demonstrate the different Linker directives you can use. The following matrix shows which set you should use, based upon LU type, whether you are writing an attached program, and the COBOL compiler you are using.

Compiler used:	COBOLA	COBOLM
ATTACHED calls used:	4	2
No ATTACHED calls used:	2	2

```
LINK DIRECTIVE SET 1

&N
&A
LINKER &1
LIB >LDD>ZCART/
LIB >LDD>ZCMRT*
LINK &1
LINK CSPHRA
MAP
QT
```

* Use either LIB, where ZCART is used for COBOLA and ZCMRT is used for COBOLM.

```
LINK DIRECTIVE SET 2

&N
&A
LINKER &1
LIB >LDD>ZCART
LINKN CSLEAD
LINK &1
LINK CSPHRA
MAP
LDEF CBLADR,&1
QT
```

NOTES

The module CSPHRZ is the parameter processing routine for LU Type 6.2 calls.

Programs compiled by COBOLM automatically have the node name, STD name, and base level moved to the LINKAGE SECTION. Programs compiled by COBOLA use the CSLEAD Linker module to perform this function. This module must be linked into the bound unit of any program that executes a CSACPT or CSATCH and is compiled using COBOLA.

Refer to the Multiuser COBOL Compiler User's Guide (HE32) for information about linking programs compiled under COBOLA and COBOLM into a single bound unit.

Checking the Return Code

On return from AIF, a COBOL interface routine fills the output parameter fields with the VPB results from the subroutine.

After the session call is made, a return code is placed in the RETURNS field. The RETURNS field is divided into RETURN-A, which consists of five yes/no conditions, and RETURN-B, which contains a four-character decimal status code, known as the return code, to provide further detail about the conditions indicated in RETURN-A.

The following values are possible for RETURN-A:

- ABEND-DEALLOCATE--The conversation has ABENDed, the LU's receive buffer has been flushed, and the conversation has been deallocated.
- STOP-RCVD--An SOPR STOP command received. If the TIME argument is supplied with the STOP command, check the TIME field for the time at which the session will be ended. This field indicates how much time you have to complete the session.
- SERV-REQ-CANC--This request has been cancelled. The application must issue it again if necessary.
- SERV-REQ-COMP--This request has been completed.
- COBOL-ERROR--Error in using COBOL interface to AIF. See RETURN-B for return code.

If the value of COBOL-ERROR is Y, then an error has occurred in the COBOL interface to AIF. Following are the general COBOL return codes that can be received in RETURN-B after executing any of the verbs to indicate a COBOL error. The value of XX is the number of the parameter in which there is an error.

Code	Meaning
XX01	Unrecognized parameter
XX02	Parameter must be 1 byte long
XX03	Parameter must be 5 bytes long
XX04	Default not acceptable
XX05	Node name error
XX06	Remote LU name error
XX07	Not session-ID
XX08	Unknown interrupt type
XX09	Nondecimal digit
XX10	Nonhexadecimal digit
XX11	Error in conversion

The values of both RETURN-A and RETURN-B should be checked after the execution of each verb. Since it is possible to have more than one Y value in RETURN-A, and to have a value greater than zero after a successfully completed call, the application should check all fields in RETURN-A and RETURN-B for all possible combinations.

If the return code contains a "no error" message, go to the next segment of the program. If the return code contains an error condition, you might decide to record it to an error-out file, go to another segment of the program, or shut down completely.

Additional return codes are listed with the individual conversation verbs to which they pertain. The return codes and their values are listed in Appendix F.

INDIVIDUAL VERB FORMAT

Table 5-3 lists the basic conversation verbs that are supported by AIF. These verbs are described in detail on the following pages.

Table 5-3. AIF LU Type 6.2 Verbs

Verb	Description
CSALLO	Allocate verb
CSATCH	Attached verb
CSCONF	Confirm verb
CSCNFD	Confirmed verb
CSDEAL	Deallocate verb
CSFLSH	Flush verb
CSPONR	Post on Receipt verb
CSPTOR	Prepare to Receive verb
CSRAW	Receive and Wait verb
CSRTOS	Request to Send verb
CSSDAT	Send Data verb
CSSERR	Send error verb
CSWAIT	Wait verb
CSACEB	ASCII-EBCDIC Conversion
CSEBAC	EBCDIC-ASCII Conversion

CSALLO - Allocate Verb

The CSALLO verb is used to allocate a conversation between a local program and a remote program.

FORMAT:

```
CALL "CSALLO" USING SNA-WORK-AREA
                    NODE-NAME
                    REMOTE-LU-NAME
                    CONVERSATION-ID
                    TRANS-PROGRAM-NAME
                    TRANS-TPN|NO-TRANS-TPN
                    STD-NAME
                    RETURN-CONTROL
                    SYNC-LEVEL
                    RETURNS
                    TIMEOUT
                    RCVD-SENSE
                    OUTPUT-CONTROL-WORD
```

DESCRIPTION:

The CSALLO verb first initiates a session between a local LU and a remote LU, then allocates a conversation over that session, between a local program and a remote program, and puts the conversation in send state. Once you have allocated a conversation over a session, that session becomes unavailable to other conversations until this conversation is deallocated.

The CSALLO verb is used to allocate conversations for either transaction programs or service component programs. The parameters issued with this verb identify the partners in the conversation and initialize the returned fields.

The CSALLO verb must be issued before any other AIF verbs that refer to the specified conversation.

When issuing the CSALLO, you have the option of whether you want to wait for an available session or to return control to the local program for processing if one is not immediately available. These options are addressed by the RETURN-CONTROL parameter.

1. The A (AVAIL) option allocates a session for the conversation before returning control to the program. If the local LU fails to obtain a session for this conversation, an allocation error is reported in the CSALLO return code.

2. The I (IMMED) option allocates a session for the conversation if one is immediately available and then returns control to the session. The following conditions are possible:

   - If a session is immediately available, the conversation is allocated and control is returned with a return code of OK. The IMMED option requests that a local LU is the contention winner.

   - If a session in not immediately available, the conversation is not allocated and control is returned with a return code of unsuccessful.

   - If a session is immediately available and an error occurs in allocating a conversation, the error is reported in the return code.

NOTE

If the LU is configured with the contention winner as nonnegotiable, an LU must be both reserved and preestablished.

The CSALLO verb must be issued before any other verbs that refer to the specified conversation. At the completion of the CSALLO verb, the conversation enters send state.

RETURN CODES:

The application should check the return code after each verb is issued to determine if the call has been completed error free. After the execution of the CSALLO verb, the following combinations are possible:

- If SERV-REQ-COMP=Y and RETURN-B=0, the conversation has been allocated.

- If the value of another field in RETURN-A is Y, the CSALLO was not allocated successfully and RETURN-B contains the return code to indicate the reason for the error.

The value that you specified for the RETURN-CONTROL parameter determines which return codes are possible. The following values are possible in RETURN-B for any value of RETURN-CONTROL.

Value	Description
0000	OK
0064	Invalid node name
0066	Invalid transaction program name (null value)
0073	Synchronization level not supported by LU
0075	Invalid return control
0150	Node not yet active
0151	No active LU for session
0152	No LU available for session
0153	Invalid STD name
0154	Invalid LU type in STD

In addition, if you specified a return control of IMMED, the following value is possible:

Value	Description
0001	Unsuccessful

CSATCH - Attached Verb

The CSATCH verb is used by an attached program to gain access to the coversation.

FORMAT:

```
CALL "CSATCH" USING SNA-WORK-AREA
                    NODE-NAME
                    REMOTE-LU-NAME
                    CONVERSATION-ID
                    STD-NAME
                    RETURN-CONTROL
                    SYNC-LEVEL
                    RETURNS
                    TIMEOUT
                    RCVD-SENSE
                    OUTPUT-CONTROL-WORD
```

DESCRIPTION:

The CSATCH verb causes an application to connect to a host-initiated conversation. When the host issues an ATTACH command to allocate a conversation, AIF loads the local transaction by spawning a group with the program as the lead task. When the program is loaded, the COBOL program must issue the CSATCH verb to direct AIF to associate the session to this COBOL program.

The CSATCH verb can be issued with the following values for SYNC-LEVEL:

- NONE - do not perform confirmation processing on this conversation. Programs that specify NONE do not issue any verbs or recognize return parameters related to synchronization.

- CONFIRM - performs confirmation processing only on this conversation. Programs that specify CONFIRM issue verbs and recognize returned confirmation parameters, but do not recognize return parameters related to synchronization.

If the application is intended to connect to a host-initiated session, the CSATCH must be the first verb executed. After the CSATCH verb is executed, the conversation enters receive state.

RETURN CODES:

The application should check the return code after each verb is issued to determine if the call has been completed error free. After the execution of the CSATCH verb, the following combinations are possible.

- If SERV-REQ-COMP=Y and RETURN-B=0, the attached program now has access to the session.
- If the value of another field in RETURN-A is Y, the CSATCH was not successful and RETURN-B contains the return code to indicate the reason for the error.

The following values are possible in RETURN-B:

Value	Description
0000	OK
0064	Invalid node name
0073	Synchronization level not supported by LU
0153	Invalid STD name
0155	No LU attached by remote TP

CSCONF - Confirm Verb

The CSCONF verb sends a confirmation request to the remote program.

FORMAT:

```
CALL "CSCONF" USING SNA-WORK-AREA
```

DESCRIPTION:

The CSCONF verb requests that the remote program send an acknowledgement, and waits for a response. The CSCONF verb is used for confirmation processing and in verifying that the conversation has been allocated or data has been received. CSCONF is not used if the conversation has been allocated with a synchronization level of NONE. This verb causes the LU to flush its send buffers.

When the CSCONF verb is issued in defer state following a CSPTOR, the conversation enters receive state. When the CSCONF verb is issued in defer state following CSDEAL, the conversation enters reset state. When the CSCONF verb is issued in send state, the state does not change.

RETURN CODES:

The application should check the return code after each verb is issued to determine if the call has been completed error free. After the execution of the CSCONF verb, the following combinations are possible.

- If SERV-REQ-COMP=Y and RETURN-B=0, the request for confirmation has been sent.
- If the value of another field in RETURN-A is Y, the CSCONF was not successful and RETURN-B contains the return code to indicate the reason for the error.

The following values are possible in RETURN-B:

Value	Description
0000	OK
0004	Program error--purging
0007	Service program error, purging
0017	Not in send/defer state
0024	Logical record not finished yet
0065	Invalid resource ID

0071	Verb not supported (conversation was allocated with a sync level of none)
0241	Remote deallocation--ABEND program
0242	Remote deallocation--ABEND service
0043	Remote deallocation--ABEND timer
0256	Session unbound by host unexpectedly
0257	Session shutdown by host orderly
0258	You are timed out by SOPR command
0259	Resource failure, no retry--session abort due to unrecoverable protocol errror
0784	ACTLU/DACTLU received
1809	Link failure
1810	ACTPU/DACTPU received
1811	$A (SOPR) ABORT AIF node
1812	$S ABORT AIF group

OUTPUT CONTROL WORD

The REQ-TO-SEND-RCVD field in the OUTPUT-CONTROL-WORD indicates whether the remote program has issued a request to send notification, requesting the local program to enter receive state. The remote program enters send state.

CSCNFD - Confirmed Verb

The CSCNFD verb sends a confirmation reply to the remote program.

FORMAT:

```
CALL "CSCNFD" USING SNA-WORK-AREA
```

DESCRIPTION:

The CSCNFD verb sends a confirmation to a remote program, always in response to a request for confirmation. The CSCNFD verb is used in confirmation processing and error detection and follows a receive-and-wait verb (CSRAW). CSCNFD is not used if the conversation has been allocated with a synchronization level of NONE.

The WHAT-RECEIVED parameter of the CSRAW verb determines what state the conversation enters after the CSCNFD is executed. If the CSRAW returned a CONFIRM indicator, the conversation enters receive state. If the CSRAW indicated CONFIRM-SEND, the conversation enters send state. If the CSRAW indicated CONFIRM-DEALLOCATE, the conversation enters reset state.

RETURN CODES:

The application should check the return code after each verb is issued to determine if the call has been completed error free. After the execution of the CSCNFD verb, the following combinations are possible.

- If SERV-REQ-COMP=Y and RETURN-B=0, the confirmation response has been sent.
- If the value of another field in RETURN-A is Y, the CSCNFD was not successful and RETURN-B contains the return code to indicate the reason for the error.

The following values are possible in RETURN-B:

Value	Description
0000	OK
0007	Service program error, purging
0018	Not in confirm state
0065	Invalid resource ID
0071	Verb not supported (conversation was allocated with a sync level of none)

0256	Session unbound by host unexpectedly
0257	Session shutdown by host orderly
0258	You are timed out by SOPR command
0784	ACTLU/DACTLU received
1809	Link failure
1810	ACTPU/DACTPU received
1811	$A (SOPR) ABORT AIF node
1812	$S ABORT AIF group

CSDEAL - Deallocate Verb

The CSDEAL verb deallocates the specified conversation from the transaction program.

FORMAT:

```
CALL "CSDEAL" USING SNA-WORK-AREA
                    TYPE
                    LOG|NO-LOG
                    LOG-DATA
```

DESCRIPTION:

The CSDEAL verb deallocates the specified conversation from the transaction program. The parameters issued with this verb identify the conversation to be deallocated and the type of deallocation to be performed.

When issuing the CSDEAL, the TYPE parameter allows you to specify whether the deallocation is to be completed as part of this verb or deferred until another verb is issued or a certain condition is met. The following options are available with the TYPE parameter.

- SYNC-LEVEL (S) - performs confirmation processing before deallocating the conversation:
  - If SYNC-LEVEL were none, CSDEAL flushes the local LU's send buffer and deallocates normally.
  - If SYNC-LEVEL were confirm, CSDEAL sends a confirmation requests to the remote LU and, if the return code is OK, deallocates the conversation normally. If the return code is UNSUCCESSFUL, CSDEAL returns the conversation to its previous state.
- FLUSH (F) - flushes the local LU's send buffer and deallocates the conversation normally.
- ABEND_PROG (P) - flushes the local LU's send buffer when the conversation is in send or defer state and deallocates normally.
- ABEND_SVC (V) - flushes the local LU's send buffer when the conversation is in send or defer state and deallocates the conversation abnormally.

- ABEND_TIM (T) - flushes the local LU's send buffer when the conversation is in send or defer state and deallocates the conversation abnormally.

If ABEND deallocation occurs when the conversation is in send state, logical record truncation can occur. When the conversation is in receive state, data purging can occur.

After the execution of the CSDEAL verb, the conversation enters reset state.

NOTE

AIF does not support a state that corresponds to the AIF deallocate state. If you receive a deallocate-confirm message after a CSCNFD verb, the conversation has been deallocated and its resources returned to the system. The conversation is then in reset state.

RETURN CODES:

The application should check the return code after each verb is issued to determine if the call has been completed error free. After the execution of the CSDEAL verb, the following combinations are possible:

- If SERV-REQ-COMP=Y and RETURN-B=0, the confirmation response has been sent.

- If the value of another field in RETURN-A is Y, the CSCNFD was not successful and RETURN-B contains the return code to indicate the reason for the error.

The SYNC-LEVEL at which the conversation was allocated determines the return codes that are possible for this call.

If you executed the CSDEAL with a type of SNCLVL and the conversation was allocated with synchronization level of NONE or a type of FLUSH, the following return codes are possible in RETURN-B.

Value	Description
0000	OK
0016	Not in send state
0024	Logical record not finished yet
0076	Invalid type specified

If you executed the CSDEAL with a type of ABEND, the following return codes are possible:

Value	Description
0000	OK (deallocation is complete)
0026	Improper state
0076	Invalid type specified

If you executed the CSDEAL with a type of SYNC-LEVEL and the conversation was allocated with synchronization level of CONFIRM, the following return codes are possible:

Value	Description
0000	OK
0007	Service program error, purging
0017	Not in send/defer state
0024	Logical record not finished yet
0071	Verb not supported (conversation was allocated with a sync level of none)
0076	Invalid type specified
0176	TPN not recognized
0192	PIP not allowed
0193	PIP not specified correctly
0194	Security not valid
0195	Conversation type mismatch
0208	Sync level not supported by program
0209	Reconnect level not supported by program
0210	TP not available--no retry
0211	TP not available--retry
0224	ACC not valid
0241	Remote deallocation--ABEND program
0242	Remote deallocation--ABEND service
0243	Remote deallocation--ABEND timer

CSFLSH - Flush Verb

The CSFLSH verb flushes the local LU's send buffer.

FORMAT:

```
CALL "CSFLSH" USING SNA-WORK-AREA
```

DESCRIPTION:

The CSFLSH verb flushes the local LU's send buffer. Any information that was in the buffer is sent to the remote LU. The CSFLSH verb is useful for transferring incomplete buffers of data to the remote LU, thus avoiding a delay in processing.

If you execute a CSFLSH when the conversation is in defer state following a CSPTOR, the conversation enters receive state. If you execute a CSFLSH when the conversation is in send state, the state of the conversation does not change.

RETURN CODES:

The application should check the return code after each verb is issued to determine if the call has been completed error free. After the execution of the CSFLSH verb, the following combinations are possible:

- If SERV-REQ-COMP=Y and RETURN-B=0, the LU's receive buffer has been flushed.
- If the value of another field in RETURN-A is Y, the CSFLSH was not successful and RETURN-B contains the return code to indicate the reason for the error.

The following values are possible for RETURN-B:

Code	Meaning
0000	OK
0017	Not in send/defer state
0065	Invalid resource ID
0256	Session unbound by host unexpectedly
0257	Session shutdown by host orderly
0258	You are timed out by SOPR command
0259	Resource failure, no retry--session abort due to unrecoverable protocol error

0784	ACTLU/DACTLU received
1809	Link failure
1810	ACTPU/DACTPU received
1811	$A (SOPR) ABORT AIF node
1812	$S ABORT AIF group

CSPONR - Post on Receipt Verb

The CSPONR verb causes the LU to post the conversation when there is information to receive.

FORMAT:

```
CALL "CSPONR" USING SNA-WORK-AREA
                    FILL
                    RECEIVE-BUFFER-SIZE
```

DESCRIPTION:

The CSPONR verb causes the LU to signal the conversation when there is information to receive. The information can be transmitted data, status information, or a request for confirmation. The CSPONR can be used with the CSWAIT verb or the CSRAW verb to allow the application to continue with other processing while waiting for data from the host.

The FILL parameter allows you to specify whether posting should occur when a logical record is received or when the receive buffer is full.

Executing the CSPONR verb does not cause the state of the conversation to change. In order to execute the CSPONR, you must be in receive state. If you are not in receive state, you must first issue the CSPTOR verb.

RETURN CODES:

The application should check the return code after each verb is issued to determine if the call has been completed error free. After the execution of the CSPONR verb, the following combinations are possible:

- If SERV-REQ-COMP=Y and RETURN-B=0, the CSPONR has been successfully issued.
- If the value of another field in RETURN-A is Y, the CSPONR was not successful and RETURN-B contains the return code to indicate the reason for the error.

If the the return code indicates OKAY and the output control word indicates that the conversation has been posted, then posting has occurred and the LU has information that the program can receive. The program has the option of issuing a CSRAW at this point or it can ignore this posting by issuing a CSWAIT and receive this data at a later time.

The following values are possible for RETURN-B:

Value	Description
0000	OK
0022	Not in receive state
0065	Invalid resource ID

OUTPUT CONTROL WORD

The CONVERSATION-POSTED field in the OUTPUT-CONTROL-WORD indicates whether the conversation has been posted. If this parameter has a value of Y, the conversation is posted and CSRAW can be used to receive data or information. If this parameter has a value of N, posting has not occurred for this conversation and CSWAIT can be used to wait for posting to occur.

CSPTOR - Prepare to Receive Verb

The CSPTOR verb changes the state of the conversation to receive state.

FORMAT:

```
CALL "CSPTOR" USING SNA-WORK-AREA
                    TYPE
                    LOCKS
```

DESCRIPTION:

The CSPTOR verb changes the state of the conversation from send to receive. The parameters issued with this verb identify the conversation whose state is being changed, the type of prepare-to-receive to be performed, and when control is to be returned to the local program after the receive.

The TYPE parameter allows you to specify whether to perform confirmation processing (SYNCLVL) before preparing to receive or to flush the send buffer (FLUSH).

The locks parameter allows you to specify whether the local program waits for a reply when a request for confirmation is executed following a CSPTOR. This parameter is relevent only if the conversation was allocated with a synchronization level of CONFIRM, and the CSPTOR is executed with a type of SYNCLVL.

After the CSPTOR is executed, the conversation enters receive state. If the CSPTOR is unsuccessful, the conversation remains in send state.

RETURN CODES:

The application should check the return code after each verb is issued to determine if the call has been completed error free. After the execution of the CSPTOR verb, the following combinations are possible:

- If SERV-REQ-COMP=Y and RETURN-B=0, the CSPTOR has been successfully issued.
- If the value of another field in RETURN-A is Y, the CSPTOR was not successful and RETURN-B contains the return code to indicate the reason for the error.

The following values are possible for RETURN-B:

Value	Description
0000	OK
0076	Invalid type specified

In addition, if you executed the CSPTOR with a type of SNCLVL and the conversation was allocated with synchronization level of CONFIRM, the following return codes are possible.

Value	Description
0004	Program error--purging
0007	Service program error, purging
0016	Not in send state
0024	Logical record not finished yet
0065	Invalid resource ID
0071	Verb not supported (conversation was allocated with a sync level of none)
0176	TPN not recognized
0192	PIP not allowed
0193	PIP not specified correctly
0194	Security not valid
0195	Conversation type mismatch
0208	Sync level not supported by program
0209	Reconnect level not supported by program
0210	TP not available--no retry
0211	TP not available--retry
0224	ACC not valid
0241	Remote deallocation--ABEND program
0242	Remote deallocation--ABEND service
0243	Remote deallocation--ABEND timer
0256	Session unbound by host unexpectedly
0257	Session shutdown by host orderly
0258	You are timed out by SOPR command
0259	Resource failure, no retry--session abort due to unrecoverable protocol error
0784	ACTLU/DACTLU received
1809	Link failure
1810	ACTPU/DACTPU received
1811	$A (SOPR) ABORT AIF node
1812	$S ABORT AIF group

CSRAW - Receive and Wait Verb

The CSRAW verb causes the LU to wait for data on the receive queue and receive it.

FORMAT:

```
CALL "CSRAW" USING  SNA-WORK-AREA
                    RECEIVE-BUFFER
                    RECEIVE-BUFFER-LENGTH
                    FILL
                    RECEIVED-DATA-LENGTH
```

DESCRIPTION:

The CSRAW verb causes the LU to wait for data to arrive at the specified conversation and receives it. The information can be data, status information, or a request for confirmation. If there is data in the receive queue when this verb is executed, the waiting time is eliminated. After CSRAW is executed, control is returned to the local program and the type of information received is indicated.

If the conversation is in send state when this verb is issued, the local LU flushes its send buffer and the conversation changes to receive state. A send indicator is sent to the remote LU, to notify the remote program that it can send data to the local program.

The receive buffer is made up of logical records. The first two bytes of the buffer indicate the length of the buffer. If you want to convert the data you receive, you must first break it down into the record length and the logical record. Do not convert the record length field.

The FILL parameter allows you to specify whether the program receives data in logical record format or buffers it.

If the conversation is in send state when this verb is issued, the local LU flushes its send buffer and the conversation changes to receive state. A send indicator is sent to the remote LU, to notify the remote program that it can send data to the local program.

The value of the WHAT-RECEIVED parameter determines the state of the conversation after the $SRAW is executed. If WHAT-RECEIVED indicates WAR-DATA, DATA-COMPLETE, DATA-INCOMPLETE, or LL-TRUNCATED, the conversation enters (or remains in) receive state. If WHAT-RECEIVED indicates WAR-SEND, the conversation enters send state. If WHAT-RECEIVED indicates CONFIRM, CONFIRM-SEND, or CONFIRM-DEALLOCATE, the conversation enters confirm state.

RETURN CODES:

The application should check the return code after each verb is issued to determine if the call has been completed error free. After the execution of the CSRAW verb, the following combinations are possible:

- If SERV-REQ-COMP=Y and RETURN-B=0, the conversation has received the data successfully.
- If the value of another field in RETURN-A is Y, the CSRAW was not successful and RETURN-B contains the return code to indicate the reason for the error.

The following values are possible for RETURN-B:

Value	Description
0000	OK
0002	Program error, not truncating
0003	Program error, truncating
0004	Program error--purging
0005	Service error, not truncating
0006	Service error, truncating
0007	Service error--purging
0020	Not in send/receive state
0024	Logical record not finished yet
0065	Invalid resource ID
0176	TPN not recognized
0192	PIP not allowed
0193	PIP not specified correctly
0194	Security not valid
0195	Conversation type mismatch
0208	Sync level not supported by program
0209	Reconnect level not supported by program
0210	TP not available--no retry
0211	TP not available--retry
0224	ACC not valid
0240	Deallocate normal
0241	Remote deallocation--ABEND program

0242	Remote deallocation--ABEND service
0243	Remote deallocation--ABEND timer
0256	Session unbound by host unexpectedly
0257	Session shutdown by host orderly
0258	You are timed out by SOPR command
0259	Resource failure, no retry--session abort due to unrecoverrable protocol error
0784	ACTLU/DACTLU received
1809	Link failure
1810	ACTPU/DACTPU received
1811	$A (SOPR) ABORT AIF node
1812	$S ABORT AIF group

RECEIVE-DATA-LENGTH

This field contains the actual length of the received data, when WHAT-RECEIVED is a DATA indicator. The RECEIVED-DATA-LENGTH includes the two-byte binary field that specifies the logical record length and the length of the record itself. The length can range from 2 to 32,767 characters.

OUTPUT-CONTROL-WORD

The REQ-TO-SEND-RCVD field in the OUTPUT-CONTROL-WORD indicates whether the remote program has issued a request to send notification, requesting the local program to enter receive state and placing itself in send state.

WHAT-RECEIVED

The WHAT-RECEIVED field defines what the transaction program has received, and should be examined when the return code is SERV-REQ-COMP. The following values are possible for WHAT-RECEIVED.

2 SEND INDICATOR RECEIVED--the remote program has entered receive state causing the local program to enter send state. The local program can now issue a CSSDAT.

4 CONFIRM REQUEST RECEIVED--the remote program has sent a confirmation request to the local program. The local program can respond by issuing a CSCNFD or another verb, such as a CSSERR.

5 CONFIRM DEALLOCATE--the remote program has issued a deallocate with type SNCLVL and a synchronization level of CONFIRM. The local program can respond by issuing a CSCNFD or another verb, such as a CSSERR.

6 CONFIRM SEND RECEIVED--the remote program has issued a prepare to receive with type SNCLVL and a synchronization level of CONFIRM. The local program can respond by issuing a CSCNFD or another verb, such as a CSSERR.

8 LL-TRUNCATED--The CSRAW was issued with the LL FILL parameter and the length field is received truncated. The program does not receive the length of the data.

9 DATA INCOMPLETE WHEN LENGTH=0--The $SRAW was issued with a LENGTH of zero and an incomplete logical record is being received by the program. No data is passed to the caller.

10 DATA AVAILABLE WHEN LENGTH=0--The $SRAW was issued with a LENGTH of zero and a complete logical record is being received by the program. No data is passed to the caller.

20 DATA--The CSRAW was issued with the buffer FILL parameter and data is being received by the program.

21 DATA-COMPLETE--The CSRAW was issued with the LL FILL parameter and a complete logical record, or the completion of a logical record, is being received by the program.

22 DATA-INCOMPLETE--The CSRAW was issued with the LL FILL parameter and an incomplete logical record is being received by the program. The program must issue one or more additional CSRAWs to receive the remainder of the logical record.

CSRTOS - Request to Send Verb

The CSRTOS verb indicates to the remote program that the local LU has data to send.

FORMAT:

```
CALL "CSRTOS" USING SNA-WORK-AREA
```

DESCRIPTION:

The CSRTOS verb indicates to the remote program that the local program is requesting to enter send state. The local LU has data to send. This data can include program data, status information, or confirmation data. When the local program receives a send indicator in response, the conversation changes to send state.

If a negative response is received, the conversation remains in receive state. If a positive response is received, the conversation enters send state.

RETURN CODES:

The application should check the return code after each verb is issued to determine if the call has been completed error free. After the execution of the CSRTOS verb, the following combinations are possible:

- If SERV-REQ-COMP=Y and RETURN-B=0, the request to send has received the data successfully.
- If the value of another field in RETURN-A is Y, the CSRTOS was not successful and RETURN-B contains the return code to indicate the reason for the error.

The following values are possible for RETURN-B:

Value	Description
0000	OK
0021	Not in receive/confirm state
0025	In confirm state (received CONFIRM SEND or CONFIRM DEALLOCATE on the preceeding CSRAW
0065	Invalid resource ID

CSSDAT - Send Data Verb

The CSSDAT verb sends data to the remote program.

FORMAT:

```
CALL "CSSDAT" USING SNA-WORK-AREA
                    SEND-BUFFER
                    SEND-BUFFER-LENGTH
```

DESCRIPTION:

The CSSDAT verb sends data to the remote program. This data can be data, status information, or confirmation. The data is formatted into logical records, which are buffered before being transmitted. A logical record includes the record being sent and the two-byte binary field specifying the length of the data being sent. A logical record, by definition, can range from 2 bytes, including only the LL field, to 7FFF bytes, including a two-byte LL field and 32765 bytes of data.

NOTE

If you are going to translate data, you must translate it before you move it to the logical record, in order not to translate the binary record length field.

Executing the CSSDAT does not change the state of the conversation.

RETURN CODES:

The application should check the return code after each verb is issued to determine if the call has been completed error free. After the execution of the CSSDAT verb, the following combinations are possible:

- If SERV-REQ-COMP=Y and RETURN-B=0, the send has been executed successfully.
- If the value of another field in RETURN-A is Y, the CSSDAT was not successful and RETURN-B contains the return code to indicate the reason for the error.

The following values are possible for RETURN-B:

Value	Description
0000	OK
0004	Program error--purging
0005	Service program error, purging
0016	Not in send state
0065	Invalid resource ID
0068	Data length error
0074	Invalid logical record length
0176	TPN not recognized
0192	PIP not allowed
0193	PIP not specified correctly
0194	Security not valid
0195	Conversation type mismatch
0208	Sync level not supported by program
0209	Reconnect level not supported by program
0210	TP not available--no retry
0211	TP not available--retry
0224	ACC not valid
0241	Remote deallocation--ABEND program
0242	Remote deallocation--ABEND service
0243	Remote deallocation--ABEND timer
0256	Session unbound by host unexpectedly
0257	Session shutdown by host orderly
0258	You are timed out by SOPR command
0259	Resource failure, no retry--session abort due to unrecoverable protocol error
0784	ACTLU/DACTLU received
1809	Link failure
1810	ACTPU/DACTPU received
1811	$A (SOPR) ABORT AIF node
1812	$S ABORT AIF group

OUTPUT CONTROL WORD

The REQ-TO-SEND-RCVD field in the OUTPUT-CONTROL-WORD indicates whether the remote program has issued a request to send notification, requesting the local program to enter receive state and placing itself in send state.

CSSERR - Send Error Verb

The CSSERR verb indicates to the remote program that an error has occurred.

FORMAT:

```
CALL "CSSERR" USING SNA-WORK-AREA
                    TYPE
                    LOG NO-LOG
                    LOG-DATA
                    LOG-DATA-SIZE
```

DESCRIPTION:

The CSSERR verb indicates to the remote program that the local program has detected an error. The parameters issued with this verb identify the conversation on which the error has occurred and the type of error which has been detected. The local LU is in send state and the remote LU in receive state. If the conversation was in send state when this verb was issued, the local LU's send buffer is flushed and the state does not change.

The TYPE parameter indicates whether you are sending a program error (ABSEND_PROG) or a service error (SVC_ERROR). These errors are application-dependent.

If the conversation is in receive or confirm state when the CSSERR is executed, the conversation enters send state.

RETURN CODES:

The application should check the return code after each verb is issued to determine if the call has been completed error free. After the execution of the CSSERR verb, the following combinations are possible:

- If SERV-REQ-COMP=Y and RETURN-B=0, the send has been executed successfully.
- If the value of another field in RETURN-A is Y, the CSSDAT was not successful and RETURN-B contains the return code to indicate the reason for the error.

The state of the conversation when you issue the CSSERR determines what return codes are possible. The following values are possible for RETURN-B, following any execution of the CSSERR verb:

Value	Description
0000	OK
0023	Not in send, receive, or confirm state
0065	Invalid resource ID
0256	Session unbound by host unexpectedly
0257	Session shutdown by host orderly
0258	You are timed out by SOPR command
0259	Resource failure, no retry--session abort due to unrecoverable protocol error
0784	ACTLU/DACTLU received
1809	Link failure
1810	ACTPU/DACTPU received
1811	$A (SOPR) ABORT AIF node
1812	$S ABORT AIF group

In addition, if the conversation is in send state when you execute the CSSERR, the following return codes are possible.

Value	Description
0004	Program error--purging
0007	Service program error, purging
0176	TPN not recognized
0192	PIP not allowed
0193	PIP not specified correctly
0194	Security not valid
0195	Conversation type mismatch
0208	Sync level not supported by program
0209	Reconnect level not supported by program
0210	TP not available--no retry
0211	TP not available--retry
0224	ACC not valid
0241	Remote deallocation--ABEND program
0242	Remote deallocation--ABEND service
0243	Remote deallocation--ABEND timer

If the conversation is in confirm state when you execute the CSSERR, the following return codes are possible.

Code	Meaning
0076	Invalid type specified

If the conversation is in receive state when you execute the CSSERR, the following return codes are possible.

Code	Meaning
0076	Invalid type specified
0176	TPN not recognized
0192	PIP not allowed
0193	PIP not specified correctly
0194	Security not valid
0095	Conversation type mismatch
0208	Sync level not supported by program
0209	Reconnect level not supported by program
0210	TP not available--no retry
0211	TP not available--retry
0224	ACC not valid
0240	Deallocate normal

OUTPUT CONTROL WORD

The REQ-TO-SEND-RCVD field in the OUTPUT-CONTROL-WORD indicates whether the remote program has issued a request to send notification, requesting the local program to enter receive state and placing itself in send state.

CSWAIT - Wait Verb

The CSWAIT verb waits for posting to occur on any of a list of conversations.

FORMAT:

```
CALL "CSWAIT" USING SNA-WORK-AREA1
                    SNA-WORK-AREA2
                          .
                          .
                    SNA-WORK-AREAN
                    POSTED-CONV-ID
```

DESCRIPTION:

The CSWAIT verb causes the local program to suspend processing and wait for posting to occur on any conversation from a list of conversations. This verb is issued after the CSPONR (Post on Receipt) verb to allow synchronous processing of multiple conversations. Following the CSWAIT verb, you must execute the CSRAW verb to access the data. If you have issued the CSPONR to allow the application to continue other program processing while waiting for data from the host, the CSWAIT brings you back to the conversation that has been posted.

Executing the CSWAIT verb does not change the state of the conversation.

RETURN CODES:

The application should check the return code after each verb is issued to determine if the call has been completed error free. After the execution of the CSWAIT verb, the following combinations are possible:

- If SERV-REQ-COMP=Y and RETURN-B=0, the CSWAIT has been executed successfully.
- If the value of another field in RETURN-A is Y, the CSWAIT was not successful and RETURN-B contains the return code to indicate the reason for the error.

The following values are possible for RETURN-B

Value	Description
0000	OK
0001	Unsuccessful
0022	Not in receive state
0065	Invalid resource ID (the verb parameter list contains an invalid resource identifier. $B4 contains the pointer to this ID)

OUTPUT-CONTROL-WORD

The CONVERSATION-POSTED parameter of the OUTPUT-CONTROL-WORD indicates whether or not a conversation has been posted. The address of the conversation that has been posted appears in POSTED-CONV-ID. If you have multiple conversations, then this parameter contains the VPB address of the conversation that has been posted.

CSACEB - ASCII-to-EBCDIC Conversion

The CSACEB verb call converts data from ASCII to EBCDIC.

FORMAT:

```
    CALL "CSACEB" USING SNA-WORK-AREA
                        CONVERT-FROM-FIELD
                        FROM-LEFT-MOST-POSITION
                        CONVERT-TO-FIELD
                        TO-LEFT-MOST-POSITION
                        CONVERSION-LENGTH
    .
```

DESCRIPTION:

The CSACEB verb converts data from ASCII to EBCDIC. The parameters used with this verb provide the buffers containing the data to be converted and the converted data.

The maximum length of data that can be converted is 32,767 bytes.

If you want to convert the data in place, specify the same dataname for the CONVERT-FROM FIELD and the CONVERT-TO-FIELD.

IMPORTANT!

Do not convert the two-byte binary length field of the logical record.

CSEBAC - EBCDIC-to-ASCII Conversion

The CSEBAC verb converts data from EBCDIC to ASCII.

FORMAT:

```
CALL "CSEBAC" USING SNA-WORK-AREA
                    CONVERT-FROM-FIELD
                    FROM-LEFT-MOST-POSITION
                    CONVERT-TO-FIELD
                    TO-LEFT-MOST-POSITION
                    CONVERSION-LENGTH
```

DESCRIPTION:

The CSEBAC verb converts data from EBCDIC to ASCII. The parameters used with this verb provide the buffers containing the data to be converted and the converted data.

The maximum length of data that can be converted is 32,767 bytes.

If you want to convert the data in place, specify the same dataname for the CONVERT-FROM FIELD and the CONVERT-TO-FIELD.

IMPORTANT!
Do not convert the two-byte binary length field of the logical record.

Section 6
RESTART

This section describes the procedures for restarting an LU Type 0 session that has been abnormally terminated. Topics include:

- Configuration Options
  - Preestablished Session Groups
  - Reserved LUs
- Normal Termination
- Abnormal Termination
- Restart Logic
- Confirmation
- Release Time
- Message Resynchronization
- Rules for restart.

CONFIGURATION OPTIONS

The application programmer has the option of allowing AIF to assign any available LU to a session or defining preestablished session groups during configuration with or without reserved LUs. A preestablished session group is a group of one or more permanent sessions that are preestablished for later use when AIF is brought up.

All sessions in the group are established with one host LU and are preestablished using one Session Type Descriptor (STD). Subsequent application calls for sessions to that host LU, which specify the appropriate STD, cause the AIF to assign an available LU from this group to the calling application.

Preestablished session groups and reserved LUs are specified during the configuration of AIF.

Preestablished Session Groups

An AIF node can be configured to contain more than one preestablished session groups. If high traffic to a particular host LU is anticipated, a number of permanent sessions can be established to reduce the overhead required to establish these sessions each time a $SINIT/CSINIT session call is executed.

When an application requests a session by executing a $SINIT or CSINIT session call, a session from a preestablished session group will be assigned if one is available. If a preestablished session is not available, AIF assigns an available LU to the application. The assigned LU then executes the procedure for establishing an LU-LU session on behalf of the caller.

When an application using a preestablished session executes a $STERM or CSTERM session call, AIF does not actually terminate the LU-LU session but makes this permanent session available for other $SINIT/CSINIT session call requests.

Reserved LUs

An LU can be reserved for special use by specifying RESERVED=Y in the LU entry of the configuration file. If an LU is reserved, an STD name must be provided. In order for an application to gain access to a reserved LU, the STD name specified with the $SINIT or CSINIT session call must be the STD name associated with this LU address in the configuration of the LU entry.

A reserved LU can also be preestablished. Preestablishing a reserved LU saves the time required to establish a session when you execute a $SINIT or CSINIT session call. Preestablishing the session for a reserved LU does not assign it to a group.

NORMAL TERMINATION

Normal termination can occur when the session is completed by the $STERM or CSTERM session call or when an SOPR command is executed. The SOPR commands, STOP, ALTER, and SHUTDOWN, initiate an orderly termination to the current session. These commands do not cause the session to be held for restart.

ABNORMAL TERMINATION

Abnormal termination can occur for any of the following reasons:

- LU is deactivated.
- Session is unbound unexpectedly.
- SDLC link failure (LU reactivated by the AIF node recovery).
- CICS/IMS transaction program ABEND.
- A DPS 6 PLUS or a DPS 6 application program issues $STERM or CSTERM abnormally.
- An operating system $S ABORT command aborts the application task group.

