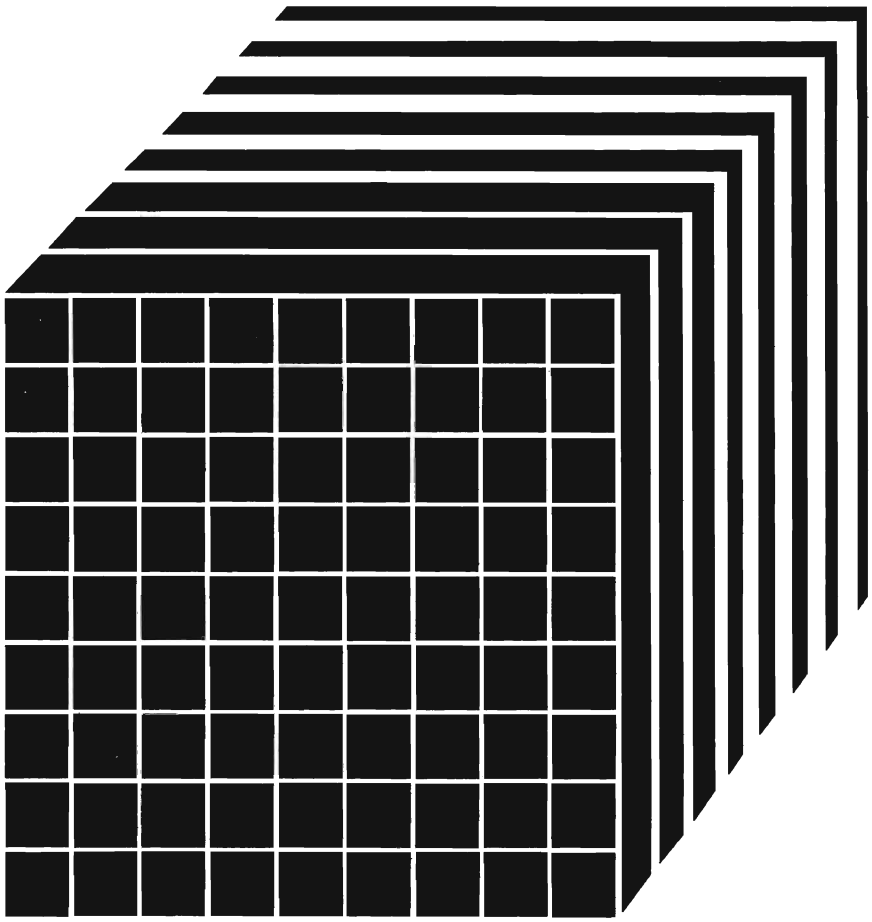
IBM

Virtual Machine/
System Product

Transparent Services Access Facility Reference

Release 5

SC24-5287-0



First Edition (December 1986)

This edition, SC24-5287-0, applies to Release 5 of Virtual Machine/System Product (VM/SP), program number 5664-167, and to all subsequent releases and modifications until otherwise indicated in new editions or Technical Newsletters. Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370, 30xx, and 4300 Processors Bibliography*, GC20-0001, for the editions that are applicable and current.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent program may be used instead.

Ordering Publications

Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality. Publications are *not* stocked at the address given below.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Information Development, Dept. G60, P.O. Box 6, Endicott, NY, U.S.A. 13760. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Preface

This reference is primarily for those persons responsible for writing Advanced Program-to-Program Communication/VM (APPC/VM) application programs to communicate within a VM processor and among interconnected VM processors. It also contains information on how to set up, run, and maintain the Transparent Services Access Facility (TSAF) virtual machines that provide communication between VM processors.

How to Use this Book

Depending on what your job is, you should find everything you need to know about TSAF in this manual. The major tasks involved are:

- Running the TSAF virtual machine and diagnosing problems involving the TSAF virtual machine, which is Part One of this book
- Writing the APPC/VM application programs, which is Part Two of this book
- Using the CP system services for VM communication, which is Part Three of this book.

If you are a system operator or system administrator in charge of running the TSAF virtual machine, use these chapters:

- "Introduction to TSAF" on page 1 for a brief introduction to the TSAF facility
- "Chapter 1. Preparing to Use TSAF" on page 11 to set up the necessary directory entries to use the TSAF facility
- "Chapter 2. Setting Up TSAF Collections and Routes" on page 25 to set up VM systems with the TSAF virtual machines to form a TSAF collection
- "Chapter 3. Running the TSAF Virtual Machine" on page 35 for descriptions of the commands to operate the TSAF virtual machine
- "Chapter 4. Generating TSAF Accounting and Link Statistics" on page 49 to collect accounting and link statistics
- "Chapter 5. Collecting TSAF Problem Diagnosis Information (Serviceability)" on page 55 for an overview of how to service the TSAF virtual machine.

You may also want to glance through "Chapter 6. APPC/VM (VM-to-VM) Communications" on page 61 for an idea of how APPC/VM provides communication within a VM system or within a collection of interconnected VM systems using TSAF.

If you are a programmer who writes the VM programs that communicate with other VM programs in the same or in different VM systems, use these chapters:

- "Introduction to TSAF" on page 1 for a brief introduction to the TSAF facility
- "Chapter 1. Preparing to Use TSAF" on page 11 for an overview of the types of directory entries that should be set up before you run your programs
- "Chapter 6. APPC/VM (VM-to-VM) Communications" on page 61 for an overview of how APPC/VM communication works
- "Chapter 7. APPC/VM and IUCV Communication Functions" on page 81 for descriptions of the APPC/VM and IUCV (VM-unique) functions to write VM application programs that communicate with each other
- "Chapter 8. APPC Verbs Mapped with APPC/VM Functions" on page 185 for a list of APPC functions mapped with APPC/VM functions
- Appendix A, "APPC/VM and IUCV Condition Codes and Return Codes" on page 225 for a summary of the APPC/VM and IUCV functions and their related error codes
- Appendix B, "APPC - APPC/VM Mapping Summary" on page 233 for a list of APPC functions as they relate to APPC/VM functions
- Appendix C, "Sample TSAF User Program" on page 249 for a sample APPC/VM user (requestor) program
- Appendix D, "Sample TSAF Resource Manager Program" on page 257 for a sample APPC/VM resource manager (server) program.

If you are a programmer or anyone else who needs to know about the CP system services for VM communications, use these chapters:

- "Chapter 9. Collection Resource Management (*CRM) System Service" on page 209 for a description of what the CP Collection Resource Management System Service does to allow a virtual machine to become a TSAF virtual machine.
- "Chapter 10. Identify (*IDENT) System Service" on page 215 for a description of what the CP Identify System Service does to allow a virtual machine to become the resource manager of one or more VM resources.

Contents

Figures

Introduction to TSAF

What is TSAF?

The Transparent Services Access Facility (TSAF) lets users connect to and communicate with local or remote virtual machines within a collection of interconnected VM systems. With TSAF, a user can connect to a program by specifying a name that the program has made known, instead of specifying a userid and node id.

TSAF provides the following support for each VM system:

- The TSAF virtual machine, which provides the ability to communicate throughout a collection of VM systems
- TSAF program communication services:
  - An Advanced Program-to-Program Communication/VM (APPC/VM) program interface for VM program-to-VM program communication
  - A set of IUCV (VM-unique) functions for use in conjunction with APPC/VM functions.
- Two CP system services, which provide TSAF communication "set-up" services within a VM system.

TSAF Virtual Machine

The TSAF virtual machine is described in part one of this book.

The TSAF virtual machine is a separate component in the TSAF facility and handles communication between VM systems by letting APPC/VM paths span more than one VM system. Each system has its own TSAF virtual machine.

The TSAF virtual machine runs on CMS (as shown in Figure 1 on page 2), and you control it using TSAF commands.

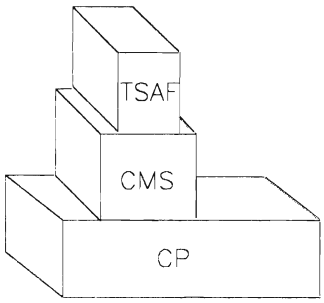


Figure 1. A TSAF Virtual Machine Running in a VM/SP System

Before you use the TSAF virtual machine, you should understand the following terms, which are described in the next few sections.

TSAF Collection

VM Resource

Resource Manager (Server)

User (Requestor) Program.

What Is a TSAF Collection?

A TSAF *collection* is a group of interconnected VM/SP systems that each have a TSAF virtual machine installed and running. Virtual machines within a collection can share data. A TSAF collection can contain up to eight interconnected VM systems.

The systems that make up the collection are connected, directly or indirectly, by either of the following TSAF-controlled links:

- Channel-to-channel (CTC) links, including 3088 links
- Binary Synchronous Communications (BSC) links.

Figure 2 on page 3 represents a sample TSAF collection made up of four VM/SP systems. In this figure, VM2 and VM3 are not physically connected, but they can communicate because a route exists through VM1 or VM4.

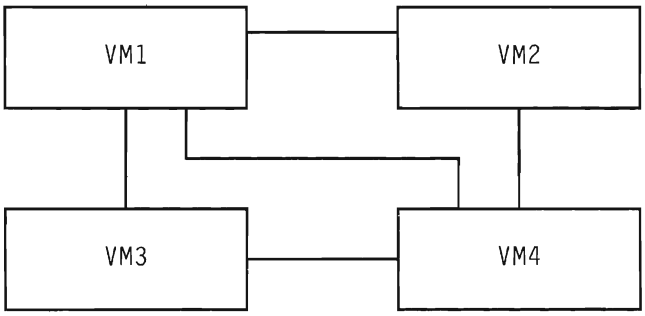


Figure 2. A Sample TSAF Collection

The TSAF virtual machine takes care of setting up the TSAF collection. A collection is started by a TSAF virtual machine wanting to communicate with a TSAF virtual machine on another system. If a link comes up or becomes available between two systems and the systems are not yet in the same collection, the TSAF virtual machines:

- Exchange information, including the names of the resources on both systems
- Dynamically configure a new collection.

Collections are discussed in more detail in "Chapter 2. Setting Up TSAF Collections and Routes" on page 25.

What Is a VM Resource?

A VM *resource* is a program, a data file, a specific set of files, a device or any other entity or set of entities that you might want to uniquely identify for purposes of application program processing in a VM system or within a TSAF collection. A VM resource is identified by a one-to-eight character name called a VM resource id. A single program may be represented by one or more resource names. For example, a data base program that manages two data bases, DB1 and DB2, could be known by the resource name DB1 for requests to data base DB1. However, the same program could be known by the resource name DB2 for requests to data base DB2.

Resources are managed by resource manager virtual machines (see "What Is a Resource Manager?" on page 4). Each TSAF virtual machine keeps an up-to-date list of all the global resources within a specified TSAF collection.

A resource can be either local or global. Only authorized users on the local system have access to local resources. An authorized user on any system in the collection has access to global resources. Refer to "Local and Global Resources" on page 18 for more information on resources.

A resource can be located on the local system or on any other system within the collection. Each global resource name within a collection of systems must be unique. You can restrict resources so that only users on

the local system can use them; these are local resources. In those cases, when resources are local, the names of the resources only need to be unique within the system, and not within the collection.

What Is a Resource Manager?

A *resource manager* (also called a *server*) is a program or set of programs executing in a virtual machine and managing access to one or more VM resources. You can add entries to the resource manager's directory to authorize other virtual machines to connect to the resource.

Some examples of resource managers are:

- A database manager
- A file server that manages a set of files
- A virtual machine that manages a high-function printer.

What Is a User Program?

A *user* program (also called a *requestor*) is a program that executes in a virtual machine, and depends upon program-to-program communications with a resource manager for some or all of its processing.

TSAF Program Communication Services

The TSAF program communication services are described in part two of this book.

TSAF provides services for any two VM programs to communicate with each other. These programs can be within the same VM processor or in different VM processors. The location of the programs can change at any time without affecting the operation of the programs; this "transparency" of access is what gives TSAF its name.

So VM programs can communicate, a logical connection must be established between them. CP provides the APPC/VM path that logically connects the two VM programs. The TSAF virtual machine provides any necessary connections along this path between VM processors. In SNA LU 6.2 (APPC architecture), this is known as establishing a conversation between programs.

In TSAF, one of the two communicating VM programs is a user program, and the other is a resource manager. The user program requests the services of the resource manager. A resource manager can connect to other resource managers; in this case, the connecting resource manager would be viewed as a user program for that connection. The TSAF virtual machine and the resource managers must each identify themselves to CP (refer to "Chapter 10. Identify (*IDENT) System Service" on page 215). User programs, on the other hand, do not have to identify themselves to CP.

User programs communicate with resource managers by using two TSAF program communication services:

- The APPC/VM program interface for VM program-to-VM program communication (provided by the APPCVM macro)
- IUCV functions used as a VM program-to-CP interface (provided by the IUCV macro).

APPC/VM Program Interface

TSAF provides an APPC/VM program interface as a means of communication between programs in two virtual machines. This APPC/VM interface provides a limited set of the SNA LU 6.2 base communication functions within a single VM system and throughout a collection of VM systems to do the following:

- Establish and sever communication paths
- Send and receive data
- Send and receive error and control information.

When a program requests, through the APPC/VM program interface, a connection to a resource manager, CP makes the connection to the resource manager. The *Systems Network Architecture Transaction Programmer's Reference Manual for LU Type 6.2* gives specific information on the LU 6.2 verb interface.

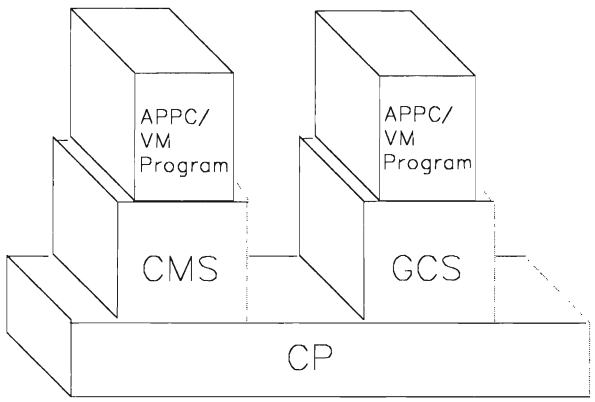


Figure 3. APPC/VM Programs Running in a VM/SP system

APPC/VM is discussed in more detail in "Chapter 6. APPC/VM (VM-to-VM) Communications" on page 61. The APPC/VM functions, as well as the necessary IUCV functions, are described in "Chapter 7. APPC/VM and IUCV Communication Functions" on page 81.

IUCV Functions for Use with APPC/VM

Applications that use APPC/VM must also use a set of IUCV functions in order to establish and control the APPC/VM environment. These IUCV functions are unique to VM and are not part of the SNA LU 6.2 (APPC architecture) verb interface. The IUCV functions provide information between a VM program and CP about the following:

- APPC/VM communication paths

  IUCV provides functions to:

  - Establish an interrupt buffer for an APPC/VM path
  - Accept an APPC/VM path connection
  - Release an interrupt buffer for an APPC/VM path.

- APPC/VM and IUCV interrupts

  IUCV provides functions to:

  - Enable and disable interrupts
  - Interrogate interrupts
  - Process interrupts.

CP System Services for TSAF

The CP system services for TSAF are described in part three of this book.

CP provides two system services for TSAF VM-to-VM communications:

- The Collection Resource Management System Service, *CRM, which gives a TSAF virtual machine the ability to be a TSAF virtual machine and to query and change the local VM resource table

- The Identify System Service, *IDENT, which allows authorized virtual machines to be a resource manager and to identify or revoke resources (i.e., begin or end management of individual VM resources).

How TSAF Enhances Your VM/SP System

TSAF provides more functions and data to more people in different systems, with less effort by users than more conventional ways of sharing data and functions.

- **More data and functions are available to your system.**

  Because TSAF lets users have access to resource managers on their own system and with other systems within the TSAF collection, users can access more information. Multiple users can have access to a

single resource manager at the same time. Each system in a TSAF collection can have up to 200 resources defined.

- **TSAF application programs are easy to program.**

  Because TSAF lets applications connect to resource managers at local or remote systems within a collection by a resource name, rather than a userid, the application can reference the resource without knowing where it is located. This makes it easier for the application programmer to write APPC/VM programs. This also makes it easy to move the actual resource to another resource manager within the collection without changing the applications that access it. Note that communication may take some additional time when the resource is on another system.

- **The TSAF virtual machine is easy to install.**

  The TSAF virtual machine is easy to install. The ITASK EXEC, documented in the *VM/SP Installation Guide*, has an option to install TSAF.

- **The TSAF virtual machine and the TSAF collections are easy to operate.**

  The TSAF virtual machine is easy to operate. When TSAF is started on a system, it automatically initiates communications with any other TSAF virtual machines it can reach over its defined links. The TSAF virtual machines dynamically compute the routes for their collections without the need for an operator or administrator.

  When a link becomes inoperative or another error condition occurs that affects the operation of the TSAF collection, the TSAF virtual machines that can still communicate with each other reconfigure the collection. In addition, if a system enters or leaves the collection, the TSAF virtual machines automatically reconfigure the collection and choose new routes for the communications to follow. Because the TSAF virtual machines are self-configuring, they do not require much operator intervention.

  "TSAF Routing" on page 32 contains more information about dynamic configuration.

- **TSAF application programs are movable.**

  A TSAF application program using the APPC/VM program interface can execute on any VM system in a TSAF collection.

Part One: TSAF Virtual Machine

This part introduces how to set up and use the TSAF virtual machine and the communicating virtual machines, and includes the tasks that the system administrator and/or operator must do to run the TSAF virtual machine.

- "Chapter 1. Preparing to Use TSAF" on page 11 introduces how to prepare to use the TSAF virtual machine:
  - Setting up the TSAF virtual machine (TSAF directory, servicing, links, and system ids)
  - Authorizing virtual machines to be resource managers
  - Authorizing virtual machines to use resources.
- "Chapter 2. Setting Up TSAF Collections and Routes" on page 25 describes how to form collections, how collections merge, information about routes, and performance considerations.
- "Chapter 3. Running the TSAF Virtual Machine" on page 35 describes how to use the TSAF commands to run and maintain the TSAF virtual machine.
- "Chapter 4. Generating TSAF Accounting and Link Statistics" on page 49 describes the contents of the TSAF accounting and link statistics records.
- "Chapter 5. Collecting TSAF Problem Diagnosis Information (Serviceability)" on page 55 gives a general overview of how to diagnose problems of the TSAF virtual machine by using dumps and/or system trace data.

Chapter 1. Preparing to Use TSAF

Before you can use TSAF, you must do the following steps:

1. Set up the TSAF virtual machine.
   a. Modify the TSAF entry in the system directory.
   b. Prepare to install and service the TSAF virtual machine.
   c. Use the TSAF message repository.
   d. Set up links through which processors can communicate. (This may have already been done during installation.)
2. Assure a secure TSAF collection by assigning unique userids and node ids within a collection.
3. Understand the difference between local and global resources.
4. Set up virtual machines to be resource managers.
5. Set up virtual machines to be able to connect to resources.

This chapter describes these steps.

Setting Up the TSAF Virtual Machine

The TSAF virtual machine component of a system keeps track of all the global resources within the system and within the collection. With the TSAF virtual machine, APPC/VM communication between systems in a collection is possible.

TSAF runs as a CMS application.

Modifying the TSAF System Directory Entry

The TSAF system directories that are provided have the following qualities:

- Privilege class G
- At least 4 MB of virtual storage
- Dedicated links.

The following statements are part of the TSAF directory entry:

1. The following OPTION statements:

   - **OPTION MAXCONN** *nnnnn*

     nnnnn is the maximum number of IUCV and APPC/VM connections allowed for this virtual machine. The value assigned to *nnnnn* should be large enough to handle all planned intersystem APPC/VM paths that start from, or end at, your system.

   - **OPTION BMX**

     This lets TSAF use CTC line drivers.

   - **OPTION ECMODE**

     This lets the TSAF virtual machine use certain S/370 instructions. These instructions are privileged operations that require extended control mode support for correct simulation by CP.

   - **OPTION COMSRV**

     This lets the TSAF virtual machine act as a communication server, routing connections on behalf of virtual machines to other servers. (COMSRV stands for *communication server*). Servers can establish connections to other servers while handling requests for other users.

     With this option, the TSAF virtual machine or any other communication server can put the userid of the virtual machine that issued the APPC/VM CONNECT in the CONNECT parameter list.

     When TSAF sends the connect request to the target resource-manager virtual machine, the request contains this information about the originating virtual machine. Without this option, CP would send the connect request with the communication server's userid.

     The authorized virtual machine can specify any SEVER or SENDERR code. CP does not verify the SEVER code. When the authorized virtual machine specifies a SENDERR code, CP does not generate a SENDERR code, but instead uses the one provided.

   - **OPTION DIAG98**

     This lets the TSAF virtual machine use DIAGNOSE code X'98'. See *VM System Facilities for Programming* for details about DIAGNOSE code X'98'.

   - **OPTION ACCT**

This causes the TSAF virtual machine to generate accounting records.

- **OPTION CONCEAL**

  This places the TSAF virtual machine in a protected application environment at logon time. Protected application environment means the following:

  - Multiple attentions do not cause the TSAF virtual machine to drop into CP mode.
  - TERMINAL BRKKEY is set to NONE.
  - If the TSAF virtual machine alters a shared page, CP tries to resume execution in the virtual machine, before it starts an automatic re-IPL.
  - CP starts an automatic re-IPL when it encounters errors such as: a virtual machine disabled wait, a paging error, an invalid PSW, an external interrupt loop, or a translation exception.
  - If a TSAF or CMS abend occurs, TSAF causes CP to start an automatic re-IPL.

- **OPTION REALTIMER**

  This ensures that the timer functions that TSAF uses operate accurately, on a real time basis.

2. The following IUCV statements:

   - **IUCV ALLOW**

     This lets any virtual machine connect to the TSAF virtual machine.

     You may not want to let every virtual machine connect to the TSAF virtual machine. Instead, you may explicitly authorize each virtual machine that wants to connect to a resource. You can do this by including "IUCV resource" or "IUCV ANY" statements in each virtual machine's directory entry. When you explicitly authorize each virtual machine this way, you should also give explicit directory authorization to the TSAF virtual machine residing on the same system as the resource, using "IUCV resource" or "IUCV ANY."

   - **IUCV *CRM**

     This lets the TSAF virtual machine connect to *CRM (Collection Resource Management System Service). Only one virtual machine in a system can connect to *CRM at any time.

3. **MDISK 191**

This sets aside room for any user EXECs and for TSAF to write its link definition file called ATSLINKS FILE A1. Sufficient storage for the virtual machine should be forty, 1024-byte blocks or one cylinder. See the *VM/SP Installation Guide* for specific device information.