RESTART LOGIC

You can restart an abnormally terminated session by executing a $SINIT or CSINIT session call. The parameters that you provide in the $SINIT or CSINIT session call determine whether the call is being used to initiate a session or restart one.

When you initiate a session using the $SINIT or CSINIT session call, you should store the two-word session id (SCCB.SC_SID in assembly language programs; the SESSION-ID field in COBOL programs). In order to restart a session after abnormal termination, you have to provide this session id. You also have to provide the most recent send and receive sequence numbers and the last message. In assembly language programs, these are found in SCCB.SC_SQN and SCCB.SC_RSQ, respectively. In COBOL programs, these numbers are found in the MSG-RESYNC-SEND- SQN and MSG-RESYNC-RCV-SQN fields. These sequence numbers should be stored after each send and receive in order to have the most current numbers available in case of abnormal termination.

NOTE

> The session id and the send and receive sequence numbers are system supplied. If a session terminates abnormally, you do not have access to these values unless you have previously stored them.

RESTART INITIALIZATION REQUEST

After a session has been successfully initiated, using the $SINIT or CSINIT session call, the application has the option of notifying AIF that the session should be held for restart. The application makes this request by executing the $SSI or CSSI session call (send interrupt) with the interrupt type ENAPRS (Enable Application Restart) to enable restart in the event of abnormal termination.

This request is required to ensure that an application can be restarted. The session is not held without confirmation regardless of the configuration of the STD "Release on Abnormal Termination" parameter.

If you decide to negate this confirmation, execute the $SSI or CSSI session call with the interrupt type DSAPRS (Disable Application Restart).

The application must restart the abnormally terminated session within the time specified in the STDs "Release on Abnormal Termination" parameter. Once the specified release time has elapsed, it is no longer possible to restart a session. If the application attempts to restart an abnormally terminated session that is not being held for restart, a return code of RCRSRF (restart failure) is returned. The sense data field shows the exact reason for this failure.

RELEASE TIME

The configuration of the STD used at session initiation determines how long the abnormally terminated session is to be held for restart. The three possibilities are:

1. If the STD "Release on Abnormal Termination" parameter is configured IMMEDIATE, the session is not held at all.

2. If the STD "Release on Abnormal Termination" parameter is configured HOLD, the session is held indefinitely.

3. If the STD "Release on Abnormal Termination" parameter is configured N(n..), the session is held for the specified n.. number of minutes.

The "Release on Abnormal Termination" can be overridden by the use of SNA Operator (SOPR) Control commands.

MESSAGE RESYNCHRONIZATION IN ASSEMBLY LANGUAGE

If the $SINIT call successfully restarts a session, the application should examine the output control word in the SCCB (SC_OCT).

An assembly language program should check for the following possible values in SCCB.SC_OCT:

1. If the output control word indicates SCRSTS, then the host has sent the "ready to send" message and SC_SQN and SC_RSQ contain the new sequence numbers for the restarted session.

2. If the output control word indicates SCL6RX, then the last message being sent by the local program was lost, and the local application must retransmit the last whole message.

3. If the output control word indicates SCHORX, then the last message being sent by the host was lost, and the host must retransmit its last whole message, and the local application must execute a receive.

MESSAGE RESYNCHRONIZATION IN COBOL

If the CSINIT call successfully restarts a session, the application should examine the OUTPUT-CONTROL-WORD field.

A COBOL program should check for the following possible values in the OUTPUT-CONTROL-WORD:

1. If the SET-SEND-RECV-SEQ = "Y", then the host has sent the "ready to send" message and MSG-RESYNC-SEND-SQN and MSG-RESYNC-RCV-SQN contain the new sequence numbers for the restarted session.

2. If APPL-RESEND-REQUIRED = "Y", then the last message being sent by the DPS 6 or DPS 6 PLUS was lost, and the local application must retransmit the last whole message.

3. If HOST-RESEND-REQUIRED = "Y", then the last message being sent by the host was lost, and the host must retransmit its last whole message, and the local application must execute a receive.

RULES FOR RESTART

When you attempt to restart a session that has abnormally terminated, you must restart it from the original task in which it was executing. The only time you may attempt to restart a session from a task other than the original task is when the application task group has been aborted.

If a session has abnormally terminated due to task group termination, the application can restart the session from any other group using the $SINIT or CSINIT session call, specifying RESTART and the correct session ID. If you must restart a session that has been terminated in this manner, the following restrictions apply:

1. The application cannot have any other sessions active when attempting to restart.

2. The session that is restarted is given the option of restarting all of the aborted sessions of its session group.

Figure 6-1 demonstrates a task restarting its sessions.

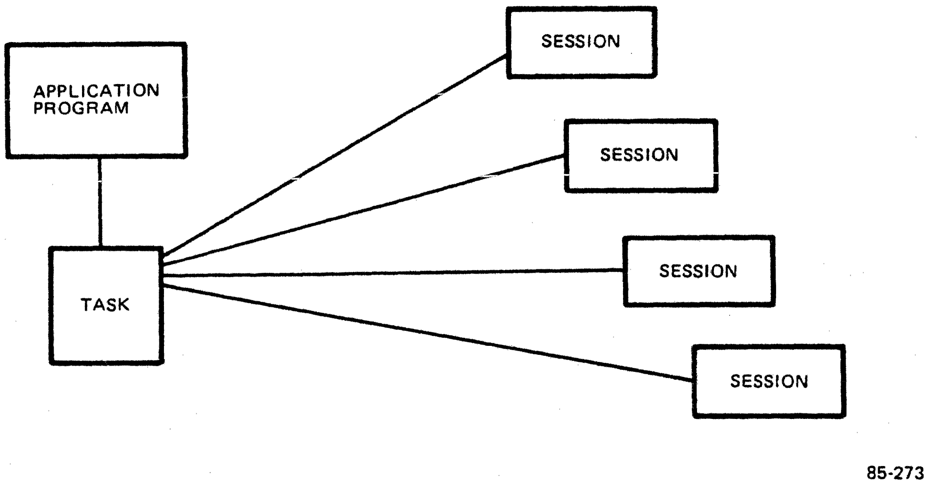


Figure 6-1. Session Restart

Each of these sessions has been running in the same task. When one or more of the sessions has been abnormally terminated, the abnormally terminated sessions must be restarted from the original task. The application program has the option of restarting each session individually.

Section 7
SUPPORT AND MAINTAINABILITY

This section discusses the role of SNA in supporting and maintaining the Application Interface Facility (AIF). Topics include the following:

- SNA Operator Services
- Maintenance Utilities
- Communications Network Management
  - AIF Alerts
  - AIF Maintenance Statistics.

SNA OPERATOR SERVICES

The control operator can use the SOPR facilities of the SNA Transport Facility for the following:

- Changing the state of an AIF LU (ALTER)
- Determining the status of an LU session (STATUS)
- Clearing the correspondence between a local LU, a local application program, and the host system (ABORT, SHUTDOWN, STOP).

These commands can be entered through the SOPR menu system or from a command line. The SOPR commands and their arguments are described in detail in the SNA6 Operator's Guide (GX10).

MAINTENANCE UTILITIES

The following SNA6 maintenance utilities are provided by the operating system (MOD 400 or HVS 6 PLUS) and aid in isolating problems:

- Trace/Software Probe Points

  AIF supports the Data Base Augmented Real-Time Tracing System (DARTS) utility that allows the user to take a snapshot of AIF activity. The utility records specific events in order to aid in problem determination.

  The Trace Information Capture Specification (TICS) file for AIF is located in:

  - >SID>AIF_L.TICS for LUs
  - >SID>AIF_P.TICS for PUs.

- SNAMAP

  AIF supports the SNAMAP utility that displays all existing SNA node structures, including AIF-specific information and journal statistics. SNAMAP commands and operating procedures are described in detail in the SNA6 Operator's Guide.

- The AIF_DUMP file

  If AIF detects an unrecoverable program error, it automatically executes SNAMAP and puts the dump file in >>CCD>AIF_DUMP. This dump file includes all existing node structures for the SNA products currently being executed. The following sequence occurs:

  1. The application program is informed that the session has been aborted. The return code RCPGER is returned.

  2. SNAMAP executes an emergency dump and puts it into the file >>CCD>AIF_DUMP. A message appears on the console to inform the operator. Processing continues after the dump is completed.

  3. The host is directed to terminate the session. The affected LU then becomes available for assignment to other callers.

  AIF limits itself to 10 dumps for the file >>CCD>AIF_DUMP. This file should be printed and forwarded to your local Honeywell representative for analysis, then deleted in order to conserve file space. It is recommended that the STARTUP.EC include directives to test for the existence of >>CCD>AIF_DUMP and print and delete its contents.

* Event Logging

  AIF makes an entry in the SNA event log when it detects system or transmission errors and when a session recovers from an error.

* SNEDIT

  The SNEDIT utility allows the user to display SNA journal files interactively . SNEDIT allows you to enter various commands to specify parameters that define journals you wish to display.

These utilities are described in detail in the _SNA6 Operator's Guide_.

COMMUNICATIONS NETWORK MANAGEMENT

The Communications Network Management (CNM) Facility allows LU Type 0 application programs to send alerts and statistics via AIF to the IBM host. Alerts are unsolicited messages that inform the host network operator of an error. AIF creates the transmission headers for these messages, but the application program must provide the message itself. The message must be formatted according to the IBM formats for alerts as detailed in Figure 7-1.

Maintenance statistics are solicited messages that supply application-dependent information, which the host requests. AIF creates the transmission headers for these messages, but the application program must provide the message itself. The format of the message is determined by the application. AIF alerts and maintenance statistics are discussed on the following pages.

AIF Alerts

As part of the program interface, AIF allows application programs to alert the host network operator that a major error has occurred by sending an SNA alert.

An SNA alert is used to inform the Network Communications Control Facility (NCCF) or Network Problem Determination Application (NPDA) that a problem exists on the DPS 6 or DPS 6 PLUS side. The AIF generates an alert on behalf of the DPS 6 or DPS 6 PLUS application program via the $SSI (send interrupt) session call where the interrupt type is specified ALERT.

The error message contents of the alert are provided by the application, which must create a buffer in the format which the IBM host can handle. AIF supplies bytes 0 through 7 and the remainder of the alert. Bytes 8 through n, which are supplied by the application program, must follow the format described in Figure 7-1.

BYTE(S)	DESCRIPTION
0-2	Network services header; x'410384'
3-7	CNM header
8-11	Node identification
	Bits 0-11: Block number
12-13	Reserved
14	X'40'
15	bits 0-3: Event type
	X'1' = Permanent error
	X'2' = Temporary error
	X'3' = Performance
15	bits 0-3: Event type (cont.)
	X'4' = Operational/Procedure
	X'5' = Customer Applications generated
	X'6' = End user generated
	X'7' = Reserved
	X'8' = Intensive mode recording
	bits 4-7: Major cause code
	X'1' = Hardware/microde - either
	X'2' = Software
	X'3' = Communications
	X'4' = Reserved
	X'5' = Environment
	X'6' = Removable media
	X'7' = Hardware/software - either
	X'8' = SNA logical
	X'9' = Operator: of sending message
	X'A' = Media/hardware - either
	X'B' = Explicitly hardware
	X'C' = Explicitly microde
	X'D' = SNA protocol
	X'E' = Link Level protocol
	X'F' = Undetermined
16	Minor Cause Code
	X'01' = Base processor
	X'02' = Service processor
	X'03' = Microde; non-customer programmable
	X'04' = Main storage
	X'05' = DASD device
	X'06' = Printer
	X'07' = Card reader/punch
	X'08' = Tape device
	X'09' = Keyboard
	X'0A' = Selector pen
	X'0B' = Magnetic stripe reader
	X'0C' = Display/printer
	X'0D' = Display device
	X'0E' = Remote Product
	X'0F' = Internal power supply
	X'10' = I/O attached controller

Figure 7-1. IBM Alert Format

BYTE(S)	DESCRIPTION
16	Minor Cause Code (cont.)
	X'11' = COMC scanner
	X'12' = COMC line adapter
	X'13' = reserved
	X'14' = Channel adapter
	X'15' = Loop adapter
	X'16' = Direct attach adapter
	X'17' = Adapter
	X'18' = Channel
	X'19' = Link
	X'1A' = Link (common carrier)
	X'1B' = Link (customer)
	X'1C' = Loop
	X'1D' = Loop (common carrier)
	X'1E' = Loop (customer)
	X'1F' = X.21 network
	X'21' = Local X.21 interface
	X'23' = Local modem
	X'24' = Remote modem
	X'25' = Local modem interface
	X'26' = Remote modem interface
	X'27' = Local probe
	X'28' = Remote probe
	X'29' = Local probe interface
	X'2A' = Remote Probe Interface
	X'2B' = Network connection
	X'2C' = IBM program SCP or major appl.
	X'2D' = IBM application program
	X'2E' = IBM communication access method
	X'2F' = Customer application program
	X'30' = IBM COMC program (T4 PU)
	X'31' = IBM control program
	X'32' = Remote/modem/interface product
	X'33' = Line/remote modem
	X'34' = SDLC data link control
	X'35' = BSC data link control
	X'36' = S/S data link control
	X'37' = Reserved
	X'38' = Power - external
	X'39' = Thermal
	X'3A' = Reserved
	X'3B' = Reserved
	X'3C' = Reserved
	X'3D' = Reserved
	X'3E' = Reserved
	X'3F' = Negative SNA Response
	X'40' = Gen or customize parameter
	X'41' = External facility
	X'42' = Component off line

Figure 7-1 (cont). IBM Alert Format

```
BYTE(S)    DESCRIPTION

16      Minor Cause Code (cont.)
           X'43' = Component busy
           X'44' = Controller or device
           X'45' = Local probe modem interface
           X'46' = Reserved
           X'47' = Card reader/punch or display/printer
           X'48' = Controller application program
           X'49' = Keyboard or display
           X'4A' = Storage Controller
           X'4B' = Channel or storage unit
           X'4C' = Reserved
           X'4D' = Controller
           X'4E' = Reserved
           X'4F' = Reserved
           X'50' = Reserved
           X'51' = Reserved
           X'52' = Maintenance device
           X'53' = Maintenance device interface
           X'67' = Sensor I/O unit
           X'68' = Magnetic stripe reader/encoder
           X'69' = Check reader
           X'6A' = Document feed
           X'6B' = Coin feed
           X'6C' = Envelope depository
           X'FF' = Undetermined
17      Reserved
18      User action code; used by NPDA, together with
        block number, to locate the alert/event
        description on the alert displays, the proper
        recommended action display, and the proper event
        detail display.
19      Reserved.
After these fields, one or more of the following
appended vectors may be included.

TEXT VECTOR
0       Vector length (binary)
1       X'00' = vector type
2-n     Text message; up to 100 bytes of customer
        defined data

DETAIL QUALIFIER VECTOR
0       Vector length (binary)
1       X'0D' = vector type
2-n     Detail qualifiers; information to be shown on
        the NPDA Event Detail screen

There may be multiple detail qualifier vectors in the
same RU.
```

Figure 7-1 (cont). IBM Alert Format

BYTE(S)	DESCRIPTION
NAME LIST VECTOR	
0	Vector length (binary)
1	X'0C' = vector type
2	X'02' = hierarchy name list in this vector is used with network names supplied by higher levels of CNM code.
3	Number of entries in the name list
4-n	Name list; identifies non NAU failing components. Each entry has the following format: Byte 0: Length Byte 1-x: Resource name Byte x+1 thru x+4: Resource type as follows: ADAP - Adapter ALA - Alternative line attachment ALS - Adjacent link stations BSC - Binary Synchronous link CHAN - Channel COMC - Communications controller CPU - Central processing unit CTF - Customer transaction facility CTRL - Controller DCA - Device cluster adapter DEV - Device DISK - Disk drive DSKT - Diskette drive IOCU - I/O control unit LCTL - Local controller LDEV - Local device LINK - Communications link LOOP - Loop NETW - Network PGM - Program PROG - Program SCF - System Control Facility SCU - Storage control unit STAT - Terminal station on loop TAPE - Magnetic Tape Drive TCU - Tape controller TTY - Teletype USER - Human or programmed operator WKST - Workstation nnnn - Machine type designator
NULL VECTOR	
0	X'00' = zero length; indicates end of vectors.

Figure 7-1 (cont). IBM Alerts Format

AIF Maintenance Statistics

As part of the program interface, AIF allows Session Type 0 programs to send maintenance statistics to the host network operator in response to a Request for Maintenance Statistic (REQMS) Type 4 made by the host.

AIF generates a reply called Records of Formatted Maintenance Statistics (RECFMS Type 4) on behalf of the local application program via the $SSI (send interrupt) session call where the interrupt type is specified STATIC.

The application must create a buffer in the format that the IBM host can handle. AIF supplies bytes 0 through 7 and the remainder of the RECFMS. Bytes 8 through n, which are supplied by the application program, are application-dependent and formatted in any way that the host program requests.

Appendix A
AIF ARCHITECTURE

The Application Interface Facility (AIF) is a general interprogram communications facility that applications on a DPS 6 or DPS 6 PLUS system can use to communicate with applications executing under the IBM host transaction processing systems Customer Information Control System (CICS) and Information Management System (IMS). The communicating programs on the DPS 6 or DPS 6 PLUS are referred to as application programs, while those on the host are referred to as transaction programs.

PROGRAM INTERFACE

AIF is a structured interface. That is, AIF specifies a number of formatted requests called session calls for LU Type 0 and verbs for LU Type 6.2 that the transaction program uses to request communication functions. These session calls equate to specific macrocalls within AIF.

AIF adheres to Honeywell's SNA6 interprogram communications architectural principals. The architecture of AIF is shown in Figure A-1 and is described below.

There are three logical subcomponents of AIF: the Physical Unit (PU) subcomponent, the LU subcomponent for each LU configured, and the monitor call handler subcomponent.

This appendix describes each of these subcomponents and the modules that make up the LU subcomponent.

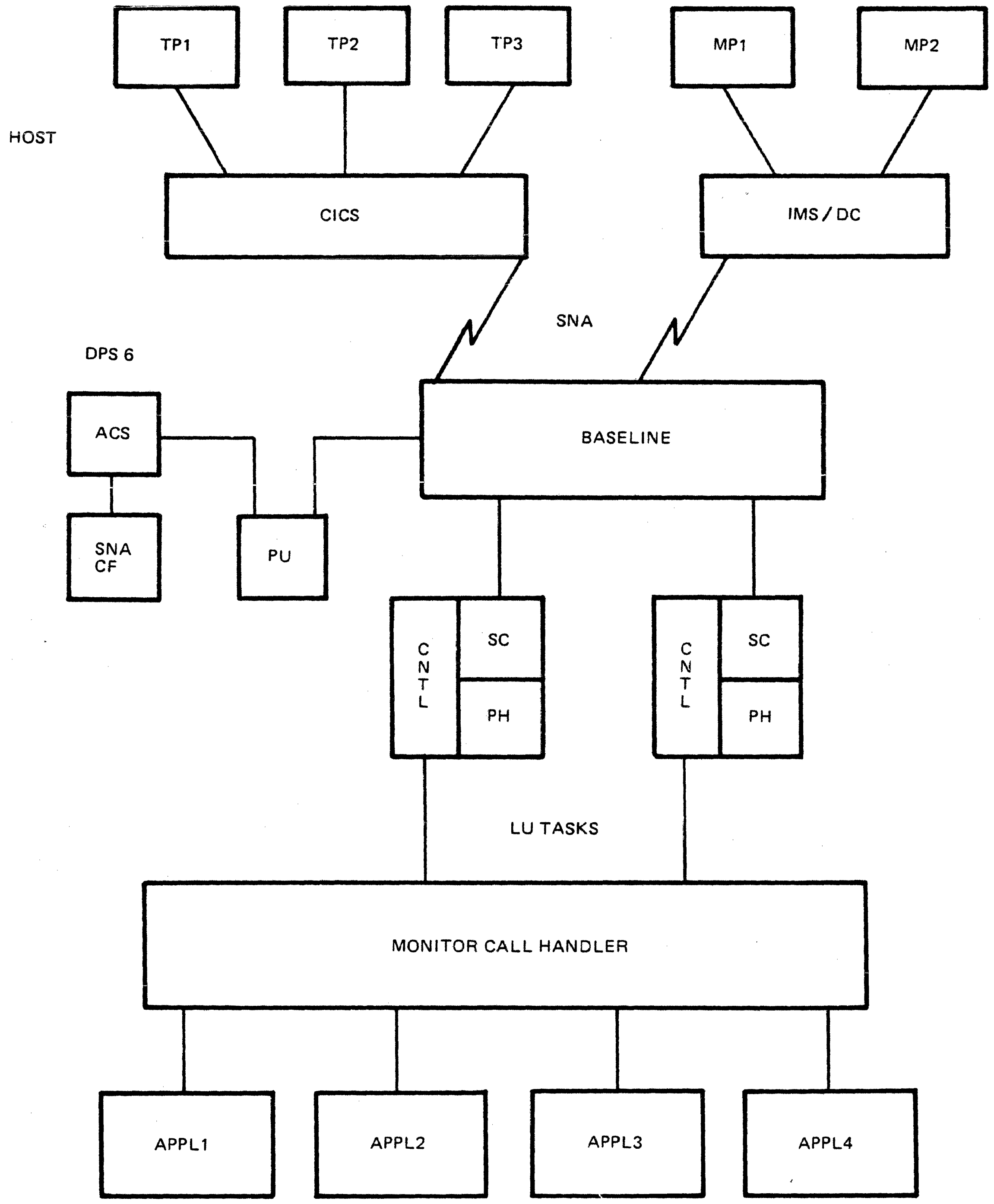


Figure A-1. Overall Architecture of AIF

Figure A-1 shows the relationship between the different components and modules which comprise an active AIF session. DPS 6 PLUS or DPS 6 transaction programs designate LU sessions through the Monitor Call Handler. The protocol handler module supervises the passing of calls from the control module to the session control module and on to the baseline.

On the host side, transaction programs communicate with the baseline through the IBM subsystems CICS and IMS.

PU SUBCOMPONENT

The PU subcomponent acts as the executive for the AIF program product. The only time the PU subcomponent is active during a session call is during the initialization or termination when it creates and terminates the LU tasks. The PU subcomponent interfaces to the Administrative Control System (ACS), SNA6 Operator Services (SOPR), and the monitor call handler. PU also sends the ALERT and maintenance statistics and Communications Network Management (CNM) commands.

LU SUBCOMPONENT

The LU subcomponent sends and receives data on behalf of the application program. It interfaces to the SNA6 network via the baseline. The LU subcomponent is comprised of three modules:

- The control module
- The protocol handler module
- The session control module.

Control Module

The LU control module has two main functions. It handles the external interfaces to the PU, the application program, or to transmission services; and it provides the mainloop processing for the LU, controlling the execution of the session control subroutines and the protocol handlers.

Protocol Handler Module

The protocol handler executes the session call subroutines on behalf of the session call executor. This module is responsible for consistent use of session calls.

Session Control Module

This module provides the subroutines that define the session call macros that actually interface to the SNA6 baseline. These subroutines are executed by protocol handlers. Subroutines are provided to receive, send, initiate a session, terminate a seession, send interrupt (control) information, and various other subroutines to support these functions.

MONITOR CALL HANDLER SUBCOMPONENT

The monitor call handler is the main interface between the application program and the LU and PU tasks. It manages the intertask group communication from the user task group, and in general acts as the interface to the AIF services.

When the monitor call handler receives a $SINIT or CSINIT session call from an application program for a session Type 0, or a $SALLO or CSALLO for Type 6.2, it sends an application service request (any session call the application passes to AIF). In response, AIF returns the SCCB or VPB to the PU subcomponent. The PU subcomponent looks for an available LU and assigns it to the application task.

For the rest of the session, the monitor call handler sends and receives application service requests/responses via the control module of the LU subcomponent. This relationship is shown in Figure A-2.

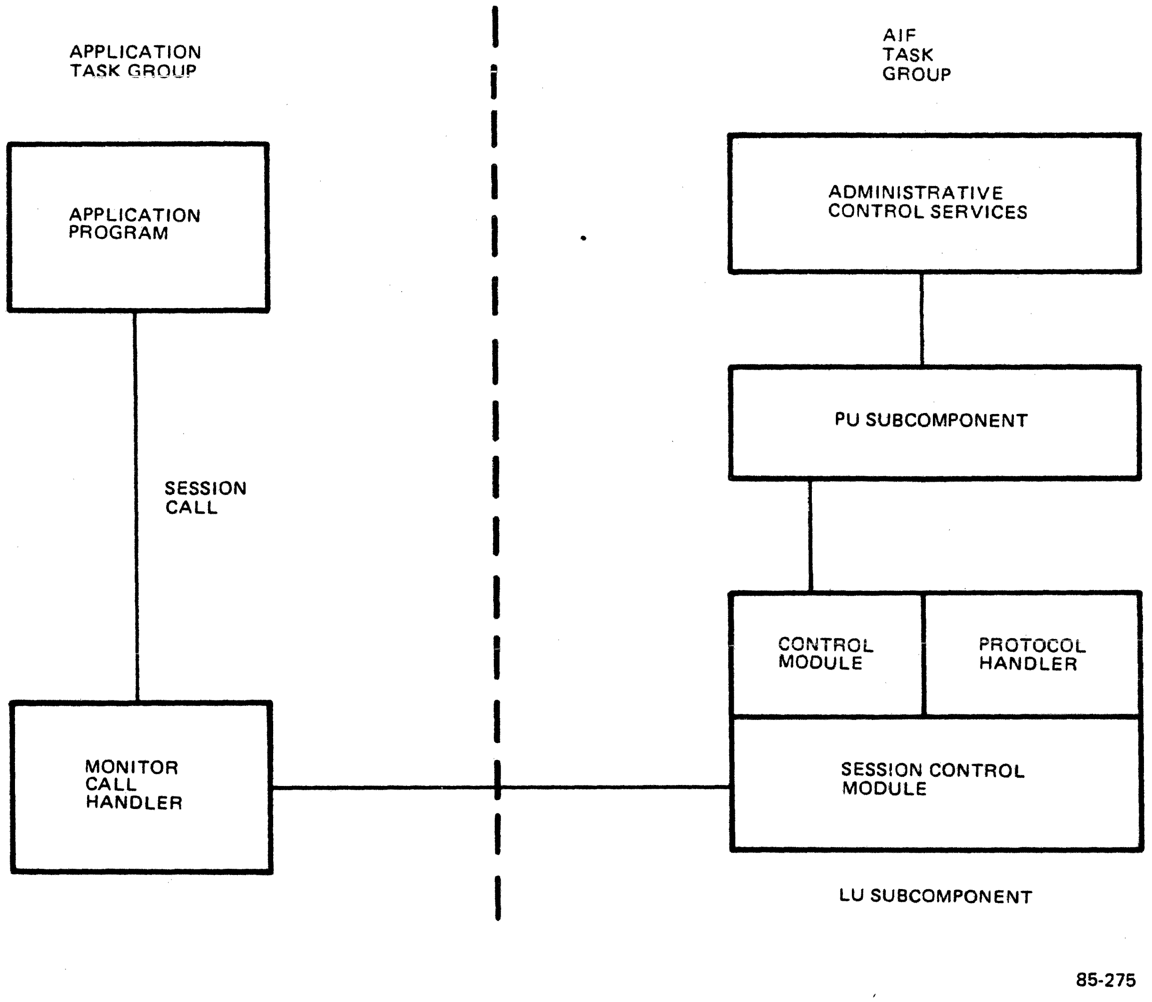


Figure A-2. Application Service Request/Reply Handling

When a session call is made, the monitor call handler checks the status of the call and issues an application service request to either the PU or the LU task, as appropriate. On the initial call for a session, the service request is directed to the PU. The monitor call handler issues service requests for subsequent calls to the LU.

When the PU or the LU completes a service request, it issues a service reply to the monitor call handler. The LU0 application must determine when an asynchronous call is completed, since the application program is not awakened until a $SWANY or CSWANY session call is executed following an asynchronous send or receive.

BASIC OPERATION

ACS is the lead task in the group running AIF. At start up, ACS creates and requests the PU task, which in turn, creates the LU tasks necessary to send and receive the data.

In order to execute a session call, the user executes a session call with parameters in the application program. These macrocalls resolve into the session call control block and a monitor call. When the monitor call is executed, a monitor call handler processes the call in the user task group.

The monitor call handler checks the state of the session, copies the user data block to a global memory block and issues an application service request to either the PU or LU task residing in the AIF task group.

Appendix B
SAMPLE ASSEMBLY LANGUAGE PROGRAMS

This appendix provides source listings of assembly language programs. These programs demonstrate the use of AIF LU Type 0 and LU Type 6.2 for both DPS 6- and Host-initiated sessions. All references to the DPS 6 system also include the DPS 6 PLUS system.

Figure B-1 is an AIF LU Type 0 sample program for a DPS 6-initiated session. Figure B-2 is an AIF LU Type 0 sample program for a Host-initiated session. Figure B-3 shows subroutines that can be linked by both program.

Figure B-4 is an AIF LU Type 6.2 sample program for a DPS 6 initiated session. Figure B-5 is an AIF LU Type 6.2 sample program for a Host initiated session. Figure B-6 are subroutines that can be linked by both programs.

```
   TITLE     TR_01,'10/22/85' SAMPLE #1  LU 0  DPS 6 INITIATED SESSION
*
*
    LIBM   '>>LDD>MACROS>MAC_USER'
          LIBM      OS_LIB
          LIBM      EXEC_LIB
*
          XDEF      AP_SCB,AP_SBF
          XLOC      GTMEM,PRTSCB
          $SSCCB
          $SAIRC
*
ADL0      TEXT      A'ADL0'        REMOTE APPLICATION NAME
BUF_SZ    EQU       141            SIZE OF RECEIVE/SEND BUFFER
RAN       EQU       2              RAN = REMOTE APPLICATION NAME LENGTH
AP_SCB    EQU       0              VPB
AP_SBF    EQU       AP_SCB+SC_SIZ  SEND BUFFER
AP_RBF    EQU       AP_SBF+BUF_SZ  RECEIVE BUFFER
AP_CW1    EQU       AP_RBF+BUF_SZ  APPLICATION CONTROL WORD 1
AP_STK    EQU       AP_CW1+1       STACK SPACE
AP_SIZ    EQU       AP_STK+100     APPLICATION WORK SIZE
*
*         DEFINITON FOR CONTROL WORD   (AP_CW1)
*
CW_RST    EQU       Z'8000'        RESTART ENABLED
*
TR_01    RESV      0
*
*
*         UPON ENTRY:
*
START     RESV      0
*
*
          LDV       $R6,0
          LDR       $R7,=AP_SIZ    AMOUNT OF MEMORY TO GET
          LDV       $R5,0          SET MEMORY TO ZEROS
          LNJ       $B5,GTMEM      GET MEMORY
          BEZ       $R1,SETREG     IF NO ERROR SET UP REGISTERS
*  E R R OR
SETREG    RESV      0
*
          LAB       $B7,$B4.AP_SIZ $B7 TO TOP OF STACK
          LAB       $B6,$B4        $B6 WORK SPACE POINTER
*         THIS TEST EXECUTES FOUR MACROS IN THIS SEQUENCE:
*
*
```

Figure B-1. Sample Assembly Language Program for LU Type 0 for DPS 6-Initiated Session

```
******************************************************************************
*                   I N I T I A L I Z E    S E S S I O N                     *
*                                                                            *
******************************************************************************
*
          $SINIT    ,'SMPLAIF','A06CICS2','BB',SYNC,NO_RESTART
INIT      RESV      0
          LB        =$R1,=(RCSCNL+RCABRT) WAS SESSION ABORT OR REQUEST CANCEL?
          BBT       TERM                      IF TRUE END
*                                                ELSE CONTINUE PROCESSING
/
*
******************************************************************************
*            S E N D      E N A B L E      R E S T A R T                     *
* TRY TO ENABLE RESTART FUNCTIONALITY FOR THIS SESSION. IF THIS SESSION IS   *
*  NOT RESTARTABLE, AN ERROR MESSAGE WILL APPEAR AND PROCESSING WILL CONTINUE.*
*  IF THIS HAPPENS, AND YOU WOULD LIKE TO ENABLE RESTART, YOU WILL HAVE TO   *
*  UPDATE YOUR AIF CONFIGURATION FILE (AIF NODE).                            *
******************************************************************************
*
          LBT       $B6.AP_CW1,=CW_RST  SET INDICATOR FOR RESTART
          $SSI      ,,,,=ENAPRS
*
          LB        =$R1,=RCABRT        WAS IT ABAND OR ?
          BBT       QUIT                 IF TRUE REPORT ERROR AND END
          LB        =$R1,=RCSCNL          ELSE WAS IT REQUEST CANCELED
          BBF       OUTMSG                 IF FALSE REQUEST DATA FROM TERM
          $USOUT    !NRSTRT,=NRSL,L          ELSE OUTPUT RESTART ERROR
          LBF       $B6.AP_CW1,=CW_RST          RESET RESTART INDICATOR
*                                               AND CONTINUE PROCESSING
/
*
******************************************************************************
*                                                                            *
*           O U T P U T    D A T A   T O   U S E R   O U T                   *
*                                                                            *
*  PLEASE ENTER DATA TO BE SENT TO THE HOST OR "END" TO END THE SESSION      *
******************************************************************************
*
OUTMSG    RESV      0
          $USOUT    !INPDAT,=INPL,L
*
/
```

Figure B-1 (cont). Sample Assembly Language Program for LU Type 0 for DPS 6-Initiated Session

```
******************************************************************************
*           G E T    D A T A   F R O M   U S E R   I N                       *
*                                                                            *
******************************************************************************
*
GETCHR    RESV      0
          LAB       $B4,$B6.AP_SBF        SET $B4 TO ADDR OF SEND BUFFER
          LAB       $B4,$B4.RAN           SET UP LENGTH FIELD FOR RECORD
          LDR       $R6,=(BUF_SZ-RAN)*2   GET RECORD SIZE IN BYTES
*
          $USIN
*
          LDR       $R5,='EN'             CHECK TO SEE IF END WAS ENTERED
          CMR       $R5,$B4               IS IT END ???
          BE        TERM0                  IF TRUE TERMINATE THE SESSION
          LDR       $R5,=(BUF_SZ-RAN)*2     ELSE GET MAXIUM LENGTH OF SEND
BUFFER
          CMR       $R5,=$R6                       WERE 0 CHAR ENTERED ??
          BNE       >CACL                          IF TRUE REPORT IT
          $USOUT    !NOCHAR,=NOL,L                    OUPUT ERROR MSG
          B         >GETCHR                           GET NEXT INPUT
CACL      SUB       $R5,=$R6                       ELSE CACULATE LL FIELD
*                                                TO THE RECORD LENGTH
          ADD       $R5,=RAN*2                   ADD RAN TO INPUT LENGTH
          STR       $R5,$B6.(AP_SCB+SC_DLG)      STORE IT IN SEND BUFFER LENGTH
*
          LAB       $B2,ADL0                     RAN TO BE STORED IN SEND BUFFER
          LDV       $R2,0                        OFFSET TO MOVE
          LAB       $B3,$B6.AP_SBF               ADDR OF SEND BUFFER
          LDV       $R3,0                        OFFSET TO MOVE TO
          LDV       $R6,RAN*2                    # OF CHARS TO MOVE
          MMM
*
/
******************************************************************************
*   T R A N S L A T E   D A T A   F R O M   A S C I I  T O   E B C D I C   *
* THIS IS AN EXAMPLE OF THE TRANSLATE CALL. IT WILL TRANSLATE FROM ASCII     *
* FROM $B2.$R2 TO $B4.$R4 EBCDIC ( THIS EXAMPLE WILL TRANSLATE IN PLACE ) .  *
* $R6 WILL CONTAIN THE # OF CHAR TO TRANSLATE                                *
******************************************************************************
*
          LAB       $B2,$B4.-RAN          $B2 = ADDR FROM BUFFER
          LDV       $R2,0                 $R2 = INDEX INTO FROM BUFFER
          LAB       $B4,$B4.-RAN          $B4 = ADDR TO BUFFER
          LDV       $R4,0                 $R4 = INDEX INTO TO BUFFER
          LDR       $R6,=$R5              $R6 = # OF CHARS TO TRANSLATE
          $SACEB
*
/
```

Figure B-1 (cont). Sample Assembly Language Program for LU Type 0 for DPS 6-Initiated Session

```
******************************************************************************
*         S E N D   D A T A   T O     T H E    H O S T                       *
* APPLICATION DATA IS STARTING AT POSITION #4 IN THE SEND BUFFER,POSITION    *
* 0-3 ARE RESERVED FOR THE TRANSACTION NAME ( THIS IS AN HOST APPLICATION    *
*  RESTRICTION), DUE TO THE FACT THAT THE HOST APPLICATION SENDS AN END      *
*  BRACKET, TERMINATING THE APPLICATION TO APPLICATION TRANSACTION ( N O T E :*
*  ( YOUR LU TO LU SESSION IS STILL ACTIVE )                                 *
******************************************************************************
*
SEND      RESV      0
          LAB       $B4,$B6.AP_SCB        $B4 == VPB POINTER
          LAB       $B2,$B6.AP_SBF        $B2 == ADDR OF SEND BUFFER
          $SSEND    ,=$B2,,L,SYNC,REPLY,MCMP
          LB        =$R1,=(RCABRT+RCSCNL) WAS IT ABORT OR REQUEST CANCEL ?
          BBT       TERM                   IF TRUE ERROR
*
/
******************************************************************************
*              R E C E I V E   D A T A     F R O M   H O S T                 *
*  DATA UNTIL THE BUFFER IS FULL, OR END OF DATA IS INDICATED.  BUFFER SIZE  *
*  IS SPECIFIED BYE DATA BUFFER LENGTH (SC_DLG)                              *
******************************************************************************
*
RECV      RESV      0
          LAB       $B2,$B6.AP_RBF+1      $B2 = ADDR OF RECEIVE BUFFER
          $SRECV    ,=$B2,=BUF_SZ,L,SYNC,MSG
          LB        =$R1,=(RCABRT+RCSCNL) WAS IT ABORT OR REQUEST CANCEL ?
          BBT       TERM                   IF TRUE ERROR
*
          LB        $B4.SC_OCT,=(SCRBOM+SCREOM) ELSE WAS BEGIN CHAIN/END CHAIN
          BBT       TRANS                        IF TRUE GO TRANSLATE
*
RECV1     RESV      0
          LB        $B4.SC_OCT,=SCRRQD          ELSE WAS A DEFINITE RESP REQ ?
          BBT       SRSP                         IF TRUE SEND +/- RESPONSE
*
RECV2     RESV      0
          LB        $B4.SC_OCT,=SCRWRP          ELSE WAS CHANGE DIR RECEIVED ?
          BBT       OUTMSG                       IF TRUE GO SEND DATA
*
          LB        $B4.SC_OCT,=SCRLST          ELSE WAS IT END BRACKET ?
          BBF       RECERR                       IF FALSE OUTPUT ERROR
          B         OUTMSG                        ELSE GET MORE DATA
*
RECERR    RESV      0
          $USOUT    !INVOCT,=IOL,L               ELSE OUTPUT ERROR
          B         OUTMSG                            CONTINUE PROCESSING
*
/
```

Figure B-1 (cont). Sample Assembly Language Program for LU Type 0 for DPS 6-Initiated Session

```
*****************************************************************************
*                                                                           *
*          COMPARE TO SEE IF SEND BUFFER EQUALS RECEIVE BUFFER              *
*                                                                           *
*****************************************************************************
*
TRANS     RESV      0
          LAB       $B2,$B6.(AP_SBF+RAN)  $B2 = ADDR OF SEND BUFFER +RAN OFFSET
          LAB       $B3,$B6.AP_RBF+1      $B3 = ADDR OF RECEIVE BUFFER
          LDR       $R3,$B4.SC_ADL        $R3 = # OF CHARS TO TRANSLATE
          ADV       $R3,-1                SUBTRACT 1 FOR BLZ
*
GETNXT    BLZ       $R3,TRANS1            IF BUFFER EQUAL DISPLAY THEM
*
          LDH       $R1,$B2.$R3           GET CHAR FROM SEND BUFFER
          CMH       $R1,$B3.$R3           COMPARE WITH CHAR FROM RECEIVE BUFFER
          ADV       $R3,-1                SUBTRACT 1 FROM INDEX
          BE        GETNXT                IF EQUAL GET NEXT CHAR
          LBT       $B4.SC_ICT,=SCSNEG     ELSE SET BIT FOR NEG RESPONSE
          LBF       $B4.SC_ICT,=SCSRSP          RESET POSSITIVE RESPONSE
          $USOUT    !BADDAT,=BADDL,L            REPORT ERROR
          B         NXTRCV
*
*****************************************************************************
*   T R A N S L A T E   D A T A   F R O M   E B C D I C   T O   A S C I I   *
* THIS IS AN EXAMPLE OF THE TRANSLATE CALL. IT WILL TRANSLATE FROM EBCDIC   *
* FROM $B2.$R2 TO $B4.$R4 ASCII ( THIS EXAMPLE WILL TRANSLATE IN PLACE ) .  *
* $R6 WILL CONTAIN THE # OF CHAR TO TRANSLATE                               *
*      SC_ADL = ACTUAL # OF CHARS RECEIVED FROM REMOTE TRANSACTION          *
*****************************************************************************
*
TRANS1    RESV      0
          LBT       $B4.SC_ICT,=SCSRSP  SET POSITIVE RESPONSE
          LBF       $B4.SC_ICT,=SCSNEG  RESET NEGITIVE RESPONSE
          LDR       $R6,$B4.SC_ADL      $R6 = # OF CHAR TO TRANSLATE
          LAB       $B2,$B6.AP_RBF+1    $B2 = ADDR FROM BUFFER
          LDV       $R2,0               $R2 = INDEX INTO FROM BUFFER
          LAB       $B4,$B2             $B4 = ADDR TO BUFFER
          LDV       $R4,0               $R4 = INDEX INTO TO BUFFER
          $SEBAC
*
          LDR       $R6,$B6.SC_ADL      GET # OF CHARS TO DISPLAY
          ADV       $R6,1               ADJUST FOR SLEW CHAR COUNT
          LAB       $B4,$B4.-1          ADJUST BUFFER ADDR FOR SLEW
          LDR       $R1,=' A'           SLEW CHAR TO OUTPUT
          STR       $R1,$B4             STORE IT IN THE BUFFER
          $USOUT    ,,R
NXTRCV    LAB       $B4,$B6.AP_SCB      $B4 = ADDR OF VPB
          B         RECV1               GET NEXT INPUT
*
/
```

Figure B-1 (cont). Sample Assembly Language Program for LU Type 0 for DPS 6-Initiated Session

```
SRSP      RESV      0
*
          $SSRSP    ,SYNC,,=0
          LB        =$R1,=(RCABRT+RCSCNL)  WAS IT ABORT OR CANCEL ?
          BBT       TERM                   IF TRUE ERROR
          B         RECV2                   ELSE CONTINUE INPUT
*
/
*******************************************************************************
*            T E R M I N A T E      T H E      S E S S I O N                  *
* TERMINATE THE SESSION NORMALLY, IF AN ERROR OCCURS, THEN AN ABNORMAL        *
* TERMINATE WILL OCCURE  (TO FREE UP THE LU )                                 *
*******************************************************************************
*
*
TERM      RESV      0
          LNJ       $B5,PRTSCB            DISPLAY VERB CALL AND ERROR
          LB        =$R1,=RCABRT          IS IT SESSION ABORTED   ?
          BBF       TERM0                 IF FALSE TERMINATE SESSION
          LB        $B6.AP_CW1,=CW_RST     ELSE SEE IF RESTART ENABLED
          BBF       QUIT                    IF NOT ENABLED QUIT
          LBF       $B6.AP_CW1,=CW_RST         RESET RESTART CONDITION
          $SINIT    ,,,,SYNC,RESTART
          B         INIT                       GO CHECK RETURN CODE
TERM0     RESV      0                      ELSE TERMINATE THE SESSION
          LAB       $B4,$B6.AP_SCB        $B4 = ADDR OF THE SCCB
          $STERM    ,NORM
          LB        =$R1,=(RCABRT+RCSCMP) WAS IT ABORT OR REQUEST COMPLETE?
          BBT       QUIT                  IF TRUE END PROGRAM
*                                          ELSE TERMINATE SESSION ABNORMALLY
          $STERM    ,ABNORM
          B         QUIT
*
/
*
```

Figure B-1 (cont). Sample Assembly Language Program for LU Type 0 for DPS 6-Initiated Session

```
QUIT      RESV     0
          LAB      $B4,$B6             SET $B4 = WORK SPACE
          $RMEM
          $TRMRQ
*
*
INPDAT    TEXT     'APLEASE ENTER DATA TO BE SENT TO HOST, OR TYPE "END" TO
END THE SESSION'
*
INPL      EQU      ($-INPDAT)*2
*
BADDAT    TEXT     'AFORMAT ERROR:RECEIVE BUFFER DOES NOT EQUAL SEND BUFFER'
*
BADDL     EQU      ($-BADDAT)*2
*
NOCHAR    TEXT     'ANO DATA HAS BEEN ENTERED, PLEASE ENTER DATA OR END TO
TERMINATE THE SESSION'
*
NOL       EQU      ($-NOCHAR)*2
*
INVOCT    TEXT     'AINVALID OUTPUT CONTROL WORD RECEIVED'
IOL       EQU      ($-INVOCT)*2
*
NRSTRT    TEXT     'ARESTART NOT POSSIBLE FOR THIS SESSION,BUT PROCESS
CONTINUES'
NRSL      EQU      ($-NRSTRT)*2
*
          END      TR_01,START                    END
```

Figure B-1 (cont). Sample Assembly Language Program for LU Type 0 for DPS 6-Initiated Session

```
    TITLE     TR_02,'10/22/85' SAMPLE #2  LU 0  HOST INITIATED SESSION
*
*
    LIBM   '>>LDD>MACROS>MAC_USER'
          LIBM      OS_LIB
          LIBM      EXEC_LIB
*
          XDEF      AP_SCB,AP_SBF
          XLOC      GTMEM,PRTSCB
          $SSCCB
          $SAIRC
*
ADLH      TEXT      A'ADLH'          REMOTE APPLICATION NAME
BUF_SZ    EQU       141              SIZE OF RECEIVE/SEND BUFFER
RAN       EQU       2                RAN = REMOTE APPLICATION NAME LENGTH
AP_SCB    EQU       0                VPB
AP_SBF    EQU       AP_SCB+SC_SIZ    SEND BUFFER
AP_RBF    EQU       AP_SBF+BUF_SZ    RECEIVE BUFFER
AP_STK    EQU       AP_RBF+BUF_SZ    STACK SPACE
AP_SIZ    EQU       AP_STK+100       APPLICATION WORK SIZE
*
*
TR_02    RESV      0
*
*
*         UPON ENTRY:
*
START     RESV      0
*
*
          LDV       $R6,0
          LDR       $R7,=AP_SIZ      AMOUNT OF MEMORY TO GET
          LDV       $R5,0            SET MEMORY TO ZEROS
          LNJ       $B5,GTMEM        GET MEMORY
          BEZ       $R1,SETREG       IF NO ERROR SET UP REGISTERS
*  E R R OR
SETREG    RESV      0
*
          LAB       $B7,$B4.AP_SIZ   $B7 TO TOP OF STACK
          LAB       $B6,$B4          $B6 WORK SPACE POINTER
*         THIS TEST EXECUTES FOUR MACROS IN THIS SEQUENCE:
*
*
```

Figure B-2. Sample Assembly Language Program for LU Type 0 for Host-Initiated Session

```
*****************************************************************************
*                    P O LL     F O R     S E S S I O N                     *
* CHECK TO SEE IF ANY LU ASSOCIATED WITH THE APPLICATION PROGRAM'S TASK GROUP *
*  HAS BEEN ATTACHED BY THE REMOTE PROGRAM, IF AN LU IS NOT PRESENT, THE     *
*  APPLICATION WILL CONTINUE TO POLL UNTIL AN LU IS PRESENT                  *
* P O L L ( WILL NOT ESTABLISH A HOST INITIATED SESSION )                    *
*****************************************************************************
*
POLL     RESV     0
         $SPOLL   ,'SMPLAIF','BB'
         LB       =$R1,=(RCSCNL+RCABRT) WAS SESSION ABORT OR REQUEST CANCEL?
         BBT      TERM                   IF TRUE END
         AND      $R1,=RCMASK             ESLE MASK OUT INDICATORS
         CMR      $R1,=RMNOAT                  NO LU TO ATTACH?
         BE       POLL                     IF TRUE POLL AGAIN
         CMR      $R1,=RMLUAT               ELSE IS THERE AN LU TO ATTACH?
         BNE      TERM                       IF FALSE ERROR
*                                             ELSE CONTINUE PROCESSING
/
*
*****************************************************************************
*               A C C E P T    A     H O S T     S E S S I ON              *
*                                                                           *
*****************************************************************************
*
         $SACPT   ,'SMPLAIF','BB'
*
         LB       =$R1,=(RCABRT+RCSCNL) WAS IT ABAND OR REQUEST CANCEL?
         BBT      QUIT                   IF TRUE REPORT ERROR AND END
*
/
*****************************************************************************
*               R E C E I V E     D A T A      F R O M    H O S T           *
*  WHEN A SESSION IS A HOST INITIATED SESSION THE DPS6 SIDE OF THE SESSION   *
*  WILL ALWAYS BE IN RECEIVE STATE. AT THIS POINT THE APPLICATION WILL       *
* RECEIVE A CHANGE DIRECTION INDICATOR, AND THE APPLICATION NAME (SC_ADL WILL *
* EQUAL 4 ).  THIS IS APPLICATION DEPENDENT, TO PUT THE DPS6 SIDE INTO       *
* SEND STATE ( DATA WILL NOT BE RECEIVED AT THIS TIME).                      *
*****************************************************************************
*
         LAB      $B2,$B6.AP_RBF        $B2 = ADDR OF RECEIVE BUFFER
         $SRECV   ,=$B2,=BUF_SZ,L,SYNC,MSG
         LB       =$R1,=(RCABRT+RCSCNL) WAS IT ABORT OR REQUEST CANCEL ?
         BBT      TERM                   IF TRUE ERROR
*
         LB       $B4.SC_OCT,=SCRWRP        ELSE WAS CHANGE DIR RECEIVED ?
         BBT      OUTMSG                     IF TRUE GO SEND DATA
*
         $USOUT   !INVOCT,=IOL,L                ELSE OUTPUT ERROR
         B        TERM                           TERMINATE THE SESSION
*
/
*
```

Figure B-2 (cont). Sample Assembly Language Program for LU Type 0 for Host-Initiated Session

```
******************************************************************************
*                                                                            *
*          O U T P U T   D A T A   T O   U S E R   O U T                     *
*                                                                            *
*  PLEASE ENTER DATA TO BE SENT TO THE HOST OR "END" TO END THE SESSION      *
******************************************************************************
*
OUTMSG    RESV      0
          $USOUT    !INPDAT,=INPL,L
*
/
******************************************************************************
*            G E T   D A T A   F R O M   U S E R   I N                       *
*                                                                            *
******************************************************************************
*
GETCHR    RESV      0
          LAB       $B4,$B6.AP_SBF           SET $B4 TO ADDR OF SEND BUFFER
          LAB       $B4,$B4.RAN              SET UP LENGTH FIELD FOR RECORD
          LDR       $R6,=(BUF_SZ-RAN)*2      GET RECORD SIZE IN BYTES
*
          $USIN
*
          LDR       $R5,='EN'            CHECK TO SEE IF END WAS ENTERED
          CMR       $R5,$B4              IS IT END ???
          BE        TERM0                 IF TRUE TERMINATE THE SESSION
          LDR       $R5,=(BUF_SZ-RAN)*2    ELSE GET MAX LENGTH OF SEND BUFFER
          CMR       $R5,=$R6                    WERE 0 CHAR ENTERED ??
          BNE       >CACL                       IF TRUE REPORT IT
          $USOUT    !NOCHAR,=NOL,L                 OUPUT ERROR MSG
          B         >GETCHR                        GET NEXT INPUT
CACL      SUB       $R5,=$R6                     ELSE CACULATE LL FIELD
*                                               TO THE RECORD LENGTH
          ADD       $R5,=RAN*2                  ADD RAN TO INPUT LENGTH
          STR       $R5,$B6.(AP_SCB+SC_DLG)     STORE IN THE SEND BUFFER LENGTH
*
          LAB       $B2,ADLH                    RAN TO BE STORED IN SEND BUFFER
          LDV       $R2,0                       OFFSET TO MOVE
          LAB       $B3,$B6.AP_SBF              ADDR OF SEND BUFFER
          LDV       $R3,0                       OFFSET TO MOVE TO
          LDV       $R6,RAN*2                    # OF CHARS TO MOVE
          MMM
*
/
```

Figure B-2 (cont). Sample Assembly Language Program for LU Type 0 for Host-Initiated Session

```
*******************************************************************************
*   T R A N S L A T E   D A T A   F R O M   A S C I I  T O   E B C D I C    *
* THIS IS AN EXAMPLE OF THE TRANSLATE CALL. IT WILL TRANSLATE FROM ASCII      *
* FROM $B2.$R2 TO $B4.$R4 EBCDIC ( THIS EXAMPLE WILL TRANSLATE IN PLACE ) .   *
* $R6 WILL CONTAIN THE # OF CHAR TO TRANSLATE                                 *
*******************************************************************************
*
          LAB       $B2,$B4.-RAN          $B2 = ADDR FROM BUFFER
          LDV       $R2,0                 $R2 = INDEX INTO FROM BUFFER
          LAB       $B4,$B2               $B4 = ADDR TO BUFFER
          LDV       $R4,0                 $R4 = INDEX INTO TO BUFFER
          LDR       $R6,=$R5              $R6 = # OF CHARS TO TRANSLATE
          $SACEB
*
/
*******************************************************************************
*         S E N D   D A T A   T O      T H E     H O S T                      *
* APPLICATION DATA IS STARTING AT POSITION #4 IN THE SEND BUFFER,POSITION     *
* 0-3 ARE RESERVED FOR THE TRANSACTION NAME ( THIS IS AN HOST APPLICATION     *
*  RESTRICTION).                                                              *
*******************************************************************************
*
SEND      RESV      0
          LAB       $B4,$B6.AP_SCB        $B4 == VPB POINTER
          LAB       $B2,$B6.AP_SBF        $B2 == ADDR OF SEND BUFFER
          $SSEND    ,=$B2,$B4.SC_DLG,L,SYNC,REPLY,MCMP
          LB        =$R1,=(RCABRT+RCSCNL) WAS IT ABORT OR REQUEST CANCEL ?
          BBT       TERM                   IF TRUE ERROR
*
/
*******************************************************************************
*              R E C E I V E    D A T A      F R O M    H O S T               *
*  DATA UNTIL THE BUFFER IS FULL, OR END OF DATA IS INDICATED.  BUFFER SIZE   *
*  IS SPECIFIED BYE DATA BUFFER LENGTH (SC_DLG)                               *
*******************************************************************************
*
RECV      RESV      0
          LAB       $B2,$B6.AP_RBF        $B2 = ADDR OF RECEIVE BUFFER
          $SRECV    ,=$B2,=BUF_SZ,L,SYNC,MSG
          LB        =$R1,=(RCABRT+RCSCNL) WAS IT ABORT OR REQUEST CANCEL ?
          BBT       TERM                   IF TRUE ERROR
*
          LB        $B4.SC_OCT,=(SCRBOM+SCREOM) ELSE WAS BEGIN CHAIN/END CHAIN
          BBT       TRANS                        IF TRUE GO TRANSLATE
*
RECV1     RESV      0
          LB        $B4.SC_OCT,=SCRWRP            ELSE WAS CHANGE DIR RECEIVED?
          BBT       OUTMSG                         IF TRUE GO SEND DATA
*
```

Figure B-2 (cont). Sample Assembly Language Program for LU Type 0 for Host-Initiated Session

```
          LB         $B4.SC_OCT,=SCRLST      ELSE WAS IT END BRACKET ?
          BBF        RECERR                   IF FALSE OUTPUT ERROR
          B          OUTMSG                    ELSE GET MORE DATA
*
RECERR    RESV       0
          $USOUT     !INVOCT,=IOL,L            ELSE OUTPUT ERROR
          B          OUTMSG                         CONTINUE PROCESSING
*
/
******************************************************************************
*                                                                            *
*          COMPARE TO SEE IF SEND BUFFER EQUALS RECEIVE BUFFER               *
*                                                                            *
******************************************************************************
*
TRANS     RESV       0
          LAB        $B2,$B6.AP_SBF          $B2 = ADDR OF SEND BUFFER
          LAB        $B3,$B6.AP_RBF          $B3 = ADDR OF RECEIVE BUFFER
          LDR        $R3,$B4.SC_ADL          $R3 = # OF CHARS TO TRANSLATE
          SUB        $R3,=1                  SUBTRACT 1 FOR BLZ
GETNXT    BLZ        $R3,TRANS1              IF BUFFER EQUAL DISPLAY THEM
*                        S
          LDH        $R1,$B2.$R3             GET CHAR FROM SEND BUFFER
          CMH        $R1,$B3.$R3             COMPARE WITH CHAR FROM RECEIVE BUFFER
          ADV        $R3,-1                  SUBTRACT 1 FROM INDEX
          BE         GETNXT                  IF EQUAL GET NEXT CHAR
          $USOUT     !BADDAT,=BADDL,L         ELSE REPORT ERROR
*         B          SNDERR                       SEND ERROR SIGNAL TO REMOTE
          B          OUTMSG
*
******************************************************************************
*    T R A N S L A T E   D A T A   F R O M   E B C D I C   T O   A S C I I   *
* THIS IS AN EXAMPLE OF THE TRANSLATE CALL. IT WILL TRANSLATE FROM EBCDIC    *
* FROM $B2.$R2 TO $B4.$R4 ASCII ( THIS EXAMPLE WILL TRANSLATE IN PLACE ) .   *
* $R6 WILL CONTAIN THE # OF CHAR TO TRANSLATE                               .*
*      SC_ADL = ACTUAL # OF CHARS RECEIVED FROM REMOTE TRANSACTION           *
******************************************************************************
*
TRANS1    RESV       0
          LDR        $R6,$B4.SC_ADL          $R6 = # OF CHAR TO TRANSLATE
          LAB        $B2,$B6.(AP_RBF+RAN)    $B2 = ADDR FROM BUFFER
          LDV        $R2,0                   $R2 = INDEX INTO FROM BUFFER
          LAB        $B4,$B2                 $B4 = ADDR TO BUFFER
          LDV        $R4,0                   $R4 = INDEX INTO TO BUFFER
          SUB        $R6,=RAN*2              SUBTRACT OUT LL FIELD
          $SEBAC
*
```

Figure B-2 (cont). Sample Assembly Language Program for LU Type 0 for Host-Initiated Session

```
          LDR        $R6,$B6.SC_ADL          GET # OF CHARS TO DISPLAY
          SUB        $R6,=RAN                SUBTRACT OUT RAN FIELD
          LAB        $B4,$B4.-(RAN-1)        ADJUST BUFFER ADDR FOR SLEW
          LDR        $R1,=' A'               SLEW CHAR TO OUTPUT
          STR        $R1,$B4                 STORE IT IN THE BUFFER
          $USOUT     ,,R
          LAB        $B4,$B6.AP_SCB          $B4 = ADDR OF VPB
          B          RECV1                   GET NEXT INPUT
*
/
******************************************************************************
*            T E R M I N A T E        T H E      S E S S I O N              *
* TERMINATE THE SESSION NORMALLY, IF AN ERROR OCCURS, THEN AN ABNORMAL       *
* TERMINATE WILL OCCURE  (TO FREE UP THE LU )                                *
******************************************************************************
*
*
TERM      RESV       0
          LNJ        $B5,PRTSCB              DISPLAY VERB CALL AND ERROR
          LB         =$R1,=RCABRT            IS IT SESSION ABORTED   ?
          BBT        QUIT                    IF TRUE END
TERM0     RESV       0                        ELSE TERMINATE THE SESSION
          LAB        $B4,$B6.AP_SCB          $B4 = ADDR OF THE SCCB
          $STERM     ,NORM
          LB         =$R1,=(RCABRT+RCSCMP)   WAS IT ABORT OR REQUEST COMPLETE ?
          BBT        QUIT                    IF TRUE END PROGRAM
*                                             ELSE TERMINATE THE SESSION
ABNORMALLY
          $STERM     ,ABNORM
          B          QUIT
*
/
*
QUIT      RESV       0
          LAB        $B4,$B6                 SET $B4 = WORK SPACE
          $RMEM
          $TRMRQ
*
*
INPDAT    TEXT       'APLEASE ENTER DATA TO BE SENT TO HOST, OR TYPE "END" TO
END THE SESSION '
*
INPL      EQU        ($-INPDAT)*2
*
BADDAT    TEXT       'AFORMAT ERROR:RECEIVE BUFFER DOES NOT EQUAL SEND BUFFER '
*
BADDL     EQU        ($-BADDAT)*2
*
NOCHAR    TEXT       'ANO DATA HAS BEEN ENTERED, PLEASE ENTER DATA OR END TO
TERMINATE THE SESSION '
*
NOL       EQU        ($-NOCHAR)*2
EBREC     TEXT       'AEND BRACKET RECEIVED,SESSION WILL BE TERMINATED '
EBL       EQU        ($-EBREC)*2
*
INVOCT    TEXT       'AINVALID OUTPUT CONTROL WORD RECEIVED '
IOL       EQU        ($-INVOCT)*2
*
          END        TR_02,START                            END
```

Figure B-2 (cont). Sample Assembly Language Program for LU Type 0 for Host-Initiated Session

```
   TITLE     TR_SUB,'85011511'  SPI TAP SUBROUTINS.
*
    LIBM   '>>LDD>MACROS>MAC_USER'
           LIBM      EXEC_LIB
           LIBM      OS_LIB
           $SSCCB
/
*
           XVAL      AP_SCB,AP_SBF
*
           XDEF      GTMEM
GTMEM      RESV      0
*
*
*THIS IS THE GET MEMORY SUBROUTINE.
*
*
*   THE SIZE OF THE BLOCK OF MEMORY AND ITS SPACE INITIALIZATION VALUE
*   ARE PROVIDED BY THE CALLER OF THIS SUBROUTINE.
*
*      $R6/$R7 -> THE SIZE OF THE BLOCK OF MEMORY TO BE OBTAINED
*      $R5 -> THE SPACE INTIALIZATION VALUE FOR THE MEMORY BLOCK
*
*
*   UPON EXIT FROM THIS SUBROUTINE, THE MEMORY BLOCK'S ADDRESS AND SIZE
*   OR IF THERE WAS A PROBLEM, THE ERROR CODE ARE RETURNED TO THE CALLER.
*
*      $B4 -> ADDRESS OF THE MEMORY BLOCK
*      $R7 -> SIZE OF THE MEMORY BLOCK
*      $R1 -> ERROR CODE
*
           $GMEM
*
*   CHECK FOR ERROR CODE RETURN FROM MACRO CALL
*
           BEZ       $R1,>+$C            IF NO ERROR ON THE GET MEM, CONTINUE
*
*   UNABLE TO GET THE BLOCK OF MEMORY, RETURN WITH THE ERROR CODE IN $R1
*
$A         RESV      0
*
*
           JMP       $B5                 RETURN TO THE CALLER
*
*   INITIALIZE THE BLOCK OBTAINED WITH THE PROVIDED VALUE ($R5)
*
CLRIT      RESV      0
$C         RESV      0
           LDR       $R2,=$R7            INIT. THE INDEX WITH THE SIZE OF TH
           LDB       $B2,=$B4            MUST USE B1, B2, OR B3 FOR B-REL.+INDEX
*
```

Figure B-3. Subroutines for LU Type 0 Assembly Language Programs

```
*   THE BLOCK INITIALIZATION LOOP
$D         RESV       0
           STR        $R5,$B2.-$R2   STORE THE PROPER VALUE IN THE NEXT BLOCK
           BGZ        $R2,>-$D       THE INDEX IS ALSO THE NUMBER OF LOCATION
*   END OF LOOP, BLOCK IS INTITIALIZED
           B          >-$A                RETURN TO THE CALLER WITH THE BLOCK
*
/
*
*          CONVERT 1 HEX WORD TO 4 ASCII BYTES
*          $R7 = WORD TO CONVERT
*          $R3 = OFFSET INTO MEMORY TO STORE CONVERTED VALUE
*          $B3 = BASE MEMORY ADDR OT STORE CONVERTED VALUE
*          $R2 = # OF CHARS TO STORE
*
ASCII      EQU        $
           LDV        $R2,3               # OF BYTES TO CONVERT -1
ASCI1      LDV        $R6,0
           DIV        $R7,=16
           ADV        $R6,=X'30'          ADD ASCII BIAS
           CMR        $R6,=X'003A'        IS IT A THROUGH F ?
           BL         >STRASC             IF NOT STORE IT
           ADV        $R6,7               ADD ALPHA OFFSET
STRASC     STH        $R6,$B3.-$R3        STORE STRING
           BDEC       $R2,>ASCI1
           JMP        $B5                 RETURN TO CALLER
/*
           XDEF       PRTSCB
PRTSCB     RESV       0
           STB        $B5,-$B7            SAVE RETURN ADDR
           LDR        $R2,$B4.SC_OPC      GET OP CODE OF VERB WITH ERROR
           LBF        =$R2,=Z'F000'       CHANGE HIGH ORDER NIBBLE TO 0
OPCFND     LAB        $B2,OPCTBL                $B2 = TABLE TO SEARCH
$A         MLV        $R2,TBLGTH                $R2 = OFFSET INTO TABLE
           LAB        $B3,$B6.AP_SBF            $B3 = ADDR OF TO BUFFER
           LDV        $R3,0                     $R3 = OFFSET INTO BUFFER
           LDV        $R6,TBLGTH                $R6 = # OF BYTES TO MOVE
           MMM
*
           LDR        $R7,$B4.SC_RCD        GET RETURN CODE
           LDV        $R3,12                OFFSET INTO BUFFER TO STORE RCD
           LNJ        $B5,ASCII             CONVERT AND STORE ASCII RETURN CODE
*
           $USOUT     =$B3,=TBLGTH+2
           LAB        $B4,$B6.AP_SCB            RESET $B4 TO SCB
           LDR        $R1,$B4.SC_RCD            RESET $R1 TO RETURN CODE
           LDB        $B5,+$B7                  RESTORE RETURN ADDR
           JMP        $B5                       RETURN TO CALLER
```

Figure B-3 (cont). Subroutines for LU Type 0 Assembly Language Programs

```
* OP CODE TABLE
OPCTBL     RESV      0
           TEXT      'A$SINIT    '
TBLGTH     EQU       ($-OPCTBL)*2
           TEXT      'A$STERM    '
           TEXT      'A$SSEND    '
           TEXT      'A$SRECV    '
           TEXT      'A$SSI      '
           TEXT      'A$SRI      '
           TEXT      'A$SCASR    '
           TEXT      'A$SWANY    '
           TEXT      'A$STEST    '
*
           END       TR_SUB
```

Figure B-3 (cont). Subroutines for LU Type 0 Assembly Language Programs

```
   TITLE     VR_01,'10/22/85' SAMPLE #1  LU 6.2 DPS 6 INITIATED SESSION
*
*
    LIBM   '>>LDD>MACROS>MAC_USER'
          LIBM      OS_LIB
          LIBM      EXEC_LIB
*
          XDEF      AP_VPB,AP_SBF
          XLOC      GTMEM,PRTVRB
          $SVPB
          $SAIVR
*
BUF_SZ    EQU       141                   SIZE OF RECEIVE/SEND BUFFER
LL        EQU       1                     LENGTH OF THE LL FIELD
*                                         LL =
AP_VPB    EQU       0                     VPB
AP_SBF    EQU       AP_VPB+VP_SIZ         SEND BUFFER
AP_RBF    EQU       AP_SBF+BUF_SZ         RECEIVE BUFFER
AP_STK    EQU       AP_RBF+BUF_SZ         STACK SPACE
AP_SIZ    EQU       AP_STK+100            APPLICATION WORK SIZE
*
*
VR_01    RESV      0
*
*
*         UPON ENTRY:
*
START     RESV      0
*
*
          LDV       $R6,0
          LDR       $R7,=AP_SIZ           AMOUNT OF MEMORY TO GET
          LDV       $R5,0                 SET MEMORY TO ZEROS
          LNJ       $B5,GTMEM             GET MEMORY
          BEZ       $R1,SETREG            IF NO ERROR SET UP REGISTERS
*  E R R OR
SETREG    RESV      0
*
          LAB       $B7,$B4.AP_SIZ        $B7 TO TOP OF STACK
          LAB       $B6,$B4               $B6 WORK SPACE POINTER
*         THIS TEST EXECUTES FOUR MACROS IN THIS SEQUENCE:
*
/
*
```

Figure B-4. Sample Assembly Language Program for LU Type 6.2 for DPS 6-Initiated Session

```
*****************************************************************************
*          A L L O C A T E     T H E     C O N V E R S A T I O N           *
*                                                                           *
*****************************************************************************
*
         $SALLO    ,'SMPLAIF','A06CICS2',=A'ADL6','AA',AVAIL,CONFIRM
*
         LB        =$R1,=(VRABND+VRSCNL) WAS IT ABAND OR REQUEST CANCEL ?
         BBT       QUIT                  IF TRUE REPORT ERROR AND END
/
*
*****************************************************************************
*          F L U S H       T H E     L U ' S   S E N D   B U F F E R       *
* FLUSH THE LOCAL LU'S SEND BUFFER,CAUSING THE ALLOCATE OF THE CONVERSATION   *
*  TO BE ESTABLISHED.  THIS COMMAND IS OPTIONAL, IF IT IS NOT USED THE        *
*  COMMAND WILL BE BUFFERED UNTIL THE PREPARE TO RECEIVE IS ISSUED IN THE APPL*
*****************************************************************************
*
         $SFLSH
         LB        =$R1,=(VRABND+VRSCNL) WAS IT ABAND OR REQUEST CANCEL ?
         BBT       QUIT                  IF TRUE REPORT ERROR AND END
*
/
*****************************************************************************
*                                                                           *
*          O U T P U T     D A T A   T O   U S E R   O U T                 *
*                                                                           *
*  PLEASE ENTER DATA TO BE SENT TO THE HOST OR "END" TO END THE CONVERSATION  *
*****************************************************************************
*
OUTMSG   RESV      0
         $USOUT    !INPDAT,=INPL,L
*
/
```

Figure B-4 (cont). Sample Assembly Language Program for LU Type 6.2 for DPS 6-Initiated Session

```
*******************************************************************************
*           G E T    D A T A   F R O M   U S E R   I N                       *
*                                                                             *
*******************************************************************************
*
GETCHR   RESV     0
         LAB      $B4,$B6.AP_SBF        SET $B4 TO ADDR OF SEND BUFFER
         LAB      $B4,$B4.LL            SET UP LENGTH FIELD FOR RECORD
         LDR      $R6,=(BUF_SZ-LL)*2    GET RECORD SIZE IN BYTES
*
         $USIN
*
         LDR      $R5,='EN'             CHECK TO SEE IF END WAS ENTERED
         CMR      $R5,$B4               IS IT END ???
         BE       DEALO                  IF TRUE DEALLOCATE THE CONVERSATION
         LDR      $R5,=(BUF_SZ-LL)*2      ELSE GET MAX LENGTH OF SEND BUFFER
         CMR      $R5,=$R6                     WERE 0 CHAR ENTERED ??
         BNE      >CACL                        IF TRUE REPORT IT
         $USOUT   !NOCHAR,=NOL,L                  OUPUT ERROR MSG
         B        >GETCHR                         GET NEXT INPUT
CACL     SUB      $R5,=$R6                      ELSE CACULATE LL FIELD
         ADD      $R5,=LL*2                    ADD THE LENGTH OF THE LL FIELD
*                                              TO THE RECORD LENGTH
         STR      $R5,$B4.-LL                  STORE IT IN THE SEND BUFFER
*
/
*******************************************************************************
*    T R A N S L A T E   D A T A   F R O M   A S C I I   T O   E B C D I C   *
* THIS IS AN EXAMPLE OF THE TRANSLATE CALL. IT WILL TRANSLATE FROM ASCII      *
* FROM $B2.$R2 TO $B4.$R4 EBCDIC ( THIS EXAMPLE WILL TRANSLATE IN PLACE ) .   *
* $R6 WILL CONTAIN THE # OF CHAR TO TRANSLATE                                 *
*******************************************************************************
*
         LAB      $B2,$B4               $B2 = ADDR FROM BUFFER
         LDV      $R2,0                 $R2 = INDEX INTO FROM BUFFER
*                                       $B4 = ADDR TO BUFFER
         LDV      $R4,0                 $R4 = INDEX INTO TO BUFFER
         LDR      $R6,=$R5              $R6 = # OF CHARS TO TRANSLATE
         SUB      $R6,=LL*2             SUBTRACT OUT LL FIELD
         $SACEB
*
/
```

Figure B-4 (cont). Sample Assembly Language Program for LU Type 6.2 for DPS 6-Initiated Session

```
*******************************************************************************
*        S E N D   D A T A   T O     T H E    H O S T                         *
*                                                                             *
*******************************************************************************
*
         LAB        $B4,$B6.AP_VPB          $B4 == VPB POINTER
         LAB        $B2,$B6.AP_SBF          $B2 == ADDR OF SEND BUFFER
         $SSDAT     ,=$B2,$B6.AP_SBF
         LB         =$R1,=(VRABND+VRSCNL)  WAS IT ABAND OR REQUEST CANCEL ?
         BBT        DEAL                     IF TRUE ERROR
*
/
*******************************************************************************
*             P R E P A R E     T O     R E C E I V E                         *
* THIS COMMAND WILL CHANGE THE  CONVERSATION STATE FROM SEND, TO RECEIVE, AND *
*  FLUSH (TRANSMIT) THE LOCAL LU'S SEND BUFFER.                               *
* THIS COMMAND IS OPTIONAL, AND THE SAME RESULT COULD OF BEEN OPTAINED BY      *
*  A $SRAW ( RECIEVE AND WAIT).  THE LOCK OPTION BEING USED (LONG) SPECIFIES  *
*  RETURN CONTROL TO THE LOCAL PROGRAM AFTER DATA AND AN ACKNOWLEDGEMENT IS   *
*  RECEIVED FROM THE REMOTE TRANSACTION.                                      *
*******************************************************************************
*
         $SPTOR     ,FLUSH,LONG
         LB         =$R1,=(VRABND+VRSCNL)  WAS IT ABAND OR REQUEST CANCEL ?
         BBT        DEAL                     IF TRUE ERROR
*
/
```

Figure B-4 (cont). Sample Assembly Language Program for LU Type 6.2 for DPS 6-Initiated Session

```
*******************************************************************************
*                   R E C E I V E     A N D     W A I T                       *
* THIS COMMAND WILL CHANGE YOUR CONVERSATION STATE TO RECEIVE , IF YOU ARE    *
*  NOT IN RECEIVE STATE, AND THEN FLUSH ITS SEND BUFFER.  THE LU THEN WAITS   *
*  FOR INFORMATION TO ARRIVE, OR RECEIVES THE DATA WITHOUT WAITING IF IT IS   *
*  CURRENTLY AVAILABLE.  THIS RECEIVE SPECIFIES BUFFER, WHICH WILL RECEIVE    *
*  DATA UNTIL THE BUFFER IS FULL, OR END OF DATA IS INDICATED.  BUFFER SIZE   *
*  IS SPECIFIED BYE DATA BUFFER LENGTH (VP_DLG)                               *
*******************************************************************************
*
RAW       RESV      0
          LAB       $B2,$B6.AP_RBF        $B2 = ADDR OF RECEIVE BUFFER
          $SRAW     ,=$B2,=BUF_SZ,BUFFER
          LB        =$R1,=(VRABND+VRSCNL) WAS IT ABAND OR REQUEST CANCEL ?
          BBT       DEAL                   IF TRUE ERROR
*
          LDR       $R2,$B4.VP_WAR          ELSE GET THE WAT RECEIVED FIELD
          CMR       $R2,=VBRDAT              IF DATA RECEIVED  TRANSLATE IT
          BE        TRANS
          CMR       $R2,=VBRCSN               ELSE WAS IT A CONFIRM WITH SEND ?
          BNE       >CHK1                      IF TRUE EXECUTE A COMFIRMED
          LNJ       $B5,CONFMD                    (SEND STATE )
          B         OUTMSG                    GO BACK TO TERMINAL FOR MORE DATA
CHK1      CMR       $R2,=VBRSND                 ELSE WAS IT A REQUEST TO SEND
          BE        OUTMSG                       IF TRUE OUTPUT PROMPT
          CMR       $R2,=VBRCDA                   ELSE WAS CONFIRM DEALOCATE?
          BNE       >CHK2                          IF TRUE EXECUTE CONFIRMED
          LNJ       $B5,CONFMD                        (RESET STATE)
          B         QUIT                              EXIT APPLICATION
CHK2      CMR       $R2,=VBRCNF                     ELSE WAS IT CONFIRM ?
          BNE       CHK3                             IF TRUE EXECUTE CONFIRMED
          LNJ       $B5,CONFMD                          (STATE DOES NOT CHANGE)
          B         RAW                                 GOTO NEXT RECEIVE
CHK3      $USOUT    !BADWHT,=WHTL,L                ELSE REPORT ERROR & CONTINUE
          B         SNDERR
*
/
*******************************************************************************
*                                                                             *
*          COMPARE TO SEE IF SEND BUFFER EQUALS RECEIVE BUFFER                *
*                                                                             *
*******************************************************************************
*
TRANS     RESV      0
          LAB       $B2,$B6.AP_SBF             $B2 = ADDR OF SEND BUFFER
          LAB       $B3,$B6.AP_RBF             $B3 = ADDR OF RECEIVE BUFFER
          LDR       $R3,$B4.VP_ADL             $R3 = # OF CHARS TO TRANSLATE
          SUB       $R3,=1                     SUBTRACT 1 FOR BLZ
GETNXT    BLZ       $R3,TRANS1                 IF BUFFER EQUAL DISPLAY THEM
*
```

Figure B-4 (cont). Sample Assembly Language Program for LU Type 6.2 for DPS 6-Initiated Session

```
          LDH       $R1,$B2.$R3            GET CHAR FROM SEND BUFFER
          CMH       $R1,$B3.$R3            COMPARE WITH CHAR FROM RECEIVE BUFFER
          ADV       $R3,-1                 SUBTRACT 1 FROM INDEX
          BE        GETNXT                 IF EQUAL GET NEXT CHAR
          $USOUT    !BADDAT,=BADDL,L        ELSE REPORT ERROR
          B         SNDERR                      SEND ERROR SIGNAL TO REMOTE
*
*******************************************************************************
*    T R A N S L A T E   D A T A   F R O M   E B C D I C   T O   A S C I I   *
* THIS IS AN EXAMPLE OF THE TRANSLATE CALL. IT WILL TRANSLATE FROM EBCDIC     *
* FROM $B2.$R2 TO $B4.$R4 ASCII ( THIS EXAMPLE WILL TRANSLATE IN PLACE ) .    *
* $R6 WILL CONTAIN THE # OF CHAR TO TRANSLATE                                 *
*       VP_ADL = ACTUAL # OF CHARS RECEIVED FROM REMOTE TRANSACTION           *
*******************************************************************************
*
TRANS1    RESV      0
          LDR       $R6,$B4.VP_ADL         $R6 = # OF CHAR TO TRANSLATE
          LAB       $B2,$B6.(AP_RBF+LL)    $B2 = ADDR FROM BUFFER
          LDV       $R2,0                  $R2 = INDEX INTO FROM BUFFER
          LAB       $B4,$B2                $B4 = ADDR TO BUFFER
          LDV       $R4,0                  $R4 = INDEX INTO TO BUFFER
          SUB       $R6,=LL*2              SUBTRACT OUT LL FIELD
          $SEBAC
*
          LDR       $R6,$B6.VP_ADL         GET # OF CHARS TO DISPLAY
          SUB       $R6,=LL                SUBTRACT OUT LL FIELD
          LAB       $B4,$B4.-LL            ADJUST BUFFER ADDR FOR SLEW
          LDR       $R1,=' A'              SLEW CHAR TO OUTPUT
          STR       $R1,$B4                STORE IT IN THE BUFFER
          $USOUT    ,,R
          LAB       $B4,$B6.AP_VPB         $B4 = ADDR OF VPB
          B         RAW                    GET NEXT INPUT
*
/
*******************************************************************************
*         D E A L L O C A T E     T H E     C O N V E R S A T I O N           *
* DEALLOCATE THE CONVERSATION NORMALLY AND FLUSH THE LOCAL LU'S SEND BUFFER   *
* IF THE DEALLOCATE NORMAL IS NOT HONORED, AN ABNORMAL DEALLOCATE TYPE =      *
* PROGRAM ERROR WILL EXECUTE, FORCING THE CONVERSATION TO DEALLOCATE          *
*******************************************************************************
*
*
DEAL      RESV      0
          LNJ       $B5,PRTVRB             DISPLAY VERB CALL AND ERROR
          LB        =$R1,=VRABND           IS IT CONVERSATION ABEND ?
          BBT       QUIT                   IF TRUE END
DEAL0     RESV      0                       ELSE DEALLOCATE THE CONVERSATION
          LAB       $B4,$B6.AP_VPB         $B4 = ADDR OF THE VPB
          $SDEAL    ,FLUSH,NO_LOG,,
          LB        =$R1,=(VRABND+VRSCMP)  WAS IT ABAND OR REQUEST COMPLETE ?
          BBT       QUIT                   IF TRUE END PROGRAM
*                                           ELSE DEALOCATE ABNORMALLY
*                                                (PROGRAM ERROR)
          $SDEAL    ,PROG_AB,NO_LOG,,
          B         QUIT
*
/
```