For more information about the system directory entries, see the *VM/SP Planning Guide and Reference.*

TSAF Virtual Machine Sample Directory

Figure 4 is a sample 3380 directory entry for a TSAF virtual machine.

```
USER TSAFVM NOLOG 4M 8M G
 ACCOUNT 1 xxxxxx
 OPTION MAXCONN 256 BMX ECMODE COMSRV DIAG98 ACCT CONCEAL REALTIMER
 IUCV ALLOW
 IUCV *CRM
 IPL CMS PARM AUTOCR
 CONSOLE 009 3215 A OPERATOR
 SPOOL 00C 2540 READER *
 SPOOL 00D 2540 PUNCH  A
 SPOOL 00E 1403 A
 LINK MAINT 190 190 RR
 LINK MAINT 19D 19D RR
 LINK MAINT 19E 19E RR
 LINK MAINT 492 492 RR
 LINK MAINT 494 494 RR
 MDISK 191 3380 675 002 VMPK01 MR
 DEDICATE 4A0 300
```

Figure 4. TSAF Virtual Machine Directory Entry Example

To access the 492 and 494 disks, you must access them in TSAF's PROFILE EXEC. The 492 disk contains the TSAF object code. The 494 disk contains updated TSAF EXECs and modules. To ensure that you are working with the most recent fixes, access the 494 disk before the 492 base TSAF disk in the PROFILE EXEC. The PROFILE EXEC must also contain a SET LANGUAGE statement (shown in "Using the TSAF Message Repository" on page 15).

The *VM/SP Planning Guide and Reference* contains more information about the directory entry statements, including the CONSOLE, SPOOL, and LINK statements.

Preparing to Install and Service TSAF

To load TSAF, you invoke the ITASK EXEC with the LOAD TSAF parameters. This loads the TSAF object code to the MAINT 492 minidisk. Because TSAF code is shipped pregenerated, no further processing is required. The *VM/SP Installation Guide* contains more information.

The most recent build for the TSAF program creates a load map. You can use this load map to process problem information (dumps) for the TSAF

virtual machine. "Chapter 5. Collecting TSAF Problem Diagnosis Information (Serviceability)" on page 55 contains more information on servicing the TSAF virtual machine.

Using the TSAF Message Repository

The TSAF message repository source file is ATSCMRx REPOS (where x is the country code for a particular language). The VMFNLS EXEC applies updates to ATSCMRx REPOS and generates a text file called ATSUME TXTlangid. Check to make sure that ATSUME TXTlangid is on a disk accessible to the TSAF virtual machine (for example, the 494 Service disk).

To use the TSAF message repository, once the necessary service has been applied, add the following to the TSAF virtual machine's PROFILE EXEC:

```
SET LANGUAGE AMENG (ADD ATS USER
```

This loads the repository and parser tables into the TSAF virtual machine. If your system is running in a language other than American English, refer to the *VM/SP CMS Command Reference* to alter this command format.

If you do not add the SET LANGUAGE entry in the PROFILE EXEC, you do not have access to the repository. In this case, when TSAF tries to issue the usual message, you will, instead, receive the following message:

```
813E  ATS repository not found, message msgid cannot be
      retrieved
```

The *VM/SP CMS Command Reference* has more information on the SET LANGUAGE command and other commands for processing message repositories.

Setting Up Links for Communication

The only information that the TSAF virtual machine needs upon installation is the identity of the communication links that it can use. TSAF stores this information in a CMS file called ATSLINKS FILE A1. You receive error messages if you store the ATSLINKS FILE file on any disk other than TSAF's A-disk. You can create this file before you start TSAF, so you do not need to issue commands to add links after TSAF is started.

The ATSLINKS FILE must contain the virtual device address for each link that TSAF can use. The virtual device address can start in any column in the ATSLINKS FILE (TSAF writes the device address in columns 2 through 4).

To physically set up the link so the system recognizes the link, follow these steps (Steps 1 and 2 may have already been done during installation):

1. Set up links between communicating processors by modifying DMKRIO to describe the physical links. The *VM/SP Planning Guide and Reference* describes DMKRIO.

2. Use the SPGEN EXEC to regenerate the real I/O configuration (DMKRIO). See the *VM/SP Installation Guide* for more specific information about the SPGEN EXEC.

3. Make sure that the TSAF virtual machine has the links it needs to communicate by doing one of the following:

   - Dedicate these links to the TSAF virtual machine in the TSAF entry of the system directory. For example, in Figure 4 on page 14,

     ```
     DEDICATE 4A0 300
     ```

     dedicates the real device at address 300 to TSAF as the virtual device with address 4A0. You specify the virtual address when you use the ADD LINKs command described in "Chapter 3. Running the TSAF Virtual Machine" on page 35.

   - Use the ATTACH command, described in the *VM/SP CP Command Reference*, to attach the links.

Note that multiple active links from one TSAF virtual machine to another TSAF virtual machine may adversely affect the ability of those TSAF virtual machines to join. "Reliability in a Collection" on page 28 contains more information about links between TSAF virtual machines.

Security Considerations when Setting Up TSAF

You can provide a secure TSAF system and collection in various ways.

Assigning Unique Userids

Your applications may rely on the userids of the connecting applications to maintain security and check authorization. The userid that TSAF presents is always the userid of the virtual machine that originated the request. Even if the connection is through the TSAF virtual machine, TSAF presents the userid of the originating virtual machine, not the TSAF virtual machine userid.

TSAF does not enforce it, but you must ensure that no two users in a collection have the same userid. The exception is when a user has the same userid on multiple nodes within the collection. In this case, the user would have the same authorization for resources from whatever system in the collection he or she is logged onto.

Assigning Unique Node Ids

There are a few different identifiers for each system:

- The processor id, or CPUID, which is preassigned
- The node id, which the system administrator assigns at installation time.

The SYSTEM NETID file, an existing CMS file, associates the CPUID of a processor with its node id. Be sure that the processors within a collection have unique node ids. For information on how to update the SYSTEM NETID file, see the *VM/SP Installation Guide*.

Assigning Unique Resource Ids

A resource can be located on the local system or on any other system within the collection. Each global resource name within a collection must be unique. For local or global resources, do not specify the name to be the same as a userid on the system. Also, do not specify a resource name as any of the following: ALLOW, ANY, SYSTEM. See "Authorizing Virtual Machines to Connect to Resources" on page 22 for more information on resource names.

When two collections are merging, and the same resource name exists on each collection, TSAF automatically awards management responsibility to one of the systems. Two systems in the same collection cannot manage the same global resource at the same time. For more information about merging collections, see "When Two Collections Merge to Form One" on page 30. "Some Rules about Resources" on page 216 describes how TSAF selects a resource manager.

Considerations for Using the APPC/VM Program Interface

The system administrator must authorize APPC/VM communications for users in their virtual machine directory entries. If you are not authorized for APPC/VM communications, you cannot communicate with virtual machines other than your own. If the target of the APPCVM CONNECT function (the resource manager) does not have IUCV ALLOW specified in its directory entry, the system administrator must specifically authorize each virtual machine to communicate within the TSAF collection. See "Authorizing Virtual Machines to Manage Resources" on page 18 and "Authorizing Virtual Machines to Connect to Resources" on page 22 for more information on the directory entries.

Security within APPC/VM Applications

When a virtual machine running an APPC/VM application tries to connect to another virtual machine, the virtual machine being connected to must run through a few security measures. Most importantly, the virtual machine must check the userid of the connecting virtual machine and the resource id for which the connection is being made. See "Accepting or Rejecting a Connection" on page 67 for more information.

Security for Communication Servers

Communication servers are authorized in the CP directory with OPTION COMSRV. See "Considerations for Communication Servers" on page 100 for more information on communication servers.

Local and Global Resources

A local resource is one that is known only to the local system. A global resource is one that is known to all systems in the TSAF collection.

A system may have both a local and global resource defined with the same name. For example, a local resource called "Count" is different than the global resource called "Count". Both resource owners can coexist on the same system. In fact, both resources can be owned by the same virtual machine. If both the local and global "Count" resources are owned by the same virtual machine, the resources would be identical. However, if the local and global "Count" resources are owned by different virtual machines, the resources may be different.

If a local and global resource are defined with the same name, the resources are accessed as follows:

- When a local user on the local VM system requests to communicate with the resource, CP routes the user to the local resource. TSAF routes the local user to a global resource only if a local resource by that name does not exist.

- When a remote user on another VM system in the collection requests to communicate with the resource, TSAF routes the user to the global resource, even if a local resource also exists on the target system.

Keep in mind, if the local resource is revoked, users on the local VM system trying to communicate with the resource are automatically connected to the global resource.

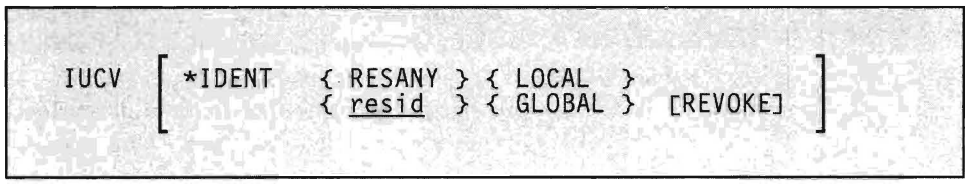
Authorizing Virtual Machines to Manage Resources

If you want to authorize a virtual machine to manage resources, you must identify it in its directory entry so the virtual machine may connect to the Identify System Service. *IDENT is the assigned system service name for the Identify System Service.

IUCV Directory Control Statement for *IDENT Authorization

To authorize a virtual machine as a resource manager, you must add an IUCV *IDENT control statement, with appropriate parameters, to the virtual machine's directory entry. Be sure the IUCV directory statement appears before any CONSOLE, SPOOL, LINK, or MDISK statements; otherwise, you get error messages.

The IUCV control statement syntax for *IDENT is:



***IDENT**
lets the virtual machine connect to the Identify System Service.

RESANY
lets the virtual machine identify any resource name.

Be careful when you assign resource names and when you give authorization for RESANY. A virtual machine that has authorization for RESANY can identify a resource name as "resany." Also, this virtual machine would be authorized to identify any other resource name.

resid
is a one-to-eight character resource name. Virtual machines can connect to the resource manager that manages the resource specified by *resid*. The first byte of the resource name should be alphanumeric. (IBM reserves names beginning with the remaining characters for its own use.)

Be sure that the resource name you specify is not the same as a userid on the system. Also, do not specify the resource name as any of the following: ALLOW, ANY, or SYSTEM.

LOCAL
authorizes the virtual machine to identify the resource as a local resource known only to the local system. If you specify LOCAL with RESANY, the virtual machine can identify any resource as a local resource.

GLOBAL
authorizes the virtual machine to identify the resource as a global resource known to all systems in the collection. This operand lets the virtual machine identify the resource as local also. If you specify GLOBAL with RESANY, the virtual machine can identify any resource either locally or globally.

REVOKE
authorizes the virtual machine to revoke the specified resource name. A virtual machine that can revoke resources can also identify them.

If you specify REVOKE with:

- LOCAL, the virtual machine can revoke and identify the resource on the local system only.
- GLOBAL, the virtual machine can do either of the following:
  - Revoke and identify the global resource
  - Revoke and identify the local resource on the local system.

  A virtual machine cannot revoke both the global and local resources at the same time. The virtual machine must specify which resource to revoke when the connection is made to *IDENT.
- RESANY and LOCAL, the virtual machine can revoke and identify any local resource.
- RESANY and GLOBAL, the virtual machine can revoke and identify any resource, local or global.

Because the TSAF virtual machines do not keep track of the local resources, a virtual machine cannot revoke a local resource on another system.

The complete IUCV control statement is described in the *VM/SP Planning Guide and Reference.*

Other Statements in the Resource Manager's Directory Entry

Specify the OPTION statement with the MAXCONN keyword in the directory entry of the resource manager. Specify a large enough number to support an additional IUCV connection for:

- Each resource that the resource manager virtual machine controls
- Each user that connects to the resources.

Though it is not a required directory statement, you should include IUCV ALLOW in the directory entry for any resource manager. This lets any virtual machine access the resource manager, instead of requiring directory authorization for each user who needs to connect to a specific resource.

However, for security reasons, you may want to, instead, explicitly authorize each virtual machine that wants to connect to a resource.

Identifying More than One Resource from the Same Virtual Machine

If a virtual machine must identify more than one resource, you can specify more than one IUCV control statement for *IDENT in the virtual machine's directory entry. *IDENT checks to see if the virtual machine is authorized to identify or revoke the resource specified on the APPCVM CONNECT function. *IDENT searches the virtual machine's directory entry for IUCV control statements in this order:

1. The first *IDENT entry that has the same resource name as specified on the CONNECT. If *IDENT does not find or finds a match, but authorization for the LOCAL/GLOBAL and REVOKE parameters does not correspond to those specified in the CONNECT parameter list, *IDENT searches for,

2. The first *IDENT entry that has the resource name RESANY. If *IDENT finds a match, it checks that the authorization for the LOCAL/GLOBAL and REVOKE parameters correspond to those specified in the CONNECT parameter list. If it does not find a match, or if the other parameters do not correspond, then it severs the requested connection.

A single system can have up to 200 resources identified.

Examples of Multiple IUCV *IDENT Control Statements

The following examples describe how to use multiple IUCV control statements for *IDENT.

Example 1: A resource manager, RESMGR1, has the following IUCV control statements:

```
IUCV *IDENT RESANY GLOBAL

IUCV *IDENT residx LOCAL REVOKE
```

RESMGR1 can identify any resource as a local or global resource, because of the first statement. This includes the local resource, residx. However, RESMGR1 can only revoke the resource, residx, when it is defined on the local system as a local resource.

Example 2: A resource manager, RESMGR2, has the following IUCV control statements:

```
IUCV *IDENT residx GLOBAL

IUCV *IDENT residy GLOBAL

IUCV *IDENT RESANY LOCAL REVOKE
```

RESMGR2 can identify the resources, residx and residy, as local or global resources. RESMGR2 is not authorized to revoke any global resources. Because of the last control statement, RESMGR2 can identify any resource as a local resource. Also, RESMGR2 can revoke any resource known on the local system as a local resource.

Example 3: A resource manager, RESMGR3, has the following IUCV control statements:

```
IUCV *IDENT residx LOCAL

IUCV *IDENT residy GLOBAL

IUCV *IDENT residx GLOBAL REVOKE
```

RESMGR3 can identify the resource, residx, as a local resource and the resource, residy, as either a local or a global resource. RESMGR3 cannot revoke any resources. The reason for this is that *IDENT searches for the first entry that matches the resource name specified on the CONNECT. If RESMGR3 tries to connect to *IDENT to identify or revoke the GLOBAL resource, residx, *IDENT severs the connection. In this case, if you want RESMGR3 to identify and revoke the global resource, residx, you would delete the first control statement.

Authorizing Virtual Machines to Connect to Resources

You only need to do this procedure if the TSAF and resource manager virtual machines do not have IUCV ALLOW statements in their directory entries. If the TSAF and resource manager virtual machines have specified IUCV ALLOW statements in their directory entries, any user in the collection can connect to any global resource.

For security reasons, you may want to explicitly authorize each virtual machine that wants to connect to a resource. In this case, you can add an IUCV control statement with the resource id parameter to the user's directory entry. If a virtual machine user has a specified alternate userid, you can authorize the user and the user's alternate userid to connect to a resource by adding the IUCV control statement to:

- The user's directory entry, or
- The user's alternate userid directory entry.

The complete IUCV control statement is described in the *VM/SP Planning Guide and Reference.*

The syntax of the statement includes:

```
IUCV    resource id
```

resource id
is a one-to-eight character resource name used to connect to a resource manager rather than to a specified virtual machine. The first byte of the resource name must be alphanumeric. (IBM reserves names beginning with the remaining characters for its own use.)

Be sure that the resource name you specify is not the same as a userid on the system. Also, do not specify the resource name as any of the following: ALLOW, ANY, or SYSTEM.

Specifying **IUCV** *resource id* does not give authority to connect by userid to the virtual machine that owns the specified resource. At the same time, specifying **IUCV** *userid* does not give authority to connect by resource-id to the specified virtual machine.

When you explicitly authorize each virtual machine (with "IUCV resource" or "IUCV ANY"), you should also give explicit directory authorization to the TSAF virtual machine residing on the same system as the resource (with "IUCV resource " or "IUCV ANY").

Figure 5 shows a typical explicitly authorized TSAF collection involving two VM/SP systems. The entries within each box represent the directory entries of the particular virtual machine.

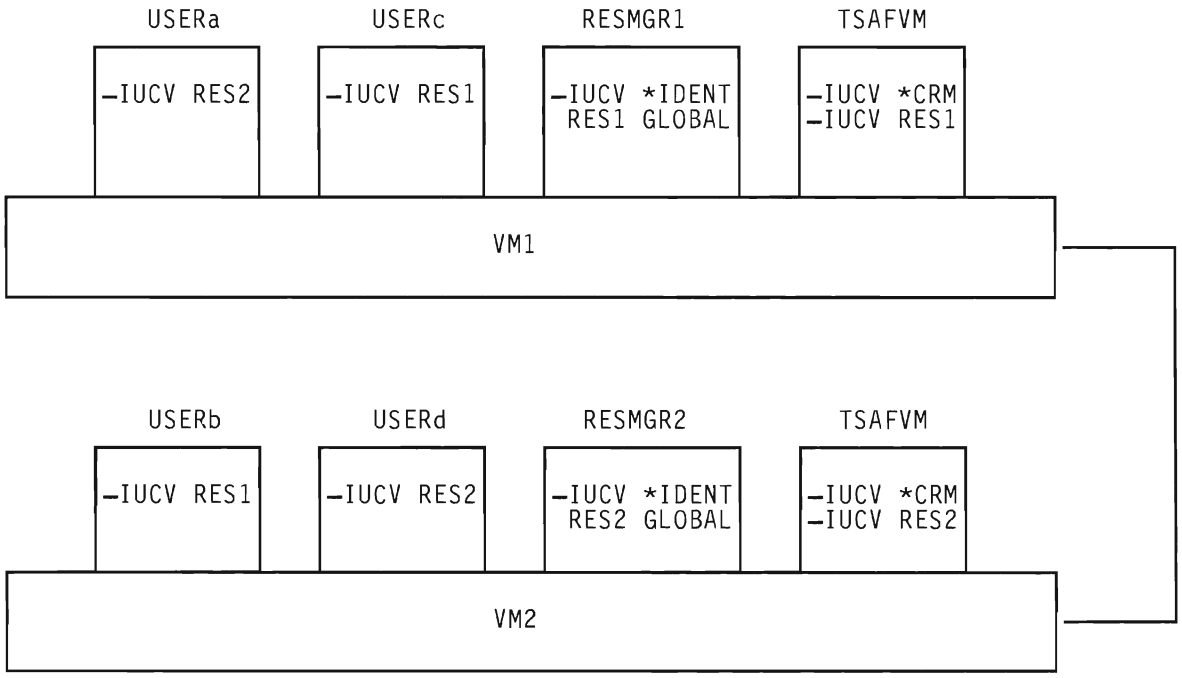


Figure 5. Two VM Systems Communicating

In this figure, users have the following authorization:

- USERa on VM1 can connect only to RES2 on VM2.
- USERb on VM2 can connect only to RES1 on VM1.
- USERc on VM1 can connect only to RES1 on VM1.
- USERd on VM2 can connect only to RES2 on VM2.

Chapter 2. Setting Up TSAF Collections and Routes

Collection Structure

A group of VM systems that each have the TSAF virtual machine component installed and running can form what is known as a collection. A collection can have up to eight systems. You must be sure of the following:

- No two users in a collection have the same userid.

  Note: TSAF does not enforce this, but you should enforce this to avoid potential problems.

- No two systems within the collection have the same node id.

See "Assigning Unique Userids" on page 16 and "Assigning Unique Node Ids" on page 17 for more information.

Collection Example

This section includes a six-step scenario that describes how to set up a collection and share resources. This particular example involves:

- One resource manager that manages a resource. The resource manager is called RESMGR and manages the resource, RES1.
- Two systems, A and B, each with a TSAF virtual machine. The TSAF virtual machines are TSAFa and TSAFb on systems A and B, respectively. The programs that want to use the resource, RES1, are PGMa and PGMb on systems A and B, respectively.

Step 1—TSAF Virtual Machine Identifies Itself

When the virtual machine, TSAFa, begins running, it requests a connection to the Collection Resource Management System Service (*CRM). Because no other local virtual machine is already connected to *CRM, and TSAFa is authorized, CP accepts TSAFa as the TSAF virtual machine and starts a connection.

If any virtual machines in the system had identified themselves as the managers of any global resources, then, upon request, CP would send those

resource names to TSAFa. In this example, there are no previously established resources.

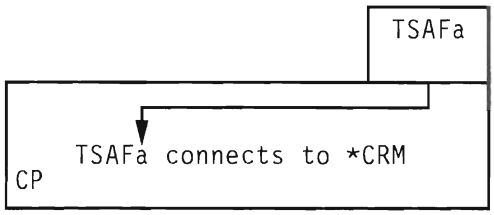


Figure 6. TSAF Virtual Machine Identifying Itself

Step 2—TSAF Virtual Machines Exchange Information

Now that TSAFa is the TSAF virtual machine for the local system, it tries to connect itself to a collection. TSAFa sends out data along each physical link that it controls. So that TSAFa can join the collection, the TSAF virtual machines on the other end of each link exchange the following information with TSAFa:

- Names of the resources that TSAFa knows
- Names of other resources in the collection that the remote TSAF virtual machines know about.

In Figure 7 there is only one other TSAF virtual machine in the collection (TSAFb) and it does not know about any resources yet.

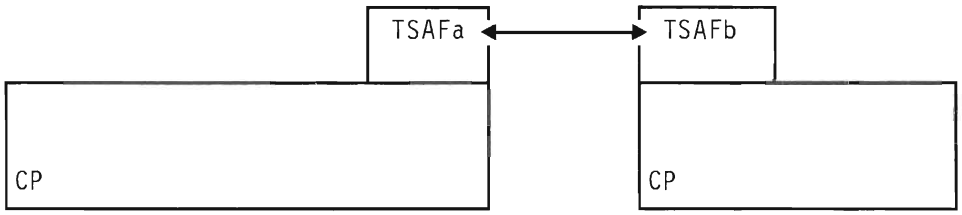


Figure 7. TSAF Virtual Machines Exchanging Information

Step 3—A Resource Manager Requests to Manage a Resource

When the resource manager, RESMGR, enters the collection, it issues a CONNECT to the Identify System Service (*IDENT). This is so CP recognizes RESMGR as the manager of the global resource, RES1. CP notifies TSAFa over its *CRM connection that RESMGR wants to manage the resource, RES1.

The TSAF virtual machines in the collection then agree that RESMGR can be the manager of the resource, RES1. TSAFa notifies the local CP. CP then adds the global resource, RES1, to its system resource table and accepts the connection from RESMGR to *IDENT. Both TSAFa and TSAFb add the resource, RES1, to their resource tables. If the resource had been

previously defined anywhere in the collection, then CP would have severed the requested connection.

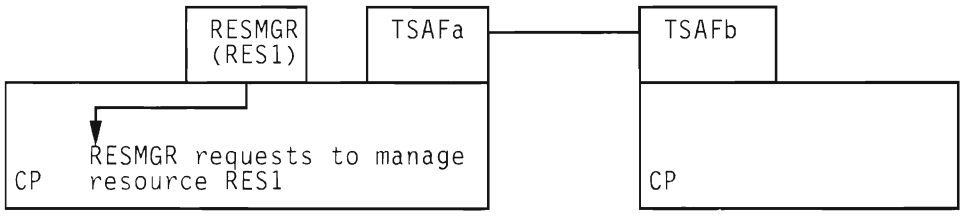


Figure 8. Resource Manager Requesting to Manage a Resource

Step 4—A Local User Connects to the Resource Manager to Share the Resource

PGMa requests to connect to resource, RES1. The local CP finds RES1 in its resource table, and connects PGMa to RESMGR. After the connection is complete, APPC/VM communication can begin over the established path. Because the resource is on the local system, there is no need to go through the TSAF virtual machine.

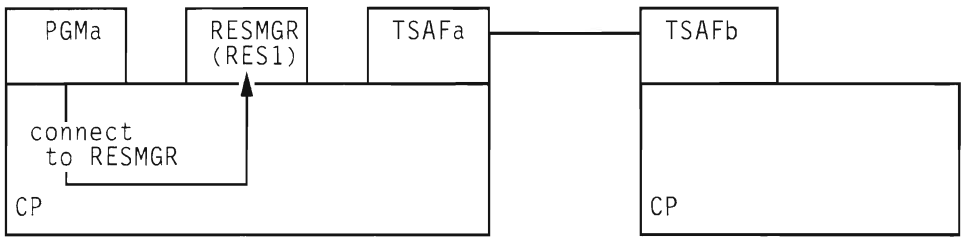


Figure 9. A Local User Sharing a Resource

Step 5—A Remote User Connects to the Resource Manager to Share the Resource

PGMb requests to connect to resource, RES1. Because the local CP does not find RES1 in its system resource table, it connects the user to TSAFb. TSAFb finds RES1 in its resource table and sends the connection request to TSAFa. TSAFa issues the CONNECT to the resource, RES1. RES1 is listed in the system resource table that TSAFa maintains, so the connection is made.

As you can see, the connection actually consists of three APPC/VM paths (between PGMb and TSAFb, between TSAFa and RESMGR, and between the TSAF virtual machines, TSAFa and TSAFb).

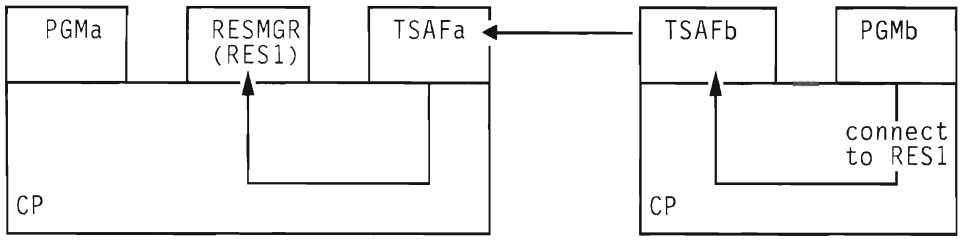


Figure 10. A Remote User Sharing a Resource

Step 6—Sends and Receives for Local and Remote Users

After the connections are complete, TSAF routes the various SENDs and RECEIVEs between the resource manager and the program that wants access to the resource.

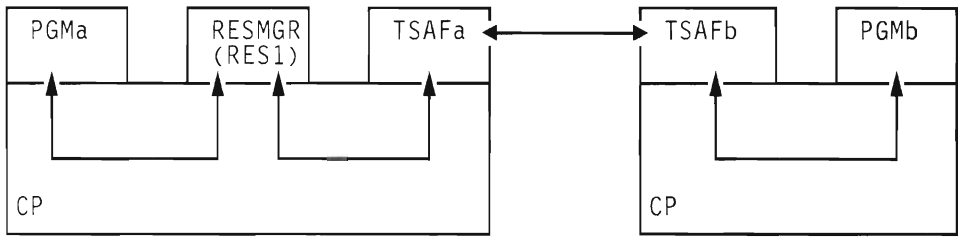


Figure 11. Sending and Receiving

Reliability in a Collection

In general, the reliability of communication within a collection depends on how the collection is set up. For example, communications from a processor where TSAF has three links to three different processors is more reliable than if the processor has only one link by which to get to the other processors. If a processor with only one link to the rest of the collection could no longer communicate through that link, the collection would be partitioned.

Multiple Links from TSAF Virtual Machine to TSAF Virtual Machine

Multiple active links from one TSAF virtual machine to another TSAF virtual machine may adversely affect the ability of those TSAF virtual machines to join. When there are multiple links, there is no guarantee that both TSAF virtual machines will use the same link to communicate.

For example, in Figure 12 on page 29, there are two dedicated links between the TSAF virtual machines (link1 and link2). If link1 and link2 were both added to TSAF, it is possible that the two TSAF virtual machines would not be able to join. This could be caused by the timing of the messages crossing those links. For example, the TSAF virtual machine on

Proc A may prefer to use link1, while the TSAF virtual machine on Proc B may prefer to use link2.

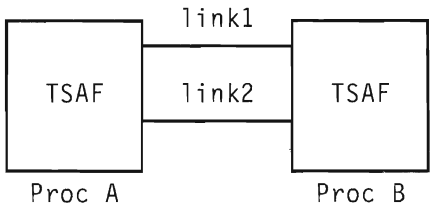


Figure 12. Multiple Connections between TSAF Virtual Machines

If you want to have more than one link available between two TSAF virtual machines, one of them should remain detached from the TSAF virtual machines or deleted from TSAF's table of communication links. Then, when needed, you can attach the link or add the link to TSAF's table of communication links. For example, in Figure 12, link2 could be unattached. But when link1 fails, you could then attach link2.

On the other hand, it is fine to have two or more links connecting the same two processors (see Figure 13), one between the TSAF virtual machines and the other links between other virtual machines, such as RSCS (Remote Spooling Communications Subsystem) or PVM (Pass-through Virtual Machine) virtual machines.

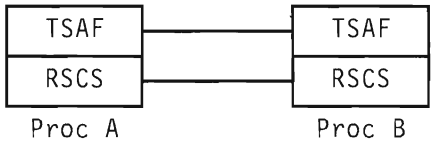


Figure 13. TSAF with RSCS

Multiple Links to Processors in a Collection

When setting up a collection of more than two processors, try to assign links from each processor to at least two other processors. This way, each processor would have at least two fully or partially distinct physical routes through which to communicate, rather than just one.

In Figure 14 on page 30, assume processors A, B, C, and D each have TSAF running. The processors, through the TSAF virtual machines, are connected by links A to B, B to C, and C to D. These systems form a collection. If the link from B to C failed for some reason, the collection would be partitioned. For example, users on A communicating with programs on C would be disconnected from those programs.

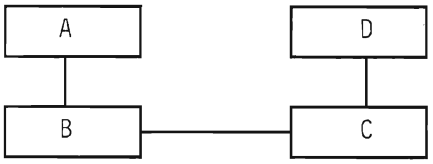


Figure 14. A TSAF Collection

On the other hand, if a link were added between processors A and D, see Figure 15, the collection would be more reliable. Again, if a user on A were communicating with programs on C, and the link from B to C failed for some reason, communication could continue on the path from A to D to C.

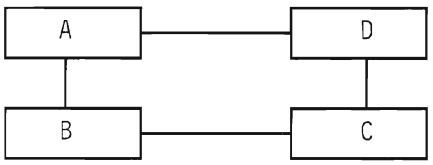


Figure 15. More Reliable TSAF Collection

When Two Collections Merge to Form One

When you make a link active or bring up a TSAF virtual machine, it may cause two or more collections to join. However, the collections may have one or more duplicate resource names. The TSAF virtual machine does not know about local resources. Therefore only global resources are affected when collections merge.

When merging collections each have a resource with duplicate resource names, TSAF determines the collection that manages the resource in the following order:

1. The largest collection (i.e., the collection with the most systems) wins management of the resource in the new merged collection.

   For example, in Figure 16 on page 31, Collection 1 and Collection 2 merge to form Collection 3. The resource ACCOUNT is defined in both Collections 1 and 2. In this case, when the collections merge, the virtual machine resource manager in Collection 1 wins ownership of the resource ACCOUNT, because Collection 1 has three nodes and Collection 2 only has two.

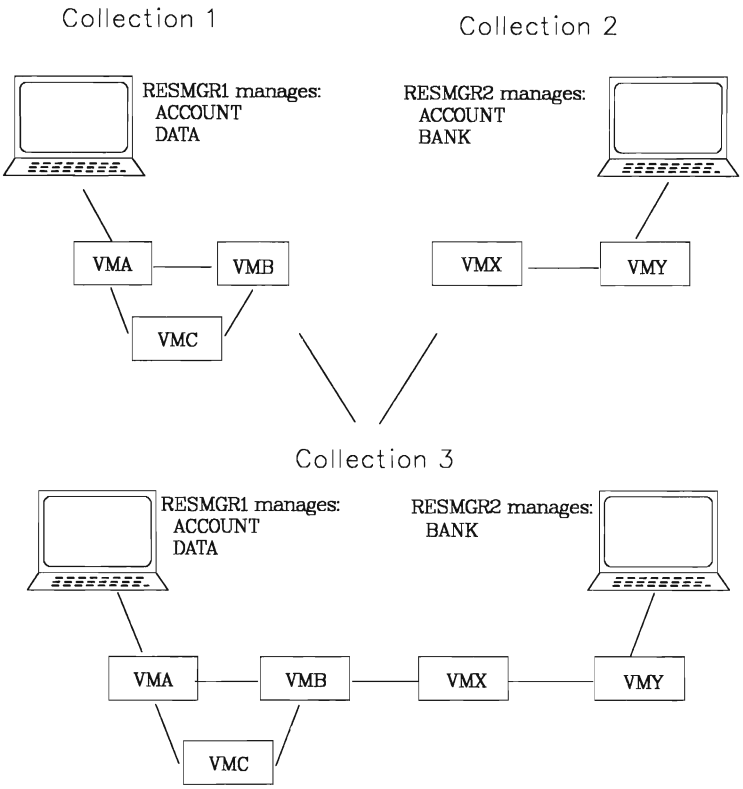


Figure 16. Two TSAF Collections Merged into One

2. If the collection sizes are the same, then TSAF looks at the two nodes that are going through the join. TSAF compares those two node ids; the first node id in alphabetical order wins management responsibility of the resource for its collection. In other words, the collection that wins management responsibility of the duplicate resource is that collection that has the first node in alphabetical order (between the two nodes involved with the join).

   In Figure 16, if the collections sizes were the same, Collection 1 would win management responsiblity for duplicate resources, since VMB and VMX are the two nodes joining, and VMB is before VMX in alphabetical order.

TSAF does not sever existing APPC/VM paths to resource managers that lose management responsibility for the resource. However, new paths will go to the resource manager in the winning collection.

Two collections may try to merge that total nine or more systems. Because only eight systems can be in a collection, any systems over eight are left out of the collection. Because of the timing involved with collection communication, you cannot predetermine the loser system. When this happens the TSAF virtual console of the system through which the ninth system was trying to join gets this message:

```
513I Node nodeid cannot join, maximum collection size
     has been reached
```

where *nodeid* is the nodeid of the system trying to join. The TSAF virtual console of the losing VM system gets this message:

```
531E Timeliness check failed on message from node nodeid
```

TSAF Routing

When the TSAF collection configures itself, the TSAF virtual machines determine the various routes that connect each TSAF virtual machine to every other TSAF virtual machine. If more than one possible route exists between two TSAF virtual machines, TSAF chooses the route with a combination of the following:

- Smallest number of intermediate systems
- Fastest links.

The TSAF virtual machines reconfigure the collection, if a route becomes unavailable because of an inoperative link, system or TSAF virtual machine. TSAF then selects a new route, if one exists.

How TSAF Dynamically Configures a Collection Using Link Information

A link is a physical connection between two systems. When you start the TSAF virtual machine, you must give it the link addresses that it needs to communicate with other systems. See "Adding Links to the TSAF Virtual Machine—ADD LINK" on page 37 for information on how to add links.

TSAF sends out messages over each link. If there is a TSAF virtual machine on the other end of the link, they exchange information about the resources they manage. The TSAF virtual machines configure their own collection, based on the information they exchange. This procedure does not require an operator.

If a particular link is not operating or if there is not an active TSAF virtual machine on the other end of the link, then TSAF does not use that link. TSAF periodically checks each link defined to it. TSAF dynamically reconfigures the collection when one of the following occurs:

- A link becomes operational or inoperative
- A TSAF virtual machine becomes active or inactive.

Route Failure

If APPC/VM data does not get to its target within a specified time, the originating TSAF virtual machine starts to reconfigure the collection. After TSAF reconfigures the collection, the originating TSAF virtual machine tries to resend the data. If the SEND fails again, the TSAF virtual machine severs the APPC/VM connection.

The TSAF virtual machines send out test messages at variable intervals over each link. If a test message indicates that a link is not operating, the TSAF virtual machines reconfigure the collection and routes.

Performance Considerations

Supported Links

The TSAF virtual machine runs on any VM/SP Release 5 supported processor. These processors must support at least one of the following connections between systems:

- Channel-to-channel (CTC) links, including 3088 links
- Binary Synchronous Communication (BSC) links.

Optimizing Performance

General Performance Characteristics for TSAF

Applications that use local paths perform faster than applications that use TSAF for remote APPC/VM paths. The reason is that when communicating with a remote resource the following are involved:

- Two or more APPC/VM paths
- Two or more TSAF virtual machines
- One or more physical connection.

With local communications, only one APPC/VM path is needed because there is no need to involve the TSAF virtual machine.

TSAF performance also depends on the speed of the communication line that is routing the path. To improve performance of the remote paths, use the class A SET command with the following parameters:

FAVORED *userid*
QDROP *userid* **OFF**

userid is the userid of the TSAF virtual machine. To add improved performance, you may want to use the SET PRIORITY command. See the *VM/SP CP Command Reference* for more details on the SET command.

Line Performance Characteristics

TSAF functions that affect all the systems in a collection include:

- Identifying a new global resource in the collection
- Revoking a global resource from the collection
- Joining another collection.

How fast any of these functions complete is directly related to the speed of the slowest line that TSAF is using in the collection. A CTC line is much faster than a BSC line. So, using a BSC line can significantly slow down TSAF functions that affect all the systems in a collection.

You should also consider the transmission error rate associated with each line in the collection. A BSC line with a fixed line speed is less reliable and not as available if the number of transmission errors increases. TSAF assumes the error rate to be less than one bit in every 500,000 bits transmitted.

An error rate that is too high causes "thrashing" on the line, and the line is useless. In other words, when the error rate is high, TSAF will tend to break the communication path and mark the line "down." The performance of the entire collection can degrade when TSAF must continually change the status of the line.

Chapter 3. Running the TSAF Virtual Machine

Because the TSAF virtual machine dynamically configures itself, you, as the operator, only need to issue a few commands to operate the virtual machine.

This chapter describes each TSAF command and gives you the possible responses for each command.

Overview - TSAF Commands

The commands to run and maintain the TSAF virtual machine are:

ADD LINK adds a link to the TSAF virtual machine.

DELETE LINK
deletes a link from the TSAF virtual machine.

QUERY gets information about the TSAF configuration.

RUNTSAF starts the TSAF virtual machine.

SET ETRACE sets external tracing on or off.

STOP TSAF stops the TSAF virtual machine.

Notes about Command Syntax

Command parameters listed in braces { }, like the QUERY command description, are not optional - you must choose one.

Command parameters listed in brackets [ ] are optional - you can enter or omit them.

A Note about Messages

The following is a list of identifiers associated with the messages introduced in this chapter. The prefix:

ATS identifies the TSAF component.

The suffices:

E represents an error message.

I represents an informational message.

R represents a response message.

T represents a terminating error message.

W represents a warning message.

For example, the following would be a TSAF error message:

```
ATSLLM701E  Driver rejected the new link vdev
```

In this example, LLM in the message identifier is the name of the module that issued the message.

Adding Links to the TSAF Virtual Machine—ADD LINK

The ADD LINK command identifies a communication link to TSAF when the TSAF virtual machine is running.

ADD LINK Syntax

ADD	**LINK** *vdev*

vdev
is the virtual device address you want to use as a link.

You only need to add a link once to the TSAF virtual machine. After that, if you do not delete the link, when the TSAF virtual machine starts, it sends out messages on that link to other TSAF virtual machines to identify itself.

What Happens When You Invoke ADD LINK

If there is a device operating at the address you specify, TSAF adds the link to the TSAF table of communication links. TSAF also adds the link information to the ATSLINKS FILE on the TSAF virtual machine's A-disk. If the TSAF A-disk is accessed as R/O, TSAF lets you use this link, but TSAF does not add this link as an entry in the ATSLINKS FILE. Therefore, the next time you initialize the TSAF virtual machine this link is not defined.

For example, if you issue:

```
ADD LINK 4A0
```

the TSAF virtual machine issues a Test I/O to test the device at that virtual address. If the test is successful, along with a group of messages, you get this message:

```
ATSLLM724I Link 4A0 added
```

For bisynchronous links, you may receive the ATSL3W795I message along with the 724I message.

If you receive a few 795I messages, communication could be slow across the links. However, if you receive a group of these (over ten), the link could be inoperative or the system on the other side of the link could be down. Check to see if the other system is running. If the system is up, delete the

link and then try adding it again. If this does not work, stop TSAF (issue STOP TSAF) and restart it (issue RUNTSAF).

When the command completes succesfully, the TSAF virtual machine then uses the link to reconfigure the collection, to join a collection, or to be joined by a collection.

Messages

You may get any of the following messages:

ATSCOP004E	Parameter *parameter* is not valid
ATSCOP005E	A required parameter is missing
ATSNHR602E	Incompatible release or service level detected on link *vdev*
ATSNHR603E	Duplicate node *nodeid* detected on link *vdev*
ATSLLM700E	Link-Definition table overflow, unable to add the new link *vdev*
ATSLLM701E	Driver rejected the new link *vdev*
ATSLLM702E	Link unit address *vdev* is not valid
ATSLLM703E	Link *vdev* is not a supported link type
ATSL1A710E	Unable to allocate control block for link *vdev*
ATSL1A711E	Unable to allocate I/O buffer for link *vdev*
ATSL1A712E	Link unit address *vdev* is a duplicate
ATSLLM715E	Failed to add the definition of link *vdev* to ATSLINKS FILE A1. Return code from FSWRITE was *nnnn*.
ATSLLM724I	Link *vdev* added
ATSL3W795I	Retry limit exceeded on unit *vdev*
ATSL1A799I	Unit *vdev* is not operational

Deleting Links from the TSAF Virtual Machine—DELETE LINK

The DELETE LINK command removes a communication link from the TSAF table of communication links when the TSAF virtual machine is running.

DELETE LINK Syntax

DELETE	LINK *vdev*

vdev
is the virtual device address you no longer want to use as a link.

What Happens When You Invoke DELETE LINK

TSAF purges any link information related to the virtual address that you specified from the TSAF table of communication links. TSAF also comments the link information out of ATSLINKS FILE on the TSAF virtual machine's A-disk. For example, if you issue

```
DELETE LINK 4A0
```

the TSAF virtual machine reconfigures the collection, if necessary. If it does this successfully, you get this message:

```
ATSLLM713I Link 4A0 deleted
```

Messages

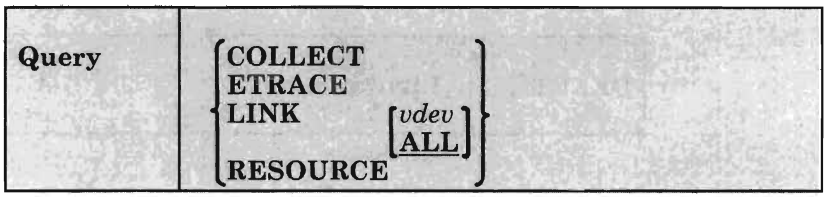
You may get any of the following messages:

ATSCOP004E	Parameter *parameter* is not valid
ATSCOP005E	A required parameter is missing
ATSLLM702E	Link unit address *vdev* is not valid
ATSLLM713I	Link *vdev* deleted
ATSLLM716E	Driver rejected the request to delete link *vdev*
ATSLLM720E	Failed to delete the definition of link *vdev* from ATSLINKS FILE A1. Return code from FSREAD was *nnnn*.
ATSLLM721E	Failed to delete the definition of link *vdev* from ATSLINKS FILE A1. Return code from FSWRITE was *nnnn*.
ATSLLM723E	Link *vaddr* not found
ATSL3W795I	Retry limit exceeded on unit *vdev*

Getting Status of the TSAF Configuration—QUERY

The QUERY command gets information about the TSAF configuration when the TSAF virtual machine is running.

QUERY Syntax



COLLECT
displays the names of the processors that are currently in the TSAF collection.

ETRACE
displays the current setting of the external tracing.

LINK
displays information about the links that TSAF currently has.

vdev
displays the link type and operational status of the link to this virtual address.

ALL
displays the link type and operational status for all of the links that TSAF currently has in its definition table.

RESOURCE
displays the current list of global resources in the collection.

What Happens When You Invoke QUERY

QUERY COLLECT

If you enter **QUERY COLLECT**, TSAF displays the node ids of the processors in the following format:

```
node01  [  node02   node03   node04   node05   node06   node07   node08  ]
```

Figure 17. Output from QUERY COLLECT

QUERY ETRACE

If you enter **QUERY ETRACE**, TSAF displays the external trace option setting as on or off. You may get either of the following responses when you issue QUERY ETRACE:

```
ETRACE ON
ETRACE OFF
```

QUERY LINK

When you issue QUERY with the LINK operand, TSAF displays the status of the link that you specify. Entering **QUERY LINK 4A0**, for example, would give you the following response:

```
Link:  4A0  Type:  CTCA  Status:  up
```

Figure 18. Output from QUERY LINK vdev

If you enter **QUERY LINK ALL**, TSAF displays the status of all the links to the TSAF virtual machine.

For two CTC links at addresses 3A0 and 4A0 and one BSC link at address 550, you get a group of messages in the following format, when entering **QUERY LINK ALL**:

```
Link:  3A0  Type:  CTCA  Status:  up
Link:  4A0  Type:  CTCA  Status:  down
Link:  550  Type:  BSC   Status:  up
```

Figure 19. Output from QUERY LINK ALL

QUERY RESOURCE

If you enter **QUERY RESOURCE**, TSAF displays the resources known in the collection. For each resource that is known throughout the collection, you may get a response in the following format:

```
resourceid at node nodeid
```

Figure 20. Output from QUERY RESOURCE

If no global resources exist in the collection, you get the following message:

```
No global resources identified
```

Messages

You may get any of the following messages:

```
ATSCOP004E Parameter parameter is not valid
ATSCOP005E A required parameter is missing
ATSLLM702E Link unit address vdev is not valid
ATSLLM722I No links are defined
ATSLLM723E Link vdev not found
```

Starting the TSAF Virtual Machine—RUNTSAF

You must be logged onto the TSAF virtual machine to issue RUNTSAF.