Figure B-4 (cont). Sample Assembly Language Program for LU Type 6.2 for DPS 6-Initiated Session

```
******************************************************************************
*                S E N D    E R R O R     TO     R E M O T E                 *
* SEND AN ERROR TO THE REMOTE. THE CONVERSATION STATE WILL CHANGE FROM       *
* RECEIVE TO SEND STATE, AND THE SEND BUFFER WILL NOT BE FLUSHED.            *
******************************************************************************
*
SNDERR    RESV      0
          LAB       $B4,$B6.AP_VPB        $B4 = ADDR OF VPB
          $SSERR    ,PROG,NO_LOG,
          LB        =$R1,=(VRABND+VRSCNL) WAS IT ABAND OR REQUEST CANCEL ?
          BBT       DEAL                   IF TRUE ERROR
*
          B         OUTMSG                  ELSE RECEIVE NEXT
*
/
******************************************************************************
*                                                                            *
*                S E N D   C O N F I R M E D  T O    R E M O T E             *
*                                                                            *
******************************************************************************
*
CONFMD    RESV      0
          $SCNFD
          LB        =$R1,=(VRABND+VRSCNL) WAS IT ABAND OR REQUEST CANCEL ?
          BBT       DEAL                   IF TRUE ERROR
          JMP       $B5                     ELSE RETURN TO CALLER
*
/
*
QUIT      RESV      0
          LAB       $B4,$B6               SET $B4 = WORK SPACE
          $RMEM
          $TRMRQ
*
*
INPDAT    TEXT      'APLEASE ENTER DATA TO BE SENT TO HOST, OR TYPE "END" TO
END THE CONVERSATION '
*
INPL      EQU       ($-INPDAT)*2
*
BADDAT    TEXT      'AFORMAT ERROR:RECEIVE BUFFER DOES NOT EQUAL SEND BUFFER '
*
BADDL     EQU       ($-BADDAT)*2
*
BADWHT    TEXT      'AUNEXPECTED WHAT RECEIVED FIELD'
*
WHTL      EQU       ($-BADWHT)*2
*
NOCHAR    TEXT      'ANO DATA HAS BEEN ENTERED, PLEASE ENTER DATA OR END TO
DEALOCATE THE CONVERSATION'
*
NOL       EQU       ($-NOCHAR)*2
          END       VR_01                               END
```

Figure B-4 (cont). Sample Assembly Language Program for LU Type 6.2 for DPS 6-Initiated Session

```
    TITLE     VR_02,'10/22/85' SAMPLE #2  LU 6.2  HOST INITIATED SESSION
*
*
          LIBM      '>>LDD>MACROS>MAC_USER'
          LIBM      OS_LIB
          LIBM      EXEC_LIB
*
          XLOC      GTMEM
          XDEF      AP_VPB,AP_SBF
          $SVPB
          $SAIVR
*
BUF_SZ    EQU       141                   SIZE OF RECEIVE/SEND BUFFER
LL        EQU       1                     LENGTH OF THE LL FIELD
*                                         LL =
AP_VPB    EQU       0                     VPB
AP_SBF    EQU       AP_VPB+VP_SIZ         SEND BUFFER
AP_RBF    EQU       AP_SBF+BUF_SZ         RECEIVE BUFFER
AP_STK    EQU       AP_RBF+BUF_SZ         STACK SPACE
AP_SIZ    EQU       AP_STK+100            APPLICATION WORK SIZE
*
*
VR_02    RESV      0
*
*
*         UPON ENTRY:
*
START     RESV      0
*
*
          LDV       $R6,0
          LDR       $R7,=AP_SIZ           AMOUNT OF MEMORY TO GET
          LDV       $R5,0                 SET MEMORY TO ZEROS
          LNJ       $B5,GTMEM             GET MEMORY
          BEZ       $R1,SETREG            IF NO ERROR SET UP REGISTERS
*  E R R OR
SETREG    RESV      0
*
          LAB       $B7,$B4.AP_SIZ        $B7 TO TOP OF STACK
          LAB       $B6,$B4               $B6 WORK SPACE POINTER
*         THIS TEST EXECUTES FOUR MACROS IN THIS SEQUENCE:
*
/
*
```

Figure B-5. Sample Assembly Language Program for LU Type 6.2 for Host-Initiated Session

```
*******************************************************************************
*          A T T A C H E D     T H E     C O N V E R S A T I O N             *
*                                                                             *
*******************************************************************************
*
         $SATCH    ,'SMPLAIF','AA',CONFIRM
*
         LB        =$R1,=VRABND   WAS IT ABAND
         BBT       QUIT              IF TRUE REPORT ERROR AND END
         LB        =$R1,=VRSCNL       ELSE WAS IT REQUEST CANCELLED ?
         BBF       RAW                 IF FALSE PREFORM A $SRAW
         AND       $R1,=VRMASK          ELSE MASK OUT RETURN CODE
         CMR       $R1,=VRSLNS               COMPARE IS MISMATCH SYNC LEVELS?
         BNE       QUIT                  IF NOT EQUAL QUIT
         $USOUT    !MISMSL,=MISL,L        ELSE ISSUE MISMATCH ERROR
         B         QUIT                        EXIT APPLICATION
*
/
*******************************************************************************
*                                                                             *
*           O U T P U T    D A T A   T O   U S E R   O U T                    *
*                                                                             *
*  PLEASE ENTER DATA TO BE SENT TO THE HOST OR "END" TO END THE CONVERSATION  *
*******************************************************************************
*
OUTMSG   RESV      0
         $USOUT    !INPDAT,=INPL,L
*
/
*******************************************************************************
*             G E T    D A T A   F R O M   U S E R   I N                      *
*                                                                             *
*******************************************************************************
*
GETCHR   RESV      0
         LAB       $B4,$B6.AP_SBF      SET $B4 TO ADDR OF SEND BUFFER
         LAB       $B4,$B4.LL          SET UP LENGTH FIELD FOR RECORD
         LDR       $R6,=(BUF_SZ-LL)*2  GET RECORD SIZE IN BYTES
*
         $USIN
*
         LDR       $R5,='EN'          CHECK TO SEE IF END WAS ENTERED
         CMR       $R5,$B4            IS IT END ???
         BE        DEAL                IF TRUE DEALLOCATE THE CONVERSATION
         LDR       $R5,=(BUF_SZ-LL)*2   ELSE GET MAXIUM LENGTH OF SEND BUFFER
         CMR       $R5,=$R6                 WERE 0 CHAR ENTERED ??
         BNE       >CACL                    IF TRUE REPORT IT
         $USOUT    !NOCHAR,=NOL,L              OUPUT ERROR MSG
         B         >GETCHR                     GET NEXT INPUT
CACL     SUB       $R5,=$R6                  ELSE CACULATE LL FIELD
         ADD       $R5,=LL*2                ADD THE LENGTH OF THE LL FIELD
*                                           TO THE RECORD LENGTH
         STR       $R5,$B4.-LL              STORE IT IN THE SEND BUFFER
```

Figure B-5 (cont). Sample Assembly Language Program for LU Type 6.2 for Host-Initiated Session

```
*
/
*******************************************************************************
*   T R A N S L A T E   D A T A   F R O M   A S C I I  T O   E B C D I C   *
* THIS IS AN EXAMPLE OF THE TRANSLATE CALL. IT WILL TRANSLATE FROM ASCII      *
* FROM $B2.$R2 TO $B4.$R4 EBCDIC ( THIS EXAMPLE WILL TRANSLATE IN PLACE ) .   *
* $R6 WILL CONTAIN THE # OF CHAR TO TRANSLATE                                 *
*******************************************************************************
*
         LAB       $B2,$B4                 $B2 = ADDR FROM BUFFER
         LDV       $R2,0                   $R2 = INDEX INTO FROM BUFFER
*                                          $B4 = ADDR TO BUFFER
         LDV       $R4,0                   $R4 = INDEX INTO TO BUFFER
         LDR       $R6,=$R5                $R6 = # OF CHARS TO TRANSLATE
         SUB       $R6,=LL*2               SUBTRACT OUT LL FIELD
         $SACEB
*
/
*******************************************************************************
*        S E N D   D A T A   T O      T H E    H O S T                        *
*                                                                             *
*******************************************************************************
*
         LAB       $B4,$B6                 $B4 == VPB POINTER
         LAB       $B2,$B6.AP_SBF          $B2 == ADDR OF SEND BUFFER
         $SSDAT    ,=$B2,$B6.AP_SBF
         LB        =$R1,=(VRABND+VRSCNL) WAS IT ABAND OR REQUEST CANCEL ?
         BBT       DEAL                     IF TRUE ERROR
*
/
*******************************************************************************
*             P R E P A R E    T O     R E C E I V E                          *
* THIS COMMAND WILL CHANGE THE  CONVERSATION STATE FROM SEND, TO RECEIVE, AND *
*  FLUSH (TRANSMIT) THE LOCAL LU'S SEND BUFFER.                               *
* THIS COMMAND IS OPTIONAL, AND THE SAME RESULT COULD OF BEEN OPTAINED BY     *
*  A $SRAW ( RECIEVE AND WAIT).  THE LOCK OPTION BEING USED (LONG) SPECIFIES  *
*  RETURN CONTROL TO THE LOCAL PROGRAM AFTER DATA AND AN ACKNOWLEDGEMENT IS   *
*  RECEIVED FROM THE REMOTE TRANSACTION.                                      *
*******************************************************************************
*
         $SPTOR    ,FLUSH,LONG
         LB        =$R1,=(VRABND+VRSCNL) WAS IT ABAND OR REQUEST CANCEL ?
         BBT       DEAL                     IF TRUE ERROR
*
/
```

Figure B-5 (cont). Sample Assembly Language Program for LU Type 6.2 for Host-Initiated Session

```
*******************************************************************************
*                  R E C E I V E    A N D    W A I T                          *
* THIS COMMAND WILL CHANGE YOUR CONVERSATION STATE TO RECEIVE , IF YOU ARE    *
*  NOT IN RECEIVE STATE, AND THEN FLUSH ITS SEND BUFFER.  THE LU THEN WAITS   *
*  FOR INFORMATION TO ARRIVE, OR RECEIVES THE DATA WITHOUT WAITING IF IT IS   *
*  CURRENTLY AVAILABLE.  THIS RECEIVE SPECIFIES BUFFER, WHICH WILL RECEIVE    *
*  DATA UNTIL THE BUFFER IS FULL, OR END OF DATA IS INDICATED.  BUFFER SIZE   *
*  IS SPECIFIED BYE DATA BUFFER LENGTH (VP_DLG)                               *
*******************************************************************************
*
RAW       RESV      0
          LAB       $B2,$B6.AP_RBF        $B2 = ADDR OF RECEIVE BUFFER
          $SRAW     ,=$B2,=BUF_SZ,BUFFER
          LB        =$R1,=(VRABND+VRSCNL) WAS IT ABAND OR REQUEST CANCEL ?
          BBT       DEAL                   IF TRUE ERROR
*
          LDR       $R2,$B4.VP_WAR          ELSE GET THE WAT RECEIVED FIELD
          CMR       $R2,=VBRDAT              IF DATA RECEIVED  TRANSLATE IT
          BE        TRANS
          CMR       $R2,=VBRCSN               ELSE WAS IT A CONFIRM WITH SEND ?
          BNE       >CHK1                      IF TRUE EXECUTE A COMFIRMED
          LNJ       $B5,CONFMD                    (SEND STATE )
          B         OUTMSG                    GO BACK TO TERMINAL FOR MORE DATA
CHK1      CMR       $R2,=VBRSND                 ELSE WAS IT A REQUEST TO SEND
          BE        OUTMSG                       IF TRUE OUTPUT PROMPT
          CMR       $R2,=VBRCDA                   ELSE WAS CONFIRM DEALOCATE?
          BNE       >CHK2                          IF TRUE EXECUTE CONFIRMED
          LNJ       $B5,CONFMD                        (RESET STATE)
          B         QUIT                              EXIT APPLICATION
CHK2      CMR       $R2,=VBRCNF                     ELSE WAS IT CONFIRM ?
          BNE       CHK3                             IF TRUE EXECUTE CONFIRMED
          LNJ       $B5,CONFMD                          (STATE DOES NOT CHANGE)
          B         RAW                                 GOTO NEXT RECEIVE
CHK3      $USOUT    !BADWHT,=WHTL,L                ELSE REPORT ERROR & CONTINUE
          B         SNDERR
*
/
*******************************************************************************
*                                                                             *
*          COMPARE TO SEE IF SEND BUFFER EQUALS RECEIVE BUFFER                *
*                                                                             *
*******************************************************************************
*
TRANS     RESV      0
          LAB       $B2,$B6.AP_SBF        $B2 = ADDR OF SEND BUFFER
          LAB       $B3,$B6.AP_RBF        $B3 = ADDR OF RECEIVE BUFFER
          LDR       $R3,$B4.VP_ADL        $R3 = # OF CHARS TO TRANSLATE
          SUB       $R3,=1                SUBTRACT 1 FOR BLZ
GETNXT    BLZ       $R3,TRANS1            IF BUFFER EQUAL DISPLAY THEM
*
```

Figure B-5 (cont). Sample Assembly Language Program for LU Type 6.2 for Host-Initiated Session

```
          LDH       $R1,$B2.$R3           GET CHAR FROM SEND BUFFER
          CMH       $R1,$B3.$R3           COMPARE WITH CHAR FROM RECEIVE BUFFER
          ADV       $R3,-1                SUBTRACT 1 FROM INDEX
          BE        GETNXT                IF EQUAL GET NEXT CHAR
          $USOUT    !BADDAT,=BADDL,L       ELSE REPORT ERROR
          B         SNDERR                     SEND ERROR SIGNAL TO REMOTE
*
********************************************************************************
*    T R A N S L A T E   D A T A   F R O M   E B C D I C   T O   A S C I I    *
* THIS IS AN EXAMPLE OF THE TRANSLATE CALL. IT WILL TRANSLATE FROM EBCDIC      *
* FROM $B2.$R2 TO $B4.$R4 ASCII ( THIS EXAMPLE WILL TRANSLATE IN PLACE ) .     *
* $R6 WILL CONTAIN THE # OF CHAR TO TRANSLATE                                  *
*       VP_ADL = ACTUAL # OF CHARS RECEIVED FROM REMOTE TRANSACTION            *
********************************************************************************
*
TRANS1    RESV      0
          LDR       $R6,$B4.VP_ADL        $R6 = # OF CHAR TO TRANSLATE
          LAB       $B2,$B6.(AP_RBF+LL)   $B2 = ADDR FROM BUFFER
          LDV       $R2,0                 $R2 = INDEX INTO FROM BUFFER
          LAB       $B4,$B2               $B4 = ADDR TO BUFFER
          LDV       $R4,0                 $R4 = INDEX INTO TO BUFFER
          SUB       $R6,=LL*2             SUBTRACT OUT LL FIELD
          $SEBAC
*
          LDR       $R6,$B6.VP_ADL        GET # OF CHARS TO DISPLAY
          SUB       $R6,=LL               SUBTRACT OUT LL FIELD
          LAB       $B4,$B4.-LL           ADJUST BUFFER ADDR FOR SLEW
          LDR       $R1,=' A'             SLEW CHAR TO OUTPUT
          STR       $R1,$B4               STORE IT IN THE BUFFER
          $USOUT    ,,R
          LAB       $B4,$B6.AP_VPB        $B4 = ADDR OF VPB
          B         RAW                   GET NEXT INPUT
*
/
********************************************************************************
*         D E A L L O C A T E     T H E     C O N V E R S A T I O N            *
* DEALLOCATE THE CONVERSATION NORMALLY AND FLUSH THE LOCAL LU'S SEND BUFFER    *
* IF THE DEALLOCATE NORMAL IS NOT HONORED, AN ABNORMAL DEALLOCATE TYPE =       *
* PROGRAM ERROR WILL EXECUTE, FORCING THE CONVERSATION TO DEALLOCATE           *
********************************************************************************
*
*
DEAL      RESV      0
          LAB       $B4,$B6.AP_VPB        $B4 = ADDR OF THE VPB
          $SDEAL    ,FLUSH,NO_LOG,,
          LB        =$R1,=(VRABND+VRSCMP) WAS IT ABAND OR REQUEST COMPLETE ?
          BBT       QUIT                  IF TRUE END PROGRAM
*                                    ELSE DEALOCATE ABNORMALLY (PROGRAM ERROR)
          $SDEAL    ,PROG_AB,NO_LOG,,
          B         QUIT
*
/
```

Figure B-5 (cont). Sample Assembly Language Program for LU Type 6.2 for Host-Initiated Session

```
*******************************************************************************
*                 S E N D    E R R O R    TO    R E M O T E                    *
* SEND AN ERROR TO THE REMOTE. THE CONVERSATION STATE WILL CHANGE FROM         *
* RECEIVE TO SEND STATE, AND THE SEND BUFFER WILL NOT BE FLUSHED.              *
*******************************************************************************
*
SNDERR    RESV      0
          LAB       $B4,$B6.AP_VPB          $B4 = ADDR OF VPB
          $SSERR    ,PROG,NO_LOG,
          LB        =$R1,=(VRABND+VRSCNL) WAS IT ABAND OR REQUEST CANCEL ?
          BBT       DEAL                    IF TRUE ERROR
*
          B         OUTMSG                   ELSE RECEIVE NEXT
*
/
*******************************************************************************
*                                                                              *
*                 S E N D    C O N F I R M E D   T O    R E M O T E            *
*                                                                              *
*******************************************************************************
*
CONFMD    RESV      0
          $SCNFD
          LB        =$R1,=(VRABND+VRSCNL) WAS IT ABAND OR REQUEST CANCEL ?
          BBT       DEAL                    IF TRUE ERROR
          JMP       $B5                      ELSE RETURN TO CALLER
*
/
*
QUIT      RESV      0
          LAB       $B4,$B6                 SET $B4 = WORK SPACE
          $RMEM
          $TRMRQ
*
*
INPDAT    TEXT      'APLEASE ENTER DATA TO BE SENT TO HOST, OR TYPE "END" TO
END THE CONVERSATION '
*
INPL      EQU       ($-INPDAT)*2
*
BADDAT    TEXT      'AFORMAT ERROR:RECEIVE BUFFER DOES NOT EQUAL SEND BUFFER '
*
BADDL     EQU       ($-BADDAT)*2
*
BADWHT    TEXT      'AUNEXPECTED WHAT RECEIVED FIELD'
*
WHTL      EQU       ($-BADWHT)*2
*
NOCHAR    TEXT      'ANO DATA HAS BEEN ENTERED, PLEASE ENTER DATA OR END TO
DEALOCATE THE CONVERSATION'
*
NOL       EQU       ($-NOCHAR)*2
*
MISMSL    TEXT      'ASYNC LEVEL MISMATCH, PLEASE CHANGE AND REASSEMBLE
APPLICATION, APPLICATION ABORTED'
MISL      EQU       ($-MISMSL)*2
*
          END       VR_02                                END
```

Figure B-5 (cont). Sample Assembly Language Program for LU Type 6.2 for Host-Initiated Session

```
    TITLE     VR_SUB,'85011511'  SPI TAP SUBROUTINES.
*
    LIBM  '>>LDD>MACROS>MAC_USER'
          LIBM      EXEC_LIB
          LIBM      OS_LIB
          $SVPB
/
*
          XVAL      AP_VPB,AP_SBF
*
          XDEF      GTMEM
GTMEM     RESV      0
*
*
*THIS IS THE GET MEMORY SUBROUTINE.
*
*
*   THE SIZE OF THE BLOCK OF MEMORY AND ITS SPACE INITIALIZATION VALUE
*   ARE PROVIDED BY THE CALLER OF THIS SUBROUTINE.
*
*       $R6/$R7 -> THE SIZE OF THE BLOCK OF MEMORY TO BE OBTAINED
*       $R5 -> THE SPACE INTIALIZATION VALUE FOR THE MEMORY BLOCK
*
*
*   UPON EXIT FROM THIS SUBROUTINE, THE MEMORY BLOCK'S ADDRESS AND SIZE
*   OR IF THERE WAS A PROBLEM, THE ERROR CODE ARE RETURNED TO THE CALLER.
*
*       $B4 -> ADDRESS OF THE MEMORY BLOCK
*       $R7 -> SIZE OF THE MEMORY BLOCK
*       $R1 -> ERROR CODE
*
          $GMEM
*
*   CHECK FOR ERROR CODE RETURN FROM MACRO CALL
*
          BEZ       $R1,>+$C     .      IF NO ERROR ON THE GET MEM, CONTINUE
*
*   UNABLE TO GET THE BLOCK OF MEMORY, RETURN WITH THE ERROR CODE IN $R1
*
$A        RESV      0
*
*
          JMP       $B5                 RETURN TO THE CALLER
*
*   INITIALIZE THE BLOCK OBTAINED WITH THE PROVIDED VALUE ($R5)
*
CLRIT     RESV      0
$C        RESV      0
          LDR       $R2,=$R7            INIT. THE INDEX WITH THE SIZE OF TH
          LDB       $B2,=$B4            MUST USE B1, B2, OR B3 FOR B-REL.+INDEX
*
```

Figure B-6. Subroutines for LU Type 6.2 Assembly Language Programs

```
*   THE BLOCK INITIALIZATION LOOP
$D        RESV      0
          STR       $R5,$B2.-$R2   STORE THE PROPER VALUE IN THE NEXT BLOCK
          BGZ       $R2,>-$D       THE INDEX IS ALSO THE NUMBER OF LOCATION
*   END OF LOOP, BLOCK IS INTITIALIZED
          B         >-$A                RETURN TO THE CALLER WITH THE BLOCK
*
/
*
*         CONVERT 1 HEX WORD TO 4 ASCII BYTES
*         $R7 = WORD TO CONVERT
*         $R3 = OFFSET INTO MEMORY TO STORE CONVERTED VALUE
*         $B3 = BASE MEMORY ADDR OT STORE CONVERTED VALUE
*         $R2 = # OF CHARS TO STORE
*
ASCII     EQU       $
          LDV       $R2,3               # OF BYTES TO CONVERT -1
ASCI1     LDV       $R6,0
          DIV       $R7,=16
          ADV       $R6,=X'30'          ADD ASCII BIAS
          CMR       $R6,=X'003A'        IS IT A THROUGH F ?
          BL        >STRASC             IF NOT STORE IT
          ADV       $R6,7               ADD ALPHA OFFSET
STRASC    STH       $R6,$B3.-$R3        STORE STRING
          BDEC      $R2,>ASCI1
          JMP       $B5                 RETURN TO CALLER
/*
          XDEF      PRTVRB
PRTVRB    RESV      0
          STB       $B5,-$B7            SAVE RETURN ADDR ON STACK
          LDR       $R2,$B4.VP_OPC      GET OP CODE OF VERB WITH ERROR
          LBF       =$R2,=Z'F000'       CHANGE HIGH ORDER NIBBLE TO 0
          CMV       $R2,=4              WAS IT A CONTROL TYPE VERB
          BNE       >OPCFND             IF FLASE SET TABLE
          LDR       $R2,$B4.VP_CTL       ELSE GET CONTROL TYPE
          LBF       =$R2,=Z'F000'             RESET HIGH ORDER NIBBLE
          LAB       $B2,CTLTBL                $B2 = TABLE TO SEARCH
          B         >+$A
OPCFND    LAB       $B2,OPCTBL                $B2 = TABLE TO SEARCH
$A        MLV       $R2,TBLGTH                $R2 = OFFSET INTO TABLE
          LAB       $B3,$B6.AP_SBF            $B3 = ADDR OF TO BUFFER
          LDV       $R3,0                     $R3 = OFFSET INTO BUFFER
          LDV       $R6,TBLGTH                $R6 = # OF BYTES TO MOVE
          MMM
*
          LDR       $R7,$B4.VP_RCD            GET RETURN CODE
          LDV       $R3,12                    OFFSET TO STORE ASCII RETURN CODE
          LNJ       $B5,ASCII                 TRANSLATE HEX TO ASCII
*
          LDR       $R7,$B4.VP_WAR        GET WHAT RECEIVED
          LDV       $R3,18                OFFSET TO STORE ASCII WHAT RECEIVED
          LNJ       $B5,ASCII             TRANSLATE HEX TO ASCII
*
```

Figure B-6 (cont). Subroutines for LU Type 6.2 Assembly Language Programs

```
          $USOUT    =$B3,=TBLGTH+2
          LAB       $B4,$B6.AP_VPB               RESET $B4 TO VPB
          LDR       $R1,$B4.VP_RCD               RESET $R1 TO RETURN CODE
          LDB       $B5,+$B7                     RESTORE RETURN ADDR
          JMP       $B5                          RETURN TO CALLER
* OP CODE TABLE
OPCTBL    RESV      0
          TEXT      'A$SALLO       '
TBLGTH    EQU       ($-OPCTBL)*2
          TEXT      'AINVLD        '
          TEXT      'A$SSDAT       '
CTLTBL    RESV      0
          TEXT      'A$SFLSH       '
          TEXT      'A$SCONF       '
          TEXT      'A$SCNFD       '
          TEXT      'A$SRTOS       '
          TEXT      'A$SSERR       '
          TEXT      'A$SPTOR       '
          TEXT      'A$SPONR       '
          TEXT      'A$SDEAL       '
*
          END       VR_SUB
```

Figure B-6 (cont). Subroutines for LU Type 6.2 Assembly Language Programs

Appendix C
SAMPLE COBOL PROGRAMS

This appendix provides compilation listings of COBOL programs. These programs demonstrate the use of AIF LU Type 0 and LU Type 6.2 for both DPS 6- and Host-initiated sessions. All references to the DPS 6 system also include the DPS 6 PLUS system.

Figure C-1 is an AIF LU Type 0 sample program for a DPS 6-initiated session. Figure C-2 is an AIF LU Type 0 sample program for a Host-initiated session.

Figure C-3 is an AIF LU Type 6.2 sample program for a DPS 6-initiated session. Figure C-4 is an AIF LU Type 6.2 sample program for a Host-initiated session.

```
PROGRAM-ID. LOS1C.

**************************************************************************
*     THIS IS A SAMPLE LU 0 PROGRAM WHICH WILL EXERCISE SOME OF THE      *
*     AIF LU0 COBOL CALLS.  THE PROGRAM WILL START A SESSION WITH THE    *
*     HOST TRANSACTION ADL0.  IT WILL READ DATA FROM THE TERMINAL,       *
*     CONVERT IT TO EBCDIC, AND SEND THE CONVERTED RECORD TO THE HOST    *
*     THEN RECEIVE THE RECORD BACK.  UPON RECEIVING THE DATA BACK, THE   *
*     PROGRAM WILL COMPARE THE DATA THAT WAS RECEIVED WITH THE DATA      *
*     SENT DIPLAYING A PROPER MESSAGE ON THE TERMINAL.  IT WILL          *
*     CONVERT THE RECEIVED DATA TO ASCII AND DISPLAY IT ON THE           *
*     TERMINAL.  IF THE TERMINAL INPUT DATA STARTS WITH: END; THE        *
*     PROGRAM WILL TERMINATE THE SESSION AND END, OTHERWISE, THE         *
*     PROGRAM WILL GO THROUGH THE SAME PROCESS WITH WHAT HAS BEEN        *
*     RECEIVED FROM THE TERMINAL.                                        *
**************************************************************************
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. LEVEL-6.
 OBJECT-COMPUTER. LEVEL-6.
*
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01   START-OF-WS PIC X(32)
                  VALUE "START OF WORKING STORAGE SECTION".
 01   AIF-PARAMETERS PIC X(21) VALUE "AIF PARAMETERS FOLLOW".
 77   SNA-WORK-AREA                   PIC X(200).
 77   AIF-NODE-NAME                   PIC X(8) VALUE "SMPLAIF".
 77   REMOTE-LU-NAME                  PIC X(8) VALUE "A06CICS2".
 77   STD-NAME                        PIC XX   VALUE "BB".
 77   SYNC-CALL                       PIC X    VALUE "S".
 77   ASYNC-CALL                      PIC X    VALUE "A".
 77   RESTART-SESSION                 PIC X    VALUE "R".
 77   NEW-SESSION                     PIC X    VALUE "N".
 77   SESSION-ID                      PIC X(4) VALUE SPACES.
 77   MSG-RESYNC-SEND-SQN             PIC 9(5) VALUE ZEROS.
 77   MSG-RESYNC-RECV-SQN             PIC 9(5) VALUE ZEROS.
 01   SEND-DATA-BUFFER                PIC X(84) VALUE SPACES.
 77   SEND-BUFFER-SIZE                PIC 9(5) VALUE ZEROS.
 77   DATA-BUFFER-ALIGNMENT           PIC X    VALUE "L".
 77   REPLY-REQUEST                   PIC X    VALUE "R".
 77   WHOLE-MSG-INDICATOR             PIC X    VALUE "Y".
 77   FMH-INDICATOR                   PIC X    VALUE "N".
 77   RQD-INDICATOR                   PIC X    VALUE "N".
 01   RECEIVE-DATA-BUFFER.
      05  RECEIVE-RECORD              PIC X(80) VALUE SPACES.
 77   RECEIVE-BUFFER-SIZE             PIC 9(5) VALUE 80.
 77   RECEIVED-DATA-LENGTH            PIC 9(5) VALUE 0.
 77   RECV-COMPLETE-MSG               PIC X    VALUE "Y".
 77   WORK-AREA-ID                    PIC X(4) VALUE SPACES.
 77   SEND-RESPONSE-TYPE              PIC X    VALUE "-".
 77   SEND-SENSE-DATA                 PIC X(8) VALUE ZEROS.
```

Figure C-1. Sample COBOL Program for LU Type 0 for DPS 6-Initiated Session

```
01  RETURN-CODE-VALUES.
    05  MAJOR-RETURN-CODES.
        10  SESSION-ABORT              PIC X     VALUE "N".
            88  SESSION-ABORTED        VALUE "Y".
        10  STOP-RECEIVED              PIC X     VALUE "N".
            88  SOPR-ISSUED-STOP       VALUE "Y".
        10  INTERRUPT-RECVD            PIC X     VALUE "N".
            88  INTERRUPT-RECEIVED     VALUE "Y".
        10  SERV-REQ-CANCELLED         PIC X     VALUE "N".
            88  CALL-WAS-CANCELLED     VALUE "Y".
        10  SERV-REQ-COMPLETE          PIC X     VALUE "N".
            88  CALL-WAS-COMPLETED     VALUE "Y".
        10  COBOL-ERROR                PIC X     VALUE "N".
            88  CALL-FORMAT-ERROR      VALUE "Y".
    05  MINOR-RETURN-CODE              PIC 9(4)  VALUE ZEROS.
77  INTERRUPT-TYPE                     PIC 99    VALUE ZEROS.
77  INTERRUPT-DATA-LENGTH              PIC 9(5)  VALUE ZEROS.
77  TERMINATE-TYPE                     PIC X     VALUE "N".
77  GET-ATTR-TYPE                      PIC 99    VALUE 01.
01  SOPR-STOP-TIME.
    05  DATE-OF-STOP.
        10  STOP-YEAR                  PIC 99.
        10  STOP-MONTH                 PIC 99.
        10  STOP-DAY                   PIC 99.
    05  TIME-OF-STOP.
        10  STOP-HOUR                  PIC 99.
        10  STOP-MINUTE                PIC 99.
        10  STOP-SECONDS               PIC 9(4).
77  RECEIVED-SENSE-DATA                PIC X(8)  VALUE ZEROS.
01  OUTPUT-CONTROL-WORD.
    05  REPLY-REQUEST-CD               PIC X.
        88  CHANGE-DIRECTION-RCVD      VALUE "Y".
    05  DEFINITE-RESPONSE-REQ          PIC X.
        88  DEFINITE-RESPONSE-RCVD     VALUE "Y".
    05  LAST-MSG-EB                    PIC X.
        88  MSG-WITH-EB-RECEIVED       VALUE "Y".
    05  FMH-IN-RCVD-DATA               PIC X.
        88  RCVD-DATA-HAS-FMH          VALUE "Y".
    05  BEGIN-MSG-RCVD-BC              PIC X.
        88  BEGIN-CHAIN-RCVD           VALUE "Y".
    05  END-MSG-RCVD-EC                PIC X.
        88  END-CHAIN-RCVD             VALUE "Y".
    05  SET-SEND-RECV-SEQ              PIC X.
        88  STSN-RECEIVED              VALUE "Y".
    05  APPL-RESEND-REQUIRED           PIC X.
        88  RESTART-LAST-MSG-INB       VALUE "Y".
    05  HOST-RESEND-REQUIRED           PIC X.
        88  RESTART-LAST-MSG-OUTB      VALUE "Y".
77  CONVERSION-LENGTH                  COMP-1.
77  TRANSLATE-FROM-POSITION            COMP-1    VALUE 1.
77  TRANSLATE-TO-POSITION              COMP-1    VALUE 1.
01  END-OF-AIF PIC X(21) VALUE "END OF AIF PARAMETERS".
```

Figure C-1 (cont). Sample COBOL Program for LU Type 0 for DPS 6-Initiated Session

```
 01  MISC-PROGRAM-VARIABLES  PIC X(26) VALUE
                             "OTHER WORKING STORAGE DATA".
 01  DATA-TO-HOST                      PIC X(84) VALUE HIGH-VALUES.
 01  DATA-TO-HOST-REDEF REDEFINES DATA-TO-HOST.
     05  HOST-TRANSACTION              PIC X(4).
     05  DPS6-DATA-RECORD.
         10  CHECK-INPUT-FIELD OCCURS 80 TIMES.
             15  DATA-FIELD-CHAR       PIC X.
 01  DATA-FROM-HOST.
     05  DATA-FIELD OCCURS 80 TIMES.
         10  DATA-FLD-CHAR      PIC X.
 01  DATA-FROM-TERMINAL.
     05  END-INDICATOR             PIC XXX.
         88  END-PROGRAM           VALUE "END".
     05  FILLER                    PIC X(77) VALUE SPACES.
 01  SWITCH-COUNT-VARIABLES.
     05  INDX1                     COMP-1    VALUE 1.
     05  INDX2                     COMP-1    VALUE 0.
     05  NUMBER-CHARS              PIC 9(4)  VALUE ZEROS.
     05  CALC-LENGTH               COMP-1    VALUE ZEROS.
     05  TEMP-LENGTH               PIC 9(5)  VALUE ZEROS.
     05  ERROR-IN-CALL-SW          PIC 9     VALUE 0.
         88  OK-TO-CONTINUE        VALUE 0.
     05  RECORD-CHECK-SW           PIC 9     VALUE 0.
         88  RECORD-CHECKED        VALUE 1.
     05  NO-INPUT-SW               PIC 9     VALUE 0.
         88  NO-INPUT-DATA         VALUE 1.
     05  COMPARE-REC-SW            PIC 9     VALUE 0.
         88  COMPARE-OK            VALUE 0.
     05  NO-MORE-SW                PIC 9     VALUE 0.
         88  NO-MORE-TO-CHECK      VALUE 1.
     05  INTERRUPT-SW              PIC 9     VALUE 0.
         88  INTERRUPT-CALL-NEXT VALUE 1.
 01  ENTER-MESSAGE                 PIC X(80) VALUE
     "PLEASE ENTER DATA TO TRANSMIT TO HOST OR END TO QUIT".
 01  END-OF-WORK-STOR  PIC X(19)  VALUE "END WORKING STORAGE".
 PROCEDURE DIVISION.
 000-BEGIN.
     DISPLAY "START OF LU 0 SAMPLE COBOL PROGRAM".
*****************************************************************
*   START BY TRYING TO INITIATE A SESSION WITH THE HOST CICS    *
*   SUBSYSTEM.                                                  *
*****************************************************************
     PERFORM 100-INITIATE-SESSION THRU 100-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         GO TO 099-TERMINATE.
```

Figure C-1 (cont). Sample COBOL Program for LU Type 0 for DPS 6-Initiated Session

```
*****************************************************************
*   IF THE SESSION IS INITIATED THEN WE CAN PROCEED WITH THE    *
*   REMAINDER OF THE PROGRAM PROCESS.                           *
*****************************************************************
 005-CONTINUE.
     MOVE "ADL0" TO HOST-TRANSACTION.
     PERFORM 200-GET-RECORD THRU 200-EXIT.
     IF END-PROGRAM
         DISPLAY "END OF RUN REQUESTED - PROGRAM WILL END"
         PERFORM 999-END-PROGRAM THRU 999-EXIT
         GO TO 099-TERMINATE
     ELSE
         NEXT SENTENCE.
     MOVE HIGH-VALUES TO DPS6-DATA-RECORD.
     MOVE SPACES TO DATA-FROM-HOST
                    RECEIVE-RECORD.
     MOVE DATA-FROM-TERMINAL TO DPS6-DATA-RECORD.
     MOVE 0 TO INDX1
               NO-INPUT-SW
               RECORD-CHECK-SW
               SEND-BUFFER-SIZE.
     PERFORM 300-CHECK-TERMINAL-DATA THRU 300-EXIT VARYING INDX1
             FROM 1 BY 1 UNTIL RECORD-CHECKED.
     IF NO-INPUT-DATA
         DISPLAY "NO DATA WAS ENTERED FROM THE TERMINAL"
         DISPLAY "PLEASE KEY SOME DATA BEFORE HITTING ENTER KEY"
         GO TO 005-CONTINUE
     ELSE
         NEXT SENTENCE.
     MOVE DATA-TO-HOST TO SEND-DATA-BUFFER.
     PERFORM 400-SEND-RECORD THRU 400-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         PERFORM 999-END-PROGRAM THRU 999-EXIT
         GO TO 099-TERMINATE.
     IF INTERRUPT-CALL-NEXT
         PERFORM 700-GET-INTERRUPT-INFO THRU 700-EXIT
     ELSE
         NEXT SENTENCE.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         PERFORM 999-END-PROGRAM THRU 999-EXIT
         GO TO 099-TERMINATE.
 010-DO-RECEIVE.
     PERFORM 500-RECEIVE-INFO THRU 500-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         PERFORM 999-END-PROGRAM THRU 999-EXIT
         GO TO 099-TERMINATE.
     PERFORM 600-COMPARE-INOUT THRU 600-EXIT.
```