The RUNTSAF command starts the TSAF virtual machine. Because TSAF runs as a CMS application, you must be in the CMS environment to start TSAF.

RUNTSAF Syntax

nnn
is the number of 1K-byte blocks of virtual storage that the TSAF virtual machine's internal trace table uses. The default is 40, and TSAF rounds up to the next 4K-byte boundary.

ETRACE
sets external tracing on. This causes TSAF to write certain internal TSAF trace records externally to a CPTRAP spool file. This is the only way to get an external trace during TSAF initialization.

If you do not specify this option, external tracing is initially off. When external tracing is off, TSAF writes trace records only to TSAF virtual storage.

What Happens When You Invoke RUNTSAF

After you enter RUNTSAF, the TSAF virtual machine gets necessary parameters, such as the local node id, using the CMS IDENTIFY command. If links have been previously defined, then the TSAF virtual machine joins the collection. It does this by exchanging data with other TSAF virtual machines over the links.

When TSAF has started all of its permanent tasks successfully, you may get a group of messages, including this message:

```
ATSCTL001I Initialization is complete.  The service level
           is ssss
```

You do not receive the CMS ready message (Ready;) at the completion of this command, since you are now in TSAF. "Scenarios" on page 47 shows a sample set of messages you may receive when issuing RUNTSAF.

Messages

You may receive a number of different TSAF messages, described in the *VM/SP System Messages and Codes*. Some of the messages are as follows:

ATSCTL001I	Initialization is complete. The service level is *ssss*
ATSCTL002T	Parameter *parameter* is a duplicate or is not valid
ATSCAC006I	TSAF link statistics and session accounting records will be generated
ATSCTL013I	Trace area size is *nnn*K
ATSMJK513I	Attempting JOIN with node *nodeid* as the agent
ATSMRZ518I	RESET: collection now has size 1
ATSMYC520I	Synchronization is now NORMAL
ATSMYC521I	Collection is roughly synchronized
ATSLMN707I	Link *vdev* came up
ATSL3Z795I	Retry limit exceeded on unit *vdev*

Setting External Tracing—SET ETRACE

The SET ETRACE command lets you enable or disable external tracing.

SET ETRACE Syntax

The command format is:

ON
causes TSAF to write certain internal TSAF trace records externally to a CPTRAP spool file.

OFF
causes TSAF to write TSAF trace records only to TSAF virtual storage. No external tracing is done.

What Happens When You Invoke SET ETRACE

If you enter **SET ETRACE ON**, TSAF starts to write trace records to a CPTRAP spool file, and you get this message:

```
ATSCOP010I  External trace started
```

If you enter **SET ETRACE OFF**, TSAF stops external tracing, and you get this message:

```
ATSCOP011I  External trace ended
```

Messages

You may get any of the following messages:

ATSCOP004E	Parameter *parameter* is not valid
ATSCOP005E	A required parameter is missing
ATSCOP010I	External trace started
ATSCOP011I	External trace ended

Stopping the TSAF Virtual Machine—STOP TSAF

The STOP TSAF command stops running the TSAF virtual machine.

STOP TSAF Syntax

STOP	TSAF

What Happens When You Invoke STOP TSAF

When TSAF accepts the STOP TSAF command, you get this message:

```
ATSCTL003I Termination is in progress
```

After the TSAF virtual machine has stopped, you get the CMS "ready" message (Ready;).

Messages

You may get any of the following messages:

ATSCTL003I	Termination is in progress
ATSCOP004E	Parameter *parameter* is not valid
ATSCOP005E	A required parameter is missing

Scenarios

The following two examples show the results of issuing some of the TSAF commands.

RUNTSAF with No Errors

The following example shows what typically could happen when you issue the RUNTSAF command:

```
RUNTSAF 250

ATSCTL013I Trace area size is 252K
ATSCAC006I TSAF link statistics and session accounting records will
           be generated
ATSMRZ518I RESET: collection now has size 1
ATSCTL001I Initialization is complete. The service level is 0500.
ATSMRX520I Synchronization is now NORMAL
ATSLMN707I Link 500 came up
ATSMDO515I JOIN in progress for node SMSNODE
ATSMYC521I Collection is roughly synchronized
ATSMYC520I Synchronization is now NORMAL

QUERY COLLECT

GDLS5    SMSNODE

QUERY LINK

Link:  500  Type:  CTCA  Status:  up

STOP TSAF

ATSCTL003I Termination is in progress.
Ready;
```

RUNTSAF with Errors

The following example shows what may happen when a link is not operational:

```
RUNTSAF 200

ATSCTL013I Trace area size is 200K
ATSCAC006I TSAF link statistics and session accounting records will
           be generated.
ATSL1Y708E An attempt to reset link 600 has failed
ATSL1A799I Unit 600 is not operational
ATSMRZ518I RESET: collection now has size 1
ATSCTL001I Initialization is complete. The service level is 0500.
ATSL1Y708E An attempt to reset link 600 has failed
ATSMRX520I Synchronization is now NORMAL

QUERY COLLECT

GDLS5

QUERY LINK

Link:  600  Type:  CTCA  Status:  down

ADD LINK 500

ATSL1Y708E An attempt to reset link 500 has failed
ATSL1A799I Unit 500 is not operational
ATSLLM724I Link 500 added
ATSL1Y708E An attempt to reset link 500 has failed

STOP TSAF

ATSCTL003I Termination is in progress
Ready;
```

Chapter 4. Generating TSAF Accounting and Link Statistics

There are four different TSAF accounting records:

1. **Initialization accounting record** — for information about when TSAF was active
2. **Session accounting record** — for information about the user virtual machine and the resource manager virtual machine
3. **Link statistics record** — for information about the load on a link over a period of time
4. **Termination accounting record** — for information about TSAF termination.

If you want TSAF to generate accounting records, you should specify the OPTION ACCT statement in the TSAF directory entry, as shown in "Modifying the TSAF System Directory Entry" on page 11. During TSAF initialization, you get one of the following messages:

- If you specify OPTION ACCT, you get:

```
ATSCAC006I TSAF link statistics and session accounting
           records will be generated
```

- If you do not specify OPTION ACCT, you get:

```
ATSCAC007I No TSAF link statistics or session accounting
           records will be generated
```

You can modify the SYSACNT macro to specify where CP should store the accounting records. See the *VM/SP Planning Guide and Reference* for more information on the SYSACNT macro.

Initialization Accounting Record

TSAF produces the initialization accounting record during its initialization. You can use this record, along with the TSAF termination record, to find out when TSAF was active.

Format of the Initialization Accounting Record

The format of the initialization accounting record is:

Column	Contents
1	Contains the CP-provided userid of the TSAF virtual machine.
9	Reserved for IBM use.
17	Lists the date and time when the accounting record was generated, 'MMDDYYHHMMSS' (month, day, year, hours, minutes, and seconds).
29	Reserved for IBM use.
75	'1ATS' identifies record as the initialization accounting record.
79	'C0' identifies the accounting record code that CP provides.

Session Accounting Record

Use the session accounting record to find out how long a user is connected to a resource manager, using a specific resource. The record also has information that tells you how much data the user virtual machine and the resource manager exchanged.

TSAF generates a session accounting record at the following times:

- Every hour
- When an APPC/VM session ends (SEVER)
- When the TSAF virtual machine stops.

The TSAF virtual machine (on the same system as the virtual machine that started the connection) issues the DIAGNOSE code X'4C' instruction to generate the records. For an application and/or server session, the system on which the application runs would generate the records.

Format of the Session Accounting Record

The format of the session accounting record is:

Column	Contents
1	Contains the CP-provided userid of the TSAF virtual machine.

Column	Contents
9	Contains the userid of the application virtual machine.
17	Lists the date and time when the accounting record was generated, 'MMDDYYHHMMSS' (month, day, year, hours, minutes, and seconds).
29	Contains the resource name.
37	Lists how long, in seconds, the session was active (unsigned binary fullword). This is the time since the session started or since the last accounting record was taken for this session.
41	Lists how many bytes of data were sent (unsigned binary fullword).
45	Lists how many bytes of data were received (unsigned binary fullword).
49	Reserved for IBM use.
75	'SATS' identifies record as a session accounting record issued by TSAF.
79	'C0' identifies the accounting record code that CP provides.

Link Statistics Record

Use the link statistics record to determine the load on the link over a period of time.

TSAF generates a link statistics record at the following times:

- Every hour when a link is up
- When the system declares a link down
- When the TSAF virtual machine stops.

Both ends of the link generate link statistics records with the DIAGNOSE code X'4C' instruction.

Format of the Link Statistics Record

The format of the link statistics record is:

Column	Contents
1	Contains the CP-provided userid of the TSAF virtual machine.
9	Reserved for IBM use.
17	Lists the date and time when the accounting record was generated, 'MMDDYYHHMMSS' (month, day, year, hours, minutes, and seconds).
29	Contains the unit address of the link, four characters in the form 'vdev'.
33	Reserved for IBM use.
37	Lists how many bytes of data were sent since the link came up, or since the last accounting record was generated for this link (unsigned binary fullword).
41	Lists how many bytes of data were received (unsigned binary fullword).
45	Reserved for IBM use.
75	'LATS' identifies record as a link statistics record issued by TSAF.
79	'C0' identifies the accounting record code that CP provides.

Termination Accounting Record

TSAF produces the termination accounting record during normal TSAF termination as well as in some types of abnormal termination. When TSAF writes the TSAF termination accounting record, you can be sure that all applicable session accounting and link statistics records have also been written. On the other hand, if TSAF abnormally terminates without writing the termination accounting record, then some session accounting and link statistics data may be lost.

Format of the Termination Accounting Record

The format of the termination accounting record is:

Column	Contents
1	Contains the CP-provided userid of the TSAF virtual machine.
9	Reserved for IBM use.

Column	Contents
17	Lists the date and time when the accounting record was generated, 'MMDDYYHHMMSS' (month, day, year, hours, minutes, and seconds).
29	Reserved for IBM use.
75	'0ATS' identifies record as the termination accounting record.
79	'C0' identifies the accounting record code that CP provides.

Chapter 5. Collecting TSAF Problem Diagnosis Information (Serviceability)

The three ways that you can collect error information for problem diagnosis within TSAF are:

1. Using console logs
2. Using dumps
3. Using system trace data.

In addition, "Interactive Service Queries" on page 58 describes how the TSAF QUERY command can also provide you with problem diagnosis information.

Note: The TSAF operator does not necessarily diagnose problems, especially from the TSAF virtual machine. Dumps and system trace data are usually used by the system programmer or whoever is responsible for diagnosing system problems.

Summary of Steps to Follow When a TSAF Abend Occurs

When a TSAF abend occurs, you must do the following steps:

1. Collect information about the error.
   - Save the console sheet or spooled console output from the TSAF virtual machine.
   - Save and process any dumps that TSAF produces.

     When an abend occurs in TSAF, either because TSAF issued an abend or because a TSAF or CMS operation caused a program exception, TSAF produces a dump via the CP VMDUMP command (described in the *VM/SP CP Command Reference*). CP sends the dump to TSAF's virtual reader.
   - Save any CPTRAP file that contains TSAF data.
2. Collect other types of information about system status.
   - The status of real and virtual devices that TSAF is using

- The system load at the time of the failure on any systems using TSAF and the status of each system (for example, did another system abend?)
- The types of applications that are using TSAF at the time and any information about them
- The physical connection configuration of the systems in use.

3. Recover from the abend to continue processing.

   After TSAF creates a dump, TSAF then issues a CP SYSTEM RESET command. If the CONCEAL option is on, as recommended, CP automatically IPLs CMS. Otherwise, you, the operator, must re-IPL CMS. Similarly, if TSAF is not invoked from the PROFILE EXEC, you must restart the TSAF virtual machine.

VM/SP System Messages and Codes lists the TSAF abend codes and their causes. The *VM Diagnosis Guide* contains a complete description of how to diagnosis problems in TSAF.

Using the Console Log

TSAF provides informational messages, as well as error messages, that may help you with problem determination. To keep track of the console messages, issue:

SPOOL CONSOLE START TO *userid*

userid can be the userid of the TSAF virtual machine or another virtual machine userid to whom you want TSAF to send the console log. You may want to add this to TSAF's PROFILE EXEC so a console log is always created.

To close the console log, issue:

SPOOL CONSOLE CLOSE

The log of messages received is sent to the specified userid. See the *VM/SP CP Command Reference* for more information on the SPOOL command.

TSAF provides additional information at the time of an abend to help you diagnose the problem. The console log contains information about the abend, such as the abend code, the program old PSW (Program Status Word), and the contents of the general purpose registers. TSAF also attempts to determine the displacement of the module in which the abend occurred and the displacement of the calling module.

The following sample shows some of the messages that TSAF may issue in response to an abend condition:

```
ATSCAC999T TSAF system error
ATSCAB017I Abend code ATS999 at 022730
ATSCAB018I Program old PSW is FFE002FF 40022730
GPR0-7  00022FFC 000003E7 00022FDA 00052BC0 00208080 00020C58 0033E811 00000001
GPR8-F  7F3B78AF 603C0000 00020B64 00022D6F 50021D70 00022B48 40022718 00023FB0
ATSCAB019I Abend modifier is ATSCAC
ATSCAB021I Failure at offset 0A06 in module ATSCAC dated 86.020
ATSCAB022I Called from offset 04B4 in module ATSSCN dated 86.078
ATSCAB023I VMDUMP ATSCAB*ATSCAB1 05/28/86 16:02:06 taken
```

Figure 21. Sample TSAF Console Log

Using TSAF Dumps to Diagnose Problems

IPCS is a dump analysis and problem tracking tool in VM/SP. You can use IPCS to collect and diagnose problem data for the TSAF virtual machine. The console listing, as described in "Using the Console Log" on page 56, may help you diagnose problems without using dumps.

Because the TSAF virtual machine is not set up to process dumps, you need to transfer the dump file to the appropriate virtual machine. You can find out the userid of the virtual machine set up to process dumps by looking in the DMKSYS ASSEMBLE file. The SYSDUMP parameter on the SYSOPR macro specifies the appropriate virtual machine.

The steps involved in using dumps to diagnose problems are:

1. Create a TSAF IPCS map (if it does not already exist), using the IPCS MAP TSAF command.

2. Create the TSAF dump, using the following CP command:

   ```
   VMDUMP SYSTEM FORMAT TSAF
   ```

3. Process the TSAF dump, using the IPCS IPCSDUMP command.

4. Diagnose the TSAF dump by doing the following:

   - Look at the symptom record.
   - Use the FDISPLAY subcommand of IPCS DUMPSCAN to display dump information.
   - Format and display trace records, using the TRACE subcommand of IPCS DUMPSCAN.

5. Optionally, print the TSAF dump, using the IPCS PRTDUMP command.

See the *VM Diagnosis Guide* for a complete description of using TSAF dumps.

Using System Trace Data to Diagnose Problems

TSAF maintains an internal trace table within the TSAF virtual machine. You can use the IPCS DUMPSCAN TRACE subcommand to display the internal trace table entries. TSAF also writes trace entries to the system CPTRAP file. You can then use TRAPRED to view TSAF entries.

The TSAF SET ETRACE command lets you enable or disable external tracing for the TSAF virtual machine. If you want to collect TSAF trace records, issue the following from the TSAF virtual machine before CPTRAP is started:

```
SET ETRACE ON
```

When you set external tracing on, certain internal TSAF trace records are written externally to a CPTRAP spool file. A complete description of the SET ETRACE command is in "Chapter 3. Running the TSAF Virtual Machine" on page 35.

The CPTRAP command collects TSAF information in a reader file. The privilege class C, QUERY CPTRAP command gets information about the CPTRAP. This information helps with problem determination. For more information about the CP commands, CPTRAP and QUERY class C, see the *VM/SP CP Command Reference.*

To access the CPTRAP reader file and review the entries contained in that file, use the TRAPRED command. For more information about CPTRAP, TRAPRED, and the trace table entry formats, see the *VM Diagnosis Guide.*

Interactive Service Queries

The TSAF QUERY command, issued from the TSAF virtual machine, can give you more information to help you diagnose problems. The TSAF QUERY command gives you data about the TSAF configuration when the TSAF virtual machine is running.

- QUERY COLLECT displays the processor names that are currently in the TSAF collection.
- QUERY ETRACE displays the current setting of the external tracing.
- QUERY LINK displays information about the links that TSAF currently has.
- QUERY RESOURCE displays the current list of global resources in the collection.

See "Getting Status of the TSAF Configuration—QUERY" on page 40 for more specific information about this command.

Part Two: TSAF Program Communication Services

This part describes the VM program-to-VM program communication functions provided by TSAF. These functions are provided by:

- The APPC/VM program interface, and
- A set of IUCV (VM-unique) functions for use with APPC/VM.

The APPC/VM program interface depends upon IUCV formats and protocols. APPC/VM provides communication only among VM processors. IUCV provides functions that are related to APPC/VM communication paths and asynchronous CP interrupts. The IUCV functions are not a part of the APPC architecture.

These chapters introduce communication functions that you can use to write APPC/VM application programs.

- "Chapter 6. APPC/VM (VM-to-VM) Communications" on page 61 gives a general overview of APPC/VM communications by describing APPC/VM paths and states. The chapter talks about the basic APPC/VM functions of:

  - Connecting to a resource
  - Sending and receiving data
  - Severing communications.

  This chapter introduces APPC/VM functions that let you send other kinds of data such as confirmation messages, error messages, and attention messages.

  In addition, this chapter describes the IUCV functions that are complementary to APPC/VM but unique to VM.

- "Chapter 7. APPC/VM and IUCV Communication Functions" on page 81 includes complete descriptions of both the APPC/VM communication functions and the related IUCV functions. Each function description includes the syntax, parameters, condition codes, exceptions, and states associated with each function.

- "Chapter 8. APPC Verbs Mapped with APPC/VM Functions" on page 185 shows the mapping of the Systems Network Architecture

Logical Unit Type 6.2 (SNA LU 6.2) APPC functions to the APPC/VM functions. The chapter begins with an overview of APPC conversations, return codes, and interrupts. It then lists the APPC/VM parameters that correspond with the APPC verbs.

Note that Appendix C, "Sample TSAF User Program" on page 249 and Appendix D, "Sample TSAF Resource Manager Program" on page 257 contain sample APPC/VM programs to help you understand the use of the various APPC/VM and IUCV functions.

Chapter 6. APPC/VM (VM-to-VM) Communications

Overview of VM-to-VM Communications

The Advanced Program-to-Program Communication/VM (APPC/VM) application program interface (API) lets TSAF support resources transparently among interconnected VM systems. Applications that use APPC/VM support can communicate with applications at remote systems within the same TSAF collection. APPC/VM lets users pass any amount of information between virtual machines within a collection of systems that each have the TSAF virtual machine running.

TSAF provides functions, with the APPCVM and IUCV macros, to:

- Establish and sever paths
- Send and receive messages
- Determine the presence of and describe pending messages.

The complete APPC/VM function descriptions and associated IUCV function descriptions are in "Chapter 7. APPC/VM and IUCV Communication Functions" on page 81.

To start APPC/VM communications, you must first issue DCLBFR to declare a buffer for receiving APPC/VM interrupts. Then, you can establish a path to a program using the CONNECT function.

The target of your CONNECT attempt may optionally receive the allocation data provided. The allocation data contains VM information and the Attach FMH5 (discussed on page 99.) If the target of your CONNECT wants to communicate, then the target should issue the ACCEPT function. When the ACCEPT completes, the target virtual machine is in RECEIVE state (unless an intermediate communication server has provided the ACCEPT). On the other hand, if the target of your CONNECT does not want to communicate, then it should issue the SEVER function.

Note: The ACCEPT function is not part of the APPC architecture and is unique to VM.

After your CONNECT completes, you can start sending data with the SENDDATA function.

APPC/VM Communications

APPC/VM Paths

An APPC/VM path is a logical connection between one or more virtual machines. Information flows on APPC/VM paths using IUCV and TSAF unique protocols. To establish an APPC/VM path between two virtual machines, at least one of the virtual machines must be authorized in the IUCV directory control statement (described in "IUCV Directory Control Statement for *IDENT Authorization" on page 19).

A single virtual machine can have up to 65,536 IUCV and APPC/VM paths defined. Two virtual machines can have more than one path between them. Communication can occur over any and all paths at the same time.

A path is created when the source virtual machine issues the CONNECT function and the target virtual machine issues the ACCEPT function. Once the path is created, communication can begin. The target virtual machine can prevent the path from being established by issuing the SEVER function. Either virtual machine can break an established path with the SEVER function.

You identify a path by including its id in the **PATHID** parameter of the pertinent APPC/VM functions.

Your Communication Partner

The virtual machine to which you are connected, through the APPC/VM path, is known as your *communication partner*, and you are that virtual machine's *communication partner*.

APPC/VM States

The APPC/VM interface depends upon a half-duplex communication protocol. This means that only one of the communication partners can send data at any given time. Because of this, APPC/VM uses states to define which communicator can issue what functions at any given time. When you or your communication partner issues an APPC/VM function, the state of the conversation may change. If your virtual machine is communicating with different virtual machines through various paths, it may be in different states on different paths at the same time.

The states that are possible within an APPC/VM application are:

- RESET
- CONNECT
- SEND
- RECEIVE
- CONFIRM
- SEVER.

Figure 22 describes each state and the commands that you may use from each state.

State	When the State Occurs	Functions You Can Issue
RESET	• Before the program sets up a path. • After the program issues a SEVER or RTRVBFR.	CONNECT
CONNECT	• After the program issues CONNECT, but before the CONNECT completes.	IUCV SEVER
	• After the program receives a connection pending interrupt, but before it ACCEPTs the connection.	IUCV SEVER RECEIVE IUCV ACCEPT
SEND	• After the CONNECT completes. • After you receive notice that your communication partner issued RECEIVE or SENDDATA RECEIVE=YES. • After SENDERR completes normally.	SENDDATA SENDCNF RECEIVE SENDERR SENDREQ SEVER TYPE=NORMAL SEVER TYPE=ABEND SENDREQ
RECEIVE	• After the program ACCEPTs a connection. • After RECEIVE completes. • After any function completes, and you receive notice that your partner has issued a SENDERR. • After you issue SENDCNFD, in response to your partner's SENDCNF TYPE=NORMAL.	RECEIVE SENDERR SENDREQ SEVER TYPE=ABEND
CONFIRM	• After the program receives a confirmation request from its communication partner.	SENDCNFD SENDERR SENDREQ SEVER TYPE=ABEND
SEVER	• After a SEND or RECEIVE completes with an indication that your communication partner issued a SEVER. • After you issue SENDCNFD, in response to your partner's SENDCNF TYPE=SEVER.	SEVER TYPE=NORMAL

Figure 22. APPC/VM States

APPC/VM Interrupts

In APPC/VM, you may receive notification of pending functions through external interrupts. The IUCV SETCMASK and SETMASK functions let you enable and disable interrupts for your virtual machine. These functions are described in in "IUCV SETCMASK" on page 161 and "IUCV SETMASK" on page 165.

Interrupts are caused by actions taken by the virtual machine on the other end of the local APPC/VM path. Interrupts indicate locally pending and locally completed functions. For example, a message pending interrupt indicates that a message was sent to you by a virtual machine on your local system. This virtual machine may be the machine you are communicating with or an intermediate communication server virtual machine.

When a function completes (you get a function complete or connection complete interrupt), you should not assume that the actual target application performed any action to cause your function to complete.

The possible interrupts you may receive in VM-to-VM communication (APPC/VM) fall into two categories. The first type of interrupt signals that your communication partner has invoked some function, independent of your actions. These interrupts are:

- Connection pending
- Message pending
- SENDREQ (Request-to-Send)
- SEVER.

The second type of interrupt signals the completion of a function that you initiated. These interrupts are:

- Connection complete
- Function complete.

Connection Pending External Interrupt

You get a connection pending interrupt, IPTYPE = X'81', when a virtual machine issues the APPCVM CONNECT function in order to connect to your virtual machine.

The connection pending external interrupt is shown in "APPCVM CONNECT" on page 97.

Message Pending External Interrupt

You get a message pending interrupt, IPTYPE = X'89', when your communication partner issues an APPCVM function which you should RECEIVE. Your communication partner issuing any of the following APPCVM functions can cause a message pending interrupt:

- RECEIVE

- SENDCNF
- SENDDATA
- SENDERR.

You only get a message pending interrupt if you are in RECEIVE state on the corresponding path. The message pending external interrupt is shown in "APPCVM SENDDATA" on page 147.

SENDREQ (Request-to-Send) External Interrupt

You get a SENDREQ interrupt, IPTYPE = X'88', when your communication partner issues the APPCVM SENDREQ function to request to send data.

The SENDREQ external interrupt is shown in "APPCVM SENDREQ" on page 160.

SEVER External Interrupt

You get a SEVER interrupt, IPTYPE = X'83', when the virtual machine to which you are connected or trying to connect to issues the APPCVM SEVER, IUCV SEVER, or IUCV RTRVBFR function. You could also get a SEVER interrupt when the virtual machine to which you are connected or trying to connect to resets its virtual machine or logs off.

The SEVER external interrupt is shown in "APPCVM SEVER" on page 175.

Connection Complete External Interrupt

You get a connection complete interrupt, IPTYPE = X'82', when you issue the APPCVM CONNECT WAIT = NO function and the virtual machine on the other end of the local APPC/VM path accepts the connection.

The connection complete external interrupt is shown in "APPCVM CONNECT" on page 95.

Function Complete External Interrupt

You get a function complete interrupt, IPTYPE = X'87', when the function that you issued completes. The completion of any of the following APPCVM functions can cause a function complete interrupt:

- RECEIVE
- SENDCNF
- SENDDATA
- SENDERR.

The function complete interrupts are shown in the following sections:

- "APPCVM RECEIVE" on page 116
- "APPCVM SENDCNF" on page 127
- "APPCVM SENDDATA" on page 141
- "APPCVM SENDERR" on page 152.

Communication Performance

You can reduce the overhead involved in reflecting APPC/VM external interrupts to the virtual machine if the buffer you declare on the DCLBFR function is entirely within one page. You can further reduce overhead if the buffer is entirely within page 0 of the virtual machine.

CMS and GCS IUCV Support

If you are writing applications for the CMS environment, use the CMS IUCV support, as described in the *VM System Facilities for Programming*. If you are writing applications for the GCS environment, use the GCS IUCV support, as described in the *VM/SP Group Control System Command and Macro Reference*. Using CMS and GCS IUCV support improves the coexistence characteristics of multiple APPC/VM and IUCV programs running in the same virtual machine.

Connecting to Another Virtual Machine

Before you can attempt to connect to a virtual machine, you should have issued DCLBFR to declare a buffer for CP to store external interrupt data.

To connect to a virtual machine and establish an APPC/VM connection, you must issue the APPC/VM CONNECT function. You can use the **RESID** parameter on the CONNECT function to specify a resource name located anywhere within the TSAF collection. Any of the following could happen:

- If a virtual machine on another system within the TSAF collection identified itself as managing the resource (**RESID** parameter specified on the IUCV *IDENT statement in its directory entry), the TSAF virtual machine routes the connection to that virtual machine.

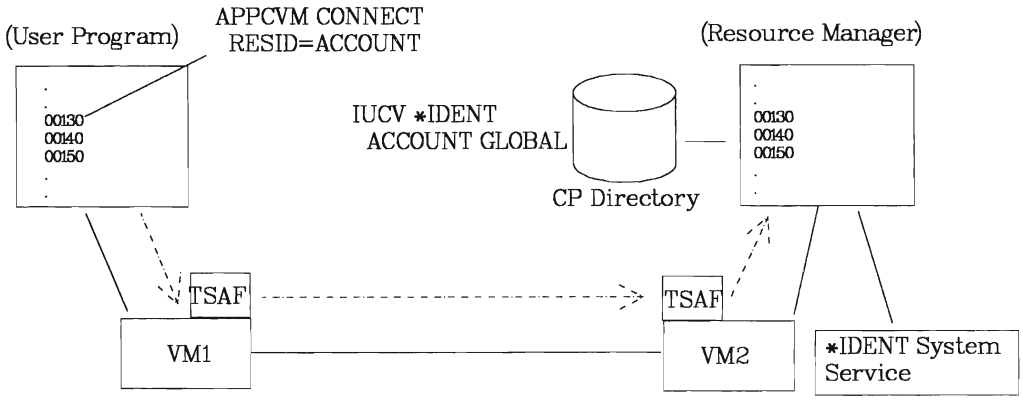


Figure 23. User Program Connecting to a Resource Manager Program

- If the resource is on your own system, CP routes you there without the need for the TSAF virtual machine.
- If no virtual machine within the collection identified itself as the resource manager, the connection fails.

The WAIT=YES or WAIT=NO parameter controls whether or not the CONNECT is synchronous or asynchronous.

When you issue an asynchronous CONNECT (WAIT=NO), your virtual machine regains control before the connection is complete. You can issue any APPC/VM function on any path except the path that you are trying to establish with the CONNECT. The only function that you can issue on the path you are trying to establish is IUCV SEVER.

Accepting or Rejecting a Connection

When you issue the CONNECT, your communication partner gets a connection pending interrupt. Your partner should examine the interrupt before accepting or rejecting the connection. The interrupt contains:

- The resource id for which the connection is being made
- An indication that an APPC path is being established
- An indication as to whether or not synchronization functions, SENDCNF and SENDCNFD, can be issued on this path
- The userid of the connecting virtual machine.

Allocation data, which contains VM information and the FMH5, is also available to your communication partner (refer to "Allocate Data That Your Communication Partner May Receive" on page 98 for more detail). Your partner can issue a RECEIVE to get the data before accepting the connection.

After examining all this data, your communication partner can do either of the following in response to your connect request:

- ACCEPT - if it wants to communicate with your virtual machine, making sure to specify the path id that was on the connection pending interrupt.

  Note: The ACCEPT function is not part of the APPC architecture and is unique to VM.

- IUCV SEVER - if it does not want to communicate with your virtual machine.

Sending and Receiving Data

When your virtual machine issues the APPCVM CONNECT and your communication partner ACCEPTs the connection:

- Your virtual machine is in SEND state.
- The communication partner is in RECEIVE state.

You can now send data, with the SENDDATA function. Remember that you can only send data when you are in the SEND state and receive data when you are in the RECEIVE state. As you send data, your communication partner is notified through one or more message pending interrupts. Your partner can then use the RECEIVE function to receive the data.

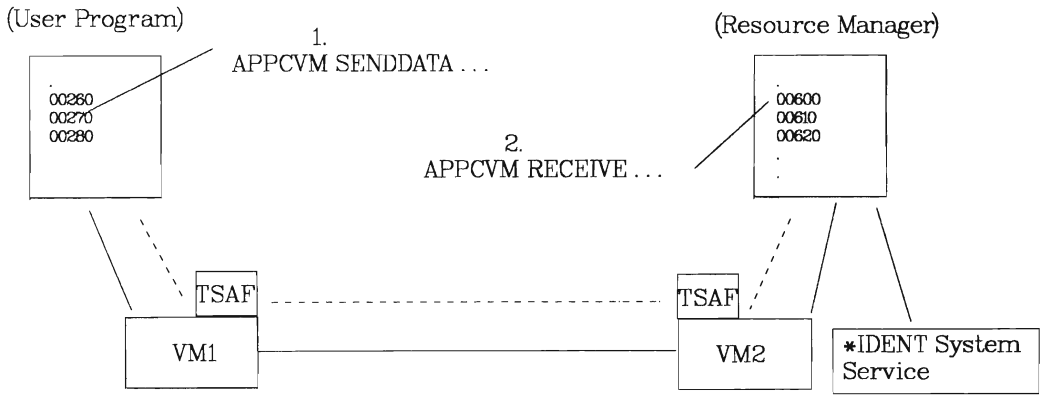


Figure 24. User Program and Resource Manager Program Sending and Receiving

How APPC Data Is Sent

In APPC/VM, you send data from source to target by issuing one or more SENDDATA commands. The data that a single SENDDATA includes is referred to as a message. Each application involved in the communication determines the amount of data sent in each message (the message size). You may choose message sizes based on whatever is important to your application, such as the size of free storage buffers or efficient storage utilization.

APPC defines a logical construct on top of the message protocol called a *logical record*. This is so applications can communicate without depending on each other's buffering priorities and the priorities of intermediate communication servers. APPC/VM requires that all data provided by the application be in a logical record format. A logical record consists of a 2-byte LL (logical record length) field followed by a data field. The logical record length has the 15-bit length of the record, plus a high-order bit (APPC/VM does not examine the high-order bit). The data field can range from 0 to 32,765 bytes long.

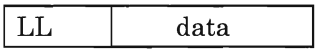


Figure 25. An APPC/VM logical record

Although the logical records are limited to 32,765 bytes, the amount of data that may be sent on a single SENDDATA is 2^{31} minus 1.

Message Lengths

Your applications should not depend on the length of APPC/VM messages. The APPC/VM message is only the vehicle for transporting the logical records. A single logical record can span multiple APPC/VM messages, or a single APPC/VM message may contain multiple logical records.

Even if you are responsible for both ends of the communication, do not set up your receiving application so that the length of the message determines the length of the logical record. Intermediate communication servers such as the TSAF virtual machine, because of their buffering needs, may break up your single SENDDATA into multiple SENDDATAs or combine numerous SENDDATAs.

When a virtual machine on the local system issues the SENDDATA, the length in the message pending interrupt is the actual length of the data sent by the SENDDATA. On the other hand, when a virtual machine on a remote system issues the SENDDATA, the length in the message pending interrupt is the length of the data sent by TSAF's SENDDATA. The length of TSAF's SENDDATA depends on the size of the TSAF virtual machine frames.

When communicating through the TSAF virtual machine, there is not a one-to-one correspondence between the number of messages you send and the number of messages that the target virtual machine receives. For example, you may send a message of zero length followed by a message of

100 bytes. However, the target virtual machine may get only one message with the length of 100 bytes.

SEND-RECEIVE Scenario

A typical SEND-RECEIVE sequence may look like the following. In this particular scenario, assume that the resource manager has already defined itself as managing the resource.

1. You declare a buffer for VM to store external interrupt data, using DCLBFR.

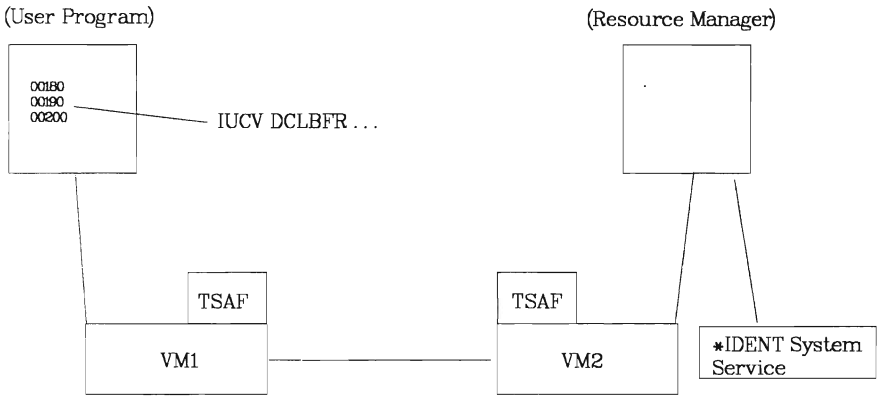


Figure 26. Declaring an APPC/VM Buffer

2. You connect to a resource manager, using CONNECT.

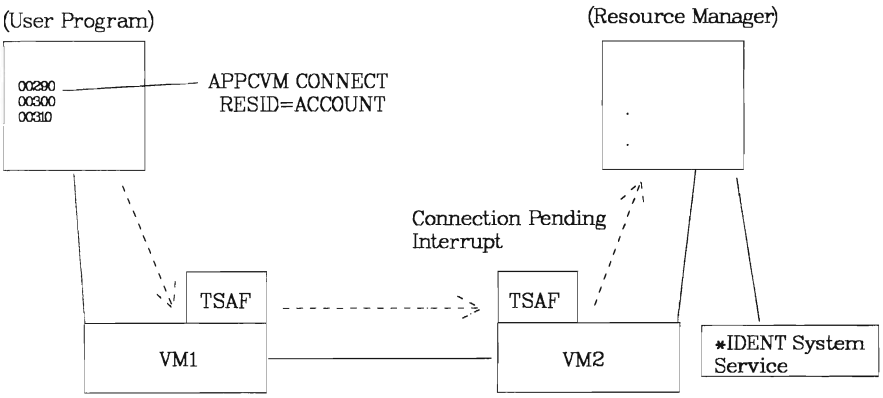


Figure 27. User Program Connecting to Resource Manager

3. Your communication partner accepts the connection, using ACCEPT.

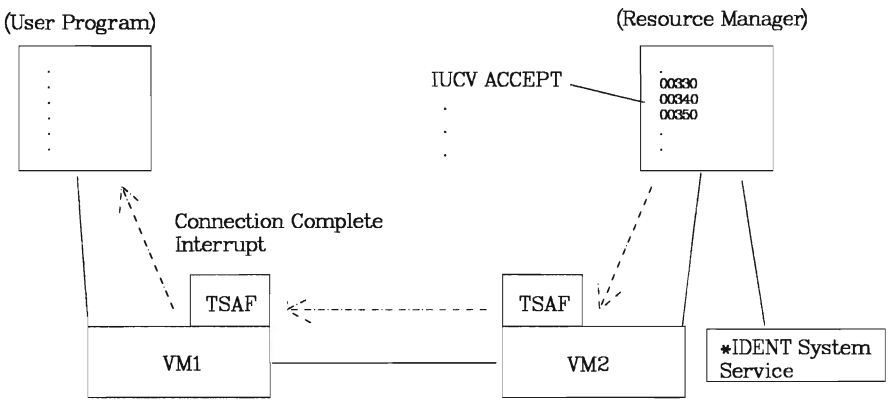


Figure 28. Resource Manager Accepting the Connection

4. You send data, using SENDDATA RECEIVE = NO.

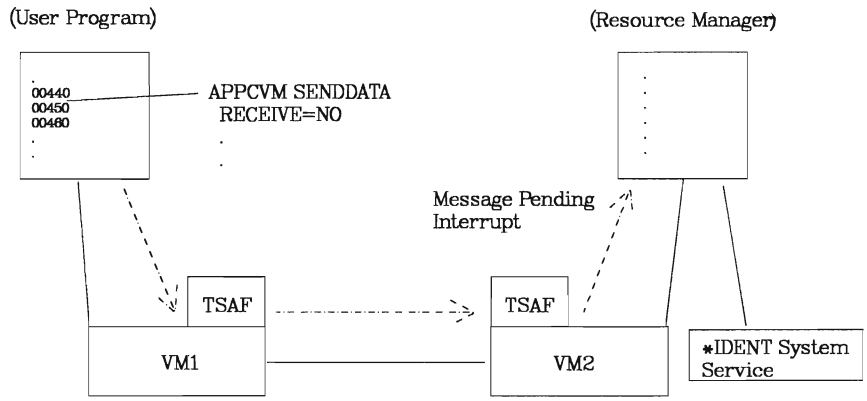


Figure 29. User Program Sending Data via SENDDATA

When the SENDDATA RECEIVE = NO completes,

- The data has been copied into your communication partner's virtual machine
- Your virtual machine remains in SEND state.

Your communication partner is in RECEIVE state as it receives the data.

Note: When you are communicating asynchronously (specified WAIT = NO on the SENDDATA statement), you must be careful not to use any buffers that are involved in IUCV functions, until the function actually completes. After you issue a SENDDATA with WAIT = NO, do not assume that the data is moved out of the buffer until you receive a function complete indication.

If your virtual machine started the connection but you do not have data to send, issue RECEIVE to switch the conversation state.

5. Your communication partner receives the data, using RECEIVE.

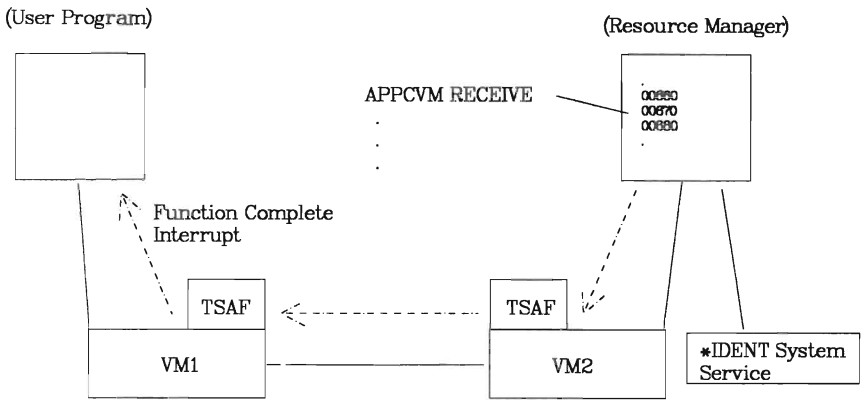


Figure 30. Resource Manager Receiving Data

Your communication partner does not get a message until it issues RECEIVE. In this example, your partner did not allocate any RECEIVE areas for a certain path, and, instead, waited until it received a message pending interrupt. Then your partner issued a RECEIVE for the amount of data pending.

The length of the pending message does not necessarily correspond to the logical record length. The length is the amount of data currently available to be received. Your communication partner can issue a RECEIVE for any length that is convenient; part of the data, all of the data, or more data than is currently pending. When your partner issues the RECEIVE for more data than is available, the RECEIVE does not complete until one of the following occurs:

- You send enough additional data to fill your communication partner's RECEIVE area
- You send your partner an indication of something other than data (for example, switch conversation states or issue a SENDERR).

6. When you finish sending all the data that you want to, issue one of the following to enter RECEIVE state:

   - RECEIVE specified with a RECEIVE area whose length could be zero.

     Your communication partner receives notice of this in IPWHATRC in the function complete data. Note that you cannot issue RECEIVE if you have started, but not yet finished, sending a logical record on the path.

   - SENDDATA RECEIVE = YES

     Your communication partner receives notice of this as if you had issued a SEND followed by a RECEIVE. See "APPCVM SENDDATA" on page 134 for more information on the SENDDATA RECEIVE = YES.

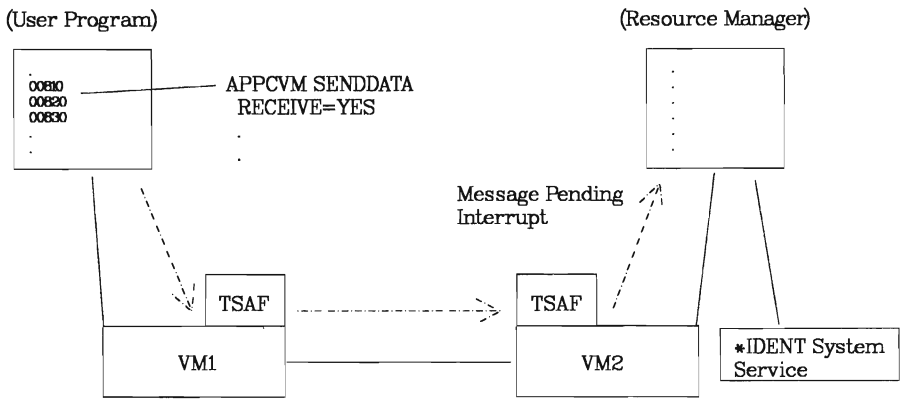


Figure 31. User Program Sending and Switching States

When your RECEIVE or SENDDATA RECEIVE = YES completes,

a. Your virtual machine enters RECEIVE state
b. Your communication partner is switched to SEND state as soon as it receives all outstanding data.

When you issue a RECEIVE with WAIT = NO, VM only notifies you when the RECEIVE is complete. See "RECEIVE Completion" on page 120 for more information on when a RECEIVE is complete.

Interrupts That You Receive

When you issue SENDDATA RECEIVE = NO WAIT = NO (Step 4 on page 71), followed by a RECEIVE (Step 5 on page 71), you get two function complete interrupts. You get the first interrupt when the data you sent is copied out of your SEND buffer. Then, when you issue the RECEIVE (Step 5), you get a second function complete interrupt when the communication partner either:

- Sends back enough data to fill your RECEIVE area, or
- Changes the conversation states.

On the other hand, when you issue SENDDATA RECEIVE = YES, you get only one function complete interrupt. You get this interrupt when the communication partner either:

- Sends back enough data to fill your RECEIVE area, or
- Changes the conversation states.

Requesting Confirmation

When you are sending data to another virtual machine, you may want to confirm that you should continue to send. To do this, you would invoke SENDCNF TYPE=NORMAL. The SENDCNF is not complete until your partner issues one of the following functions:

- SENDCNFD, to indicate that the sender may continue
- SENDERR, to indicate that something is wrong
- SEVER, to end communications.

When you are sending data to another virtual machine, you may want to sever the connection. You can confirm this decision with your communication partner, by issuing SENDCNF TYPE=SEVER.

To use SENDCNF and SENDCNFD, you or your communication partner must have specified SYNCLVL=CONFIRM on the CONNECT function.

Note: You cannot issue SENDCNF if you have started, but not yet finished, sending a logical record on the path.