Figure C-1 (cont). Sample COBOL Program for LU Type 0 for DPS 6-Initiated Session

```
 020-CHECK-COMPARE.
     IF COMPARE-OK
         DISPLAY "PROGRAM WILL CONTINUE"
     ELSE
         DISPLAY "CHECK PROGRAM LOGIC - SESSION WILL TERMINATE"
         PERFORM 999-END-PROGRAM THRU 999-EXIT
         GO TO 099-TERMINATE.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         PERFORM 999-END-PROGRAM THRU 999-EXIT
         GO TO 099-TERMINATE.
     IF INTERRUPT-CALL-NEXT
         PERFORM 700-GET-INTERRUPT-INFO THRU 700-EXIT
     ELSE
         NEXT SENTENCE.
     GO TO 005-CONTINUE.
 099-TERMINATE.
     STOP RUN.
*
*
*
 100-INITIATE-SESSION.
*****************************************************************
*   THIS ROUTINE WILL ISSUE A CSINIT TO ATTEMPT TO START A      *
*   SESSION WITH THE HOST CICS SUBSYSTEM. THIS CALL WILL BE     *
*   MADE SYNCHRONOUSLY BECAUSE WE WANT TO MAKE SURE A SESSION   *
*   IS AVAILABLE BEFORE ATTEMPTING TO START A PROGRAM TO        *
*   PROGRAM CONVERSATION WITH A HOST TRANSACTION.               *
*****************************************************************
*****DISPLAY "GOING TO DO CSINIT NOW".
     CALL "CSINIT" USING SNA-WORK-AREA
                         AIF-NODE-NAME
                         REMOTE-LU-NAME
                         STD-NAME
                         SYNC-CALL
                         NEW-SESSION
                         SESSION-ID
                         MSG-RESYNC-SEND-SQN
                         MSG-RESYNC-RECV-SQN
                         RETURN-CODE-VALUES
                         INTERRUPT-TYPE
                         SOPR-STOP-TIME
                         RECEIVED-SENSE-DATA.
```

Figure C-1 (cont). Sample COBOL Program for LU Type 0 for DPS 6-Initiated Session

```
*****************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                               *
*****************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "ERRORS FROM CSINIT REQUEST - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL END - NO SESSION"
         GO TO 100-EXIT.
     DISPLAY "SESSION HAS BEEN ESTABLISHED - ID IS: "
             SESSION-ID.
     IF INTERRUPT-CALL-NEXT
         PERFORM 700-GET-INTERRUPT-INFO THRU 700-EXIT
     ELSE
         NEXT SENTENCE.
 100-EXIT.
     EXIT.
*EJECT
 200-GET-RECORD.
     MOVE HIGH-VALUES TO DATA-FROM-TERMINAL.
*****************************************************************
*   NOW GET SOME DATA FROM THE TERMINAL OPERATOR TO SEND TO THE *
*   HOST REMOTE PROGRAM.                                        *
*****************************************************************
     DISPLAY ENTER-MESSAGE.
     ACCEPT DATA-FROM-TERMINAL.
 200-EXIT.
     EXIT.
*SKIP3
 300-CHECK-TERMINAL-DATA.
*****************************************************************
*   NOW CHECK THE INPUT FROM THE TERMINAL TO SEE IF ANY DATA    *
*   WAS ENTERED AND CALCULATE THE LENGTH OF THE DATA ENTERED    *
*   THEN CONVERT THE DATA TO EBCDIC.                            *
*****************************************************************
     IF CHECK-INPUT-FIELD (INDX1) IS EQUAL TO HIGH-VALUES
         MOVE 1 TO RECORD-CHECK-SW
         COMPUTE CALC-LENGTH = INDX1 - 1
         IF CALC-LENGTH IS EQUAL TO ZEROS OR
            CALC-LENGTH IS LESS THAN ZEROS
             MOVE 1 TO NO-INPUT-SW
             GO TO 300-EXIT
         ELSE
             ADD 4 TO SEND-BUFFER-SIZE
             COMPUTE CONVERSION-LENGTH = SEND-BUFFER-SIZE
             PERFORM 305-CONVERT-RECORD THRU 305-EXIT
     ELSE
         ADD 1 TO SEND-BUFFER-SIZE.
 300-EXIT.
     EXIT.
*SKIP3
 305-CONVERT-RECORD.
```

Figure C-1 (cont). Sample COBOL Program for LU Type 0 for DPS 6-Initiated Session

```
*****************************************************************
*   THIS ROUTINE WILL ISSUE THE CSACEB CALL TO CONVERT THE DATA *
*   FROM THE TERMINAL AND THE HOST TRANSACTION NAME TO EBCDIC   *
*   BEFORE THE DATA IS SENT TO THE HOST CICS SYSTEM.            *
*****************************************************************
     CALL "CSACEB" USING SNA-WORK-AREA
                         DATA-TO-HOST
                         TRANSLATE-FROM-POSITION
                         DATA-TO-HOST
                         TRANSLATE-TO-POSITION
                         CONVERSION-LENGTH.
     IF CALL-FORMAT-ERROR
         DISPLAY "COBOL ERROR IN CSACEB CALL - CHECK RETURN CODES"
         DISPLAY "COBOL RETURN CODE IS: " MINOR-RETURN-CODE
         DISPLAY "PROGRAM WILL TERMINATE"
         MOVE 1 TO ERROR-IN-CALL-SW
     ELSE
         NEXT SENTENCE.
 305-EXIT.
     EXIT.
*EJECT
 400-SEND-RECORD.
*****************************************************************
*   THIS ROUTINE WILL ISSUE THE CSSEND CALL TO SEND THE DATA    *
*   TO THE HOST.  THE FIRST FOUR BYTES OF THE DATA CONTAIN THE  *
*   HOST CICS TRANSACTION CODE (ADL0) WHICH CAUSES CICS TO LOAD *
*   THE PROGRAM ASSOCIATED WITH THAT TRANSACTION AND BEGINS THE *
*   PROGRAM TO PROGRAM CONVERSATION.  THIS CALL IS MADE         *
*   SYNCHRONOUSLY SINCE THE DESIGN OF THE PROGRAMS IS TO SEND   *
*   A MESSAGE THEN WAIT FOR THE RETURN MESSAGE.  ALSO, THE      *
*   ENTIRE MESSAGE IS DELIVERED TO AIF, NOT MESSAGE SEGMENTS.   *
*****************************************************************
*****DISPLAY "GOING TO DO CSSEND NOW".
     CALL "CSSEND" USING SNA-WORK-AREA
                         SEND-DATA-BUFFER
                         SEND-BUFFER-SIZE
                         DATA-BUFFER-ALIGNMENT
                         SYNC-CALL
                         REPLY-REQUEST
                         WHOLE-MSG-INDICATOR
                         FMH-INDICATOR
                         RQD-INDICATOR.
*****************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                               *
*****************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "ERRORS FROM CSSEND REQUEST - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL TERMINATE".
 400-EXIT.
     EXIT.
*EJECT
 500-RECEIVE-INFO.
```

Figure C-1 (cont). Sample COBOL Program for LU Type 0 for DPS 6-Initiated Session

```
*****************************************************************
*   THIS ROUTINE WILL ISSUE THE CSRECV CALL TO RECEIVE THE      *
*   DATA FROM THE HOST TRANSACTION PROGRAM.  THIS CALL IS MADE *
*   SYNCHRONOUSLY AND THE PROGRAM EXPECTS THE ENTIRE MESSAGE    *
*   TO BE DELIVERED.                                            *
*****************************************************************
*****DISPLAY "GOING TO DO CSRECV"
     CALL "CSRECV" USING SNA-WORK-AREA
                         RECEIVE-DATA-BUFFER
                         RECEIVE-BUFFER-SIZE
                         DATA-BUFFER-ALIGNMENT
                         SYNC-CALL
                         WHOLE-MSG-INDICATOR
                         RECEIVED-DATA-LENGTH
                         OUTPUT-CONTROL-WORD.
******************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                                *
******************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "ERRORS FROM CSRECV - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL TERMINATE"
         GO TO 500-EXIT.
 505-CHECK-STATUS-WORD.
******************************************************************
*   THIS ROUTINE WILL CHECK THE OUTPUT CONTROL WORD STATUS       *
*   FIELDS TO DETERMINE WHAT CONTROL INFORMATION WAS RETURNED    *
*   TO THE PROGRAM BESIDES THE DATA.  THE CONTROL INFORMATION    *
*   WOULD INDICATE ADDITIONAL PROCESSING THIS PROGRAM WOULD      *
*   HAVE TO DO BEFORE CONTINUING NORMAL PROCESSING.  THE         *
*   DESIGN OF THE TWO COMPLEMENTARY PROGRAMS WOULD INDICATE      *
*   WHETHER ANY SPECIAL PROCESSING, LIKE CHAINING OR DEFINITE    *
*   RESPONSE, WOULD HAVE TO BE HANDLED.                          *
******************************************************************
     IF CHANGE-DIRECTION-RCVD
         DISPLAY "HOST PROGRAM IS WAITING TO RECEIVE NOW"
     ELSE
         NEXT SENTENCE.
     IF MSG-WITH-EB-RECEIVED
         DISPLAY "HOST TRANSACTION HAS ENDED - PROGRAM CAN SEND"
     ELSE
         NEXT SENTENCE.
     IF DEFINITE-RESPONSE-RCVD
         DISPLAY "HOST PROGRAM IS EXPECTING A RESPONSE"
         DISPLAY "ISSUE A CSSRSP CALL NEXT"
     ELSE
         NEXT SENTENCE.
```

Figure C-1 (cont). Sample COBOL Program for LU Type 0 for DPS 6-Initiated Session

```
     IF RCVD-DATA-HAS-FMH
         DISPLAY "DATA FROM HOST CONTAINS FMH INFORMATION"
         DISPLAY "CHECK THE FMH DATA BEFORE CONTINUING"
     ELSE
         NEXT SENTENCE.
     IF BEGIN-CHAIN-RCVD
         DISPLAY "HOST PROGRAM HAS SENT THE BEGINNING OF A CHAIN"
         DISPLAY " OF DATA - MULTIPLE RECEIVES MAY BE REQUIRED"
     ELSE
         NEXT SENTENCE.
     IF END-CHAIN-RCVD
         DISPLAY "LAST RECEIVE CALL HAS ENDED THE CHAIN"
     ELSE
         NEXT SENTENCE.
 500-EXIT.
     EXIT.
*EJECT
 600-COMPARE-INOUT.
****************************************************************
*  THIS ROUTINE WILL COMPARE THE DATA RECEIVED FROM THE HOST   *
*  WITH THE DATA ORIGINALLY SENT.  IF THEY ARE NOT THE SAME    *
*  A SWITCH IS SET AND ERROR MESSAGES ARE DISPLAYED.           *
****************************************************************
     DISPLAY "GOING TO COMPARE RECORD SENT TO RECEIVED NOW".
     MOVE RECEIVE-RECORD TO DATA-FROM-HOST.
     COMPUTE SEND-BUFFER-SIZE = SEND-BUFFER-SIZE - 4.
     IF SEND-BUFFER-SIZE IS EQUAL TO RECEIVED-DATA-LENGTH
         NEXT SENTENCE
     ELSE
         DISPLAY "BUFFER LENGTHS ARE NOT THE SAME"
         DISPLAY "SEND LENGTH: " SEND-BUFFER-SIZE
                 " RECEIVE LENGTH: " RECEIVED-DATA-LENGTH.
     MOVE 0 TO COMPARE-REC-SW
               NUMBER-CHARS
               NO-MORE-SW
               INDX1.
     PERFORM 800-COMPARE-EACH-FIELD THRU 800-EXIT
         VARYING INDX1 FROM 1 BY 1
             UNTIL NO-MORE-TO-CHECK.
     IF COMPARE-OK
         DISPLAY "DATA FROM HOST IS THE SAME AS DATA SENT"
     ELSE
         DISPLAY "DATA FROM HOST IS NOT THE SAME AS DATA SENT"
         DISPLAY "POSSIBLE LOGIC ERROR".
 605-CONVERT-DATA.
```

Figure C-1 (cont). Sample COBOL Program for LU Type 0 for DPS 6-Initiated Session

```
*******************************************************************
*  THIS ROUTINE WILL CONVERT THE RECEIVED DATA FROM EBCDIC TO    *
*  ASCII AND DISPLAY THE RECORD ON THE TERMINAL.                 *
*******************************************************************
      COMPUTE CONVERSION-LENGTH = RECEIVED-DATA-LENGTH.
      CALL "CSEBAC" USING SNA-WORK-AREA
                          DATA-FROM-HOST
                          TRANSLATE-FROM-POSITION
                          DATA-FROM-HOST
                          TRANSLATE-TO-POSITION
                          CONVERSION-LENGTH.
     IF CALL-FORMAT-ERROR
         DISPLAY "COBOL ERROR IN CSEBAC CALL - CHECK RETURN CODES"
         DISPLAY "COBOL RETURN CODE IS: " MINOR-RETURN-CODE
         DISPLAY "PROGRAM WILL TERMINATE"
         MOVE 1 TO ERROR-IN-CALL-SW
         GO TO 600-EXIT
     ELSE
         NEXT SENTENCE.
     DISPLAY "RECIEVED DATA IS: ".
     DISPLAY DATA-FROM-HOST.
 600-EXIT.
     EXIT.
*SKIP3
 700-GET-INTERRUPT-INFO.
******************************************************************
*  THIS ROUTINE WILL ISSUE A CSRI CALL IN ORDER TO PICK UP THE *
*  LENGTH OF ANY INTERRUPT INFORMATION THAT IS BEING RETURNED  *
*  TO THE PROGRAM.  AFTER THIS CALL IS COMPLETED A CSWANY MUST *
*  BE ISSUED BECAUSE A CSRI IS AN ASYNCHRONOUS CALL.  A CSRECV *
*  WOULD BE ISSUED AFTER THAT IF THERE IS AN INTERRUPT MESSAGE *
*  TO PICK UP. THE INTERRUPT TYPE RETURNED ON THE ORIGINAL     *
*  SESSION CALL WILL INDICATE WHAT FURTHER PROCESSING THE      *
*  PROGRAM SHOULD DO NEXT.  WE JUST DISPLAY ANY INFORMATION    *
*  RETURNED TO THE PROGRAM THEN CONTINUE NORMAL PROCESSING.    *
*  SOME INTERRUPTS MAY REQUIRE OTHER PROCESSING LOGIC.         *
******************************************************************
*****DISPLAY "GOING TO ISSUE CSRI CALL NOW"
     CALL "CSRI" USING SNA-WORK-AREA
                       INTERRUPT-DATA-LENGTH.
*******************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                                *
*******************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "ERRORS FROM CSRI - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL TERMINATE"
         GO TO 700-EXIT.
```

Figure C-1 (cont). Sample COBOL Program for LU Type 0 for DPS 6-Initiated Session

```
*****************************************************************
*   ISSUE THE CSWANY CALL TO FORCE THE PROGRAM TO WAIT FOR THE  *
*   RETURN FROM THE CSRI CALL.                                  *
*****************************************************************
     CALL "CSWANY" USING SNA-WORK-AREA.
*****************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                               *
*****************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "ERRORS FROM CSWANY - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL TERMINATE"
         GO TO 700-EXIT.
     IF INTERRUPT-DATA-LENGTH IS EQUAL TO ZERO
         DISPLAY "NO INTERRUPT MESSAGE RECEIVED - CONTINUE"
         GO TO 700-EXIT
     ELSE
         DISPLAY "NEED TO DO CSRECV FOR INTERRUPT MESSAGE".
     MOVE INTERRUPT-DATA-LENGTH TO RECEIVE-BUFFER-SIZE.
     CALL "CSRECV" USING SNA-WORK-AREA
                         RECEIVE-DATA-BUFFER
                         RECEIVE-BUFFER-SIZE
                         DATA-BUFFER-ALIGNMENT
                         SYNC-CALL
                         WHOLE-MSG-INDICATOR
                         RECEIVED-DATA-LENGTH
                         OUTPUT-CONTROL-WORD.
*****************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                               *
*****************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "ERRORS FROM CSRECV (I) - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL TERMINATE"
         GO TO 700-EXIT.
*****************************************************************
*  THIS ROUTINE WILL CONVERT THE RECEIVED DATA FROM EBCDIC TO   *
*  ASCII AND DISPLAY THE RECORD ON THE TERMINAL.                *
*****************************************************************
      COMPUTE CONVERSION-LENGTH = RECEIVED-DATA-LENGTH.
      CALL "CSEBAC" USING SNA-WORK-AREA
                          RECEIVE-DATA-BUFFER
                          TRANSLATE-FROM-POSITION
                          RECEIVE-DATA-BUFFER
                          TRANSLATE-TO-POSITION
                          CONVERSION-LENGTH.
```

Figure C-1 (cont). Sample COBOL Program for LU Type 0 for DPS 6-Initiated Session

```
     IF CALL-FORMAT-ERROR
         DISPLAY "COBOL ERROR IN CSEBAC CALL - CHECK RETURN CODES"
         DISPLAY "COBOL RETURN CODE IS: " MINOR-RETURN-CODE
         DISPLAY "PROGRAM WILL TERMINATE"
         MOVE 1 TO ERROR-IN-CALL-SW
         GO TO 600-EXIT
     ELSE
         NEXT SENTENCE.
     DISPLAY "INTERRUPT INFORMATION IS: " RECEIVE-DATA-BUFFER.
 700-EXIT.
     EXIT.
*EJECT
 800-COMPARE-EACH-FIELD.
     IF CHECK-INPUT-FIELD (INDX1) IS EQUAL TO DATA-FIELD (INDX1)
         ADD 1 TO NUMBER-CHARS
     ELSE
         ADD 1 TO NUMBER-CHARS
         DISPLAY "CHARACTER NOT THE SAME IS: "
                 NUMBER-CHARS
         MOVE 1 TO COMPARE-REC-SW.
     IF INDX1 IS EQUAL TO RECEIVED-DATA-LENGTH
         MOVE 1 TO NO-MORE-SW
         DISPLAY "END OF COMPARE"
     ELSE
         ADD 1 TO INDX2.
 800-EXIT.
     EXIT.
*SKIP3
 900-CHECK-RETURN.
*****************************************************************
*   THIS ROUTINE WILL CHECK THE RETURN CODES FROM THE VARIOUS   *
*   AIF CALLS.  A SWITCH IS SET TO INDICATE WHETHER THE CALL    *
*   WAS OK OR NOT.  WHEN THE RETURN CODES ARE NOT OK THEY       *
*   WILL BE DISPLAYED ON THE TERMINAL.                          *
*****************************************************************
     MOVE 0 TO ERROR-IN-CALL-SW.
     IF CALL-FORMAT-ERROR
         MOVE 1 TO ERROR-IN-CALL-SW
         DISPLAY "COBOL FORMAT ERROR IN CALL - RETURN CODE IS: "
                 MINOR-RETURN-CODE
         DISPLAY "NEXT MESSAGE INDICATES CALL IN ERROR"
         GO TO 900-EXIT
     ELSE
         NEXT SENTENCE.
     IF SOPR-ISSUED-STOP
         DISPLAY "SOPR OPERATOR HAS ISSUED A STOP COMMAND"
         DISPLAY "STOP TIME IS: " SOPR-STOP-TIME
     ELSE
         NEXT SENTENCE.
     IF SESSION-ABORTED
         DISPLAY "LU SESSION HAS BEEN ABORTED - REINIT REQUIRED"
         MOVE 1 TO ERROR-IN-CALL-SW
     ELSE
         NEXT SENTENCE.
```

Figure C-1 (cont). Sample COBOL Program for LU Type 0 for DPS 6-Initiated Session

```
    IF INTERRUPT-RECEIVED
        DISPLAY "INTERRUPT FROM HOST OR AIF RECEIVED"
        DISPLAY "INTERRUPT TYPE IS: " INTERRUPT-TYPE
                " RECEIVED SENSE DATA IS: " RECEIVED-SENSE-DATA
        DISPLAY "DO A CSRI FOR ADDITIONAL INFORMATION"
        MOVE 1 TO INTERRUPT-SW
    ELSE
        MOVE 0 TO INTERRUPT-SW.
    IF CALL-WAS-COMPLETED AND
        MINOR-RETURN-CODE IS EQUAL TO ZEROS
        GO TO 900-EXIT
    ELSE
        NEXT SENTENCE.
    DISPLAY "SESSION CALL CONTAINS ERRORS - RETURN CODE IS: "
        MINOR-RETURN-CODE " MAJOR RETURN CODE IS: "
        MAJOR-RETURN-CODES.
    MOVE 1 TO ERROR-IN-CALL-SW.
900-EXIT.
EXIT.
*SKIP3
999-END-PROGRAM.
*****************************************************************
*   THIS ROUTINE WILL BE USED TO ISSUE A CSTERM CALL TO END THE *
*   CONVERSATION WITH THE HOST TRANSACTION AND THE LU SESSION.  *
*   A NORMAL TERMINATE IS ATTEMPTED FIRST BUT IF ERRORS ARE     *
*   RETURNED THEN AND ABNORMAL TERMINATE IS ATTEMPTED.          *
*****************************************************************
*****DISPLAY "GOING TO TRY A NORMAL TERMINATE NOW".
     MOVE "N" TO TERMINATE-TYPE
     CALL "CSTERM" USING SNA-WORK-AREA
                         TERMINATE-TYPE.
*****************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                               *
*****************************************************************
PERFORM 900-CHECK-RETURN THRU 900-EXIT.
IF OK-TO-CONTINUE
    DISPLAY "SESSION TERMINATION COMPLETE"
    GO TO 999-EXIT
ELSE
    DISPLAY "ERRORS FROM CSTERM N - CHECK RETURN CODES"
          DISPLAY "PROGRAM WILL ISSUE ABNORMAL TERMINATE".
    MOVE "A" TO TERMINATE-TYPE.
    CALL "CSTERM" USING SNA-WORK-AREA
                        TERMINATE-TYPE.
999-EXIT.
    EXIT.
```

Figure C-1 (cont). Sample COBOL Program for LU Type 0 for DPS 6-Initiated Session

```
 PROGRAM-ID. LOS2CH.

************************************************************************
*     THIS IS A SAMPLE LU 0 PROGRAM WHICH WILL EXERCISE SOME OF THE    *
*     AIF LU0 COBOL CALLS.  THE PROGRAM WILL START A SESSION WITH THE  *
*     HOST TRANSACTION ADL0.  IT WILL READ DATA FROM THE TERMINAL,     *
*     CONVERT IT TO EBCDIC, AND SEND THE CONVERTED RECORD TO THE HOST  *
*     THEN RECEIVE THE RECORD BACK.  UPON RECEIVING THE DATA BACK, THE *
*     PROGRAM WILL COMPARE THE DATA THAT WAS RECEIVED WITH THE DATA    *
*     SENT DIPLAYING A PROPER MESSAGE ON THE TERMINAL.  IT WILL        *
*     CONVERT THE RECEIVED DATA TO ASCII AND DISPLAY IT ON THE         *
*     TERMINAL.  IF THE TERMINAL INPUT DATA STARTS WITH: END; THE      *
*     PROGRAM WILL TERMINATE THE SESSION AND END, OTHERWISE, THE       *
*     PROGRAM WILL GO THROUGH THE SAME PROCESS WITH WHAT HAS BEEN      *
*     RECEIVED FROM THE TERMINAL.                                      *
************************************************************************
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. LEVEL-6.
 OBJECT-COMPUTER. LEVEL-6.
*
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  START-OF-WS PIC X(32)
                VALUE "START OF WORKING STORAGE SECTION".
 01  AIF-PARAMETERS PIC X(21) VALUE "AIF PARAMETERS FOLLOW".
 77  SNA-WORK-AREA                  PIC X(200).
 77  AIF-NODE-NAME                  PIC X(8) VALUE "SMPLAIF".
 77  REMOTE-LU-NAME                 PIC X(8) VALUE "A06CICS2".
 77  STD-NAME                       PIC XX   VALUE "BB".
 77  SYNC-CALL                      PIC X    VALUE "S".
 77  ASYNC-CALL                     PIC X    VALUE "A".
 77  RESTART-SESSION                PIC X    VALUE "R".
 77  NEW-SESSION                    PIC X    VALUE "N".
 77  SESSION-ID                     PIC X(4) VALUE SPACES.
 77  MSG-RESYNC-SEND-SQN            PIC 9(5) VALUE ZEROS.
 77  MSG-RESYNC-RECV-SQN            PIC 9(5) VALUE ZEROS.
 01  SEND-DATA-BUFFER               PIC X(84) VALUE SPACES.
 77  SEND-BUFFER-SIZE               PIC 9(5) VALUE ZEROS.
 77  DATA-BUFFER-ALIGNMENT          PIC X    VALUE "L".
 77  REPLY-REQUEST                  PIC X    VALUE "R".
 77  WHOLE-MSG-INDICATOR            PIC X    VALUE "Y".
 77  FMH-INDICATOR                  PIC X    VALUE "N".
 77  RQD-INDICATOR                  PIC X    VALUE "N".
 01  RECEIVE-DATA-BUFFER.
     05  RECIEVE-TRAN               PIC X(4)  VALUE SPACES.
     05  RECEIVE-RECORD             PIC X(80) VALUE SPACES.
 77  RECEIVE-BUFFER-SIZE            PIC 9(5) VALUE 84.
 77  RECEIVED-DATA-LENGTH           PIC 9(5) VALUE 0.
```

Figure C-2. Sample COBOL Program for LU Type 0 for Host-Initiated Session

```
77  RECV-COMPLETE-MSG               PIC X    VALUE "Y".
77  WORK-AREA-ID                    PIC X(4) VALUE SPACES.
77  SEND-RESPONSE-TYPE              PIC X    VALUE "-".
77  SEND-SENSE-DATA                 PIC X(8) VALUE ZEROS.
01  RETURN-CODE-VALUES.
    05  MAJOR-RETURN-CODES.
        10  SESSION-ABORT           PIC X    VALUE "N".
            88  SESSION-ABORTED     VALUE "Y".
        10  STOP-RECEIVED           PIC X    VALUE "N".
            88  SOPR-ISSUED-STOP    VALUE "Y".
        10  INTERRUPT-RECVD         PIC X    VALUE "N".
            88  INTERRUPT-RECEIVED  VALUE "Y".
        10  SERV-REQ-CANCELLED      PIC X    VALUE "N".
            88  CALL-WAS-CANCELLED  VALUE "Y".
        10  SERV-REQ-COMPLETE       PIC X    VALUE "N".
            88  CALL-WAS-COMPLETED  VALUE "Y".
        10  COBOL-ERROR             PIC X    VALUE "N".
            88  CALL-FORMAT-ERROR   VALUE "Y".
    05  MINOR-RETURN-CODE           PIC 9(4) VALUE ZEROS.
77  INTERRUPT-TYPE                  PIC 99   VALUE ZEROS.
77  INTERRUPT-DATA-LENGTH           PIC 9(5) VALUE ZEROS.
77  TERMINATE-TYPE                  PIC X    VALUE "N".
77  GET-ATTR-TYPE                   PIC 99   VALUE 01.
01  SOPR-STOP-TIME.
    05  DATE-OF-STOP.
        10  STOP-YEAR               PIC 99.
        10  STOP-MONTH              PIC 99.
        10  STOP-DAY                PIC 99.
    05  TIME-OF-STOP.
        10  STOP-HOUR               PIC 99.
        10  STOP-MINUTE             PIC 99.
        10  STOP-SECONDS            PIC 9(4).
77  RECEIVED-SENSE-DATA             PIC X(8) VALUE ZEROS.
01  OUTPUT-CONTROL-WORD.
    05  REPLY-REQUEST-CD            PIC X.
        88  CHANGE-DIRECTION-RCVD   VALUE "Y".
    05  DEFINITE-RESPONSE-REQ       PIC X.
        88  DEFINITE-RESPONSE-RCVD  VALUE "Y".
    05  LAST-MSG-EB                 PIC X.
        88  MSG-WITH-EB-RECEIVED    VALUE "Y".
    05  FMH-IN-RCVD-DATA            PIC X.
        88  RCVD-DATA-HAS-FMH       VALUE "Y".
    05  BEGIN-MSG-RCVD-BC           PIC X.
        88  BEGIN-CHAIN-RCVD        VALUE "Y".
    05  END-MSG-RCVD-EC             PIC X.
        88  END-CHAIN-RCVD          VALUE "Y".
    05  SET-SEND-RECV-SEQ           PIC X.
        88  STSN-RECEIVED           VALUE "Y".
    05  APPL-RESEND-REQUIRED        PIC X.
        88  RESTART-LAST-MSG-INB    VALUE "Y".
    05  HOST-RESEND-REQUIRED        PIC X.
        88  RESTART-LAST-MSG-OUTB   VALUE "Y".
77  CONVERSION-LENGTH               COMP-1.
77  TRANSLATE-FROM-POSITION         COMP-1   VALUE 1.
```

Figure C-2 (cont). Sample COBOL Program for LU Type 0 for Host-Initiated Session

```
77  TRANSLATE-TO-POSITION           COMP-1   VALUE 1.
01  END-OF-AIF PIC X(21) VALUE "END OF AIF PARAMETERS".
01  MISC-PROGRAM-VARIABLES  PIC X(26) VALUE
                            "OTHER WORKING STORAGE DATA".
01  DATA-TO-HOST                    PIC X(84) VALUE HIGH-VALUES.
01  DATA-TO-HOST-REDEF REDEFINES DATA-TO-HOST.
    05  HOST-TRANSACTION            PIC X(4).
    05  DPS6-DATA-RECORD.
        10  CHECK-INPUT-FIELD OCCURS 80 TIMES.
            15  DATA-FIELD-CHAR     PIC X.
01  DATA-FROM-HOST.
    05  DATA-FIELD OCCURS 80 TIMES.
        10  DATA-FLD-CHAR     PIC X.
01  DATA-FROM-TERMINAL.
    05  END-INDICATOR             PIC XXX.
        88  END-PROGRAM           VALUE "END".
    05  FILLER                    PIC X(77) VALUE SPACES.
01  SWITCH-COUNT-VARIABLES.
    05  INDX1                     COMP-1   VALUE 1.
    05  INDX2                     COMP-1   VALUE 0.
    05  NUMBER-CHARS              PIC 9(4) VALUE ZEROS.
    05  CALC-LENGTH               COMP-1   VALUE ZEROS.
    05  TEMP-LENGTH               PIC 9(5) VALUE ZEROS.
    05  ERROR-IN-CALL-SW          PIC 9    VALUE 0.
        88  OK-TO-CONTINUE        VALUE 0.
    05  RECORD-CHECK-SW           PIC 9    VALUE 0.
        88  RECORD-CHECKED        VALUE 1.
    05  NO-INPUT-SW               PIC 9    VALUE 0.
        88  NO-INPUT-DATA         VALUE 1.
    05  COMPARE-REC-SW            PIC 9    VALUE 0.
        88  COMPARE-OK            VALUE 0.
    05  NO-MORE-SW                PIC 9    VALUE 0.
        88  NO-MORE-TO-CHECK      VALUE 1.
    05  INTERRUPT-SW              PIC 9    VALUE 0.
        88  INTERRUPT-CALL-NEXT VALUE 1.
01  ENTER-MESSAGE                 PIC X(80) VALUE
    "PLEASE ENTER DATA TO TRANSMIT TO HOST OR END TO QUIT".
01  END-OF-WORK-STOR  PIC X(19) VALUE "END WORKING STORAGE".
LINKAGE SECTION.
77  NODE-NAME                     PIC X(8).
77  STD                           PIC XX.
77  BASE-LEVEL                    PIC 99.
PROCEDURE DIVISION USING NODE-NAME
                         STD
                         BASE-LEVEL.
000-BEGIN.
    DISPLAY "START OF LU 0 SAMPLE COBOL PROGRAM".
    MOVE NODE-NAME TO AIF-NODE-NAME.
    MOVE STD TO STD-NAME.
    DISPLAY "AIF NODE IS: " NODE-NAME " STD IS: " STD.
```

Figure C-2 (cont). Sample COBOL Program for LU Type 0 for Host-Initiated Session

```
*****************************************************************
*   START BY TRYING TO ATTACH TO A SESSION THAT WAS STARTED BY  *
*   THE HOST CICS TRANSACTION ADLH.                             *
*****************************************************************
     PERFORM 100-ACCEPT-SESSION THRU 100-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         GO TO 099-TERMINATE.
*****************************************************************
*   IF THE SESSION IS CONNECTED THEN WE MUST ISSUE A RECEIVE    *
*   CALL SINCE A HOST INITIATED PROGRAM COMES UP IN RECEIVE     *
*   STATE TO RECEIVE AT A MININUM THE TRANSACTION NAME SENT     *
*   BY THE HOST.                                                *
*****************************************************************
     PERFORM 500-RECEIVE-INFO THRU 500-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "INITIAL CSRECV PROBLEM - PROGRAM WILL TERMINATE"
         PERFORM 999-END-PROGRAM THRU 999-EXIT
         GO TO 099-TERMINATE.
*****************************************************************
*   IF THE SESSION IS CONNECTED THEN WE CAN PROCEED WITH THE    *
*   REMAINDER OF THE PROGRAM PROCESS.                           *
*****************************************************************
 005-CONTINUE.
     MOVE "ADLH" TO HOST-TRANSACTION.
     PERFORM 200-GET-RECORD THRU 200-EXIT.
     IF END-PROGRAM
         DISPLAY "END OF RUN REQUESTED - PROGRAM WILL END"
         PERFORM 999-END-PROGRAM THRU 999-EXIT
         GO TO 099-TERMINATE
     ELSE
         NEXT SENTENCE.
     MOVE HIGH-VALUES TO DPS6-DATA-RECORD.
     MOVE SPACES TO DATA-FROM-HOST
                    RECEIVE-RECORD.
     MOVE DATA-FROM-TERMINAL TO DPS6-DATA-RECORD.
     MOVE 0 TO INDX1
               NO-INPUT-SW
               RECORD-CHECK-SW
               SEND-BUFFER-SIZE.
     PERFORM 300-CHECK-TERMINAL-DATA THRU 300-EXIT VARYING INDX1
             FROM 1 BY 1 UNTIL RECORD-CHECKED.
     IF NO-INPUT-DATA
         DISPLAY "NO DATA WAS ENTERED FROM THE TERMINAL"
         DISPLAY "PLEASE KEY SOME DATA BEFORE HITTING ENTER KEY"
         GO TO 005-CONTINUE
     ELSE
         NEXT SENTENCE.
     MOVE DATA-TO-HOST TO SEND-DATA-BUFFER.
     PERFORM 400-SEND-RECORD THRU 400-EXIT.
```

Figure C-2 (cont). Sample COBOL Program for LU Type 0 for Host-Initiated Session

```
           IF OK-TO-CONTINUE
               NEXT SENTENCE
           ELSE
               PERFORM 999-END-PROGRAM THRU 999-EXIT
               GO TO 099-TERMINATE.
           IF INTERRUPT-CALL-NEXT
               PERFORM 700-GET-INTERRUPT-INFO THRU 700-EXIT
           ELSE
               NEXT SENTENCE.
           IF OK-TO-CONTINUE
               NEXT SENTENCE
           ELSE
               PERFORM 999-END-PROGRAM THRU 999-EXIT
               GO TO 099-TERMINATE.
       010-DO-RECEIVE.
           PERFORM 500-RECEIVE-INFO THRU 500-EXIT.
           IF OK-TO-CONTINUE
               NEXT SENTENCE
           ELSE
               PERFORM 999-END-PROGRAM THRU 999-EXIT
               GO TO 099-TERMINATE.
           PERFORM 600-COMPARE-INOUT THRU 600-EXIT.
       020-CHECK-COMPARE.
           IF COMPARE-OK
               DISPLAY "PROGRAM WILL CONTINUE"
           ELSE
               DISPLAY "CHECK PROGRAM LOGIC - SESSION WILL TERMINATE"
               PERFORM 999-END-PROGRAM THRU 999-EXIT
               GO TO 099-TERMINATE.
           IF OK-TO-CONTINUE
               NEXT SENTENCE
           ELSE
               PERFORM 999-END-PROGRAM THRU 999-EXIT
               GO TO 099-TERMINATE.
           IF INTERRUPT-CALL-NEXT
               PERFORM 700-GET-INTERRUPT-INFO THRU 700-EXIT
           ELSE
               NEXT SENTENCE.
           GO TO 005-CONTINUE.
       099-TERMINATE.
           STOP RUN.
      *
      *
      *
       100-ACCEPT-SESSION.
```

Figure C-2 (cont). Sample COBOL Program for LU Type 0 for Host-Initiated Session

```
*****************************************************************
*   THIS ROUTINE WILL ISSUE A CSACPT TO ATTEMPT TO CONNECT TO   *
*   AN AIF SESSION THAT HAS A BIND PENDING FROM CICS.  THIS CALL*
*   IS ALWAYS MADE SYNCHRONOUSLY.                               *
*****************************************************************
     DISPLAY "GOING TO DO CSACPT NOW".
     CALL "CSACPT" USING SNA-WORK-AREA
                         AIF-NODE-NAME
                         REMOTE-LU-NAME
                         STD-NAME
                         SYNC-CALL
                         NEW-SESSION
                         SESSION-ID
                         MSG-RESYNC-SEND-SQN
                         MSG-RESYNC-RECV-SQN
                         RETURN-CODE-VALUES
                         INTERRUPT-TYPE
                         SOPR-STOP-TIME
                         RECEIVED-SENSE-DATA.
*****************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                               *
*****************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "ERRORS FROM CSINIT REQUEST - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL END - NO SESSION"
         GO TO 100-EXIT.
     DISPLAY "SESSION HAS BEEN ESTABLISHED - ID IS: "
             SESSION-ID.
     IF INTERRUPT-CALL-NEXT
         PERFORM 700-GET-INTERRUPT-INFO THRU 700-EXIT
     ELSE
         NEXT SENTENCE.
 100-EXIT.
     EXIT.
*EJECT
 200-GET-RECORD.
     MOVE HIGH-VALUES TO DATA-FROM-TERMINAL.
*****************************************************************
*   NOW GET SOME DATA FROM THE TERMINAL OPERATOR TO SEND TO THE *
*   HOST REMOTE PROGRAM.                                        *
*****************************************************************
     DISPLAY ENTER-MESSAGE.
     ACCEPT DATA-FROM-TERMINAL.
 200-EXIT.
     EXIT.
*SKIP3
 300-CHECK-TERMINAL-DATA.
```

Figure C-2 (cont). Sample COBOL Program for LU Type 0 for Host-Initiated Session

```
******************************************************************
*   NOW CHECK THE INPUT FROM THE TERMINAL TO SEE IF ANY DATA     *
*   WAS ENTERED AND CALCULATE THE LENGTH OF THE DATA ENTERED     *
*   THEN CONVERT THE DATA TO EBCDIC.                             *
******************************************************************
     IF CHECK-INPUT-FIELD (INDX1) IS EQUAL TO HIGH-VALUES
         MOVE 1 TO RECORD-CHECK-SW
         COMPUTE CALC-LENGTH = INDX1 - 1
         IF CALC-LENGTH IS EQUAL TO ZEROS OR
            CALC-LENGTH IS LESS THAN ZEROS
             MOVE 1 TO NO-INPUT-SW
             GO TO 300-EXIT
         ELSE
             COMPUTE SEND-BUFFER-SIZE = SEND-BUFFER-SIZE + 4
             COMPUTE CONVERSION-LENGTH = SEND-BUFFER-SIZE
             PERFORM 305-CONVERT-RECORD THRU 305-EXIT
     ELSE
         ADD 1 TO SEND-BUFFER-SIZE.
 300-EXIT.
     EXIT.
*SKIP3
 305-CONVERT-RECORD.
******************************************************************
*   THIS ROUTINE WILL ISSUE THE CSACEB CALL TO CONVERT THE DATA *
*   FROM THE TERMINAL AND THE HOST TRANSACTION NAME TO EBCDIC    *
*   BEFORE THE DATA IS SENT TO THE HOST CICS SYSTEM.             *
******************************************************************
     CALL "CSACEB" USING SNA-WORK-AREA
                         DATA-TO-HOST
                         TRANSLATE-FROM-POSITION
                         DATA-TO-HOST
                         TRANSLATE-TO-POSITION
                         CONVERSION-LENGTH.
     IF CALL-FORMAT-ERROR
         DISPLAY "COBOL ERROR IN CSACEB CALL - CHECK RETURN CODES"
         DISPLAY "COBOL RETURN CODE IS: " MINOR-RETURN-CODE
         DISPLAY "PROGRAM WILL TERMINATE"
         MOVE 1 TO ERROR-IN-CALL-SW
     ELSE
         NEXT SENTENCE.
 305-EXIT.
     EXIT.
*EJECT
 400-SEND-RECORD.
```