Signalling an Error

When you sense that there is an error in the communication, whether you are in SEND or RECEIVE state, you can issue SENDERR. This signals your communication partner and causes a break in the normal SEND/RECEIVE logical record sequence.

If you are in RECEIVE state, and issue SENDERR:

1. The error notice goes to your communication partner.
2. Your virtual machine enters SEND state, when the SENDERR completes.

If you are in SEND state, and issue SENDERR:

1. The error notice goes to your communication partner.
2. Your virtual machine remains in SEND state.

There may be a time when your communication partner, who is in RECEIVE state, sends a SENDERR before it receives your SENDERR notice. Because your partner is in RECEIVE state, its SENDERR is invoked over yours. In this case, your communication partner would enter SEND state and you would be switched to RECEIVE state.

The SENDERR code that your partner receives (IPCODE) depends on your conversation state and whether or not your SENDERR truncated a logical record. For example, if you are in SEND state in the middle of sending a logical record, and you issue SENDERR, your communication partner would get an IPCODE of X'0420' (PROG_ERROR_TRUNC).

Requesting to Send

At some point, when your partner is sending data, you may want to interrupt. You could issue a SENDREQ, which would be presented to your partner as a SENDREQ interrupt. Because APPC/VM presents a SENDREQ as an interrupt, your partner cannot receive it. Your partner can even choose to ignore your SENDREQ.

SENDREQ does not cause state changes. However, if your partner decides to change states (by issuing RECEIVE or SENDDATA RECEIVE = YES) because of your SENDREQ, then you would be able to send some data.

Synchronous APPC/VM Support

When you issue a synchronous APPC/VM function (WAIT = YES), your virtual machine remains in a WAIT state until your function completes. The following can cause your function to complete:

- Your partner issues a function to complete your function
- Your partner logs off or resets its virtual machine
- You log off or reset your virtual machine.

The synchronous functions are not interruptible. Even if your virtual machine is enabled for interrupts, you are not given control until your function completes. Your communication partner sees no difference if you issue functions synchronously (WAIT = YES) or asynchronously (WAIT = NO).

Use synchronous functions with caution! If your communication partner does not respond, logoff, or do a system reset, your virtual machine cannot execute any instructions until you do a logoff or reset your virtual machine. Applications should take responsibility to avoid deadlock situations. A deadlock situation is when two virtual machines are waiting for an action by or a response from the other. The WAIT = NO option may help you handle a deadlock situation.

To compare synchronous with asynchronous:

- With synchronous functions, your virtual machine is in a WAIT state. This means you cannot issue any APPC/VM functions to any paths until the synchronous function that you issued completes. When the function completes you do not get an interrupt; instead, the function complete data goes to your output parameter list.

  Do not use the synchronous option for a program that must serve more than one user at the same time, or for a program that must run in a multitasking environment within a virtual machine.

- With asynchronous functions, the function may or may not complete immediately. When the function does not complete immediately, your

virtual machine still has to wait before issuing other APPC/VM functions on the path. However, you can issue APPC/VM functions to other paths while waiting for the asynchronous function to complete. In this case, you get an interrupt when the asynchronous function completes.

On the other hand, when the function does complete immediately, you get the function complete data in the output parameter list. In this case, you would not receive a function complete interrupt.

You can receive notice that any asynchronous function has completed by doing either of the following:

- Enable your virtual machine for function complete interrupts using the SETMASK or SETCMASK commands. These are described in "IUCV SETMASK" on page 165 and "IUCV SETCMASK" on page 161.

- Use the TESTCMPL function, described in "IUCV TESTCMPL" on page 177.

How APPC/VM Differs from General IUCV

APPC/VM makes use of the general IUCV support. APPC/VM depends on a half duplex protocol, while IUCV communication uses a full duplex protocol. In support of half duplex protocol, APPC/VM defines and enforces states on each path. In APPC/VM, the high order bit of the IPTYPE field is set for APPC/VM interrupts.

APPC/VM functions may not be issued in CP code except by the IUCV support. This section outlines the differences between APPC/VM functions and general IUCV functions.

Shared APPC/VM and IUCV Functions

The functions that are shared for both APPC/VM and IUCV are:

- **ACCEPT**

  When ACCEPT is issued to establish an APPC path,

  - ACCEPT does not have a message limit parameter, whereas non-APPC ACCEPT does. This is because the message limit is always one on APPC/VM paths.

  - ACCEPT does not have a user data field.

- **DCLBFR** and **RTRVBFR**

Both APPC/VM and IUCV interrupts are presented in the same buffers. You declare these buffers with DCLBFR and release them with RTRVBFR.

- **DESCRIBE**

  DESCRIBE gives the following information:

  - The next message pending on non-APPC paths
  - The next message pending on an APPC path that is in RECEIVE state
  - A SENDREQ on an APPC path.

- **QUERY**

  QUERY gives you the following information about a virtual machine:

  - The size of the external interrupt buffer
  - The maximum number of communication paths that can be established for the virtual machine.

- **SETMASK** and **SETCMASK**

  You can use SETMASK and SETCMASK to mask on and mask off APPC and non-APPC interrupts.

- **TESTCMPL**

  You can use TESTCMPL to determine the next APPC or non-APPC function that has completed.

- **TESTMSG**

  TESTMSG waits for the following:

  - A message pending or message complete on non-APPC paths
  - A message pending on an APPC path that is in RECEIVE state
  - A SENDREQ on an APPC path
  - A function complete on an APPC path.

APPC/VM and IUCV Functions That Work Differently

The functions that work differently between APPC/VM and IUCV are:

- **CONNECT**

  The *resource id* in the APPC/VM CONNECT defines the target of the connection. In the IUCV CONNECT, the *userid* defines the target. The location of the resource id within the APPC/VM CONNECT parameter list corresponds to the program name of CMS and GCS IUCV support.

  In addition, APPC/VM differences are:

- APPC/VM CONNECT does not have a message limit parameter, whereas IUCV CONNECT does. This is because the message limit is always one on APPC/VM paths.
- APPC/VM CONNECT does not have a user data field.
- You cannot use APPC/VM CONNECT to connect to a CP system service.
- A connection extension is defined for APPC/VM CONNECT that lets you specify an LU name and mode name.

- **RECEIVE**

  APPC/VM differences are:

  - You must provide a path id on APPC/VM RECEIVE.
  - You can issue APPC/VM RECEIVE before data arrives on a path.
  - APPC/VM RECEIVE has a WAIT=YES option.

- **SEND** (for APPC/VM, **SENDDATA**)

  You can use both APPC/VM SENDDATA and IUCV SEND to send data to your communication partner. APPC/VM differences are:

  - APPC/VM SENDDATA does not have a parameter list data option.
  - APPC/VM SENDDATA does not have a priority message option.
  - APPC/VM SENDDATA does not have any special message identifiers (a message class or a message tag).
  - APPC/VM SENDATA has a WAIT=YES option.
  - APPC/VM SENDDATA has a RECEIVE=YES option that lets you define an answer area. IUCV SEND has a TYPE=2WAY option that lets you define an answer area.

    The APPC/VM user responds to a SENDDATA RECEIVE=YES with a SENDDATA. The length of the response does not depend on the size of the answer area. The IUCV user, on the other hand, responds to a SEND TYPE=2WAY with a REPLY. The length of the response cannot be bigger than the size of the answer area.
  - With APPC/VM, the data sent is in logical record format. With IUCV, the data can be in any format.

- **SEVER**

  APPC/VM differences are:

- APPC/VM SEVER does not have a user data field.
- There are two APPCVM SEVER types, TYPE=NORMAL and TYPE=ABEND. There is only one IUCV SEVER type.

IUCV Functions Not Supported on APPC/VM Paths

The following IUCV functions are not supported on APPC/VM paths:

- **PURGE**
- **QUIESCE** and **RESUME**

  Because the message limit on APPC/VM paths is one, an application can "quiesce" a path by not receiving a message pending on that path.

- **REJECT**

  APPC/VM SENDERR is similar to IUCV REJECT.

- **REPLY**

  APPC/VM SENDDATA can be used in place of IUCV REPLY. Refer to "APPC/VM and IUCV Functions That Work Differently" on page 77 for more information.

APPC/VM Functions Not Supported on IUCV Paths

The following APPC/VM functions are not supported on IUCV paths:

- **SENDCNF** and **SENDCNFD**

  IUCV does not have any equivalent functions.

- **SENDERR**

  IUCV REJECT is similar to APPC/VM SENDERR.

- **SENDREQ**

  APPC/VM SENDREQ is similar to an IUCV priority 1WAY parameter data SEND, when that SEND is used as a signal and does not contain any data.

APPC/VM Local Communication vs. Remote Communication

With APPC/VM, you can write applications the same whether or not the communicating virtual machines are on the same VM system or different VM systems. However, a program that works when communicating with a program on the same system may not always work when communicating with the identical program on a different system in the TSAF collection.

In order to ensure that an application running within a system continues to run when communicating outside of a single system, you should follow these guidelines when writing applications:

- Write your application to handle all possible completion indications for each verb your application issues. Do not assume that only a subset of completion indications can occur.

- In general, you cannot determine when a function completes immediately or when a function completes asynchronously. Your application should be prepared for both immediate and asynchronous completions where appropriate. Appendix D, "Sample TSAF Resource Manager Program" on page 257 shows how to handle both types of completions in a simple manner.

You, as the application programmer, should be familiar with the general APPC program interface as defined in the *Systems Network Architecture Transaction Programmer's Reference Manual for LU Type 6.2.*

Chapter 7. APPC/VM and IUCV Communication Functions

This chapter describes the following functions:

- VM program-to-VM program communication functions provided by the APPC/VM program interface
- The set of IUCV (VM-unique) functions for use in conjunction with APPC/VM functions.

APPC/VM Communication Functions

The APPC/VM communication functions are provided as parameters of the APPCVM macro. These functions provide program-to-program communication over an APPC/VM path that has a unique path id and that exists for the exclusive use of a pair of communicating VM programs. VM programs at each end of the APPC/VM path use APPC/VM functions to communicate with each other.

The APPC/VM functions are available only for communications between VM programs. These functions relate only to APPC/VM paths. The APPC/VM interface provides a limited set of the SNA LU 6.2 base communication functions.

Supported Functions

The APPC/VM communication functions are:

- CONNECT
- RECEIVE
- SENDCNF
- SENDCNFD
- SENDDATA
- SENDERR
- SENDREQ
- SEVER.

IUCV Functions Associated with APPC/VM

The VM-unique IUCV functions are provided as parameters of the IUCV macro. These functions provide a program-to-CP (not program-to-program) interface to do the following types of functions:

- Establish a VM path
- Handle asynchronous APPC/VM interrupts that occur on a VM path
- Terminate a VM path.

The functions relate to both APPC/VM and IUCV paths. The IUCV functions are not defined by the SNA LU 6.2 (APPC architecture) verb interface, but instead are a necessary complement for APPC programs executing in a VM processor. The asynchronous capability of APPC/VM is not based on APPC architecture, but, instead, is a VM-unique asynchronous implementation.

Supported Functions

The IUCV functions are:

- ACCEPT
- DCLBFR
- DESCRIBE
- QUERY
- RTRVBFR
- SETCMASK
- SETMASK
- TESTCMPL
- TESTMSG.

APPC/VM and IUCV Functions Reference List

The APPC/VM and IUCV functions are listed alphabetically in this chapter by function name, without regard to the macro on which the parameter (function) name is found. The functions defined in this chapter are:

ACCEPT (IUCV)
accepts the connection from a virtual machine.

CONNECT (APPCVM)
establishes and reserves a path to communicate with another program.

DCLBFR (IUCV)
declares a buffer to store external interrupt data for APPC/VM and IUCV functions.

DESCRIBE (IUCV)
gets an indication and description of a pending message without receiving it.

QUERY (IUCV)
gets information about the external interrupt buffer and to find out how many paths can be established.

RECEIVE (APPC/VM)
receives data and information sent to your program.

RTRVBFR (IUCV)
ends APPC/VM and IUCV communications by releasing (retrieving) the buffer defined by DCLBFR.

SENDCNF (APPC/VM)
sends a confirmation request.

SENDCNFD (APPC/VM)
sends a response to a confirmation request.

SENDDATA (APPC/VM)
sends data to another program.

SENDERR (APPC/VM)
sends notice that an error has been detected.

SENDREQ (APPC/VM)
requests permission to send data.

SETCMASK (IUCV)
enables or disables IUCV control external interrupts.

SETMASK (IUCV)
enables or disables general IUCV external interrupts.

SEVER (APPC/VM)
ends communication with another program.

TESTCMPL (IUCV)
determines if any messages or functions have been completed.

TESTMSG (IUCV)
tests if any messages or functions are pending or complete.

For the non-APPC IUCV versions of these functions, unrelated to APPC/VM, and the other functions that IUCV supports, see *VM System Facilities for Programming*.

Appendix C, "Sample TSAF User Program" on page 249 and
Appendix D, "Sample TSAF Resource Manager Program" on page 257
illustrate the various APPC/VM functions and their parameters.

Some General Information about the APPCVM and IUCV Macros

In most cases, the APPCVM and IUCV macros move a fixed amount of data as specified with each parameter definition. Unlike the IUCV macro, all lengths in the APPCVM macro are fullwords.

Both the APPCVM and IUCV macros are in the DMKSP MACLIB, along with the IPARML DSECT.

Completing the Parameter List (Parameter Addressability)

The APPCVM and IUCV macros use labels defined in IPARML DSECT to complete the parameter list. You need to provide a USING for the IPARML DSECT when you invoke the macro.

To reference fields in the APPCVM and IUCV parameter lists, always use the name defined for that field in the IPARML DSECT, rather than using the displacement within IPARML DSECT. The parameter lists (IPARML DSECT mappings) are shown for each function. These are provided so you can visualize the IPARML DSECT. However, these parameter list pictures are not meant to define the interface.

Setting Defaults for Optional Parameters

If you do not specify a parameter marked as "Optional" when you invoke the APPCVM macro, the macro assumes that you have stored a value in the parameter list before invoking the APPCVM macro.

Note: All fields and flag bits in the parameter list that are not defined for a particular function should be set to zero. This helps ensure that if these fields are defined in the future, applications will continue to work.

Expanding the Macro with MF=L

With the MF=L parameter on the APPC/VM and IUCV functions, you have a choice of the following:

- Formatting the parameter list (specifying MF=L)
- Formatting and executing or just executing the parameter list (not specifying MF=L).

If you do specify MF=L, then the macro generates the instructions necessary to format the parameter list, using the parameter values provided on the macro. However, the macro does not generate any instructions to

execute the parameter list. In other words, parameter list formatting is provided, and parameter list execution is not.

You can use the IUCV or APPCVM macro to issue an APPC/VM function with a parameter list that is already formatted for the function. You can also use the CMS IUCV macros or the GCS IUCV macros to issue an APPC/VM function with a parameter list that is already formatted for the function. Do not use the APPCVM macro to issue non-APPC functions.

Note: The CMS IUCV macros are described in the *VM System Facilities for Programming*, and the GCS IUCV macros are described in the *VM/SP Group Control System Command and Macro Reference.*

If you do not specify MF=L on the APPCVM macro, the macro generates the instructions necessary to:

1. Format the parameter list as specified by parameter values on the macro.
2. Execute the APPC/VM function.

When you do not specify MF=L, R0 is altered.

When you want to invoke the APPC/VM SEND functions (SENDCNF, SENDCNFD, SENDDATA, SENDERR, SENDREQ) with the IUCV or IUCVCOM macro, specify the SEND function on the IUCV or IUCVCOM macro.

Declaring the Buffer for Interrupts

At the start of your program, you declare a buffer, with the DCLBFR function, to hold the APPC/VM and IUCV interrupt information for the established path. You can specify two types of buffers:

- Control Buffer - DCLBFR CONTROL=YES
- Application Buffer - DCLBFR CONTROL=NO.

The control buffer and control paths are for CMS and GCS use, and not for general application use. CMS and GCS declare a control buffer during their initialization process. CMS and GCS do not let applications that use the control buffer establish paths. See the *VM System Facilities for Programming* and the *VM/SP Group Control System Command and Macro Reference* for more specific information.

However, for applications not running with CMS or GCS, you can use both buffers. If you declare a control buffer with the DCLBFR function, then you can establish control paths with CONNECT CONTROL=YES. When you specify CONTROL=YES, only you view the path as a control path. Your communication partner views it as an application path.

When an interrupt for a control path is presented to your virtual machine, it goes to the control buffer. When an interrupt for an application path

comes in, it goes to the application buffer, and the path id is stored in the control buffer. The rest of the control buffer contains zeroes.

Enabling or Disabling for Interrupts

How you communicate with another virtual machine depends on how you have your virtual machine set up for external interrupts. You can use the SETCMASK and SETMASK functions to control what types of APPC/VM and IUCV interrupts your virtual machine can receive. SETCMASK and SETMASK affect both APPC/VM and IUCV interrupts.

Condition Codes and Return Codes That You May Receive

The condition codes and return codes that you may receive for the various APPC/VM functions are listed under each function description. In addition, Appendix A, "APPC/VM and IUCV Condition Codes and Return Codes" on page 225 summarizes the codes in a chart that lists every APPC/VM function. Return codes are stored in the IPRCODE field, which is a field defined in IPARML DSECT when CC = 1.

At some point, when running an APPC/VM application, you may receive a return code that is not documented under the APPC/VM functions. In this case, the return code is most likely issued from an IUCV function. See the *VM System Facilities for Programming* for details on these return codes.

IUCV ACCEPT

Function Code: X'0A'

This function accepts a connection from another virtual machine or from your own virtual machine. You should use ACCEPT only after your virtual machine gets a connection pending external interrupt.

The *VM System Facilities for Programming* describes the IUCV version of this function unrelated to APPC/VM. Parameters other than those listed here are available, but have no meaning on APPC/VM paths and are ignored. The ACCEPT function is not part of the APPC architecture and is unique to VM.

The IUCV ACCEPT syntax is:

label **IUCV**	**ACCEPT,**	Required
	PRMLIST = *label*/(*reg*),	Required
	MF = **L,**	Optional
	PATHID = *label*/(*reg*)	Optional

PRMLIST = *label*/(*reg*)
lets you specify the address of the IUCV parameter list. The address must be a guest real address (real to the virtual machine), and the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

reg
is the register number that contains the address of the parameter list.

MF = **L**
expands the IUCV macro to generate the instructions necessary to initialize the IUCV parameter list as specified, but not to invoke the IUCV function.

PATHID = *label*/(*reg*)
lets you identify the path on which to accept the connection.

label
is the relocatable label of a halfword that contains the path id.

reg
is the register number that contains the path id in the low-order halfword.

ACCEPT Parameter List

The IUCV ACCEPT parameter list has the following input format when accepting a connection on an APPC path:

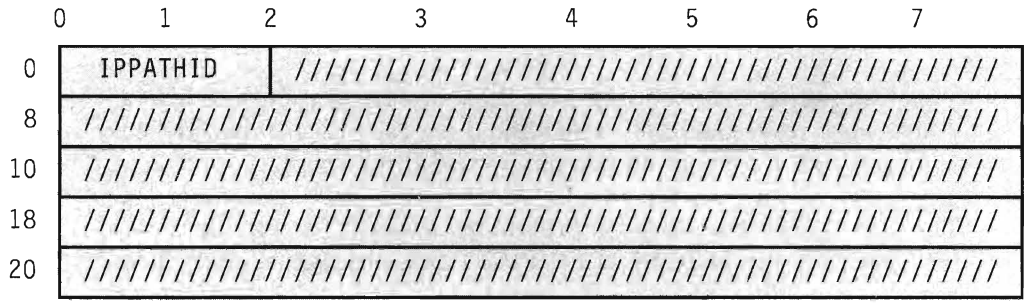


Figure 32. IUCV ACCEPT Input Parameter List

The supplied parameters are:

IPPATHID
: is the path id on which to accept the connection.

Error Codes and Exceptions

Condition Codes

CC = 0	CC = 1	CC = 2	CC = 3
Not Possible	X	X	Not Possible

CC = 1
An error occurred. The output parameter list is the same as the input shown in "ACCEPT Parameter List," except the return code is stored in IPRCODE.

You may get either of these return codes (listed here in decimal):

01 You specified a path id that is not yet established.

20 Connection cannot be completed—originator has severed the path.

Note: You must still issue IUCV SEVER to clean up your side of the path.

CC = 2
Accept is complete. The output parameter list is the same as the input, shown in "ACCEPT Parameter List."

ACCEPT Program Exceptions

The program exceptions for ACCEPT are:

Type	Description
Addressing	The parameter list address is outside of the virtual machine.
Operation	A noncontrol external interrupt buffer has not been declared with the DCLBFR function, or the invoker is not in supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user.
Specification	The parameter list is not on a doubleword boundary.

State Changes

Your virtual machine path is in:

- ***CONNECT state***—after you receive the connection pending interrupt. You may only issue ACCEPT, APPCVM RECEIVE, or IUCV SEVER from CONNECT state.
- ***RECEIVE state***—after the ACCEPT completes successfully.

ACCEPT Completion

Because you can issue ACCEPT only when there is a connection pending, the ACCEPT function completes immediately. If you have not yet received the allocation data pending for this path, the data is purged when ACCEPT completes. "Allocate Data That Your Communication Partner May Receive" on page 98 describes allocation data in more detail.

What Happens to Your Communication Partner

When you issue the ACCEPT function, the virtual machine that issued the CONNECT (your communication partner) gets a connection complete indication. This ACCEPT function can be issued by an intermediate communication server that is not the target of the CONNECT. In this case, accepting a connection does not necessarily mean that the APPC/VM path has been completed between two communicating programs. See "APPCVM CONNECT" on page 90 for more details on the connection complete indication.

APPCVM CONNECT

Function Code: X'0B'

This function establishes a communications path with a program residing in another virtual machine or your own virtual machine.

The APPCVM CONNECT syntax is:

label **APPCVM**	**CONNECT,**	Required
	PRMLIST = *label*/(*reg*),	Required
	MF = **L,**	Optional
	CONTROL = **YES/NO,**	Optional
	RESID = *label*/(*reg*),	Optional
	SYNCLVL = **NONE/CONFIRM,**	Optional
	WAIT = **YES/NO,**	Optional
	BUFFER = *label*/(*reg*),	Optional
	BUFLEN = *label*/(*reg*),	Optional
	ALTID = *label*/(*reg*)	Restricted

PRMLIST = *label*/(*reg*)
lets you specify the address of the APPC/VM parameter list. The address must be a guest real address (real to the virtual machine), and the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

reg
is the register number that contains the address of the parameter list.

MF = **L**
expands the APPCVM macro to generate the instructions necessary to initialize the APPC/VM parameter list as specified, but not to invoke the APPC/VM function.

CONTROL = **YES/NO**
lets you specify whether or not a control path is being established and if all interrupt information for your half of the path is be placed in the control buffer for control paths.

- CONTROL = YES sends APPC/VM interrupt information on this path to the control buffer.
- CONTROL = NO sends APPC/VM interrupt information on this path to the application buffer.

RESID = *label*/(*reg*)
lets you specify a one-to-eight character resource name. Your virtual machine is connected to the resource manager that manages the resource specified by IPRESID. If the resource id you specify is less than 8 bytes, left justify the value in this field and pad the right with blanks.

label
is the relocatable label of the storage area that contains the resource id.

reg
is the register number that contains the address of the storage area. This storage area contains the resource id.

Note: The location of the resource id in the parameter list corresponds to the program name required by CMS and GCS IUCV support.

SYNCLVL = NONE/CONFIRM
lets you specify the synchronization level for the path being established.

- SYNCLVL = NONE does not let either communication partner issue SENDCNF or SENDCNFD on the path that this connection is establishing.
- SYNCLVL = CONFIRM lets either communication partner issue SENDCNF and SENDCNFD on the path that this connection is establishing.

WAIT = YES/NO
lets you specify when control is returned to your virtual machine.

- WAIT = YES returns control to your virtual machine after the CONNECT completes.
- WAIT = NO returns control to your virtual machine as soon as the CONNECT request is initiated. When the CONNECT completes, you are notified via an interrupt.

BUFFER = *label*/(*reg*)
lets you specify the address of the area that contains the connection parameter list extension.

label
is the relocatable label of the storage area that contains the connection parameter list extension.

reg
is the register number that contains the address of the storage area. This storage area contains the connection parameter list extension.

The connection extension supplied by the virtual machine is currently 16 bytes long:

1. The first 8 bytes, which must be all zeroes to signify LU_NAME (OWN).

2. The second 8 bytes represent the mode name, which also must be all zeroes.

Note: Information on LU_NAME and mode name is in the *Systems Network Architecture Transaction Programmer's Reference Manual for LU Type 6.2.*

BUFLEN = *label*/(*reg*)
lets you specify the 4-byte length of the area that has the connection parameter list extension. Valid lengths are 0 and 16. If you specify 0, APPC/VM ignores the address that you specified with **BUFFER** = and defaults the data in the connection extension.

label
is the relocatable label of the storage area that contains the length.

reg
is the register number that contains the length of the storage area.

ALTID = *label*/(*reg*)
is the 8-byte userid of the virtual machine for which the communication server is establishing the path. If the userid that you specify is less than 8 bytes, left justify the value in this field and pad the right with blanks.

label
is the relocatable label of the storage area that contains the userid.

reg
is the register number that contains the address of the storage area. This storage area contains the userid.

Note: The ALTID parameter is for communication servers, like the TSAF virtual machine. You must have directory authorization to use it (OPTION COMSRV). When you specify ALTID, APPC/VM sets the IPCOMSRV flag. For more details, see "Considerations for Communication Servers" on page 100.

CONNECT Parameter List

The APPC/VM CONNECT parameter list has the following input format when establishing APPC paths:

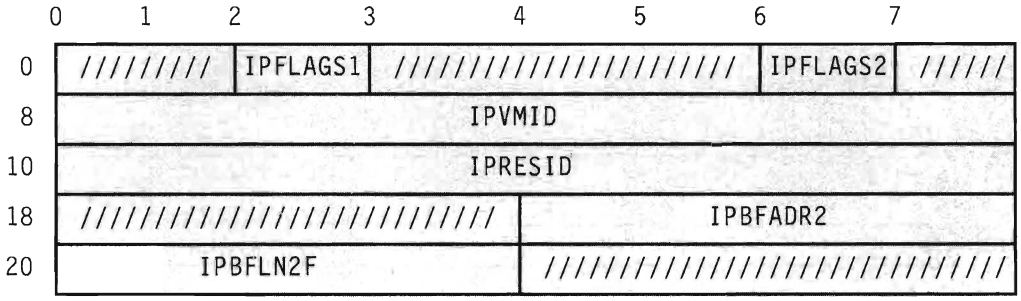


Figure 33. APPCVM CONNECT Input Parameter List

The supplied parameters are:

IPFLAGS1
: contains the input flags:

  IPAPPC (X'08')—APPC protocol is used on the path.
  IPCNTRL (X'04')—A control path is being established.

IPFLAGS2
: contains the input flags:

  IPWAIT (X'80')—A synchronous return was specified.
  IPLVLCF (X'40')—The synchronization functions, SENDCNF and SENDCNFD, are permitted.
  IPCOMSRV (X'20')—This connection is being made for another user. See "Considerations for Communication Servers" on page 100.

IPVMID
: is the userid that this connection is being made for. It is only valid when IPCOMSRV is set. See "Considerations for Communication Servers" on page 100.

  Only communication servers can supply this parameter.

IPRESID
: is the name of the resource for this connection.

  Note: The location of the resource id in the parameter list corresponds to the program name required by CMS and GCS IUCV support.

IPBFADR2
: is the address of the connection parameter list extension.

IPBFLN2F
is the length of the connection parameter list extension.

Error Codes and Exceptions

Condition Codes

CC = 0	CC = 1	CC = 2	CC = 3
X	X	X	Not Possible

CC = 0
When you specify WAIT = YES, CC = 0 is not possible. CC = 0 means that the CONNECT started successfully, but has not completed. The output parameter list field, IPPATHID, identifies the path being started by this connection.

When the function does complete and your virtual machine is properly enabled for interrupts (see "IUCV SETCMASK" on page 161), you get a connection complete or SEVER interrupt. Both the connection complete and the SEVER interrupt have the same format as the output CONNECT parameter list (see CC = 2).

CC = 1
An error occurred before the CONNECT was initiated. The output parameter list is the same as the input shown in "CONNECT Parameter List" on page 93, except the return code is stored in IPRCODE.

You may get any of these return codes (listed here in decimal):

11 CP could not find the resource, or the resource is not available for connections, and no TSAF virtual machine is currently operating on your system.

12 Your communication partner has not invoked the DCLBFR function.

13 Your virtual machine already has the maximum number of connections.

14 Your communication partner already has the maximum number of connections.

15 Your virtual machine is not authorized to connect to the resource

28 You specified CONTROL = YES, but no control buffer exists.

29 You are not authorized to act for another user.

39 You specified an invalid connection parameter list extension length.

40 You specified an invalid LU_NAME.

41 You specified an invalid mode name.

CC = 2

Connect completed (see section "Connection Completion" on page 96). When you specify WAIT = NO, CC = 2 is not possible. The output parameter list when CC = 2 is:

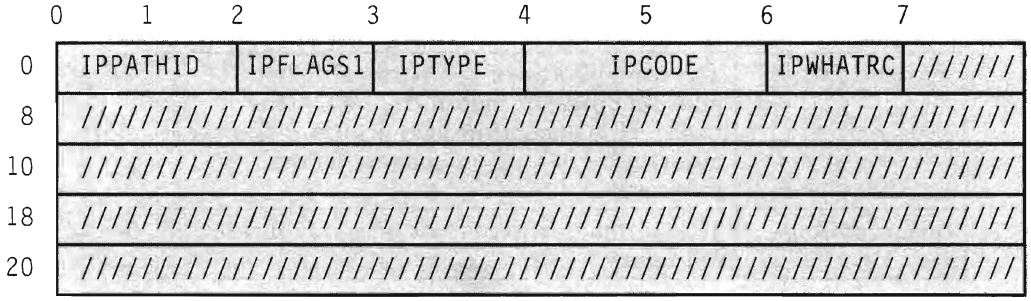


Figure 34. APPCVM CONNECT Output Parameter List (Connection Complete Interrupt)

The parameters are:

IPPATHID
: contains the path id on which the connection was completed or severed.

IPFLAGS1
: contains the flag:

  IPCNTRL (X'04')—A connection complete is on a control path. This flag is not set if IPTYPE = X'83'.

IPTYPE
: if your partner or an intermediate communication server accepted the connection, this field contains the connection complete interrupt code (IPTYPCCA, X'82'). If your partner or an intermediate communication server rejected the connection with the SEVER function, IPTYPE is the SEVER interrupt code (IPTYPSVA, X'83').

IPCODE
: contains the SEVER code from the partner's SEVER. IPCODE is only valid when IPTYPE = X'83'. See "APPC/VM Error/SEVER Codes" on page 183 for a description of the error/SEVER codes.

IPWHATRC
: contains the what-received code. IPWHATRC is only valid when IPTYPE = X'83'.

IPSABEND (X'09')—Your partner issued SEVER TYPE = ABEND.

CONNECT Program Exceptions

The program exceptions for CONNECT are:

Type	Description
Addressing	The parameter list address or connection extension address is outside of the virtual machine.
Operation	An external interrupt buffer has not been declared with the DCLBFR function, or the invoker is not in supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user. This exception also occurs if the connection extension address is fetch protected.
Specification	The parameter list is not on a doubleword boundary.

State Changes

Your virtual machine path is in one of the following states:

- ***CONNECT state***—for either of the following situations:
  - After you issue the CONNECT, but before your communication partner or an intermediate communication server ACCEPTS the connection (CC = 0).
  - If your communication partner or an intermediate communication server issues a SEVER when you issue the CONNECT, you get a SEVER interrupt and no path is established. You remain in CONNECT state, and you must issue IUCV SEVER to delete your side of the path.
- ***SEND state***—after you receive the connection complete indication.

Connection Completion

Connection complete data can be in different forms, depending on whether you specify WAIT = YES or WAIT = NO on the CONNECT.

If you specify WAIT = YES on the CONNECT, and your communication partner or an intermediate communication server:

- ACCEPTs the connection, you get a connection complete indication.

- SEVERs the connection, you get a SEVER indication in the connection complete data.

The connection complete data goes to the parameter list that you specified on the APPCVM CONNECT macro. The format of the connection complete interrupt data is the same as described under CC=2 in Figure 34 on page 95.

On the other hand, if you specify WAIT=NO on the CONNECT, and your communication partner or an intermediate communication server:

- ACCEPTs the connection, you get a connection complete interrupt.
- SEVERs the connection, you get a SEVER interrupt.

When you specify WAIT=NO on the APPCVM CONNECT and your virtual machine is enabled for the proper external interrupts, the connection complete data goes to the buffer that you specified with the DCLBFR function. When you specify CONTROL=YES on the CONNECT function, the interrupt data goes to the control buffer. When you specify CONTROL=NO on the CONNECT function, the interrupt data goes to the application buffer. The format of the connection complete interrupt data is the same as described under CC=2 in Figure 34 on page 95. All subsequent interrupts for the established path are presented in the same buffer as the connection complete interrupt.

Note: The connection complete interrupt indicates only that your CONNECT has completed and that you are now in SEND state on the path. A connection complete interrupt does not necessarily indicate that the actual target of your CONNECT has ACCEPTed the connection, or even that the target of the CONNECT has been invoked.

What Happens to Your Communication Partner

When you issue CONNECT, if your communication partner is enabled for connection pending interrupts, your partner gets a connection pending external interrupt. The connection pending external interrupt data always goes to the APPC/VM interrupt buffer that your communication partner defined with DCLBFR, and never into the control buffer. The connection pending external interrupt format is the following:

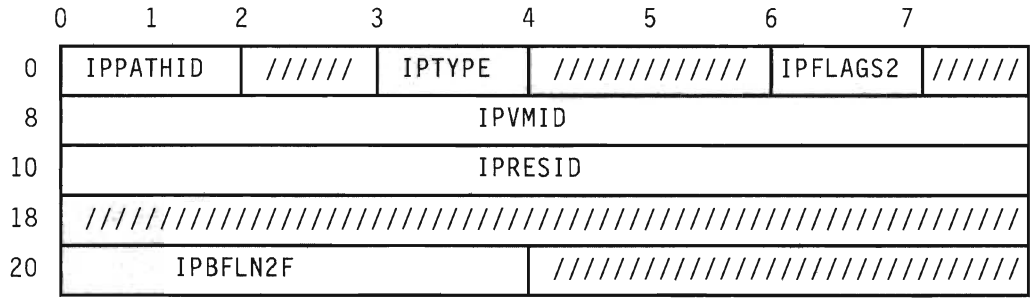


Figure 35. Connection Pending External Interrupt

The parameters are:

IPPATHID
: contains the path id on which a connection is pending.

IPTYPE
: contains the interrupt type for a connection pending (X'81').

IPFLAGS2
: contains the flag:

  IPLVLCF (X'40')—SENDCNF and SENDCNFD functions are permitted on this path.

IPVMID
: contains the userid of the virtual machine that wants to connect. If the connecting virtual machine has an alternate userid specified, this field contains the alternate userid. The field may be zero, which indicates that the identity of the connecting virtual machine is unknown.

IPRESID
: contains the name of the resource that this connection is for.

IPBFLN2F
: contains the length of the allocate data that is pending.

Allocate Data That Your Communication Partner May Receive

After your partner gets a connection pending interrupt, your partner may issue a RECEIVE. Your partner would receive the following variable-length allocate data, consisting of two parts:

1. **A VM-architected area (variable).**

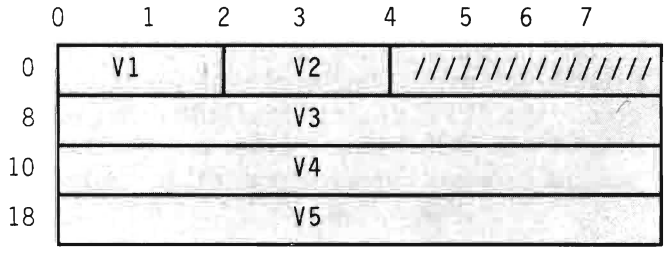


Figure 36. A VM-Architected Area

The fields are the following:

V1 is the total length of the VM architected area, including the length fields. This is now 32 bytes, but may change in the future.

V2 is the length of the fixed length fields area. This is now 24 bytes, but may change in the future.

V3 is zero, indicating LU_NAME (OWN).

V4 is the mode name.

V5 is the PARTNER_LU_NAME.

2. **The Attach FMH5.** CP specifies the values shown below when creating an FMH5. FMH5 is a Function Management Header Type 5. LU type 6.2 uses this header to carry a request for a conversation to be established between two transaction programs. This header identifies the transaction program that is to be put into execution and connected to the receiving half-session. Treat the FMH5 as a variable length construct. See the *Systems Network Architecture Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2* for more information on the FMH5.

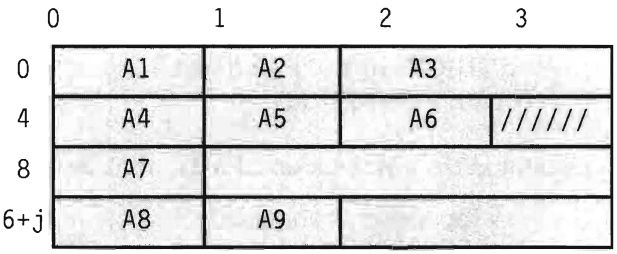


Figure 37. Attach FMH5 Record for APPC Conversations

The fields are the following:

A1 is the total length of the Attach FMH5, including the length fields. This is 18 bytes, but may change in the future.

A2 is the type code, B'00000101' (FMH5).

A3 is the command code, X'02FF' (Attach).

A4 is a flag byte.

- Bit 0—is the security indicator. CP sets it to 0 to indicate the userid and password are not verified.
- Bits 1-3—are reserved and set to 0.
- Bit 4—is on when a PIP (Program Initialization Parameter) is present. CP sets this field to 0 to indicate no PIP is present.
- Bits 5-7—are reserved and set to 0.

A5 is the length of the fixed length parameters field, which is currently 3 bytes.

A6 is the conversation type. X'D0' indicates a basic conversation.

A7 is a flag byte.

- Bits 0-1—is the synchronization level. When you specified SYNCLVL=NONE, this field is 00. When you specified SYNCLVL=CONFIRM, this field is 01.
- Bit 2—is on for reconnect support. CP sets this field to 0 to indicate there is no reconnect support.
- Bits 3-7—are reserved and set to 0.

A8 is the length of the transaction program name, which is currently 8 bytes.

A9 is the transaction program name. CP fills in the resource id that you specified.

All reserved fields are set to zero.

For example, if you specify the following:

- An LU_NAME and MODE NAME of zeroes
- A RESOURCE ID of "PAYROLL"
- SYNCLVL=CONFIRM

you would get a VM area and FMH5 that looks like the following:

```
x'0020001800000000'
x'0000000000000000'
x'0000000000000000'
x'0000000000000000'
x'120502FF0003D00001'
x'08D7C1E8D9D6D3D340'
```

Your partner may or may not receive the data. If your partner wants to receive the data, your partner must do so before issuing ACCEPT. CP purges the data that it created on CONNECT after your partner issues ACCEPT.

Considerations for Communication Servers

Communication servers must be authorized in the CP directory with OPTION COMSRV. When a communication server is establishing a connection for another virtual machine, the communication server may specify the userid (IPVMID) of the virtual machine for which the connection is being made.

The communication server must turn on the IPCOMSRV flag before issuing the CONNECT for another user. Note that IPCOMSRV set to "on" tells CP the following:

- The communication server is authorized to connect for other users.

- CP should pass the userid supplied by the communication server in IPVMID on to the communication partner. The userid is that which initiated the CONNECT.

Note that the userid in the connection pending interrupt should always be the userid of the virtual machine that issued the original CONNECT. Also, when you do specify IPCOMSRV, you can only connect to globally defined resources.

IUCV DCLBFR

Function Code: X'0C'

Use DCLBFR before you use any other APPC/VM or IUCV functions (except QUERY) to set the address of a buffer that APPC/VM and IUCV can use to store external interrupt data. After you receive an external interrupt, this buffer contains information about the message, reply, or control function that caused the interrupt.

To improve performance, declare a buffer entirely within one page. To further reduce overhead, declare the buffer entirely within page 0 of the virtual machine.

When you issue DCLBFR, the virtual machine is enabled for all types of APPC/VM and IUCV external interrupts. Use the SETMASK and the SETCMASK functions to change these initial settings.

Note: The IUCV interrupt mask in control register 0 is not affected by DCLBFR. See "IUCV SETMASK" on page 165 for more details.

The *VM System Facilities for Programming* describes the IUCV version of this function unrelated to APPC/VM.

The IUCV DCLBFR syntax is:

label **IUCV**	**DCLBFR,**	Required
	PRMLIST = *label*/(*reg*),	Required
	BUFFER = *label*/(*reg*),	Optional
	CONTROL = **YES/NO,**	Optional
	MF = **L**	Optional

PRMLIST = *label*/(*reg*)
lets you specify the address of the APPC/VM parameter list. The address must be a guest real address (real to the virtual machine), and the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

reg
is the register number that contains the address of the parameter list.

BUFFER = *label*/(*reg*)
identifies the external interrupt buffer. When an external interrupt is sent to the virtual machine, APPC/VM stores information about the message or the control interrupt in this buffer.

label
is the relocatable label of the storage area that is used as the external interrupt buffer.

reg
is the register number that contains the address of the storage area that is used as the external interrupt buffer.

CONTROL = **YES/NO**
lets you declare an application buffer or a control buffer.

- CONTROL = YES declares a control buffer.
- CONTROL = NO declares an application buffer.

MF = **L**
expands the IUCV macro to generate the instructions necessary to initialize the IUCV parameter list as specified, but not to invoke the IUCV function.

DCLBFR Parameter List

The IUCV DCLBFR parameter list has the following input format:

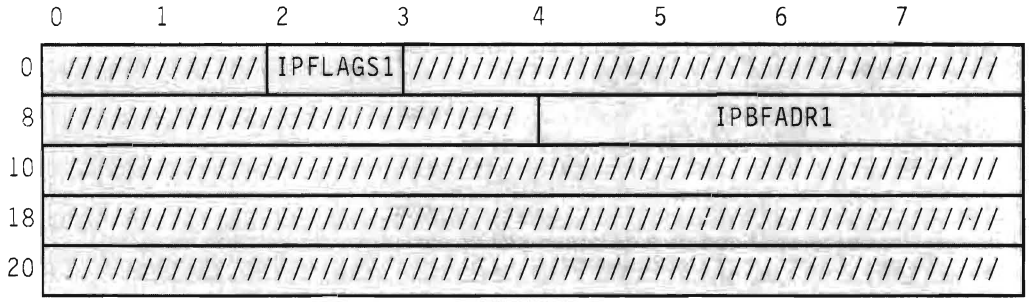


Figure 38. IUCV DCLBFR Input Parameter List

The supplied parameters are:

IPFLAGS1
contains the flag:

IPCNTRL (X'04')—A control buffer is being established.

IPBFADR1
identifies the area in which IUCV stores information about an APPC/VM or IUCV external interrupt.

Error Codes and Exceptions

Condition Codes

CC = 0	CC = 1	CC = 2	CC = 3
X	X	Not Possible	X

CC = 0
DCLBFR completed with no errors.

CC = 1
An error occurred before the DCLBFR was initiated. The output parameter list is the same as the input shown in "DCLBFR Parameter List" on page 103, except the return code is stored in IPRCODE. You may get the following return code (listed here in decimal):

19 depends on how you specified CONTROL=:

- If you specified CONTROL=YES, a control buffer has already been defined.
- If you specified CONTROL=NO, an application buffer has already been defined.

CC = 3
IUCV found errors while reading the directory. The output parameter list is the same as the input shown in "DCLBFR Parameter List" on page 103.

DCLBFR Program Exceptions

The program exceptions for DCLBFR are:

Type	Description
Addressing	The parameter list address or specified buffer address is outside the virtual machine.
Operation	The invoker is not in supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user.
Specification	The parameter list is not on a doubleword boundary.

State Changes

DCLBFR does not act on any one path and does not cause any state changes.

DCLBFR Completion

The DCLBFR function completes immediately.

What Happens to Your Communication Partner

Not applicable.

IUCV DESCRIBE

Function Code: X'03'

This function can do the following:

- Get a description of a pending IUCV or APPC/VM message without receiving it.
- Get a SENDREQ indication.

DESCRIBE returns information on a given message only once. The next time you invoke DESCRIBE, you get a description of the next "unDESCRIBEd" message. You can use the RECEIVE function to receive messages after you have performed a DESCRIBE on the message. However, it is not necessary to DESCRIBE a message before receiving it.

The *VM System Facilities for Programming* describes the IUCV version of this function unrelated to APPC/VM. Parameters other than those listed here are available, but have no meaning on APPC/VM paths and are ignored.

The IUCV DESCRIBE syntax is:

label **IUCV**	**DESCRIBE,**	Required
	PRMLIST = *label*/(*reg*)	Required

PRMLIST = *label*/(*reg*)
lets you specify the address of the IUCV parameter list. The address must be a guest real address (real to the virtual machine), and the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

reg
is the register number that contains the address of the parameter list.

Some Notes about DESCRIBE

1. For APPC messages, the message is only described if the corresponding path is in RECEIVE state. APPC/VM presents SENDREQ indications no matter what the state is of the corresponding path.

2. DESCRIBE does not describe messages that are pending on control paths.

3. If there is a function outstanding on a path, APPC/VM may report the message on the completion of that function, and not on DESCRIBE.