Figure C-2 (cont). Sample COBOL Program for LU Type 0 for Host-Initiated Session

```
*****************************************************************
*   THIS ROUTINE WILL ISSUE THE CSSEND CALL TO SEND THE DATA    *
*   TO THE HOST.  THE FIRST FOUR BYTES OF THE DATA CONTAIN THE  *
*   HOST CICS TRANSACTION CODE (ADL0) WHICH CAUSES CICS TO LOAD *
*   THE PROGRAM ASSOCIATED WITH THAT TRANSACTION AND BEGINS THE *
*   PROGRAM TO PROGRAM CONVERSATION.  THIS CALL IS MADE         *
*   SYNCHRONOUSLY SINCE THE DESIGN OF THE PROGRAMS IS TO SEND   *
*   A MESSAGE THEN WAIT FOR THE RETURN MESSAGE.  ALSO, THE      *
*   ENTIRE MESSAGE IS DELIVERED TO AIF, NOT MESSAGE SEGMENTS.   *
*****************************************************************
*****DISPLAY "GOING TO DO CSSEND NOW".
     CALL "CSSEND" USING SNA-WORK-AREA
                         SEND-DATA-BUFFER
                         SEND-BUFFER-SIZE
                         DATA-BUFFER-ALIGNMENT
                         SYNC-CALL
                         REPLY-REQUEST
                         WHOLE-MSG-INDICATOR
                         FMH-INDICATOR
                         RQD-INDICATOR.
*****************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                               *
*****************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "ERRORS FROM CSSEND REQUEST - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL TERMINATE".
 400-EXIT.
     EXIT.
*EJECT
 500-RECEIVE-INFO.
****************************************************************
*   THIS ROUTINE WILL ISSUE THE CSRECV CALL TO RECEIVE THE     *
*   DATA FROM THE HOST TRANSACTION PROGRAM.  THIS CALL IS MADE *
*   SYNCHRONOUSLY AND THE PROGRAM EXPECTS THE ENTIRE MESSAGE   *
*   TO BE DELIVERED.                                           *
****************************************************************
*****DISPLAY "GOING TO DO CSRECV"
     CALL "CSRECV" USING SNA-WORK-AREA
                         RECEIVE-DATA-BUFFER
                         RECEIVE-BUFFER-SIZE
                         DATA-BUFFER-ALIGNMENT
                         SYNC-CALL
                         WHOLE-MSG-INDICATOR
                         RECEIVED-DATA-LENGTH
                         OUTPUT-CONTROL-WORD.
```

Figure C-2 (cont). Sample COBOL Program for LU Type 0 for Host-Initiated Session

```
*****************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                               *
*****************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "ERRORS FROM CSRECV - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL TERMINATE"
         GO TO 500-EXIT.
 505-CHECK-STATUS-WORD.
*****************************************************************
*   THIS ROUTINE WILL CHECK THE OUTPUT CONTROL WORD STATUS      *
*   FIELDS TO DETERMINE WHAT CONTROL INFORMATION WAS RETURNED   *
*   TO THE PROGRAM BESIDES THE DATA.  THE CONTROL INFORMATION   *
*   WOULD INDICATE ADDITIONAL PROCESSING THIS PROGRAM WOULD     *
*   HAVE TO DO BEFORE CONTINUING NORMAL PROCESSING.  THE        *
*   DESIGN OF THE TWO COMPLEMENTARY PROGRAMS WOULD INDICATE     *
*   WHETHER ANY SPECIAL PROCESSING, LIKE CHAINING OR DEFINITE   *
*   RESPONSE, WOULD HAVE TO BE HANDLED.                         *
*****************************************************************
     IF CHANGE-DIRECTION-RCVD
         DISPLAY "HOST PROGRAM IS WAITING TO RECEIVE NOW"
     ELSE
         NEXT SENTENCE.
     IF MSG-WITH-EB-RECEIVED
         DISPLAY "HOST TRANSACTION HAS ENDED - PROGRAM CAN SEND"
     ELSE
         NEXT SENTENCE.
     IF DEFINITE-RESPONSE-RCVD
         DISPLAY "HOST PROGRAM IS EXPECTING A RESPONSE"
         DISPLAY "ISSUE A CSSRSP CALL NEXT"
     ELSE
         NEXT SENTENCE.
     IF RCVD-DATA-HAS-FMH
         DISPLAY "DATA FROM HOST CONTAINS FMH INFORMATION"
         DISPLAY "CHECK THE FMH DATA BEFORE CONTINUING"
     ELSE
         NEXT SENTENCE.
     IF BEGIN-CHAIN-RCVD
         DISPLAY "HOST PROGRAM HAS SENT THE BEGINNING OF A CHAIN"
         DISPLAY " OF DATA - MULTIPLE RECEIVES MAY BE REQUIRED"
     ELSE
         NEXT SENTENCE.
     IF END-CHAIN-RCVD
         DISPLAY "LAST RECEIVE CALL HAS ENDED THE CHAIN"
     ELSE
         NEXT SENTENCE.
 500-EXIT.
     EXIT.
*EJECT
 600-COMPARE-INOUT.
```

Figure C-2 (cont). Sample COBOL Program for LU Type 0 for Host-Initiated Session

```
****************************************************************
*  THIS ROUTINE WILL COMPARE THE DATA RECEIVED FROM THE HOST   *
*  WITH THE DATA ORIGINALLY SENT.  IF THEY ARE NOT THE SAME    *
*  A SWITCH IS SET AND ERROR MESSAGES ARE DISPLAYED.           *
****************************************************************
     DISPLAY "GOING TO COMPARE RECORD SENT TO RECEIVED NOW".
     MOVE RECEIVE-RECORD TO DATA-FROM-HOST.
     IF SEND-BUFFER-SIZE IS EQUAL TO RECEIVED-DATA-LENGTH
         NEXT SENTENCE
     ELSE
         DISPLAY "BUFFER LENGTHS ARE NOT THE SAME"
         DISPLAY "SEND LENGTH: " SEND-BUFFER-SIZE
                  " RECEIVE LENGTH: " RECEIVED-DATA-LENGTH.
     MOVE 0 TO COMPARE-REC-SW
               NUMBER-CHARS
               NO-MORE-SW
               INDX1.
     COMPUTE RECEIVED-DATA-LENGTH = RECEIVED-DATA-LENGTH - 4.
     PERFORM 800-COMPARE-EACH-FIELD THRU 800-EXIT
         VARYING INDX1 FROM 1 BY 1
             UNTIL NO-MORE-TO-CHECK.
     IF COMPARE-OK
         DISPLAY "DATA FROM HOST IS THE SAME AS DATA SENT"
     ELSE
         DISPLAY "DATA FROM HOST IS NOT THE SAME AS DATA SENT"
         DISPLAY "POSSIBLE LOGIC ERROR".
 605-CONVERT-DATA.
*****************************************************************
*  THIS ROUTINE WILL CONVERT THE RECEIVED DATA FROM EBCDIC TO   *
*  ASCII AND DISPLAY THE RECORD ON THE TERMINAL.                *
*****************************************************************
      COMPUTE CONVERSION-LENGTH = RECEIVED-DATA-LENGTH + 4.
      CALL "CSEBAC" USING SNA-WORK-AREA
                          RECEIVE-DATA-BUFFER
                          TRANSLATE-FROM-POSITION
                          RECEIVE-DATA-BUFFER
                          TRANSLATE-TO-POSITION
                          CONVERSION-LENGTH.
     IF CALL-FORMAT-ERROR
         DISPLAY "COBOL ERROR IN CSEBAC CALL - CHECK RETURN CODES"
         DISPLAY "COBOL RETURN CODE IS: " MINOR-RETURN-CODE
         DISPLAY "PROGRAM WILL TERMINATE"
         MOVE 1 TO ERROR-IN-CALL-SW
         GO TO 600-EXIT
     ELSE
         NEXT SENTENCE.
     DISPLAY "RECIEVED DATA IS: ".
     DISPLAY RECEIVE-RECORD.
 600-EXIT.
     EXIT.
*SKIP3
 700-GET-INTERRUPT-INFO.
```

Figure C-2 (cont). Sample COBOL Program for LU Type 0 for Host-Initiated Session

```
*****************************************************************
*  THIS ROUTINE WILL ISSUE A CSRI CALL IN ORDER TO PICK UP THE *
*  LENGTH OF ANY INTERRUPT INFORMATION THAT IS BEING RETURNED  *
*  TO THE PROGRAM.  AFTER THIS CALL IS COMPLETED A CSWANY MUST *
*  BE ISSUED BECAUSE A CSRI IS AN ASYNCHRONOUS CALL.  A CSRECV *
*  WOULD BE ISSUED AFTER THAT IF THERE IS AN INTERRUPT MESSAGE *
*  TO PICK UP. THE INTERRUPT TYPE RETURNED ON THE ORIGINAL     *
*  SESSION CALL WILL INDICATE WHAT FURTHER PROCESSING THE      *
*  PROGRAM SHOULD DO NEXT.  WE JUST DISPLAY ANY INFORMATION    *
*  RETURNED TO THE PROGRAM THEN CONTINUE NORMAL PROCESSING.    *
*  SOME INTERRUPTS MAY REQUIRE OTHER PROCESSING LOGIC.         *
*****************************************************************
*****DISPLAY "GOING TO ISSUE CSRI CALL NOW"
     CALL "CSRI" USING SNA-WORK-AREA
                       INTERRUPT-DATA-LENGTH.
******************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                                *
******************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "ERRORS FROM CSRI - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL TERMINATE"
         GO TO 700-EXIT.
******************************************************************
*   ISSUE THE CSWANY CALL TO FORCE THE PROGRAM TO WAIT FOR THE  *
*   RETURN FROM THE CSRI CALL.                                   *
******************************************************************
     CALL "CSWANY" USING SNA-WORK-AREA.
******************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                                *
******************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "ERRORS FROM CSWANY - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL TERMINATE"
         GO TO 700-EXIT.
     IF INTERRUPT-DATA-LENGTH IS EQUAL TO ZERO
         DISPLAY "NO INTERRUPT MESSAGE RECEIVED - CONTINUE"
         GO TO 700-EXIT
     ELSE
         DISPLAY "NEED TO DO CSRECV FOR INTERRUPT MESSAGE".
     MOVE INTERRUPT-DATA-LENGTH TO RECEIVE-BUFFER-SIZE.
```

Figure C-2 (cont). Sample COBOL Program for LU Type 0 for Host-Initiated Session

```
     CALL "CSRECV" USING SNA-WORK-AREA
                         RECEIVE-DATA-BUFFER
                         RECEIVE-BUFFER-SIZE
                         DATA-BUFFER-ALIGNMENT
                         SYNC-CALL
                         WHOLE-MSG-INDICATOR
                         RECEIVED-DATA-LENGTH
                         OUTPUT-CONTROL-WORD.
*****************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                               *
*****************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "ERRORS FROM CSRECV (I) - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL TERMINATE"
         GO TO 700-EXIT.
*****************************************************************
*  THIS ROUTINE WILL CONVERT THE RECEIVED DATA FROM EBCDIC TO   *
*  ASCII AND DISPLAY THE RECORD ON THE TERMINAL.                *
*****************************************************************
      COMPUTE CONVERSION-LENGTH = RECEIVED-DATA-LENGTH.
      CALL "CSEBAC" USING SNA-WORK-AREA
                          RECEIVE-DATA-BUFFER
                          TRANSLATE-FROM-POSITION
                          RECEIVE-DATA-BUFFER
                          TRANSLATE-TO-POSITION
                          CONVERSION-LENGTH.
     IF CALL-FORMAT-ERROR
         DISPLAY "COBOL ERROR IN CSEBAC CALL - CHECK RETURN CODES"
         DISPLAY "COBOL RETURN CODE IS: " MINOR-RETURN-CODE
         DISPLAY "PROGRAM WILL TERMINATE"
         MOVE 1 TO ERROR-IN-CALL-SW
         GO TO 600-EXIT
     ELSE
         NEXT SENTENCE.
     DISPLAY "INTERRUPT INFORMATION IS: "
               RECEIVE-DATA-BUFFER.
 700-EXIT.
     EXIT.
*EJECT
 800-COMPARE-EACH-FIELD.
     IF CHECK-INPUT-FIELD (INDX1) IS EQUAL TO DATA-FIELD (INDX1)
         ADD 1 TO NUMBER-CHARS
     ELSE
         ADD 1 TO NUMBER-CHARS
         DISPLAY "CHARACTER NOT THE SAME IS: "
                 NUMBER-CHARS
         MOVE 1 TO COMPARE-REC-SW.
```

Figure C-2 (cont). Sample COBOL Program for LU Type 0 for Host-Initiated Session

```
      IF INDX1 IS EQUAL TO RECEIVED-DATA-LENGTH
          MOVE 1 TO NO-MORE-SW
          DISPLAY "END OF COMPARE"
      ELSE
          ADD 1 TO INDX2.
  800-EXIT.
      EXIT.
 *SKIP3
  900-CHECK-RETURN.
 *****************************************************************
 *   THIS ROUTINE WILL CHECK THE RETURN CODES FROM THE VARIOUS   *
 *   AIF CALLS.  A SWITCH IS SET TO INDICATE WHETHER THE CALL    *
 *   WAS OK OR NOT.  WHEN THE RETURN CODES ARE NOT OK THEY       *
 *   WILL BE DISPLAYED ON THE TERMINAL.                          *
 *****************************************************************
      MOVE 0 TO ERROR-IN-CALL-SW.
      IF CALL-FORMAT-ERROR
          MOVE 1 TO ERROR-IN-CALL-SW
          DISPLAY "COBOL FORMAT ERROR IN CALL - RETURN CODE IS: "
                  MINOR-RETURN-CODE
          DISPLAY "NEXT MESSAGE INDICATES CALL IN ERROR"
          GO TO 900-EXIT
      ELSE
          NEXT SENTENCE.
      IF SOPR-ISSUED-STOP
          DISPLAY "SOPR OPERATOR HAS ISSUED A STOP COMMAND"
          DISPLAY "STOP TIME IS: " SOPR-STOP-TIME
      ELSE
          NEXT SENTENCE.
      IF SESSION-ABORTED
          DISPLAY "LU SESSION HAS BEEN ABORTED - REINIT REQUIRED"
          MOVE 1 TO ERROR-IN-CALL-SW
      ELSE
          NEXT SENTENCE.
      IF INTERRUPT-RECEIVED
          DISPLAY "INTERRUPT FROM HOST OR AIF RECEIVED"
          DISPLAY "INTERRUPT TYPE IS: " INTERRUPT-TYPE
                  " RECEIVED SENSE DATA IS: " RECEIVED-SENSE-DATA
          DISPLAY "DO A CSRI FOR ADDITIONAL INFORMATION"
          MOVE 1 TO INTERRUPT-SW
      ELSE
          MOVE 0 TO INTERRUPT-SW.
      IF CALL-WAS-COMPLETED AND
         MINOR-RETURN-CODE IS EQUAL TO ZEROS
          GO TO 900-EXIT
      ELSE
          NEXT SENTENCE.
      DISPLAY "VERB CALL CONTAINS ERRORS - RETURN CODE IS: "
              MINOR-RETURN-CODE " MAJOR RETURN CODE IS: "
              MAJOR-RETURN-CODES.
      MOVE 1 TO ERROR-IN-CALL-SW.
  900-EXIT.
      EXIT.
```

Figure C-2 (cont). Sample COBOL Program for LU Type 0 for Host-Initiated Session

```
*SKIP3
 999-END-PROGRAM.
*****************************************************************
*   THIS ROUTINE WILL BE USED TO ISSUE A CSTERM CALL TO END THE *
*   CONVERSATION WITH THE HOST TRANSACTION AND THE LU SESSION.  *
*   AN ABNORMAL TERMINATE IS DONE SINCE THE HOST TRANSACTION    *
*   IS DESIGNED TO NOT END THE BRACKET.                         *
*****************************************************************
*****DISPLAY "GOING TO TRY A NORMAL TERMINATE NOW".
     MOVE "A" TO TERMINATE-TYPE
     CALL "CSTERM" USING SNA-WORK-AREA
                         TERMINATE-TYPE.
*****************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                               *
*****************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         DISPLAY "SESSION TERMINATION COMPLETE"
         GO TO 999-EXIT
     ELSE
         DISPLAY "ERRORS FROM CSTERM A - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL ISSUE ABNORMAL TERMINATE AGAIN".
     MOVE "A" TO TERMINATE-TYPE.
     CALL "CSTERM" USING SNA-WORK-AREA
                         TERMINATE-TYPE.
 999-EXIT.
     EXIT.
```

Figure C-2 (cont). Sample COBOL Program for LU Type 0 for Host-Initiated Session

```
 PROGRAM-ID. L6S1C.

************************************************************************
*     THIS IS A SAMPLE LU 6.2 PROGRAM WHICH WILL EXERCISE SOME         *
*     OF THE AIF 6.2 VERBS.  THE PROGRAM WILL ALLOCATE A               *
*     CONVERSATION WITH THE HOST TRANSACTION ADL6.  IT WILL            *
*     READ DATA FROM THE TERMINAL, CONVERT IT TO EBCDIC, BUILD         *
*     THE LOGICAL RECORD TO SEND TO THE HOST, SEND THE RECORD          *
*     TO THE HOST AND RECEIVE THE RECORD BACK.  UPON RECEIVING         *
*     THE DATA BACK, THE PROGRAM WILL COMPARE THE DATA THAT            *
*     WAS RECEIVED WITH THE DATA SENT AND SEND EITHER A CON-           *
*     FIRMATION OR AN ERROR MESSAGE TO THE HOST DEPENDING ON           *
*     WHETHER THE TWO COMPARED THE SAME.  IT WILL CONVERT THE          *
*     RECEIVED DATA TO ASCII AND DISPLAY IT ON THE TERMINAL.           *
*     IF THE TERMINAL INPUT DATA STARTS WITH: END; THE PROGRAM         *
*     WILL DEALLOCATE THE CONVERSTATION AND END OTHERWISE IT           *
*     WILL DO THE SAME PROCESS WITH WHAT HAS BEEN RECEIVED             *
*     FROM THE TERMINAL.                                               *
************************************************************************
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. LEVEL-6.
 OBJECT-COMPUTER. LEVEL-6.
*
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01   START-OF-WS PIC X(32)
                 VALUE "START OF WORKING STORAGE SECTION".
 01   AIF-PARAMETERS PIC X(21) VALUE "AIF PARAMETERS FOLLOW".
 77   SNA-WORK-AREA                    PIC X(200).
 77   AIF-NODE-NAME                    PIC X(8) VALUE "SMPLAIF".
 77   REMOTE-LU-NAME                   PIC X(8) VALUE "A06CICS2".
 77   STD-NAME                         PIC XX   VALUE "AA".
 77   SYNC-LEVEL                       PIC X    VALUE "C".
 77   HOST-TRANSACTION-NAME            PIC X(4) VALUE "ADL6".
 77   TRANSLATE-TRAN-NAME              PIC X    VALUE "Y".
 77   RETURN-CONTROL                   PIC X    VALUE "A".
 77   CONVERSATION-ID                  PIC X(4).
 77   POSTED-CONVERSATION-ID           PIC X(4).
 01   LOGICAL-DATA-BUFFER.
      05  LOGICAL-REC-LENGTH           COMP-1.
      05  LOGICAL-RECORD               PIC X(80) VALUE SPACES.
 77   DATA-BUFFER-LENGTH               PIC 9(5) VALUE 82.
 77   DATA-BUFFER-ALIGNMENT            PIC X    VALUE "L".
 01   RECEIVE-DATA-BUFFER.
      05  RECEIVE-REC-LENGTH           COMP-1.
      05  RECEIVE-RECORD               PIC X(80) VALUE SPACES.
 77   TYPE-OF-RECEIVE                  PIC X    VALUE "B".
 77   RECEIVE-BUFFER-SIZE              PIC 9(5) VALUE 82.
 77   RECEIVED-DATA-LENGTH             PIC 9(5) VALUE 0.
 77   SEND-SENSE-DATA                  PIC X(8) VALUE ZEROS.
```

Figure C-3. Sample COBOL Program for LU Type 6.2 for DPS 6-Initiated Session

```
01  RETURN-CODE-VALUES.
    05  MAJOR-RETURN-CODES.
        10  ABEND-DEALLOCATE          PIC X.
            88  ABEND-RECEIVED        VALUE "Y".
        10  STOP-RECEIVED             PIC X.
            88  SOPR-ISSUED-STOP      VALUE "Y".
        10  SERV-REQ-CANCELLED        PIC X.
            88  CALL-WAS-CANCELLED    VALUE "Y".
        10  SERV-REQ-COMPLETE         PIC X.
            88  CALL-WAS-COMPLETED    VALUE "Y".
        10  COBOL-ERROR               PIC X.
            88  CALL-FORMAT-ERROR     VALUE "Y".
    05  MINOR-RETURN-CODE             PIC 9(4) VALUE ZEROS.
01  SOPR-STOP-TIME.
    05  DATE-OF-STOP.
        10  STOP-YEAR                 PIC 99.
        10  STOP-MONTH                PIC 99.
        10  STOP-DAY                  PIC 99.
    05  TIME-OF-STOP.
        10  STOP-HOUR                 PIC 99.
        10  STOP-MINUTE               PIC 99.
        10  STOP-SECONDS              PIC 9(4).
77  RECEIVED-SENSE-DATA               PIC X(8) VALUE ZEROS.
01  OUTPUT-CONTROL-WORD.
    05  REQUEST-SEND-RECVD            PIC X.
        88  REQUEST-TO-SEND           VALUE "Y".
    05  CONVERSATION-POSTED           PIC X.
        88  POSTED-CONVERSATION       VALUE "Y".
    05  WHAT-RECEIVED                 PIC 99.
        88  DATA-RECEIVED             VALUE 20.
        88  LL-DATA-RECEIVED-COMP     VALUE 21.
        88  LL-DATA-RECEIVED-INCOMP   VALUE 22.
        88  LL-FIELD-TRUNCATED        VALUE 08.
        88  CONFIRM-REQUEST           VALUE 02.
        88  CONFIRM-ON-HOST-PTOR      VALUE 06.
        88  SEND-REQUEST-RECVD        VALUE 04.
        88  DEALLOCATE-CONFIRM        VALUE 05.
        88  DATA-INC-LENG-0           VALUE 09.
        88  DATA-AVAIL-LENG-0         VALUE 10.
77  LOG-SWITCH                        PIC X    VALUE "N".
77  LOG-DATA                          PIC X(80) VALUE
           "ERROR IN PROGRAM".
77  TYPE-SWITCH                       PIC X    VALUE "S".
77  CONVERSION-LENGTH                 COMP-1.
77  CONFIRMATION-LOCKS                PIC X    VALUE "L".
77  TRANSLATE-FROM-POSITION           COMP-1   VALUE 1.
77  TRANSLATE-TO-POSITION             COMP-1   VALUE 1.
01  END-OF-AIF PIC X(21) VALUE "END OF AIF PARAMETERS".
01  MISC-PROGRAM-VARIABLES  PIC X(26) VALUE
                            "OTHER WORKING STORAGE DATA".
01  DATA-TO-HOST                    PIC X(80) VALUE HIGH-VALUES.
```

Figure C-3 (cont). Sample COBOL Program for LU Type 6.2 for DPS 6-Initiated Session

```
 01  DATA-TO-HOST-REDEF REDEFINES DATA-TO-HOST.
     05  CHECK-INPUT-FIELD OCCURS 80 TIMES.
         10  DATA-FIELD-CHAR      PIC X.
 01  DATA-FROM-HOST.
     05  DATA-FIELD OCCURS 80 TIMES.
         10  DATA-FLD-CHAR      PIC X.
 01  DATA-FROM-TERMINAL.
     05  END-INDICATOR            PIC XXX.
         88  END-PROGRAM          VALUE "END".
     05  FILLER                   PIC X(77) VALUE SPACES.
 01  SWITCH-COUNT-VARIABLES.
     05  INDX1                    COMP-1   VALUE 1.
     05  NUMBER-CHARS             PIC 9(4) VALUE ZEROS.
     05  CALC-LENGTH              COMP-1   VALUE ZEROS.
     05  TEMP-LENGTH              PIC 9(5) VALUE ZEROS.
     05  ERROR-IN-CALL-SW         PIC 9    VALUE 0.
         88  OK-TO-CONTINUE       VALUE 0.
     05  RECORD-BUILT-SW          PIC 9    VALUE 0.
         88  RECORD-BUILT         VALUE 1.
     05  NO-INPUT-SW              PIC 9    VALUE 0.
         88  NO-INPUT-DATA        VALUE 1.
     05  COMPARE-REC-SW           PIC 9    VALUE 0.
         88  COMPARE-OK           VALUE 0.
     05  NO-MORE-SW               PIC 9    VALUE 0.
         88  NO-MORE-TO-CHECK     VALUE 1.
 01  ENTER-MESSAGE                PIC X(80) VALUE
     "PLEASE ENTER DATA TO TRANSMIT TO HOST OR END TO QUIT".
 01  END-OF-WORK-STOR  PIC X(19)  VALUE "END WORKING STORAGE".
 PROCEDURE DIVISION.
 000-BEGIN.
     DISPLAY "START OF LU 6.2 SAMPLE COBOL PROGRAM".
******************************************************************
*   START BY TRYING TO ALLOCATE A CONVERSATION WITH HOST CICS    *
*   TRANSACTION ADL6.                                            *
******************************************************************
     PERFORM 100-ALLOCATE-CONVERSATION THRU 100-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         GO TO 099-TERMINATE.
******************************************************************
*   IF THE CONVERSATION IS ALLOCATED THEN WE CAN PROCEED WITH    *
*   THE REMAINDER OF THE PROGRAM PROCESS.                        *
******************************************************************
 005-CONTINUE.
     PERFORM 200-GET-RECORD THRU 200-EXIT.
     IF END-PROGRAM
         DISPLAY "END OF RUN REQUESTED - PROGRAM WILL END"
         PERFORM 999-END-PROGRAM THRU 999-EXIT
         GO TO 099-TERMINATE
     ELSE
         NEXT SENTENCE.
     MOVE HIGH-VALUES TO DATA-TO-HOST.
```

Figure C-3 (cont). Sample COBOL Program for LU Type 6.2 for DPS 6-Initiated Session

```
     MOVE SPACES TO DATA-FROM-HOST
                    RECEIVE-RECORD.
     MOVE DATA-FROM-TERMINAL TO DATA-TO-HOST.
     MOVE 0 TO INDX1
               NO-INPUT-SW
               RECORD-BUILT-SW
               LOGICAL-REC-LENGTH
               DATA-BUFFER-LENGTH.
     PERFORM 300-BUILD-LOGICAL THRU 300-EXIT VARYING INDX1 FROM 1
             BY 1 UNTIL RECORD-BUILT.
     IF NO-INPUT-DATA
         DISPLAY "NO DATA WAS ENTERED FROM THE TERMINAL"
         DISPLAY "PLEASE KEY SOME DATA BEFORE HITTING ENTER KEY"
         GO TO 005-CONTINUE
     ELSE
         NEXT SENTENCE.
     MOVE DATA-TO-HOST TO LOGICAL-RECORD.
     PERFORM 400-SEND-RECORD THRU 400-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         PERFORM 999-END-PROGRAM THRU 999-EXIT
         GO TO 099-TERMINATE.
 010-DO-RECEIVE.
     PERFORM 500-RECEIVE-INFO THRU 500-EXIT.
 0101-NEXT-RECEIVE.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         PERFORM 999-END-PROGRAM THRU 999-EXIT
         GO TO 099-TERMINATE.
 015-CHECK-WHAT-RECEIVED.
     IF DATA-RECEIVED
         PERFORM 600-COMPARE-INOUT THRU 600-EXIT
         PERFORM 505-ISSUE-CSRAW THRU 500-EXIT
         GO TO 0101-NEXT-RECEIVE
     ELSE
         IF DEALLOCATE-CONFIRM
             PERFORM 700-ISSUE-CONFIRMED THRU 700-EXIT
             GO TO 099-TERMINATE
         ELSE
             IF CONFIRM-ON-HOST-PTOR
                 GO TO 020-CHECK-COMPARE
         ELSE
             NEXT SENTENCE.
     DISPLAY "UNEXPECTED WHAT RECEIVED FIELD".
     DISPLAY "WHAT RECEIVED IS: " WHAT-RECEIVED.
     PERFORM 705-SEND-ERROR THRU 705-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         PERFORM 999-END-PROGRAM THRU 999-EXIT
         GO TO 099-TERMINATE.
     GO TO 0101-NEXT-RECEIVE.
```

Figure C-3 (cont). Sample COBOL Program for LU Type 6.2 for DPS 6-Initiated Session

```
 020-CHECK-COMPARE.
     IF COMPARE-OK
         PERFORM 700-ISSUE-CONFIRMED THRU 700-EXIT
     ELSE
         PERFORM 705-SEND-ERROR THRU 705-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         PERFORM 999-END-PROGRAM THRU 999-EXIT
         GO TO 099-TERMINATE.
     GO TO 005-CONTINUE.
 099-TERMINATE.
     STOP RUN.
*
*
*
 100-ALLOCATE-CONVERSATION.
*****************************************************************
*   THIS ROUTINE WILL ISSUE A CSALLO TO ATTEMPT TO ALLOCATE A   *
*   LU 6.2 CONVERSATION WITH THE HOST CICS TRANSACTION ADL6.    *
*   A CSFLSH IS ISSUED TO FORCE AIF TO SEND THE ATTACH REQUEST  *
*   TO CICS IMMEDIATELY, INSTEAD OF WAITING FOR THE SEND BUFFER *
*   TO FILL UP OR ANOTHER VERB BEING ISSUED WITH A FLUSH OPTION.*
*   WE WANT TO FIND OUT IF A CONVERSATION CAN BE STARTED BEFORE *
*   PROCEEDING FURTHER.                                         *
*****************************************************************
*****DISPLAY "GOING TO DO CSALLO NOW".
     CALL "CSALLO" USING SNA-WORK-AREA
                         AIF-NODE-NAME
                         REMOTE-LU-NAME
                         CONVERSATION-ID
                         HOST-TRANSACTION-NAME
                         TRANSLATE-TRAN-NAME
                         STD-NAME
                         RETURN-CONTROL
                         SYNC-LEVEL
                         RETURN-CODE-VALUES
                         SOPR-STOP-TIME
                         RECEIVED-SENSE-DATA
                         OUTPUT-CONTROL-WORD.
*****************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                               *
*****************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "ERRORS FROM CSALLO REQUEST - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL END - NO CONVERSATION"
         GO TO 100-EXIT.
```

Figure C-3 (cont). Sample COBOL Program for LU Type 6.2 for DPS 6-Initiated Session

```
*****************************************************************
*   NOW ISSUE THE CSFLSH TO FORCE AIF TO SEND THE ATTACH REQUEST*
*   TO THE HOST CICS SYSTEM.                                    *
*****************************************************************
*****DISPLAY "GOING TO DO CSFLSH NOW".
     CALL "CSFLSH" USING SNA-WORK-AREA.
*****************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                               *
*****************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "ERRORS FROM CSFLSH - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL END - NO CONVERSATION"
         GO TO 100-EXIT.
     DISPLAY "CONVERSATION HAS BEEN ALLOCATED - ID IS: "
             CONVERSATION-ID.
 100-EXIT.
     EXIT.
*EJECT
 200-GET-RECORD.
     MOVE HIGH-VALUES TO DATA-FROM-TERMINAL.
*****************************************************************
*   NOW GET SOME DATA FROM THE TERMINAL OPERATOR TO SEND TO THE *
*   HOST REMOTE PROGRAM.                                        *
*****************************************************************
     DISPLAY ENTER-MESSAGE.
     ACCEPT DATA-FROM-TERMINAL.
 200-EXIT.
     EXIT.
*SKIP3
 300-BUILD-LOGICAL.
*****************************************************************
*   NOW BUILD THE LOGICAL RECORD THAT WILL BE SENT TO THE HOST  *
*   BY CALCULATING THE LENGTH OF THE DATA RECEIVED THEN CONVERT *
*   THE DATA TO EBCDIC.                                         *
*****************************************************************
     IF CHECK-INPUT-FIELD (INDX1) IS EQUAL TO HIGH-VALUES
         MOVE 1 TO RECORD-BUILT-SW
         COMPUTE CALC-LENGTH = INDX1 - 1
         IF CALC-LENGTH IS EQUAL TO ZEROS OR
            CALC-LENGTH IS LESS THAN ZEROS
             MOVE 1 TO NO-INPUT-SW
             GO TO 300-EXIT
         ELSE
             ADD 2 TO DATA-BUFFER-LENGTH
                      LOGICAL-REC-LENGTH
             MOVE CALC-LENGTH TO CONVERSION-LENGTH
             MOVE LOGICAL-REC-LENGTH TO TEMP-LENGTH
             PERFORM 305-CONVERT-RECORD THRU 305-EXIT
```

Figure C-3 (cont). Sample COBOL Program for LU Type 6.2 for DPS 6-Initiated Session

```
     ELSE
         ADD 1 TO DATA-BUFFER-LENGTH
                  LOGICAL-REC-LENGTH.
 300-EXIT.
     EXIT.
*SKIP3
 305-CONVERT-RECORD.
*****************************************************************
*   THIS ROUTINE WILL ISSUE THE CSACEB CALL TO CONVERT THE DATA *
*   FROM THE TERMINAL TO EBCDIC BEFORE IT IS SENT TO THE HOST.  *
*****************************************************************
     CALL "CSACEB" USING SNA-WORK-AREA
                         DATA-TO-HOST
                         TRANSLATE-FROM-POSITION
                         DATA-TO-HOST
                         TRANSLATE-TO-POSITION
                         CONVERSION-LENGTH.
     IF CALL-FORMAT-ERROR
         DISPLAY "COBOL ERROR IN CSACEB CALL - CHECK RETURN CODES"
         DISPLAY "COBOL RETURN CODE IS: " MINOR-RETURN-CODE
         DISPLAY "PROGRAM WILL TERMINATE"
         MOVE 1 TO ERROR-IN-CALL-SW
     ELSE
         NEXT SENTENCE.
 305-EXIT.
     EXIT.
*EJECT
 400-SEND-RECORD.
*****************************************************************
*   THIS ROUTINE WILL ISSUE THE CSSDAT CALL TO SEND THE DATA    *
*   TO AIF.  AIF WILL NOT SEND THE DATA TO THE HOST UNTIL WE    *
*   ISSUE ANOTHER CALL TO FORCE A FLUSH OF THE BUFFERS. THIS    *
*   WILL BE DONE IN THE NEXT ROUTINE.                           *
*****************************************************************
*****DISPLAY "GOING TO DO CSSDAT NOW".
     CALL "CSSDAT" USING SNA-WORK-AREA
                         LOGICAL-DATA-BUFFER
                         DATA-BUFFER-LENGTH.
*****************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                               *
*****************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "ERRORS FROM CSSDAT REQUEST - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL TERMINATE".
 400-EXIT.
     EXIT.
*EJECT
 500-RECEIVE-INFO.
```

Figure C-3 (cont). Sample COBOL Program for LU Type 6.2 for DPS 6-Initiated Session

```
*****************************************************************
*   THIS ROUTINE WILL ISSUE A NUMBER OF AIF VERBS.  FIRST IT    *
*   WILL DO A CSPTOR WHICH WILL CAUSE AIF TO FLUSH THE SEND     *
*   BUFFER SENDING THE DATA FROM THE CSSDAT CALL AND A SEND     *
*   INDICATOR TO THE HOST PROGRAM TO TELL THAT PROGRAM IT CAN   *
*   TURN AROUND AND SEND TO THIS PROGRAM.                       *
*   AFTER THE CSPTOR, THE PROGRAM WILL ISSUE A CSRAW TO WAIT    *
*   FOR THE DATA TO COME BACK FROM THE HOST AND RECEIVE IT.     *
*****************************************************************
*****************************************************************
*   THE TYPE OF PREPARE TO RECEIVE IS A FLUSH (TYPE-SWITCH=F)   *
*   THE TYPE OF LOCKS IS LONG (CONFIRMATION-LOCKS=L)            *
*****************************************************************
*****DISPLAY "GOING TO DO CSPTOR TYPE F NOW".
     MOVE "F" TO TYPE-SWITCH.
     CALL "CSPTOR" USING SNA-WORK-AREA
                         TYPE-SWITCH
                         CONFIRMATION-LOCKS.
******************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                                *
******************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "ERRORS FROM CSPTOR - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL TERMINATE"
         GO TO 500-EXIT.
 505-ISSUE-CSRAW.
******************************************************************
*   ISSUE THE CSRAW TO CAUSE THE PROGRAM TO WAIT FOR A RECEIVE   *
*   AND RECEIVE THE DATA COMING BACK FROM THE HOST TRANSACTION. *
*   THE TYPE OF RECEIVE IS A BUFFER (TYPE-OF-RECEIVE=B) SO       *
*   AIF WILL PASS AN ENTIRE BUFFER'S WORTH OF DATA AS OPPOSED    *
*   TO A LOGICAL RECORD.  THIS ROUTINE WILL ALSO BE USED TO      *
*   RECEIVE STATUS OR STATE CHANGE INFORMATION.                  *
******************************************************************
*****DISPLAY "GOING TO DO CSRAW NOW".
     CALL "CSRAW" USING SNA-WORK-AREA
                        RECEIVE-DATA-BUFFER
                        RECEIVE-BUFFER-SIZE
                        TYPE-OF-RECEIVE
                        RECEIVED-DATA-LENGTH.
******************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                                *
******************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "ERRORS FROM CSRAW - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL TERMINATE".
```

Figure C-3 (cont). Sample COBOL Program for LU Type 6.2 for DPS 6-Initiated Session

```
 500-EXIT.
     EXIT.
*EJECT
 600-COMPARE-INOUT.
*****************************************************************
*  THIS ROUTINE WILL COMPARE THE DATA RECEIVED FROM THE HOST    *
*  WITH THE DATA ORIGINALLY SENT.  IF THEY ARE NOT THE SAME     *
*  A SWITCH IS SET AND ERROR MESSAGES ARE DISPLAYED.            *
*****************************************************************
     DISPLAY "GOING TO COMPARE RECORD SENT TO RECEIVED NOW".
     MOVE RECEIVE-RECORD TO DATA-FROM-HOST.
     IF DATA-BUFFER-LENGTH IS EQUAL TO RECEIVED-DATA-LENGTH
         NEXT SENTENCE
     ELSE
         DISPLAY "BUFFER LENGTHS ARE NOT THE SAME"
         DISPLAY "SEND LENGTH: " DATA-BUFFER-LENGTH
                 " RECEIVE LENGTH: " RECEIVED-DATA-LENGTH.
     IF LOGICAL-REC-LENGTH IS EQUAL TO RECEIVE-REC-LENGTH
         NEXT SENTENCE
     ELSE
         DISPLAY "LOGICAL LENGTHS ARE NOT THE SAME".
     MOVE 0 TO COMPARE-REC-SW
               NUMBER-CHARS
               NO-MORE-SW
               INDX1.
     COMPUTE RECEIVE-REC-LENGTH = RECEIVE-REC-LENGTH - 2.
     PERFORM 800-COMPARE-EACH-FIELD THRU 800-EXIT
         VARYING INDX1 FROM 1 BY 1
             UNTIL NO-MORE-TO-CHECK.
     IF COMPARE-OK
         DISPLAY "DATA FROM HOST IS THE SAME AS DATA SENT"
     ELSE
         DISPLAY "DATA FROM HOST IS NOT THE SAME AS DATA SENT"
         DISPLAY "POSSIBLE LOGIC ERROR".
 605-CONVERT-DATA.
*****************************************************************
*  THIS ROUTINE WILL CONVERT THE RECEIVED DATA FROM EBCDIC TO   *
*  ASCII AND DISPLAY THE RECORD ON THE TERMINAL.                *
*****************************************************************
      COMPUTE CONVERSION-LENGTH = RECEIVE-REC-LENGTH.
      CALL "CSEBAC" USING SNA-WORK-AREA
                          DATA-FROM-HOST
                          TRANSLATE-FROM-POSITION
                          DATA-FROM-HOST
                          TRANSLATE-TO-POSITION
                          CONVERSION-LENGTH.
     IF CALL-FORMAT-ERROR
         DISPLAY "COBOL ERROR IN CSEBAC CALL - CHECK RETURN CODES"
         DISPLAY "COBOL RETURN CODE IS: " MINOR-RETURN-CODE
         DISPLAY "PROGRAM WILL TERMINATE"
         MOVE 1 TO ERROR-IN-CALL-SW
         GO TO 600-EXIT
```

Figure C-3 (cont). Sample COBOL Program for LU Type 6.2 for DPS 6-Initiated Session

```
     ELSE
         NEXT SENTENCE.
     DISPLAY "RECIEVED DATA IS: ".
     DISPLAY DATA-FROM-HOST.
 600-EXIT.
     EXIT.
*EJECT
 700-ISSUE-CONFIRMED.
*****************************************************************
*   THIS ROUTINE WILL ISSUE A CSCNFD CALL.  THIS WILL CAUSE AIF *
*   TO SEND A CONFIRMATION TO THE HOST TRANSACTION.             *
*****************************************************************
*****DISPLAY "GOING TO DO CSCNFD NOW".
     CALL "CSCNFD" USING SNA-WORK-AREA.
*****************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                               *
*****************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "ERRORS FROM CSCNFD - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL TERMINATE".
 700-EXIT.
     EXIT.
*SKIP3
 705-SEND-ERROR.
*****************************************************************
*   THIS ROUTINE WILL ISSUE A CSSERR CALL TO NOTIFY THE HOST    *
*   TRANSACTION OF AN ERROR IN PROCESSING.  THE TYPE OF ERROR   *
*   IS PROG (TYPE-SWITCH=P).  THE PROGRAM WILL NOT REQUEST THE  *
*   LOGGING OF DATA (LOG-SWITCH=N).                             *
*****************************************************************
*****DISPLAY "GOING TO DO CSSERR TYPE P NOW".
     MOVE "P" TO TYPE-SWITCH.
     CALL "CSSERR" USING SNA-WORK-AREA
                         TYPE-SWITCH
                         LOG-SWITCH
                         LOG-DATA.
*****************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                               *
*****************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "ERRORS FROM CSSERR - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL TERMINATE".
 705-EXIT.
     EXIT.
```

Figure C-3 (cont). Sample COBOL Program for LU Type 6.2 for DPS 6-Initiated Session

```
*EJECT
 800-COMPARE-EACH-FIELD.
     IF CHECK-INPUT-FIELD (INDX1) IS EQUAL TO DATA-FIELD (INDX1)
         ADD 1 TO NUMBER-CHARS
     ELSE
         ADD 1 TO NUMBER-CHARS
         DISPLAY "CHARACTER NOT THE SAME IS: "
                 NUMBER-CHARS
         MOVE 1 TO COMPARE-REC-SW.
     IF INDX1 IS EQUAL TO RECEIVE-REC-LENGTH
         MOVE 1 TO NO-MORE-SW
         DISPLAY "END OF COMPARE"
     ELSE
         NEXT SENTENCE.
 800-EXIT.
     EXIT.
*SKIP3
 900-CHECK-RETURN.
*****************************************************************
*   THIS ROUTINE WILL CHECK THE RETURN CODES FROM THE VARIOUS   *
*   AIF VERB CALLS.  A SWITCH IS SET TO INDICATE WHETHER THE    *
*   CALL WAS OK OR NOT.  WHEN THE RETURN CODES ARE NOT OK THEY  *
*   WIL BE DISPLAYED ON THE TERMINAL.                           *
*****************************************************************
     MOVE 0 TO ERROR-IN-CALL-SW.
     IF CALL-FORMAT-ERROR
         MOVE 1 TO ERROR-IN-CALL-SW
         DISPLAY "COBOL FORMAT ERROR IN CALL - RETURN CODE IS: "
                 MINOR-RETURN-CODE
         DISPLAY "NEXT MESSAGE INDICATES CALL IN ERROR"
         GO TO 900-EXIT
     ELSE
         NEXT SENTENCE.
     IF SOPR-ISSUED-STOP
         DISPLAY "SOPR OPERATOR HAS ISSUED A STOP COMMAND"
         DISPLAY "STOP TIME IS: " SOPR-STOP-TIME
     ELSE
         NEXT SENTENCE.
     IF ABEND-RECEIVED
         DISPLAY "AN ABEND/DEALLOCATE HAS BEEN RECEIVED"
         DISPLAY "SESSION WILL BE TERMINATED"
         MOVE 1 TO ERROR-IN-CALL-SW
     ELSE
         NEXT SENTENCE.
     IF CALL-WAS-COMPLETED AND
        MINOR-RETURN-CODE IS EQUAL TO ZEROS
         GO TO 900-EXIT
     ELSE
         NEXT SENTENCE.
     DISPLAY "VERB CALL CONTAINS ERRORS - RETURN CODE IS: "
             MINOR-RETURN-CODE.
     MOVE 1 TO ERROR-IN-CALL-SW.
```

Figure C-3 (cont). Sample COBOL Program for LU Type 6.2 for DPS 6-Initiated Session

```
 900-EXIT.
     EXIT.
*SKIP3
 999-END-PROGRAM.
*****************************************************************
*   THIS ROUTINE WILL BE USED TO ISSUE A CSDEAL CALL ENDING THE *
*   CONVERSATION WITH THE HOST TRANSACTION.  THE TYPE OF DE-    *
*   ALLOCATE IS FLUSH (TYPE-SWITCH=F) ON THE FIRST ATTEMPT IF   *
*   THAT HAS AN ERROR THEN AND ABEND PROG TYPE WILL BE ISSUED   *
*   (TYPE-SWITCH=P).  THE PROGRAM WILL NOT REQUEST THE LOGGING  *
*   OF ERROR DATA (LOG-SWITCH=N).                               *
*****************************************************************
*****DISPLAY "GOING TO TRY A NORMAL DEALLOCATE NOW".
     MOVE "F" TO TYPE-SWITCH.
     CALL "CSDEAL" USING SNA-WORK-AREA
                         TYPE-SWITCH
                         LOG-SWITCH
                         LOG-DATA.
*****************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                               *
*****************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         DISPLAY "CONVERSATION HAS BEEN DEALLOCATED"
         GO TO 999-EXIT
     ELSE
         DISPLAY "ERRORS FROM CSDEAL F - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL ISSUE DEALLOCATE/ABEND".
     MOVE "P" TO TYPE-SWITCH.
     CALL "CSDEAL" USING SNA-WORK-AREA
                         TYPE-SWITCH
                         LOG-SWITCH
                         LOG-DATA.
 999-EXIT.
     EXIT.
```

Figure C-3 (cont). Sample COBOL Program for LU Type 6.2 for DPS 6-Initiated Session

```
 PROGRAM-ID. L6S2CH.

****************************************************************************
*     THIS IS A SAMPLE LU 6.2 PROGRAM WHICH WILL EXERCISE SOME             *
*     OF THE AIF 6.2 VERBS.  THE PROGRAM WILL ATTACH TO A CONVERSATION *
*     THAT IS ALLOCATED BY THE  HOST TRANSACTION ADL6.  IT WILL            *
*     READ DATA FROM THE TERMINAL, CONVERT IT TO EBCDIC, BUILD             *
*     THE LOGICAL RECORD TO SEND TO THE HOST, SEND THE RECORD              *
*     TO THE HOST AND RECEIVE THE RECORD BACK.  UPON RECEIVING             *
*     THE DATA BACK, THE PROGRAM WILL COMPARE THE DATA THAT                *
*     WAS RECEIVED WITH THE DATA SENT AND SEND EITHER A CON-               *
*     FIRMATION OR AN ERROR MESSAGE TO THE HOST DEPENDING ON               *
*     WHETHER THE TWO COMPARED THE SAME.  IT WILL CONVERT THE              *
*     RECEIVED DATA TO ASCII AND DISPLAY IT ON THE TERMINAL.               *
*     IF THE TERMINAL INPUT DATA STARTS WITH: END; THE PROGRAM             *
*     WILL DEALLOCATE THE CONVERSTATION AND END OTHERWISE IT               *
*     WILL DO THE SAME PROCESS WITH WHAT HAS BEEN RECEIVED                 *
*     FROM THE TERMINAL.                                                   *
****************************************************************************
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. LEVEL-6.
 OBJECT-COMPUTER. LEVEL-6.
*
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01   START-OF-WS PIC X(32)
                VALUE "START OF WORKING STORAGE SECTION".
 01   AIF-PARAMETERS PIC X(21) VALUE "AIF PARAMETERS FOLLOW".
 77   SNA-WORK-AREA                     PIC X(200).
 77   AIF-NODE-NAME                     PIC X(8) VALUE "SMPLAIF".
 77   REMOTE-LU-NAME                    PIC X(8) VALUE "A06CICS2".
 77   STD-NAME                          PIC XX   VALUE "AA".
 77   SYNC-LEVEL                        PIC X    VALUE "C".
 77   HOST-TRANSACTION-NAME             PIC X(4) VALUE "ADL6".
 77   TRANSLATE-TRAN-NAME               PIC X    VALUE "Y".
 77   RETURN-CONTROL                    PIC X    VALUE "A".
 77   CONVERSATION-ID                   PIC X(4).
 77   POSTED-CONVERSATION-ID            PIC X(4).
 01   LOGICAL-DATA-BUFFER.
      05  LOGICAL-REC-LENGTH            COMP-1.
      05  LOGICAL-RECORD                PIC X(80) VALUE SPACES.
 77   DATA-BUFFER-LENGTH                PIC 9(5) VALUE 82.
 77   DATA-BUFFER-ALIGNMENT             PIC X    VALUE "L".
 01   RECEIVE-DATA-BUFFER.
      05  RECEIVE-REC-LENGTH            COMP-1.
      05  RECEIVE-RECORD                PIC X(80) VALUE SPACES.
 77   TYPE-OF-RECEIVE                   PIC X    VALUE "B".
 77   RECEIVE-BUFFER-SIZE               PIC 9(5) VALUE 82.
 77   RECEIVED-DATA-LENGTH              PIC 9(5) VALUE 0.
 77   SEND-SENSE-DATA                   PIC X(8) VALUE ZEROS.
```

Figure C-4. Sample COBOL Program for LU Type 6.2 for Host-Initiated Session

```
01  RETURN-CODE-VALUES.
    05  MAJOR-RETURN-CODES.
        10  ABEND-DEALLOCATE          PIC X    VALUE "N".
            88  ABEND-RECEIVED        VALUE "Y".
        10  STOP-RECEIVED             PIC X    VALUE "N".
            88  SOPR-ISSUED-STOP      VALUE "Y".
        10  SERV-REQ-CANCELLED        PIC X    VALUE "N".
            88  CALL-WAS-CANCELLED    VALUE "Y".
        10  SERV-REQ-COMPLETE         PIC X    VALUE "N".
            88  CALL-WAS-COMPLETED    VALUE "Y".
        10  COBOL-ERROR               PIC X    VALUE "N".
            88  CALL-FORMAT-ERROR     VALUE "Y".
    05  MINOR-RETURN-CODE             PIC 9(4) VALUE ZEROS.
01  SOPR-STOP-TIME.
    05  DATE-OF-STOP.
        10  STOP-YEAR                 PIC 99.
        10  STOP-MONTH                PIC 99.
        10  STOP-DAY                  PIC 99.
    05  TIME-OF-STOP.
        10  STOP-HOUR                 PIC 99.
        10  STOP-MINUTE               PIC 99.
        10  STOP-SECONDS              PIC 9(4).
77  RECEIVED-SENSE-DATA               PIC X(8) VALUE ZEROS.
01  OUTPUT-CONTROL-WORD.
    05  REQUEST-SEND-RECVD            PIC X.
        88  REQUEST-TO-SEND           VALUE "Y".
    05  CONVERSATION-POSTED           PIC X.
        88  POSTED-CONVERSATION       VALUE "Y".
    05  WHAT-RECEIVED                 PIC 99.
        88  DATA-RECEIVED             VALUE 20.
        88  LL-DATA-RECEIVED-COMP     VALUE 21.
        88  LL-DATA-RECEIVED-INCOMP   VALUE 22.
        88  LL-FIELD-TRUNCATED        VALUE 08.
        88  CONFIRM-REQUEST           VALUE 04.
        88  CONFIRM-ON-HOST-PTOR      VALUE 06.
        88  SEND-REQUEST-RECVD        VALUE 02.
        88  DEALLOCATE-CONFIRM        VALUE 05.
        88  DATA-INC-LENG-0           VALUE 09.
        88  DATA-AVAIL-LENG-0         VALUE 10.
77  LOG-SWITCH                        PIC X    VALUE "N".
77  LOG-DATA                          PIC X(80) VALUE
           "ERROR IN PROGRAM".
77  TYPE-SWITCH                       PIC X    VALUE "S".
77  CONVERSION-LENGTH                 COMP-1.
77  CONFIRMATION-LOCKS                PIC X    VALUE "L".
77  TRANSLATE-FROM-POSITION           COMP-1   VALUE 1.
77  TRANSLATE-TO-POSITION             COMP-1   VALUE 1.
01  END-OF-AIF PIC X(21) VALUE "END OF AIF PARAMETERS".
01  MISC-PROGRAM-VARIABLES  PIC X(26) VALUE
                            "OTHER WORKING STORAGE DATA".
01  DATA-TO-HOST                PIC X(80) VALUE HIGH-VALUES.
01  DATA-TO-HOST-REDEF REDEFINES DATA-TO-HOST.
    05  CHECK-INPUT-FIELD OCCURS 80 TIMES.
        10  DATA-FIELD-CHAR     PIC X.
```

Figure C-4 (cont). Sample COBOL Program for LU Type 6.2 for Host-Initiated Session

```
01  DATA-FROM-HOST.
    05  DATA-FIELD OCCURS 80 TIMES.
        10  DATA-FLD-CHAR      PIC X.
01  DATA-FROM-TERMINAL.
    05  END-INDICATOR              PIC XXX.
        88  END-PROGRAM            VALUE "END".
    05  FILLER                     PIC X(77) VALUE SPACES.
01  SWITCH-COUNT-VARIABLES.
    05  INDX1                      COMP-1    VALUE 1.
    05  NUMBER-CHARS               PIC 9(4)  VALUE ZEROS.
    05  CALC-LENGTH                COMP-1    VALUE ZEROS.
    05  TEMP-LENGTH                PIC 9(5)  VALUE ZEROS.
    05  ERROR-IN-CALL-SW           PIC 9     VALUE 0.
        88  OK-TO-CONTINUE         VALUE 0.
    05  RECORD-BUILT-SW            PIC 9     VALUE 0.
        88  RECORD-BUILT           VALUE 1.
    05  NO-INPUT-SW                PIC 9     VALUE 0.
        88  NO-INPUT-DATA          VALUE 1.
    05  COMPARE-REC-SW             PIC 9     VALUE 0.
        88  COMPARE-OK             VALUE 0.
    05  NO-MORE-SW                 PIC 9     VALUE 0.
        88  NO-MORE-TO-CHECK       VALUE 1.
01  ENTER-MESSAGE                  PIC X(80) VALUE
    "PLEASE ENTER DATA TO TRANSMIT TO HOST OR END TO QUIT".
01  END-OF-WORK-STOR  PIC X(19) VALUE "END WORKING STORAGE".
LINKAGE SECTION.
77  NODE-NAME                      PIC X(8).
77  STD                            PIC XX.
77  BASE-LEVEL                     PIC 99.
PROCEDURE DIVISION USING NODE-NAME
                         STD
                         BASE-LEVEL.
000-BEGIN.
    DISPLAY "START OF LU 6.2 SAMPLE COBOL PROGRAM".
    MOVE NODE-NAME TO AIF-NODE-NAME.
    MOVE STD TO STD-NAME.
    DISPLAY "AIF NODE IS: " NODE-NAME " STD IS: " STD.
*****************************************************************
*   START BY TRYING TO ATTACH TO A CONVERSATION WITH HOST CICS  *
*   TRANSACTION ADL6.                                           *
*****************************************************************
    PERFORM 100-ATTACH-CONVERSATION THRU 100-EXIT.
    IF OK-TO-CONTINUE
        NEXT SENTENCE
    ELSE
        GO TO 099-TERMINATE.
```

Figure C-4 (cont). Sample COBOL Program for LU Type 6.2 for Host-Initiated Session

```
*****************************************************************
*   IF THE CONVERSATION IS ATTACHED THEN WE MUST ISSUE A        *
*   RECEIVE AND WAIT SINCE A HOST INITIATED PROGRAM ALWAYS      *
*   COMES UP IN RECEIVE STATE.                                  *
*****************************************************************
     PERFORM 505-ISSUE-CSRAW THRU 500-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "INITIAL CSRAW PROBLEM - PROGRAM WILL TERMINATE"
         PERFORM 999-END-PROGRAM THRU 999-EXIT
         GO TO 099-TERMINATE.
*****************************************************************
*   CHECK THE WHAT RECEIVED VALUE TO MAKE SURE WE HAVE BEEN     *
*   PUT INTO A SEND STATE.                                      *
*****************************************************************
     IF SEND-REQUEST-RECVD
         NEXT SENTENCE
     ELSE
         DISPLAY "INITIAL WHAT RECEIVED IS UNEXPECTED"
         DISPLAY "WHAT RECEIVED VALUE IS: " WHAT-RECEIVED
         DISPLAY "PROGRAM WILL TERMINATE"
         PERFORM 999-END-PROGRAM THRU 999-EXIT
         GO TO 099-TERMINATE.
*****************************************************************
*   AT THIS POINT THE CONVERSATION HAS BEEN ATTACHED AND WE     *
*   ARE IN A SEND STATE THAT ALLOWS US TO PROCEED WITH THE      *
*   REMAINDER OF THE PROGRAM PROCESS.                           *
*****************************************************************
 005-CONTINUE.
     PERFORM 200-GET-RECORD THRU 200-EXIT.
     IF END-PROGRAM
         DISPLAY "END OF RUN REQUESTED - PROGRAM WILL END"
         PERFORM 999-END-PROGRAM THRU 999-EXIT
         GO TO 099-TERMINATE
     ELSE
         NEXT SENTENCE.
     MOVE HIGH-VALUES TO DATA-TO-HOST.
     MOVE SPACES TO DATA-FROM-HOST
                    RECEIVE-RECORD.
     MOVE DATA-FROM-TERMINAL TO DATA-TO-HOST.
     MOVE 0 TO INDX1
               NO-INPUT-SW
               RECORD-BUILT-SW
               LOGICAL-REC-LENGTH
               DATA-BUFFER-LENGTH.
     PERFORM 300-BUILD-LOGICAL THRU 300-EXIT VARYING INDX1 FROM 1
             BY 1 UNTIL RECORD-BUILT.
     IF NO-INPUT-DATA
         DISPLAY "NO DATA WAS ENTERED FROM THE TERMINAL"
         DISPLAY "PLEASE KEY SOME DATA BEFORE HITTING ENTER KEY"
         GO TO 005-CONTINUE
```

Figure C-4 (cont). Sample COBOL Program for LU Type 6.2 for Host-Initiated Session

```
        ELSE
            NEXT SENTENCE.
        MOVE DATA-TO-HOST TO LOGICAL-RECORD.
        PERFORM 400-SEND-RECORD THRU 400-EXIT.
        IF OK-TO-CONTINUE
            NEXT SENTENCE
        ELSE
            PERFORM 999-END-PROGRAM THRU 999-EXIT
            GO TO 099-TERMINATE.
    010-DO-RECEIVE.
        PERFORM 500-RECEIVE-INFO THRU 500-EXIT.
    0101-NEXT-RECEIVE.
        IF OK-TO-CONTINUE
            NEXT SENTENCE
        ELSE
            PERFORM 999-END-PROGRAM THRU 999-EXIT
            GO TO 099-TERMINATE.
    015-CHECK-WHAT-RECEIVED.
        IF DATA-RECEIVED
            PERFORM 600-COMPARE-INOUT THRU 600-EXIT
            PERFORM 505-ISSUE-CSRAW THRU 500-EXIT
            GO TO 0101-NEXT-RECEIVE
        ELSE
            IF DEALLOCATE-CONFIRM
                PERFORM 700-ISSUE-CONFIRMED THRU 700-EXIT
                GO TO 099-TERMINATE
            ELSE
                IF CONFIRM-ON-HOST-PTOR
                    GO TO 020-CHECK-COMPARE
            ELSE
                NEXT SENTENCE.
        DISPLAY "UNEXPECTED WHAT RECEIVED FIELD".
        DISPLAY "WHAT RECEIVED IS: " WHAT-RECEIVED.
        PERFORM 705-SEND-ERROR THRU 705-EXIT.
        IF OK-TO-CONTINUE
            NEXT SENTENCE
        ELSE
            PERFORM 999-END-PROGRAM THRU 999-EXIT
            GO TO 099-TERMINATE.
        GO TO 0101-NEXT-RECEIVE.
    020-CHECK-COMPARE.
        IF COMPARE-OK
            PERFORM 700-ISSUE-CONFIRMED THRU 700-EXIT
        ELSE
            PERFORM 705-SEND-ERROR THRU 705-EXIT.
        IF OK-TO-CONTINUE
            NEXT SENTENCE
        ELSE
            PERFORM 999-END-PROGRAM THRU 999-EXIT
            GO TO 099-TERMINATE.
        GO TO 005-CONTINUE.
    099-TERMINATE.
        STOP RUN.
```

Figure C-4 (cont). Sample COBOL Program for LU Type 6.2 for Host-Initiated Session

```
*
*
*
 100-ATTACH-CONVERSATION.
*****************************************************************
*   THIS ROUTINE WILL ISSUE A CSATCH TO ATTEMPT TO ATTACH AN    *
*   LU 6.2 CONVERSATION WITH THE HOST CICS TRANSACTION ADL6.    *
*   SINCE THE CONVERSATION WAS ALLOCATED BY THE HOST TRANS-     *
*   ACTION WE MUST DO AN ATTACH COMMAND SO AIF CAN PUT US IN    *
*   CONVERSTAION WITH THE HOST TRANSACTION.                     *
*****************************************************************
*****DISPLAY "GOING TO DO CSATCH NOW".
     CALL "CSATCH" USING SNA-WORK-AREA
                         AIF-NODE-NAME
                         REMOTE-LU-NAME
                         CONVERSATION-ID
                         STD-NAME
                         RETURN-CONTROL
                         SYNC-LEVEL
                         RETURN-CODE-VALUES
                         SOPR-STOP-TIME
                         RECEIVED-SENSE-DATA
                         OUTPUT-CONTROL-WORD.
*****************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                               *
*****************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "ERRORS FROM CSATCH REQUEST - CHECK RETURN CODES"
         DISPLAY "MAJOR RETURN CODES ARE: " MAJOR-RETURN-CODES
         DISPLAY "PROGRAM WILL END - NO CONVERSATION"
         GO TO 100-EXIT.
*****************************************************************
     DISPLAY "CONVERSATION HAS BEEN ATTACHED - ID IS: "
             CONVERSATION-ID.
 100-EXIT.
     EXIT.
*EJECT
 200-GET-RECORD.
     MOVE HIGH-VALUES TO DATA-FROM-TERMINAL.
*****************************************************************
*   NOW GET SOME DATA FROM THE TERMINAL OPERATOR TO SEND TO THE *
*   HOST REMOTE PROGRAM.                                        *
*****************************************************************
     DISPLAY ENTER-MESSAGE.
     ACCEPT DATA-FROM-TERMINAL.
 200-EXIT.
     EXIT.
*SKIP3
 300-BUILD-LOGICAL.
```

Figure C-4 (cont). Sample COBOL Program for LU Type 6.2 for Host-Initiated Session

```
******************************************************************
*   NOW BUILD THE LOGICAL RECORD THAT WILL BE SENT TO THE HOST   *
*   BY CALCULATING THE LENGTH OF THE DATA RECEIVED THEN CONVERT *
*   THE DATA TO EBCDIC.                                          *
******************************************************************
     IF CHECK-INPUT-FIELD (INDX1) IS EQUAL TO HIGH-VALUES
         MOVE 1 TO RECORD-BUILT-SW
         COMPUTE CALC-LENGTH = INDX1 - 1
         IF CALC-LENGTH IS EQUAL TO ZEROS OR
            CALC-LENGTH IS LESS THAN ZEROS
             MOVE 1 TO NO-INPUT-SW
             GO TO 300-EXIT
         ELSE
             ADD 2 TO DATA-BUFFER-LENGTH
                      LOGICAL-REC-LENGTH
             MOVE CALC-LENGTH TO CONVERSION-LENGTH
             MOVE LOGICAL-REC-LENGTH TO TEMP-LENGTH
             PERFORM 305-CONVERT-RECORD THRU 305-EXIT
     ELSE
         ADD 1 TO DATA-BUFFER-LENGTH
                  LOGICAL-REC-LENGTH.
 300-EXIT.
     EXIT.
*SKIP3
 305-CONVERT-RECORD.
******************************************************************
*   THIS ROUTINE WILL ISSUE THE CSACEB CALL TO CONVERT THE DATA *
*   FROM THE TERMINAL TO EBCDIC BEFORE IT IS SENT TO THE HOST.  *
******************************************************************
     CALL "CSACEB" USING SNA-WORK-AREA
                         DATA-TO-HOST
                         TRANSLATE-FROM-POSITION
                         DATA-TO-HOST
                         TRANSLATE-TO-POSITION
                         CONVERSION-LENGTH.
     IF CALL-FORMAT-ERROR
         DISPLAY "COBOL ERROR IN CSACEB CALL - CHECK RETURN CODES"
         DISPLAY "COBOL RETURN CODE IS: " MINOR-RETURN-CODE
         DISPLAY "PROGRAM WILL TERMINATE"
         MOVE 1 TO ERROR-IN-CALL-SW
     ELSE
         NEXT SENTENCE.
 305-EXIT.
     EXIT.
*EJECT
 400-SEND-RECORD.
```

Figure C-4 (cont). Sample COBOL Program for LU Type 6.2 for Host-Initiated Session

```
*****************************************************************
*   THIS ROUTINE WILL ISSUE THE CSSDAT CALL TO SEND THE DATA    *
*   TO AIF.  AIF WILL NOT SEND THE DATA TO THE HOST UNTIL WE    *
*   ISSUE ANOTHER CALL TO FORCE A FLUSH OF THE BUFFERS. THIS    *
*   WILL BE DONE IN THE NEXT ROUTINE.                           *
*****************************************************************
*****DISPLAY "GOING TO DO CSSDAT NOW".
     CALL "CSSDAT" USING SNA-WORK-AREA
                         LOGICAL-DATA-BUFFER
                         DATA-BUFFER-LENGTH.
*****************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                               *
*****************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "ERRORS FROM CSSDAT REQUEST - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL TERMINATE".
 400-EXIT.
     EXIT.
*EJECT
 500-RECEIVE-INFO.
****************************************************************
*   THIS ROUTINE WILL ISSUE A NUMBER OF AIF VERBS.  FIRST IT   *
*   WILL DO A CSPTOR WHICH WILL CAUSE AIF TO FLUSH THE SEND    *
*   BUFFER SENDING THE DATA FROM THE CSSDAT CALL AND A SEND    *
*   INDICATOR TO THE HOST PROGRAM TO TELL THAT PROGRAM IT CAN  *
*   TURN AROUND AND SEND TO THIS PROGRAM.                      *
*   AFTER THE CSPTOR, THE PROGRAM WILL ISSUE A CSRAW TO WAIT   *
*   FOR THE DATA TO COME BACK FROM THE HOST AND RECEIVE IT.    *
****************************************************************
****************************************************************
*   THE TYPE OF PREPARE TO RECEIVE IS A FLUSH (TYPE-SWITCH=F)  *
*   THE TYPE OF LOCKS IS LONG (CONFIRMATION-LOCKS=L)           *
****************************************************************
*****DISPLAY "GOING TO DO CSPTOR TYPE F NOW".
     MOVE "F" TO TYPE-SWITCH.
     CALL "CSPTOR" USING SNA-WORK-AREA
                         TYPE-SWITCH
                         CONFIRMATION-LOCKS.
*****************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                               *
*****************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "ERRORS FROM CSPTOR - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL TERMINATE"
         GO TO 500-EXIT.
 505-ISSUE-CSRAW.
```

Figure C-4 (cont). Sample COBOL Program for LU Type 6.2 for Host-Initiated Session

```
*****************************************************************
*   ISSUE THE CSRAW TO CAUSE THE PROGRAM TO WAIT FOR A RECEIVE  *
*   AND RECEIVE THE DATA COMING BACK FROM THE HOST TRANSACTION. *
*   THE TYPE OF RECEIVE IS A BUFFER (TYPE-OF-RECEIVE=B) SO      *
*   AIF WILL PASS AN ENTIRE BUFFER'S WORTH OF DATA AS OPPOSED   *
*   TO A LOGICAL RECORD.  THIS ROUTINE WILL ALSO BE USED TO     *
*   RECEIVE STATUS OR STATE CHANGE INFORMATION.                 *
*****************************************************************
*****DISPLAY "GOING TO DO CSRAW NOW".
     CALL "CSRAW" USING SNA-WORK-AREA
                        RECEIVE-DATA-BUFFER
                        RECEIVE-BUFFER-SIZE
                        TYPE-OF-RECEIVE
                        RECEIVED-DATA-LENGTH.
*****************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                               *
*****************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "ERRORS FROM CSRAW - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL TERMINATE".
 500-EXIT.
     EXIT.
*EJECT
 600-COMPARE-INOUT.
****************************************************************
*  THIS ROUTINE WILL COMPARE THE DATA RECEIVED FROM THE HOST   *
*  WITH THE DATA ORIGINALLY SENT.  IF THEY ARE NOT THE SAME    *
*  A SWITCH IS SET AND ERROR MESSAGES ARE DISPLAYED.           *
****************************************************************
     DISPLAY "GOING TO COMPARE RECORD SENT TO RECEIVED NOW".
     MOVE RECEIVE-RECORD TO DATA-FROM-HOST.
     IF DATA-BUFFER-LENGTH IS EQUAL TO RECEIVED-DATA-LENGTH
         NEXT SENTENCE
     ELSE
         DISPLAY "BUFFER LENGTHS ARE NOT THE SAME"
         DISPLAY "SEND LENGTH: " DATA-BUFFER-LENGTH
               " RECEIVE LENGTH: " RECEIVED-DATA-LENGTH.
     IF LOGICAL-REC-LENGTH IS EQUAL TO RECEIVE-REC-LENGTH
         NEXT SENTENCE
     ELSE
         DISPLAY "LOGICAL LENGTHS ARE NOT THE SAME".
     MOVE 0 TO COMPARE-REC-SW
               NUMBER-CHARS
               NO-MORE-SW
               INDX1.
     COMPUTE RECEIVE-REC-LENGTH = RECEIVE-REC-LENGTH - 2.
     PERFORM 800-COMPARE-EACH-FIELD THRU 800-EXIT
         VARYING INDX1 FROM 1 BY 1
             UNTIL NO-MORE-TO-CHECK.
```

Figure C-4 (cont). Sample COBOL Program for LU Type 6.2 for Host-Initiated Session

```
     IF COMPARE-OK
         DISPLAY "DATA FROM HOST IS THE SAME AS DATA SENT"
     ELSE
         DISPLAY "DATA FROM HOST IS NOT THE SAME AS DATA SENT"
         DISPLAY "POSSIBLE LOGIC ERROR".
 605-CONVERT-DATA.
*****************************************************************
*  THIS ROUTINE WILL CONVERT THE RECEIVED DATA FROM EBCDIC TO   *
*  ASCII AND DISPLAY THE RECORD ON THE TERMINAL.                *
*****************************************************************
      COMPUTE CONVERSION-LENGTH = RECEIVE-REC-LENGTH.
      CALL "CSEBAC" USING SNA-WORK-AREA
                          DATA-FROM-HOST
                          TRANSLATE-FROM-POSITION
                          DATA-FROM-HOST
                          TRANSLATE-TO-POSITION
                          CONVERSION-LENGTH.
     IF CALL-FORMAT-ERROR
         DISPLAY "COBOL ERROR IN CSEBAC CALL - CHECK RETURN CODES"
         DISPLAY "COBOL RETURN CODE IS: " MINOR-RETURN-CODE
         DISPLAY "PROGRAM WILL TERMINATE"
         MOVE 1 TO ERROR-IN-CALL-SW
         GO TO 600-EXIT
     ELSE
         NEXT SENTENCE.
     DISPLAY "RECIEVED DATA IS: ".
     DISPLAY DATA-FROM-HOST.
 600-EXIT.
     EXIT.
*EJECT
 700-ISSUE-CONFIRMED.
*****************************************************************
*   THIS ROUTINE WILL ISSUE A CSCNFD CALL.  THIS WILL CAUSE AIF *
*   TO SEND A CONFIRMATION TO THE HOST TRANSACTION.             *
*****************************************************************
*****DISPLAY "GOING TO DO CSCNFD NOW".
     CALL "CSCNFD" USING SNA-WORK-AREA.
*****************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                               *
*****************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "ERRORS FROM CSCNFD - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL TERMINATE".
 700-EXIT.
     EXIT.
*SKIP3
 705-SEND-ERROR.
```

Figure C-4 (cont). Sample COBOL Program for LU Type 6.2 for Host-Initiated Session

```
*******************************************************************
*   THIS ROUTINE WILL ISSUE A CSSERR CALL TO NOTIFY THE HOST     *
*   TRANSACTION OF AN ERROR IN PROCESSING.  THE TYPE OF ERROR    *
*   IS PROG (TYPE-SWITCH=P).  THE PROGRAM WILL NOT REQUEST THE   *
*   LOGGING OF DATA (LOG-SWITCH=N).                               *
*******************************************************************
*****DISPLAY "GOING TO DO CSSERR TYPE P NOW".
     MOVE "P" TO TYPE-SWITCH.
     CALL "CSSERR" USING SNA-WORK-AREA
                         TYPE-SWITCH
                         LOG-SWITCH
                         LOG-DATA.
*******************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                                 *
*******************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         NEXT SENTENCE
     ELSE
         DISPLAY "ERRORS FROM CSSERR - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL TERMINATE".
 705-EXIT.
     EXIT.
*EJECT
 800-COMPARE-EACH-FIELD.
     IF CHECK-INPUT-FIELD (INDX1) IS EQUAL TO DATA-FIELD (INDX1)
         ADD 1 TO NUMBER-CHARS
     ELSE
         ADD 1 TO NUMBER-CHARS
         DISPLAY "CHARACTER NOT THE SAME IS: "
                 NUMBER-CHARS
         MOVE 1 TO COMPARE-REC-SW.
     IF INDX1 IS EQUAL TO RECEIVE-REC-LENGTH
         MOVE 1 TO NO-MORE-SW
         DISPLAY "END OF COMPARE"
     ELSE
         NEXT SENTENCE.
 800-EXIT.
     EXIT.
*SKIP3
 900-CHECK-RETURN.
*******************************************************************
*   THIS ROUTINE WILL CHECK THE RETURN CODES FROM THE VARIOUS    *
*   AIF VERB CALLS.  A SWITCH IS SET TO INDICATE WHETHER THE     *
*   CALL WAS OK OR NOT.  WHEN THE RETURN CODES ARE NOT OK THEY   *
*   WIL BE DISPLAYED ON THE TERMINAL.                             *
*******************************************************************
     MOVE 0 TO ERROR-IN-CALL-SW.
     IF CALL-FORMAT-ERROR
         MOVE 1 TO ERROR-IN-CALL-SW
         DISPLAY "COBOL FORMAT ERROR IN CALL - RETURN CODE IS: "
                 MINOR-RETURN-CODE
         DISPLAY "NEXT MESSAGE INDICATES CALL IN ERROR"
```

Figure C-4 (cont). Sample COBOL Program for LU Type 6.2 for Host-Initiated Session

```
            GO TO 900-EXIT
        ELSE
            NEXT SENTENCE.
        IF SOPR-ISSUED-STOP
            DISPLAY "SOPR OPERATOR HAS ISSUED A STOP COMMAND"
            DISPLAY "STOP TIME IS: " SOPR-STOP-TIME
        ELSE
            NEXT SENTENCE.
        IF ABEND-RECEIVED
            DISPLAY "AN ABEND/DEALLOCATE HAS BEEN RECEIVED"
            DISPLAY "SESSION WILL BE TERMINATED"
            MOVE 1 TO ERROR-IN-CALL-SW
        ELSE
            NEXT SENTENCE.
        IF CALL-WAS-COMPLETED AND
           MINOR-RETURN-CODE IS EQUAL TO ZEROS
            GO TO 900-EXIT
        ELSE
            NEXT SENTENCE.
        DISPLAY "VERB CALL CONTAINS ERRORS - RETURN CODE IS: "
                MINOR-RETURN-CODE.
        MOVE 1 TO ERROR-IN-CALL-SW.
    900-EXIT.
        EXIT.
   *SKIP3
    999-END-PROGRAM.
   *****************************************************************
   *   THIS ROUTINE WILL BE USED TO ISSUE A CSDEAL CALL ENDING THE *
   *   CONVERSATION WITH THE HOST TRANSACTION.  THE TYPE OF DE-    *
   *   ALLOCATE IS FLUSH (TYPE-SWITCH=F) ON THE FIRST ATTEMPT IF   *
   *   THAT HAS AN ERROR THEN AND ABEND PROG TYPE WILL BE ISSUED   *
   *   (TYPE-SWITCH=P).  THE PROGRAM WILL NOT REQUEST THE LOGGING  *
   *   OF ERROR DATA (LOG-SWITCH=N).                               *
   *****************************************************************
   *****DISPLAY "GOING TO TRY A NORMAL DEALLOCATE NOW".
        MOVE "F" TO TYPE-SWITCH.
        CALL "CSDEAL" USING SNA-WORK-AREA
                            TYPE-SWITCH
                            LOG-SWITCH
                            LOG-DATA.
```

Figure C-4 (cont). Sample COBOL Program for LU Type 6.2 for Host-Initiated Session

```
*****************************************************************
*   CHECK THE RETURN CODE VALUES NEXT TO MAKE SURE THE CALL HAS *
*   COMPLETED WITHOUT ANY ERRORS.                               *
*****************************************************************
     PERFORM 900-CHECK-RETURN THRU 900-EXIT.
     IF OK-TO-CONTINUE
         DISPLAY "CONVERSATION HAS BEEN DEALLOCATED"
         GO TO 999-EXIT
     ELSE
         DISPLAY "ERRORS FROM CSDEAL F - CHECK RETURN CODES"
         DISPLAY "PROGRAM WILL ISSUE DEALLOCATE/ABEND".
     MOVE "P" TO TYPE-SWITCH.
     CALL "CSDEAL" USING SNA-WORK-AREA
                         TYPE-SWITCH
                         LOG-SWITCH
                         LOG-DATA.
 999-EXIT.
     EXIT.
```

Figure C-4 (cont). Sample COBOL Program for LU Type 6.2 for Host-Initiated Session

Appendix D
SESSION CALL RETURN CODES

The following pages show the unique return codes that are returned by AIF after the execution of each call or verb. As described in the Assembly language sections, bits 0 through 4 of the return code have special meaning. The tables in this section present the return codes both after these bits have been masked out.

The following tables are included in this appendix:

- Table D-1 provides the AIF session call return codes
- Table D-2 provides the individual return codes
- Table D-3 provides the COBOL RETURNS fields
- Table D-4 provides the general COBOL RETURN-B codes
- Table D-6 provides the interrupt types
- Table D-7 provides the attribute types.

Table D-1. AIF Session Call Return Codes

Mask	Label	Meaning
8000	RCABRT	SESSION ABORTED, CHECK SC_ABT FIELD FOR REASON
*4000	RCSTOP	SOPR COMMAND RECEIVED
2000	RCRINT	INTERRUPT RECEIVED
1000	RCSCNL	SERVICE REQUEST NOT PROCESSED OR CANCELLED
0800	RCSCMP	SERVICE REQUEST COMPLETED
07FF	RCMASK	MASK FOR INDIVIDUAL RETURN CODES (SEE TABLE D-2)
*Return codes marked with an asterisk can be received after any session call.		

Table D-2. Individual Return Codes

COBOL RETURN-B	Assembly Language	Macro Label	Meaning
0000	0000	RMNOER	NO ERROR
0001	0001	RMPTSN	PERMISSION TO SEND
0002	0002	RMDRNR	DATA RECEIVED BUT NO READ
0003	0003	RMRNEG	NEGATIVE RESPONSE RECEIVED FROM HOST
0004	0004	RMNBIF	BIND NEGOTIATION FAILED
0005	0005	RMLUAT	LU ATTACHED BY REMOTE
0016	0010	RMIMPS	IMPROPER STATE
0018	0012	RMIRHI	INVALID INPUT CONTROL INDICATORS
0019	0013	RMRB2S	RECEIVE BUFFER TOO SMALL
0020	0014	RMIINT	INVALID INTERRUPT TYPE
0021	0015	RMICOD	INVALID STATUS VALUE OR USER CODE
0023	0017	RMNOUT	NO OUTSTANDING ASYNCHRONOUS ORDER
0025	0019	RMACTO	ACCEPT TIMED OUT
0032	0020	RMRSRF	RESTART NOT POSSIBLE
*0048	0030	RMSYSE	SYSTEM ERROR
*0049	0031	RMRNAV	RESOURCE NOT AVAILABLE
0050	0032	RMDTCL	SEND/RECEIVE REJECT, DATA TRAFFIC INACTIVE/RESET
0064	0040	RMINOD	INVALID NODE NAME
*0065	0041	RMINVS	INVALID SESSION ID
*0066	0042	RMASYN	ASYNCHRONOUS SERVICE REQUEST OUTSTANDING
*0067	0043	RMIVSR	INVALID SERVICE REQUEST (OPERATION CODE)
0068	0044	RMLNER	DATA LENGTH ERROR ON SEND
0069	0045	RMIVFC	INVALID FUNCTION CODE ON $SWANY/CSWANY
0070	0046	RMIMCS	IMPROPER CALLING SEQUENCE
0150	0096	RMNNAC	NODE NOT YET ACTIVE
0151	0097	RMNLAC	NO ACTIVE LU FOR SESSION
0152	0098	RMNOAV	NO LU AVAILABLE FOR SESSION
0153	0099	RMISTD	INVALID STD NAME
0154	009A	RMILUT	INVALID LU TYPE IN STD
0155	009B	RMNOAT	NO LU ATTACHED FOR $SACPT
*0256	0100	RMUNBI	SESSION UNBOUND BY HOST UNEXPECTEDLY
*0257	0101	RMSSHU	SESSION SHUTDOWN BY HOST ORDERLY
*0258	0102	RMURTO	YOU ARE TIMED OUT BY SOPR COMMAND
*0259	0103	RMPGER	SESSION ABORT DUE TO UNRECOVERABLE PROGRAM ERROR

Table D-2 (cont). Individual Return Codes

COBOL RETURN-B	Assembly Language	Macro Label	Meaning
*0784	0301	RMADLU	ACTLU/DACTLU RECEIVED
*1809	0711	RMLKFL	LINK FAILURE
*1810	0712	RMADPU	ACTPU/DACTPU RECEIVED
*1811	0713	RMACSA	$A (SOPR) 'ABORT' AIF NODE
*1812	0714	RMSABT	$S ABORT AIF GROUP

*Return codes marked with an asterisk can be received after any session call.

Table D-3. COBOL Session Call RETURNS fields.

Fields	Value	Meaning
SESSION-ABORT	Y	LU-LU session or node has been aborted
STOP-RCVD	Y	SOPR STOP command received.
INTRPT-RCVD	Y	Interrupt received. See INTERRUPT output parameter
SERV-REQ-CANCLD	Y	This request has been cancelled. The application must issue it again if necessary.
SERV-REQ-COMPLETE	Y	This request has been completed.
COBOL-INT-ERROR	Y	Error in using COBOL interface to AIF. See RETURN-B for return code.

Table D-4. General COBOL RETURN-B Values

Code	Meaning
XX01	Unrecognized parameter
XX02	Parameter must be 1 byte long
XX03	Parameter must be 5 bytes long
XX04	Default not acceptable
XX05	Node name error
XX06	Remote LU name error
XX07	Not session-ID
XX08	Unknown interrupt type
XX09	Nondecimal digit
XX10	Nonhexadecimal digit
XX11	Error in conversion

Table D-5. Interrupt-Type Correspondence

COBOL Value	Hex Value	Label	Comment
01	40C0	SHUTD	Shutdown
02	40C1	SHUTC	Shutdown complete
03	40C2	RSHUTD	Request shutdown
04	40C9	SIGNAL	Signal
05	4080	QEC	Quiesce at end of chain
06	0081	QCOMPL	Quiesce complete
07	4082	RELQ	Release quiesce
08	4071	SBI	Stop bracket initialization
09	0070	BIS	Bracket initiation stopped
10	0083	CANCEL	Cancel
11	0084	CHASE	Chase
12	00C8	BID	Bid
13	0004	LUSTAT	LU status
14	0005	RTR	Ready to receive
15	8001	CLEAR	Data traffic cleared/reset by host
16	8010	ENAPRS	Enable restart for DPS 6 or DPS 6 PLUS application
17	8011	DSAPRS	Disable restart for DPS 6 or DPS 6 PLUS application
18	8012	RQRCVR	DPS 6 or DPS 6 PLUS application request for receive
19	2008	ALERT	Alert
20	200E	STAT	Statistics

Table D-6. Attribute Types

COBOL Value	Hex Value	Label	Comment
01	0001	BINDIM	Bind image attribute

Appendix E
$SSCCB TEMPLATE

This appendix contains the template for $SSCCB, the Session Call Control Block (SCCB). This template is used in creating an SCCB for your LU Type 0 application.

Table E-1. $SSCCB Template.

Offset	Label	Meaning
0000	$SSCCB	
0000	SC_SID	SESSION ID
0000	SC_SGP	SESSION GROUP NAME
0001	SC_SES	SESSION NAME
0002	SC_APS	(FOR AIF USE ONLY)
0004	SC_OPC	OPERATION CODE SC_OPC IS NORMALLY LOADED BY AN AIF MACROCALL
0005	SC_RF1	RESERVED FOR FUTURE USE (1)
0008	SC_STD	STD
0009	SC_RLN	REMOTE LU NAME IN ASCII
000D	SC_NOD	SNAPI NODE NAME IN ASCII
0011	SC_TPN	TRANSACTION PROGRAM NAME IN ASCII
0019	SCOUPT	SC OUTPUT PARAMETER AREA
0019	SC_OCT	SC OUTPUT CONTROL WORD
001A	SC_PHB	FOR AIF USE ONLY
001A	SC_ADL	ACTUAL DATA LENGTH RECEIVED
001C	SC_INT	RECEIVED INTERRUPT TYPE
001D	SC_SQN	SEQUENCE NUMBER OF LAST SENT RU
001E	SC_RSQ	SEQUENCE NUMBER OF LAST RECEIVED RU
001F	SC_ESD	ERROR CODE OR SENSE DATA RECEIVED
001F	SC_MRU	MAXIMUM RU SIZE
0021	SC_RCD	RETURN CODE OF SESSION CALL
0022	SC_ABT	SESSION ABORT REASON WHEN RCABRT SET IN SC_RCD - REFER TO MACRO $SAIRC FOR DEFINITION
0023	SC_TIM	TIME OF SESSION TERMINATION WHEN RCSTOP SET IN SC_RCD OR TIME TO RELEASE ABNORMALLY TERMINATED SESSION
0026	SC_RF2	RESERVED FOR FUTURE USE (2)
0010	SCOUPS	SIZE OF SCCB OUTPUT AREA
0029	SCINPT	SESSION CALL INPUT PARAMETER AREA
0029	SC_ICT	SESSION CALL INPUT CONTROL WORD
002A	SC_BUF	-> SEND/RCV DATA BUFFER
002C	SC_DLG	SEND/RECIEVE DATA BUFFER LENGTH
002D	SC_SSD	SENSE DATA FOR SENDING INTERRUPT, -RSP OR ABNORMAL TERMINATION
002D	SC_MRS	SEND_SQN FOR MESSAGE RESYNCHRONIZATION
002E	SC_MRR	RCV_SQN FOR MESSAGE RESYNCHRONIZATION
002F	SC_SIN	SEND INTERRUPT TYPE
0030	SC_RF3	RESERVED FOR FUTURE USE (3)
000A	SCINPS	SIZE OF SCCB INPUT AREA
0033	SC_REG	SAVE REGISTER SPACE
0041	SC_SIZ	SCCB SIZE

Table E-1 (cont.). $SSCCB Template

Offset	Label	Meaning
OPERATION CODE (SC_OPC)		
4000	ASCINI	$SINIT
4001	ASCTER	$STERM
0002	ASCSND	$SSEND
0003	ASCRCV	$SRECV
0004	ASCSIN	$SSI
0005	ASCRIN	$SRI
0006	ASCASR	$SCASR
000A	ASCWAN	$SWANY
000B	ASCTST	$STEST
SESSION CALL INPUT CONTROL WORD (SC_ICT)		
0800	SCRTNS	RETURN CONTROL WHEN SESSION CALL COMPLETED (SYNC.)
0400	SCRHBI	DATA START AT RIGHT BYTE OF BUFFER
0200	SCRMSG	USED FOR $SRECV TO WAIT FOR WHOLE MESSAGE
0100	SCRSTR	RESTART, USED ONLY FOR $SINIT AFTER SESSION HAS BEEN ABNORMALLY TERMINATED
8000	SCSWRP	SEND WITH REPLY (SET CD IN RH)
4000	SCSRQD	SEND WITH DEFINITE RESP REQUIRED
2000	SCSLST	SEND LAST MESSAGE (SET EB IN RH)
1000	SCSFMH	SEND WITH FMH IN DATA RU
0080	SCSRSP	SEND +RSP
0040	SCSNEG	SEND -RSP
0020	SCSMNC	MESSAGE (CHAIN) NOT COMPLETE
0010	SCACPT	1=ACCEPT, $SACPT WITH SC_OPC THIS BIT SHOULD BE 0 IF $SIN
0008	SCGTAT	1=GET_ATTRIBUTE, 0=RECEIVE_D
0004	SCPOLL	1=POLL, $SPOLL WITH SC_OPC THIS BIT SHOULD BE 0 IF $SIN
0001	SCATRM	ABNORMAL TERMINATION
SESSION CALL OUTPUT CONTROL WORD (SC_OCT)		
0008	SCRWRP	REPLY REQUESTED (CD RECEIVED IN RH)
0001	SCRRQD	DEFINITE RESPONSE REQUESTED
2000	SCRLST	LAST MESSAGE RECEIVED (EB RECEIVED IN RH)
1000	SCRFMH	FMH IN RECEIVED DATA
0400	SCRBOM	BEGINNING OF MESSAGE RECEIVED (BC IN RH)
0200	SCREOM	END OF MESSAGE RECEIVED (EC IN RH)

Table E-1 (cont.). $SSCCB Template.

Offset	Label	Meaning
BITS USED FOR SESSION RESTART		
0008	SCRSTS	STATION RECEIVED FOR MSG_RESYNC, SET SQN TO SC_SQN, SC_RSQ
0004	SCL6RX	DPS 6 OR DPS 6 PLUS APPLICATION RETRANSMIT REQUIRED
0002	SCHORX	HOST APPLICATION RETRANSMIT REQUIRED, READY TO RECEIVE
INTERRUPT TYPE THERE ARE 3 CATEGORIES OF INTERRUPT: 1. EXPEDITED OR NORMAL FLOW DFC COMMAND 2. CNM DATA 3. INFORMATION PASSED TO OR FROM APPLICATION PROGRAM		
FF00	INTCAT	CATEGORY
8000	APPINF	APPLICATION INFORMATION
4000	EXPDFC	EXPEDITED DFC COMMAND
2000	INTBUF	INTERRUPT WITH BUFFER FOR DATA
00FF	INTCOD	INTERRUPT TYPE CODE
DATA FLOW CONTROL COMMANDS		
40C0	SHUTD	SHUTDOWN
40C1	SHUTC	SHUTDOWN COMPLETE
40C2	RSHUTD	REQUEST SHUTDOWN
40C9	SIGNAL	SIGNAL
4080	QEC	QUIESCE AT END OF CHAIN
0081	QCOMPL	QUIESCE COMPLETE
4082	RELQ	RELEASE QUIESCE
4071	SBI	STOP BRACKET INITIALIZATION
0070	BIS	BRACKET INITIALIZATION STOPPED
0083	CANCEL	CANCEL
0084	CHASE	CHASE
00C8	BID	BID
0004	LUSTAT	LU STATUS
0005	RTR	READY TO RECEIVE
INFORMATION PASSED TO OR FROM APPLICATION		
8001	CLEAR	DATA TRAFFIC CLEARED/RESET BY HOST
8010	ENAPRS	ENABLE RESTART FOR DPS 6 OR DPS 6 PLUS APPLICATION
8011	DSAPRS	DISABLE RESTART FOR DPS 6 OR DPS 6 PLUS APPLICATION
8012	RQRCVR	DPS 6 OR DPS 6 PLUS APPLICATION REQUEST FOR RECOVERY

Table E-1 (cont.). $SSCCB Template.

Offset	Label	Meaning
CNM DATA		
2008	ALERT	ALERT
200E	STATIC	STATISTICS OF REQMS (TYPE 4) SC_ESD=0; NO PARAMETER IN REQMS TO PASS TO APPLICATION
0001	SD0001	SC_ESD=1: REQMS RECEIVED IN RECEIVE BUFFER TO PASS PARAMETER SC_ESD=2: PARAMETER IN REQMS NEEDS TO BE PASSED
GET ATTRIBUTE TYPE		
0001	BINDM	BIND IMAGE STARTING FROM BYTE1

Appendix F *CONVERSATION VERB RETURN CODES*

The following pages show the unique return codes that are returned by AIF after the execution of each call or verb. As described in the Assembly language sections, bits 0 through 4 of the return code have special meaning. The tables in these section present the return codes both before and after these bits have been masked out.

The following tables are included in this appendix:

- Table F-1 provides the general return codes for the conversation verb
- Table F-2 provides the individual return codes
- Table F-3 provides sense data
- Table F-4 provides COBOL RETURN-A fields
- Table F-5 provides general COBOL RETURN-B codes.

Table F-1. Individual Return Codes

Hex Value	Label	Meaning
8000	VRABND	CONVERSATION ABEND/DEALLOCATED
*4000	VRSTOP	SOPR STOP COMMAND RECEIVED; CHECK VP_TIM FOR TIME
2000	VRRINT	RESERVED WHEN USING VERB
1000	VRSCNL	SERV. REQ. NOT PROCESSED OR CANCELLED
0800	VRSCMP	SERVICE REQUEST COMPLETED
07FF	VRMASK	MASK FOR INDIVIDUAL RETURN CODES (SEE TABLE F-2)

*Return codes noted by an asterisk can be received after the execution of any verb.

Table F-2. Individual Return Codes

COBOL RETURN-B	Assembly Language	Macro Label	Meaning
0000	0000	VROKAY	O.K. (NO ERROR)
0001	0001	VRUNSU	UNSUCCESSFUL
0002	0002	VRPENT	PROG_ERROR_NO_TRUNC
0003	0003	VRPETR	PROG_ERROR_TRUNC
0004	0004	VRSEPR	PROG_ERROR_PURGING
0005	0005	VRSENT	SVC_ERROR_NO_TRUNC
0006	0006	VRSETR	SVC_ERROR_TRUNC
0007	0007	VRSEPR	SVC_ERROR_PURGING
0008	0008	VRIHLN	INVALID HOST LU NAME
0009	0009	VRHLNA	HOST LU NOT AVAILABLE
0016	0010	VRNSND	CONV. NOT IN SEND STATE
0017	0011	VRNSDF	CONV. NOT IN SEND OR DEFER STATE
0018	0012	VRNCNF	CONV. NOT IN CONFIRM STATE
0019	0013	VRRB2S	RECEIVE BUFFER TOO SMALL
0020	0014	VRNSOR	CONV. NOT IN SEND OR RECEIVE STATE
0021	0015	VRNSCS	CONV. NOT IN RECEIVE OR CONFIRM STATE
0022	0016	VRNRCV	CONV. NOT IN RECEIVE STATE
0023	0017	VRNSRC	CONV. NOT IN SEND, RECEIVE OR CONFIRM STATE
0024	0018	VRLRNF	LOGICAL RECORD NOT FINISHED YET
0025	0019	VRCSCD	CONV. IN CONFIRM_SEND OR CONFIRM_DEALLOCATE RECEIVED
0026	001A	VRPDEA	CONV. IN PEND_DEALLOCATE STATE
*0048	0030	VRSYSE	SYSTEM ERROR
*0049	0031	VRRNAV	RESOURCE NOT AVAILABLE
0064	0040	VRINOD	INVALID NODE NAME
*0065	0041	VRIRID	INVALID RESOURCE ID
0066	0042	VRITPN	INVALID TPN (LENGTH OF TPN = 0)
*0067	0043	VRIVSR	INVALID SERVICE REQ.(OPERATION CODE)
0068	0044	VRLNER	DATA LENGTH ERROR ON SEND_DATA
*0069	0045	VRIVFC	INVALID FUNCTION CODE ON MCL 2319
*0070	0046	VRIMCS	IMPROPER CALLING SEQUENCE
*0071	0047	VRVBNS	VERB_NOT_SUPPORTED
*0072	0048	VRSRMU	ASR (VERB/SC) USAGE MIXED
0073	0049	VRSLNS	SYNC. LEVEL NOT SUPPORTED BY LU
0074	004A	VRIVLL	INVALID LOGICAL RECORD LENGTH
0075	004B	VRIRTC	INVALID RETURN_CONTROL FOR ALLOCATE
0076	004C	VRITYP	INVALID TYPE SPECIFIED

Table F-2 (cont). Individual Return Codes

COBOL RETURN-B	Assembly Language	Macro Label	Meaning
0150	0096	VRNNAC	NODE NOT YET ACTIVE
0151	0097	VRNLAC	NO ACTIVE LU FOR SESSION
0152	0098	VRNOAV	NO LU AVAILABLE FOR SESSION
0153	0099	VRISTD	INVALID STD NAME
0154	009A	VRILUT	INVALID LU TYPE IN STD
0155	009B	VRNOAT	NO LU ATTATCHED BY REMOTE TP
0176	00B0	VRAETN	TPN_NOT_RECONIZED
0192	00C0	VRAEPI	PIP_NOT_ALLOWED
0193	00C1	VRAEIP	PIP_NOT_SPECIFIED_CORRECTLY
0194	00C2	VRAESI	SECURITY_NOT_VALID
0195	00C3	VRAECM	CONVERSATION_TYPE_MISMATCH
0208	00D0	VRAESP	SYNC._LEVEL_NOT_SUPPORTED_BY PROGRAM
0209	00D1	VRAERP	RECONNECT_LEVEL_NOT_SUPPORTED_BY PROGRAM
0210	00D2	VRAENR	TRANS_PRG_NOT_AVAILABLE_NO_RETRY
0211	00D3	VRAETR	TRANS_PRG_NOT_AVAILABLE_RETRY
0224	00E0	VRAEAN	ACC_NOT_VALID
*0240	00F0	VRDANM	DEALLOCATE_NORMAL
*0241	00F1	VRDAPG	DEALLOCATE_ABEND_PROGRAM
*0242	00F2	VRDASV	DEALLOCATE_ABEND_SERVICE
*0243	00F3	VRDATM	DEALLOCATE_ABEND_TIMER
*0256	0100	VRUNBI	SESSION UNBOUND BY HOST UNEXPECTEDLY
*0257	0101	VRSSHU	SESSION SHUTDOWN BY HOST ORDERLY
*0258	0102	VRURTO	YOU ARE TIMED OUT BY SOPR COMMAND
*0259	0103	VRPGER	SESSION ABORT DUE TO UNRECOVERABLE PROTOCOL ERROR
*0784	0310	VRADLU	ACTLU/DACTLU RECEIVED
*1809	0711	VRLKFL	LINK FAILURE
*1810	0712	VRADPU	ACTPU/DACTPU RECEIVED
*1811	0713	VRACSA	$A (SOPR) 'ABORT' AIF NODE
*1812	0714	VRSABT	$S ABORT AIF GROUP

*Return codes noted by an asterisk can be received after the execution of any verb.

Table F-3 contains AIF specific sense data that is associated with certain AIF return codes. For sense codes not listed, refer to sense codes listed in the SNA6 Reference Summary or in the SNA6 Operator's Guide.

Table F-3. Sense Data

Macro Label	Hex Value	Sense Data	Meaning
VRRNAV	74C1	VR74C1	INVALID CALLER
	74C2	VR74C2	NO ASRBS AVAILABLE ON NODE
	74C6	VR74C6	EXCEEDED MAX. NO. OF SESSION GROUPS
	74C9	VR74C9	TIME OUT PASSING A REQUEST TO PU
VRIRID	74C0	VR74C0	ASRB NOT FOUND
	74C7	VR74C7	CAN'T FIND A VALID SESSION GROUP
VRIMCS	74CB	VR74CB	CAN'T PROCESS THIS CALL AT THIS TIME
	74D3	VR74D3	CALL WHEN NOT IN SESSION ERROR

Table F-4. COBOL Session Call RETURNS fields.

Fields	Value	Meaning
ABEND-DEALLOCATE	Y	The conversation has ABENDed and therefore been deallocated
STOP-RCVD	Y	SOPR STOP command received.
SERV-REQ-CANC	Y	This request has been cancelled. The application must issue it again if necessary.
SERV-REQ-COMP	Y	This request has been completed.
COBOL-ERROR	Y	Error in using COBOL interface to the AIF. See RETURN-B for return code.

Table F-5. General COBOL RETURN-B Values

Code	Meaning
XX01	Unrecognized parameter
XX02	Parameter must be 1 byte long
XX03	Parameter must be 5 bytes long
XX04	Default not acceptable
XX05	Node name error
XX06	Remote LU name error
XX07	Not session-ID
XX08	Unknown interrupt type
XX09	Nondecimal digit
XX10	Nonhexadecimal digit
XX11	Error in conversion

Appendix G
$SVPB TEMPLATE

Table G-1 contains the template for $SVPB, the Verb Parameter Block (VPB). This template is used in creating a VPB for your LU Type 6.2 application.

Table G-1. $SVPB Template

Offset	Label	Meaning
0000	$SVPB	
0000	VP_SID	SESSION ID
0000	VP_SGP	SESSION GROUP NAME
0001	VP_SES	SESSION NAME
0002	VP_APS	(FOR AIF USE ONLY)
0004	VP_OPC	OPERATION CODE
0005	VP_RF1	RESERVED FOR FUTURE USE 1
0007	VP_SLV	SYNC. LEVEL USED BY CONVERSATION 0 = NONE, 1 = CONFIRM
0008	VP_STD	STD NAME IN ASCII
0009	VP_RLN	REMOTE LU NAME IN ASCII
000D	VP_NOD	AIF NODE NAME IN ASCII
0011	VP_TPL	LENGTH OF TRANSACTION PROG NAME
8000	VBTPNT	DO NOT TRANSLATE TP NAME WHEN SET
0012	VP_TPN	TP NAME (MAX. 14 BYTES)
0019	VPOUPT	VERB OUTPUT PARAMETER AREA
0019	VP_OCT	VERB OUTPUT CONTROL WORD
001A	VP_PHB	(FOR AIF USE ONLY)
001A	VP_ADL	ACTUAL DATA LENGTH RECEIVED
001C	VP_WAR	WHAT RECEIVED
001D	VP_CST	CONVERSATION STATE
001E	VP_RFU	RFU
001F	VP_ESD	ERROR CODE OR SENSE DATA FOR SOME RETURN CODES. REFER TO MACRO $SAIVR.
0021	VP_RCD	RETURN CODE OF VERB CALL
0022	VP_CAR	CONVERSATION ABEND REASON WHEN VRBAND SET IN VP_RCD; REFER TO $SAIVR
0023	VP_TIM	TIME OF SESSION TERMINATION WHEN VRSTOP SET IN VP_RCD, INDICATING THAT THE SOPR STOP COMMAND WAS RECEIVED
0026	VP_RF2	RFU 2
0010	VPOUPS	SIZE OF VPB OUTPUT AREA
0029	VPINPT	VERB INPUT PARAMETER AREA
0029	VP_ICT	VERB INPUT CONTROL WORD
002A	VP_BUF	-> SEND/RCV/LOG DATA BUFFER
002C	VP_DLG	SEND/RCV DATA LENGTH
002D	VP_TYP	USED BY DEALLOCATE, PREPARE_TO_RECEIVE, AND SEND_ERROR VERBS TO SPECIFY TYPE
002F	VP_CTL	SEND_CONTROL_INFOMATION TYPE
0030	VP_RF3	RFU 3
000A	VPINPS	SIZE OF VPB INPUT AREA
0033	VP_REG	SAVE REG. SPACE
0041	VP_SIZ	VPB SIZE

Table G-1 (cont). $SVPB Template

Offset	Label	Meaning
OPERATION CODE		
C000	VBALLO	$SALLO, ALLOCATE
C001	VBCVDA	USED BY AIF ONLY
8002	VBSNDA	$SSDTA, SEND_DATA
8003	VBRANW	$SRAW, RECEIVE_AND_WAIT
8004	VBSCTL	USED BY $SFLSH, $SCONF, $SCNFD, $SSERR, $SDEAL, $SPONR, $SPTOR, $SRTOS, WITH TYPE SET IN VP_CTL
VERB CALL INPUT CONTROL WORD (VP_ICT)		
0800	VBRTNS	1=SYNC. PROC. (VERB ALWAYS SYNC.)
0400	VBRHBI	DATA START AT RT. BYTE OF BUFF.
0200	VBFILL	FILL FOR $SRAW AND $SPONR VERB 0 = BUFFER, 1 = LL
0100	VBRSTR	RESERVED WHEN USING VERB (0 ALWAYS)
8000	VBSWRP	SEND WITH REPLY (SET CD IN RH)
2000	VBSLST	SEND LAST MSG (SET EB IN RH)
1000	VBLGDA	LOG_DATA PRESENT (USED BY SEND_ERROR OR DEALLOCATE_ABEND)
0080	VBSRSP	RESERVED WHEN USING VERB (0 ALWAYS)
0040	VBSNEG	RESERVED WHEN USING VERB (0 ALWAYS)
0010	VBATCH	ATTATCHED, $SATCH W/ VP_OPC = VBALLO
0008	VBLOCK	LOCK FOR PREPARE_TO_RECEIVE VERB ($SPTOR): 0 = SHORT, 1 = LONG
0006	VBRCTL	RETURN CNT'L (USED BY ALLOCATE ONLY)
0004	VBWALL	WHEN ALLOCATED
0002	VBIMMD	IMMEDIATE
0001	VBATRM	RESERVED WHEN USING VERB (0 ALWAYS)
VERB CONTROL INFORMATION TYPE (VP_CTL)		
0000	VBFLSH	FLUSH SEND BUFFER, $SFLSH
0001	VBCONF	CONFIRM, $SCONF
0002	VBCNFD	CONFIRMED, $SCNFD
0003	VBRTOS	REQUEST_TO_SEND, $SRTOS
2004	VBSERR	SEND_ERROR, $SSERR
0005	VBPTOR	PREPARE_TO_RECEIVE, $SPTOR
0006	VBPONR	POST_ON_RECEIPT, $SPONR
2007	VBDEAL	DEALLOCATE, $SDEAL
0007	VBCTLM	MAX VALUE OF CONTROL_INFORMATION TYPE

Table G-1 (cont). $SVPB Template

Offset	Label	Meaning
VERB OUTPUT CONTROL WORD		
0080	VBRRTS	REQUEST_TO_SEND RECEIVED WHEN SET
0040	VBPOST	CONV. POSTED (USED BY POST_ON_RECEIPT ONLY)
WHAT-RECEIVED INDICATORS		
2	0002	VBRSND SEND INDICATOR RCV'D
4	0004	VBRCNF CONFIRM REQ. RCV'D
5	0005	VBRCDA CONFIRM_DEALLOCATE RCV'D
6	0006	VBRCSN CONFIRM_SEND RCV'D
8	0008	VBRLLT LL_TRUNCATED
10	000A	VBRDAT0 DATA AVAILABLE WHEN LENGTH=0
14	0014	VBRDAT DATA RECEIVED
15	0015	VBRDCP DATA_COMPLETE
16	0016	VBRDIC DATA_INCOMPLETE
TYPE VALUES (VP_TYP)		
0000	VBTPFL	FLUSH
0001	VBTPSL	SYNC_LEVEL
0002	VBTPAP	ABEND_PROGRAM
0003	VBTPAS	ABEND_SERVICE
0004	VBTPAT	ABEND_TIMER
0005	VBTPPG	PROGRAM
0006	VBTPSV	SERVICE
CONVERSATION STATE (VP_CST)		
0000	VBCRST	RESET
0100	VBCSND	SEND STATE
0200	VBCRCV	RECEIVE STATE
0300	VBCCNF	RCV'D CONFIRM
0400	VBCCSN	RCV'D CONFIRM_SEND
0500	VBCCDA	RCV'D CONFIRM_DEALLOCATE
0600	VBCDPR	DEFER STATE--PREPARE TO RECEIVE
0700	VBCDDA	DEFER STATE--DEALLOCATE
0800	VBCPDA	PEND DEALLOCATE
0900	VBCSPT	SYNC. POINT
0A00	VBCBOT	BACKED OUT STATE

GLOSSARY

basic information unit (BIU)

The unit of data and control information that is passed between half-sessions. It consists of a request/response header (RH) followed by a request/response unit (RU).

class of service

A designation of the path control network characteristics, such as path security, transmission priority, and bandwidth, that apply to a particular session. The end user designates class of service at session initiation by using a symbolic name that is mapped into a list of virtual routes, any one of which can be selected for the session to provide the requested level oF service.

configuration services

One of the types of network services in the system services control point (SSCP) and in the physical unit (PU); configuration services activate, deactivate, and maintain the status of physical units, links, and link stations.

contention state

The state in which neither half-session is transmitting data or in which both half-sessions are transmitting data simultaneously. The contention winner can be configured to be non-negotiable, in which case the specified primary or secondary LU would always be the winner when a contention state occurred.

data flow control (DFC)

A request/response unit (RU) category used for requests and responses exchanged between the data flow control layer in the session partner.

end user

The ultimate source or destination of application dataflowing through an SNA6 network. An end user may be an application program or a terminal operator.

function management (fm) header

One or more headers, optionally present in the leading request units (RUs) of an RU chain, that is provides information to: (1) select a destination at the session partner, (2) control the way that end-user data is handled at the destination, (3) change the characteristics of the data during the session, and (4) transmit status or user information about the destination (for example, a program or device).

half-session

A component that provides FMD services, data flow control, and transmission control for one of the sessions of a network addressable unit (NAU).

host node

A subarea node that contains a system services control point (SSCP); for example, a System/370 with OS/VS2 and ACF/TCAM.

interrupt type

The type of flag which is sent by either the host or the DPS 6 or DPS 6 PLUS during the session. These flage can be SNA6 commands or indicators or SPI control information,

link

The combination of the link connection and the link stations joining network nodes; for example, (1) a System/370 channel and its associated protocols, (2) a serial-by-bit connection under the control of synchronous data link control (SDLC).

link connections

The physical equipment providing two-way communication between one link station and one or more other link stations; for example, a communication line and data circuit terminating equipment (DCE).

link station

The combination of hardware and software that allows a node to attach to and provide control for a link.

logical unit (LU)

A port through which an end user accesses the SNA6 network the functions provided by system services control points (SSCPs). An LU is capable of supporting at least two sessions--one with an SSCP and one with another logical unit--and may be capable of supporting many sessions with other logical units.

LU-LU session

A session between two logical units in an SNA6 network. It provides communication between two end users or between an end user and an LU services component.

network addressable unit (NAU)

A logical unit, a physical unit, or a system services control point. It is the origin or the destination of information transmitted by the path control network.

node

An endpoint of a link or a junction common to two or more links in a network. Nodes can be distributed or host processors, communication controllers, cluster controllers, or terminals. Nodes can vary in routing and other functional capbilities.

pacing

A technique by which a receiving component controls the rate of transmission of a sending component to prevent overrun or congestion.

parallel sessions

Two or more currently active sessions between the same two logical units (LU's) using different pairs of network addresses. Each session can have independent session parameters.

physical unit (PU)

The component that manages and monitors the resources of a node, as requested by an SSCP via an SSCP-PU session. Each node of an SNA6 network contains a physical unit.

protocol

The meaning of, and the sequencing rules for, requests and responses used for managing the network, transferring data, and synchronizing the states of network components.

request header (RH)

A request unit (ru) header preceding a request unit.

request unit (RU)

A message unit that contains control information such as a request code of FM header, end-user data, or both.

request/response header (RH)

Control information, preceding a request/response unit (RU), that specifies the type of RU (request unit or response unit) and contains control in formation associated with that RU.

request/response unit (RU)

A generic term for a request unit or a response unit.

response

(1) A message unit that acknowledges receipt of request; a response consists of a response header (RH), a response unit (RU), or both. (2) in SDLC, the control information sent from the secondardy station to the primary station.

response header (RH)

A header, optionally followed by a response unit (RU), that indicates whether the response is positive or negative and that may contain a pacing response.

response unit (RU)

A message unit that acknowledges a request unit; it may contain prefix information received in a request unit. If positive, the response unit may contain additional information (such as session parameters in response to bind session), or if negative, contains sense data defining the exception condition.

session

A logical connection between two network addressable units (NAUs) that can be activated, tailored to provide various protocols, and deactivated, as requested. The session activation request and response can determine options relating to such things as the rate and concurrency of data exchange, the control of contention and error recovery, and the characteristics of the data stream. Sessions compete for network resources such as the links within the path control network.

session partner

One of the two network addressable units having an active session.

SNA6 network

The part of a user-application network that conforms to the formats and protocols of Systems Network Architecture. It enables reliable transfer of data among end users and provides protocols for controlling the resources of various network configurations. The SNA6 network consists of network addressable units, boundary function components, and the path control network.

SNA6 node

A node that supports SNA6 protocols

SSCP-PU session

A session between a system services control point (SSCP) and a physical unit (PU). SSCP-PU sessions allow SSCP's to send requests to and receive status information from individual nodes in order to control network configuration.

SSCP-SSCP session

A session between the system services control point (SSCP) in one domain and the SSCP in another domain. An SSCP-SSCP session is used to initiate and terminate cross-domain LU-LU sessions.

Synchronous Data Link Control (SDLC)

A discipline for managing synchronous, code-transparent, serial-by-bit information transfer over a link connection. transmission exchanges may be duplex or half duplex over switched or nonswitched links. The configuration of the link connection may be point-to-point, multipoint, or loop.

System Services Control Point (SSCP)

A focal point withing an SNA6 network for managing the configuration, coordinating network operator and problem determination requests, and providing directory support and other session services for end users of the network. Multiple SSCPs cooperating as peers with one another, can divide the network into domains of control, with each SSCP having a hierarcical control relationship to the physical units and logical units within its own domain.

MANUAL DIRECTORY

SNA6 MANUALS

Base Publication Number	Manual Title
CR56	IBM Distributed Data Processing Overview
CR57	SNA6 Network Configuration
CR58	SNA6 Interactive Terminal Facility User's Guide
CR59	SNA6 Remote Job Entry Facility User's Guide
CR60	SNA6 File Transfer Facility User's Guide
GR11	SNA6 Application Programmer's Guide
GB88	SNA6 Host System Programmer's Guide
GX10	SNA6 Operator's Guide
GX11	SNA6 Host System Operator's Guide
GX12	SNA6 Internetworking User's Guide

IBM MANUALS

Refer to these IBM documents for host programming, operating, application, and configuration information:

Base Publication Number	Manual Title
SC27-0164	ACF/VTAM Version 2 Messages and Codes
SC27-0610	ACF/VTAM Version 2 Installation/Resource Definition
SC27-0611	ACF/VTAM Version 2 Programming
SC30-3142	ACF/NCP/VS & SSP Installation (Release 2.1)
SC30-3143	ACF/NCP/VS & SSP Utilities (Release 2.1)
SC30-3145	ACF/NCP/VS & SSP Messages (Release 2.1)
SC23-0046	JES2 Initialization and Tuning
SC33-0149	CICS Resource Definition Guide
SH20-9081	IMS/VS Installation Guide
SH20-9045	IMS/VS Programming Guide for Remote SNA Systems
SC33-0077	CICS Application Programmer's Reference Manual
SC33-0133	CICS Intercommunication Facilities Guide
GC30-3084	Transaction Programmer's Reference Manual for LU Type 6.2

HVS 6 PLUS OPERATING SYSTEM MANUALS

Base Publication Number	Manual Title
HE01	ONE PLUS Guide to Software Documentation
HE02	HVS 6 PLUS System Building and Administration
HE03	HVS 6 PLUS System Concepts
HE04	HVS 6 PLUS System User's Guide
HE05	HVS 6 PLUS System Programmer's Guide - Volume I
HE06	HVS 6 PLUS System Programmer's Guide - Volume II
HE07	HVS 6 PLUS Programmer's Pocket Guide
HE09	HVS 6 PLUS System Maintenance Facility Administrator's Guide
HE10	HVS 6 PLUS Menu System User's Guide
HE11	HVS 6 PLUS Software Installation Guide
HE13	HVS 6 PLUS Migration Guide
HE14	HVS 6 PLUS Application Development Overview
HE15	HVS 6 PLUS Application Developer's Guide
HE16	HVS 6 PLUS System Messages
HE17	HVS 6 PLUS Commands
HE18	HVS 6 PLUS Sort/Merge
HE19	HVS 6 PLUS Data File Organizations and Formats
HE21	HVS 6 PLUS Display Formatting and Control
HE22	HVS 6 PLUS VISION Reference Manual
HE23	HVS 6 PLUS Editors Manual

MOD 400 OPERATING SYSTEM MANUALS

Base Publication Number	Manual Title
CZ02	GCOS 6 MOD 400 System Building and Administration
CZ03	GCOS 6 MOD 400 System Concepts
CZ04	GCOS 6 MOD 400 System User's Guide
CZ05	GCOS 6 MOD 400 System Programmer's Guide - Volume I
CZ06	GCOS 6 MOD 400 System Programmer's Guide - Volume II
CZ07	GCOS 6 MOD 400 Programmer's Pocket Guide
CZ09	GCOS 6 MOD 400 System Maintenance Facility Administrator's Guide
CZ10	GCOS 6 MOD 400 Menu System User's Guide
CZ11	GCOS 6 MOD 400 Software Installation Guide
CZ15	GCOS 6 MOD 400 Application Developer's Guide
CZ16	GCOS 6 MOD 400 System Messages
CZ17	GCOS 6 MOD 400 Commands
CZ18	GCOS 6 Sort/Merge
CZ19	GCOS 6 Data File Organizations and Formats
CZ20	GCOS 6 MOD 400 Transaction Control Language Facility
CZ21	GCOS 6 MOD 400 Display Formatting and Control
CZ22	GCOS 6 VISION Reference Manual
GZ13	GCOS 6 MOD 400 R3.1 to R4.0 Migration Guide
HC01	GCOS 6 MOD 400 Application Development Overview

INDEX

INDEX

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE	COMMUNICATIONS SNA6 APPLICATION PROGRAMMER'S GUIDE	ORDER NO.	GR11-02
		DATED	SEPTEMBER 1986

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME ______________________ DATE ____________

TITLE ______________________

COMPANY ______________________

ADDRESS ______________________

____

PLEASE FOLD AND TAPE–
NOTE: U. S. Postal Service will not deliver stapled forms

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

Honeywell

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE	COMMUNICATIONS SNA6 APPLICATION PROGRAMMER'S GUIDE	ORDER NO.	GR11-02
		DATED	SEPTEMBER 1986

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME ______________________ DATE ____________

TITLE ______________________

COMPANY ______________________

ADDRESS ______________________

____

PLEASE FOLD AND TAPE—
NOTE: U. S. Postal Service will not deliver stapled forms

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

Honeywell

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE	COMMUNICATIONS SNA6 APPLICATION PROGRAMMER'S GUIDE	ORDER NO.	GR11-02
		DATED	SEPTEMBER 1986

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME ______________________ DATE ____________

TITLE ______________________

COMPANY ______________________

ADDRESS ______________________

____

PLEASE FOLD AND TAPE—
NOTE: U. S. Postal Service will not deliver stapled forms

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

Honeywell

46625, 1086, Printed in U.S.A.

GR11-02