4. CP considers a message described if you do one of the following:

   - Completely or partially receive a message
   - Get a message pending interrupt
   - Get a SENDREQ interrupt.

DESCRIBE Parameter List

There are no input parameters for DESCRIBE.

Error Codes and Exceptions

Condition Codes

CC = 0	CC = 1	CC = 2	CC = 3
X	Not Possible	X	Not Possible

CC = 0
DESCRIBE completed with no errors. The output parameter list when an APPC/VM message is being described follows. See the *VM System Facilities for Programming* for non-APPC messages.

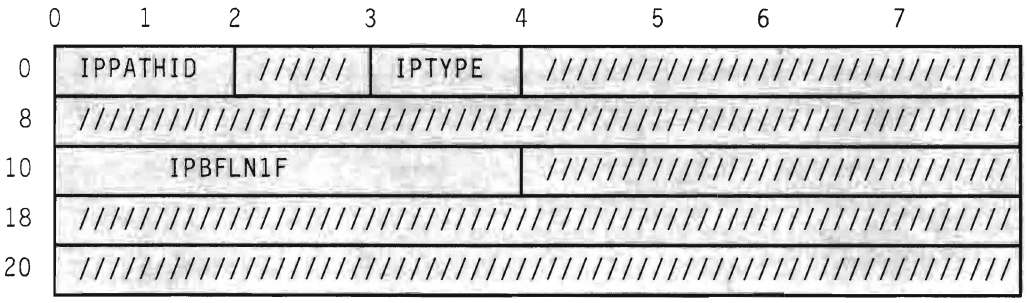


Figure 39. IUCV DESCRIBE Output Parameter List

The parameters are:

IPPATHID
contains the path id on which the message is pending.

IPTYPE contains the interrupt type for a message pending (IPTYPMPA, X'89') or SENDREQ (IPTYPSRA, X'88').

IPBFLN1F
contains the length of the message that is pending. This length has no relationship to the length of the APPC data stream being sent. It is only the length of the data that has arrived and is ready to receive. If the interrupt type is not X'89', this field has no meaning.

CC = 2
IUCV did not find any APPC/VM or IUCV "unDESCRIBEd" messages.

DESCRIBE Program Exceptions

The program exceptions for DESCRIBE are:

Type	Description
Addressing	The parameter list address is outside of the virtual machine.
Operation	A normal interrupt buffer has not been declared with the DCLBFR function, or the invoker is not in supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user.
Specification	The parameter list is not on a doubleword boundary.

State Changes

No state changes occur.

DESCRIBE Completion

The DESCRIBE function completes immediately with CC = 0 or CC = 2.

What Happens to Your Communication Partner

Not applicable.

IUCV QUERY

Function Code: X'00'

This function gets the following type of information about a virtual machine:

- The size of the external interrupt buffer (returned in register 0)
- The maximum number of communication paths that can be established for your virtual machine (returned in register 1).

You can issue QUERY before DECLARE BUFFER to determine the buffer size, and allocate the buffer before declaring it to APPC/VM.

The *VM System Facilities for Programming* describes the IUCV version of this function unrelated to APPC/VM.

The IUCV QUERY syntax is:

label	**IUCV**	**QUERY**	Required

QUERY Parameter List

There are no input parameters for QUERY.

Error Codes and Exceptions

Condition Codes

CC = 0	CC = 1	CC = 2	CC = 3
X	Not Possible	X	X

CC = 0
QUERY completed with no errors.

CC = 2
The user is not found in the CP directory.

CC = 3
CP found errors when reading the CP user directory.

QUERY Program Exceptions

The program exceptions for QUERY are:

Type	Description
Operation	The invoker is not in supervisor state.

State Changes

No state changes occur.

QUERY Completion

The QUERY function completes immediately.

What Happens to Your Communication Partner

Not applicable.

APPCVM RECEIVE

Function Code: X'05'

This function receives any of the following:

- Data sent to your virtual machine from another virtual machine
- Data sent to your virtual machine from your own virtual machine
- Allocation data, before accepting a connection.

When you issue RECEIVE and no message is currently pending on the path, the buffer that you specified is allocated for future messages on the path. See "RECEIVE Completion" on page 120 for more details.

The APPCVM RECEIVE syntax is:

label **APPCVM**	**RECEIVE,**	Required
	PRMLIST = *label*/(*reg*),	Required
	MF = **L,**	Optional
	PATHID = *label*/(*reg*),	Optional
	BUFLIST = **YES/NO,**	Optional
	BUFFER = *label*/(*reg*),	Optional
	BUFLEN = *label*/(*reg*),	Optional
	WAIT = **YES/NO**	Optional

PRMLIST = *label*/(*reg*)
lets you specify the address of the APPC/VM parameter list. The address must be a guest real address (real to the virtual machine), and the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

reg
is the register number that contains the address of the parameter list.

MF = **L**
expands the APPCVM macro to generate the instructions necessary to initialize the APPC/VM parameter list as specified, but not to invoke the APPC/VM function.

PATHID = *label*/(*reg*)
lets you identify the path on which to receive data.

label
is the relocatable label of a halfword that contains the path id.

reg
is the register number that contains the path id in the low-order halfword.

BUFLIST = YES/NO
specifies the type of buffer address that the BUFFER parameter refers to.

- BUFLIST = YES refers to a list of addresses.
- BUFLIST = NO refers to a single address.

For more information, see "Specifying Buffers on RECEIVE" on page 113.

BUFFER = *label*/(*reg*)
specifies the address of the area(s) into which CP places the data received. For more information, see "Specifying Buffers on RECEIVE" on page 113.

label
is the relocatable label in storage where CP places the data received.

reg
is the register number that contains the address of the storage area. This storage area is where CP places the message.

BUFLEN = *label*/(*reg*)
specifies the length of the area(s) into which APPC/VM places the message. For more information, see "Specifying Buffers on RECEIVE" on page 113.

label
is the relocatable label of the fullword that contains the length.

reg
is the register number that contains the length.

WAIT = YES/NO
lets you specify when control is returned to your virtual machine.

- WAIT = YES returns control to your virtual machine after the receive completes.
- WAIT = NO returns control to your virtual machine when you initiate the RECEIVE. If the RECEIVE does not complete immediately, when it does complete, you are notified with a function complete interrupt.

Specifying Buffers on RECEIVE

For APPCVM RECEIVE, you can specify the buffers with a single address and a single length (BUFLIST = NO) or with a list of addresses and lengths (BUFLIST = YES).

When you specify the buffer with a single address and a single length, BUFFER specifies the address, and BUFLEN specifies the length. When you specify the buffer with a list of addresses and lengths, BUFFER specifies the address of the list and BUFLEN specifies the sum of the lengths of the buffers in the list.

When specifying address lists, note the following:

1. When you use an address list (BUFLIST = YES), you must follow these rules:
   - The list must begin on a doubleword boundary.
   - Each list entry must be two fullwords:
     - The first fullword is the address of that portion of the list.
     - The second fullword is the length of that portion of the list.
2. The addresses and lengths in the address list are updated during APPC/VM processing. Do not alter them during processing or assume that they are unchanged when APPC/VM processing is complete.
3. APPC/VM assumes that there is another entry in the list until the sum of the lengths of the entries processed is equal to the total length specified by BUFLEN.

RECEIVE Parameter List

The APPCVM RECEIVE parameter list has the following input format:

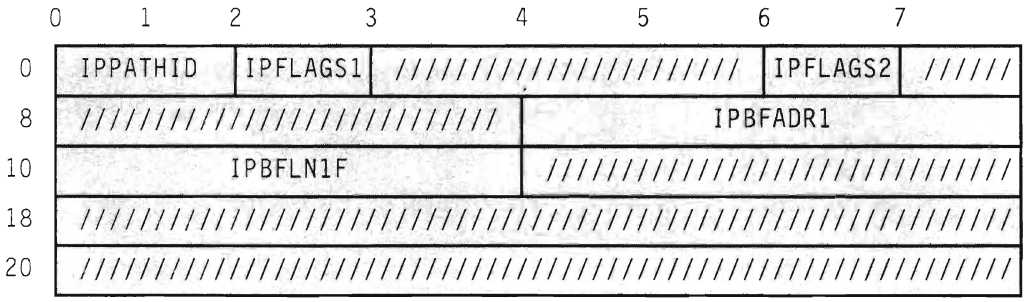


Figure 40. APPCVM RECEIVE Input Parameter List

The supplied parameters are:

IPPATHID
contains the path id over which you receive data.

IPFLAGS1
contains the flags:

IPBUFLST (X'40')—A buffer list was specified.
IPAPPC (X'08')—An APPC function was issued.

IPFLAGS2
contains the input flag:

IPWAIT (X'80')—A synchronous return is desired.

IPBFADR1
contains the address of the area where APPC/VM stores the received data or the address of the list. See "Specifying Buffers on RECEIVE" on page 113.

IPBFLN1F
contains receive area length that IPBFADR1 specifies. See "Specifying Buffers on RECEIVE" on page 113.

Error Codes and Exceptions

Condition Codes

CC = 0	CC = 1	CC = 2	CC = 3
X	X	X	X

CC = 0
RECEIVE started successfully, but has not yet completed. If your virtual machine is enabled for function complete interrupts, one is sent to your virtual machine when RECEIVE completes. The interrupt format is the same as the RECEIVE output parameter list (see CC = 2,3). When you get the function complete interrupt, check the IPAUDIT field for error information.

When control is returned to your virtual machine with CC = 0, the parameter list may have been altered.

Note: When you specify WAIT = YES, CC = 0 is not possible.

CC = 1
An error occurred before the RECEIVE was initiated. The parameter list format is the same as the input shown in "RECEIVE Parameter List" on page 113, except that the return code is stored in IPRCODE. Other fields in the parameter list may also have been altered.

You may get the following return codes (listed here in decimal):

01 You specified a path id that is not yet established.

03 A function is pending on this path.

06 A storage protection exception occurred on your partner's SEND buffer.

07 An addressing exception occurred on your partner's SEND buffer.

10 The buffer length is negative.

22 Your communication partner's SEND list is invalid.

23 A length specified in the RECEIVE list is negative.

24 The total length specified is not the total of the lengths in your list.

26 The buffer list address is not on a doubleword boundary.

30 You specified an APPC/VM function on a non-APPC path.

32 RECEIVE is an invalid function from CONNECT state. (See "State Changes" on page 119.)

35 RECEIVE is an invalid function from CONFIRM state.

36 RECEIVE is an invalid function from SEVER state.

43 There is an invalid logical record length in your communication partner's data stream.

44 Before issuing RECEIVE, you started, but did not finish, sending a logical record.

45 Your communication partner started, but did not finish, sending a logical record and tried to change to RECEIVE state.

Return codes 6, 7, 22, 23, 24, 43 and 45 can only occur if a message is pending for the specified path at the time you issued the RECEIVE. If no message is pending for the specified path at the time you issued the RECEIVE, CP reports those error conditions in the corresponding audit flags when the RECEIVE completes. For return codes 22, 23, 24, 43 and 45, some data may have been received; however, the results are unpredictable.

CC = 2 or
CC = 3

Function is complete (also see "RECEIVE Completion" on page 120). When CC = 2, the function completed with no errors. When CC = 3, there is error information in IPAUDIT. When you specify WAIT = NO, CC = 3 is not possible. The output parameter list when CC = 2 or 3 is:

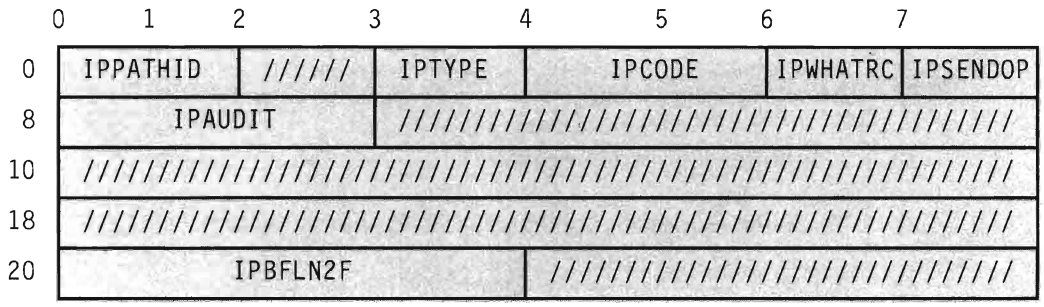


Figure 41. APPCVM RECEIVE Output Parameter List (Function Complete Interrupt)

The parameters are:

IPPATHID
: contains the path id on which the function is complete.

IPTYPE
: contains the function complete interrupt code (IPTYPFCA, X'87').

IPCODE
: contains the error/SEVER code from the partner's SENDERR or SEVER. IPCODE is only valid when IPWHATRC = IPERROR or IPSABEND. See "APPC/VM Error/SEVER Codes" on page 183 for a description of the error/SEVER codes.

IPWHATRC
: contains the what-received code:

  IPDATA (01)—Only data was received, with no other indications.
  IPSEND (02)—Your partner has switched the conversation around and you are now in SEND state.
  IPERROR (03)—Your partner issued SENDERR.
  IPCNFRM (04)—Your partner is requesting confirmation.
  IPCNFSEV (05)—Your partner is requesting confirmation that it can issue a SEVER.
  IPSNORM (08)—Your partner issued a SEVER TYPE = NORMAL.
  IPSABEND (09)—Your partner issued a SEVER TYPE = ABEND.
  IPALLOCD (11)—The allocate data was received.

IPSENDOP
: contains the SEND option code:

IPRECV (10)—The RECEIVE is being completed.

IPAUDIT
contains the following flags (only when CC = 3):

IPAUDIT1
IPADANPX (X'10')—A protection exception occurred on your RECEIVE buffer (IPBFADR1).
IPADANAX (X'08')—An addressing exception occurred on your RECEIVE buffer (IPBFADR1).
IPAUDIT2
IPADRPPX (X'20')—A protection exception occurred on your communication partner's SEND data area.
IPADRPAX (X'10')—An addressing exception occurred on your communication partner's SEND data area.
IPADRLST (X'04')—Your communication partner had an invalid SEND list.
IPAUDIT3
IPADALEN (X'40')—A bad length is in your RECEIVE buffer list.
IPADATOT (X'10')—Your RECEIVE buffer length is invalid.
IPADTINV (X'08')—Your communication partner's data stream has an invalid logical record length.
IPADTTRN (X'02')—Your communication partner started, but did not finish, sending a logical record and tried to change to RECEIVE state.

CP reflects the exception to you, if a message is pending for the specified path when you issue the RECEIVE, and one of the following is present:

- Protection exception on the RECEIVE buffer
- Addressing exception on the RECEIVE buffer.

CP reflects the error in IPAUDIT1 when the RECEIVE completes, if there is no message pending for the specified path when you issue the RECEIVE, and one of the following is present:

- Protection exception on the RECEIVE buffer
- Addressing exception on the RECEIVE buffer.

IPBFLN2F
contains one of the following depending on the value of IPWHATRC:

- If IPWHATRC is equal to IPDATA, then IPBFLN2F contains the number of bytes that were sent but did not fit into the defined RECEIVE area. This length is the byte length of your partner's SENDDATA minus the length that you already received.

For example, your partner issues a SENDDATA with a data length of 100, and you issue a RECEIVE with a buffer length of 40. In this case, IPBFLN2F would contain 60.

- If IPWHATRC is not equal to IPDATA, then IPBFLN2F contains the number of bytes left in the defined RECEIVE area.

Note: Nondata indications such as IPSEND do not appear in IPWHATRC until all data sent with or before the nondata function notice has been received. For example, IPWHATRC would be IPDATA if the following occurred:

1. Your communication partner issued SENDDATA RECEIVE = YES with a data length of 200 bytes, and
2. You did a RECEIVE for 199 bytes.

When you issue a RECEIVE for the 200th byte, then IPWHATRC would become IPSEND.

Data may have been received for any IPWHATRC value.

RECEIVE Program Exceptions

The program exceptions for RECEIVE are:

Type	Description
Addressing	The parameter list address is outside of the virtual machine. An addressing exception also occurs for any of the following: • An invalid buffer address in the parameter list • An invalid buffer address in the buffer list • An invalid buffer list address.
Operation	An external interrupt buffer has not been declared with the DCLBFR function, or the invoker is not in supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user. A protection exception also occurs for any of the following: • Buffer address in the parameter list is protected • Buffer address in the buffer list is protected • Buffer list address is protected.

Type	Description
Specification	The parameter list is not on a doubleword boundary.

State Changes

A state check occurs (CC = 1 and IPRCODE = 32, 35 or 36) if your virtual machine is not in SEND or RECEIVE state on this path. If you are the target of a CONNECT on the path, no state check occurs on RECEIVE while you are in CONNECT state, unless you have received all of the allocate data. If you issue a RECEIVE from CONNECT state after you have received all of the allocate data, you get a state check (IPRCODE = 32).

A state check also occurs (IPRCODE = 44) if you start, but do not finish, sending a logical record on this path.

No state change occurs when CC = 1. State changes can occur when either of the following occurs:

- The function completes, that is, control is returned to the virtual machine (CC = 2 or 3)
- The function complete interrupt is accepted by your virtual machine or you use TESTCMPL to discover that the function was completed.

The state change depends on the IPWHATRC value:

IPWHATRC Value	State	Cause
IPDATA	RECEIVE or CONNECT	Depends on your state when you invoke RECEIVE: • If you are *not* in CONNECT state, you enter RECEIVE state. The RECEIVE completed without you receiving any non-data indications. Your communication partner sent data to complete the RECEIVE, or the receive length was zero. • If you are in CONNECT state, you remain in CONNECT state. An IPDATA indication means that there is still some allocation data left for you to receive.
IPSEND	SEND	Your communication partner issued RECEIVE or SENDDATA RECEIVE = YES to complete the RECEIVE.
IPERROR	RECEIVE	Your communication partner issued a SENDERR to complete the RECEIVE.
IPCNFRM	CONFIRM	Your communication partner issued a SENDCNF TYPE = NORMAL to complete the RECEIVE.

IPWHATRC Value	State	Cause
IPCNFSEV	CONFIRM	Your communication partner issued a SENDCNF TYPE = SEVER to complete the RECEIVE.
IPSNORM	SEVER	Your communication partner issued a SEVER TYPE = NORMAL to complete the RECEIVE.
IPSABEND	SEVER	Your communication partner issued a SEVER TYPE = ABEND to complete the RECEIVE.
IPALLOCD	CONNECT	You completed the RECEIVE of the allocate data.

RECEIVE Completion

You cannot issue another SEND[1], RECEIVE, or SEVER TYPE = NORMAL on the same path, until the outstanding RECEIVE is complete. The RECEIVE is complete when your communication partner does one of the following:

- Sends a message or messages to your virtual machine to completely fill the receive area specified by the RECEIVE
- Issues RECEIVE, SENDCNF, SENDDATA RECEIVE = YES, SENDERR, or SEVER.

When the receive area has zero length and you are in:

- RECEIVE state, the RECEIVE completes immediately
- SEND state, the RECEIVE completes when your communication partner or an intermediate communication server receives notice that it is in SEND state.

When the RECEIVE is for the allocate data created by the CONNECT, the RECEIVE always completes immediately. See "Allocate Data That Your Communication Partner May Receive" on page 98 for information on the type of data you may receive. If your communication partner severs, the RECEIVE completes immediately with no indication of the SEVER in the output parameter list. However, you will get a SEVER interrupt.

The RECEIVE completes immediately, when a message is pending with one of the following:

- No data
- Data greater than the size of the RECEIVE area that you specified
- Data equal to the size of the RECEIVE area that you specified.

---

1 SEND generally refers to all of the APPC/VM "SEND" functions: SENDCNF, SENDCNFD, SENDDATA, SENDERR, and SENDREQ.

What Happens to Your Communication Partner

Your communication partner may get a function complete or a message pending interrupt, or your partner may not receive any indications.

Your partner gets a function complete interrupt, for either of the following:

- Your partner has a SENDDATA RECEIVE = NO or SENDDATA RECEIVE = YES with a zero answer length outstanding on its half of the path, and you have received all the data sent.
- Your partner has a SENDERR outstanding on its half of the path.

If your communication partner has a SENDCNF or SENDDATA RECEIVE = YES outstanding on its half of the path, your partner does not get any notification of your actions on that path until you respond.

Your partner gets a message pending interrupt, if your partner has the following qualities:

- Has no function outstanding on its half of the path
- Is in RECEIVE state on its half of the path
- Is enabled for message pending interrupts.

See "Message Pending External Interrupt" on page 147.

IUCV RTRVBFR

Function Code: X'02'

RTRVBFR does the following:

- Stops all IUCV and APPC/VM outstanding messages on non-control paths
- Severs all IUCV and APPC/VM non-control communication paths
- Ends IUCV and APPC/VM communications, except for IUCV and APPC/VM communications on control paths.

RTRVBFR has no effect on control paths. Control paths are paths that your virtual machine establishes by issuing the CONTROL = YES parameter on CONNECT.

The *VM System Facilities for Programming* describes the IUCV version of this function unrelated to APPC/VM.

The IUCV RTRVBFR syntax is:

label **IUCV**	**RTRVBFR**	Required

RTRVBFR Parameter List

The RTRVBFR function does not use a parameter list.

Error Codes and Exceptions

Condition Codes

CC = 0	CC = 1	CC = 2	CC = 3
X	Not Possible	Not Possible	Not Possible

CC = 0
Normal Completion.

RTRVBFR Program Exceptions

The program exception for RTRVBFR is:

Type	Description
Operation	An application interrupt buffer has not been declared with the DCLBFR function, or the invoker is not in supervisor state.

State Changes

RTRVBFR does not act on any one path; therefore no state checks occur. When RTRVBFR is done executing, you are in RESET state on all paths, except your control paths. Your control paths experience no state change. Reset state means that the path no longer exists.

RTRVBFR Completion

The RTRVBFR function completes immediately.

What Happens to Your Communication Partner

When you invoke RTRVBFR, all your non-control APPC/VM and IUCV communication paths are severed. APPC/VM informs your communication partners as if you issued a SEVER TYPE=ABEND with SEVER code X'0610'.

APPCVM SENDCNF

Function Code: X'04'

Use this function to send a confirmation request from your virtual machine to another virtual machine or to your own virtual machine.

The APPCVM SENDCNF syntax is:

label **APPCVM**	**SENDCNF,**	Required
	PRMLIST = *label*/(*reg*),	Required
	TYPE = NORMAL/SEVER,	Required
	MF = L,	Optional
	PATHID = *label*/(*reg*),	Optional
	WAIT = YES/NO	Optional

PRMLIST = *label*/(*reg*)
lets you specify the address of the APPC/VM parameter list. The address must be a guest real address (real to the virtual machine), and the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

reg
is the register number that contains the address of the parameter list.

TYPE = NORMAL/SEVER
specifies what type of confirmation is being requested.

- TYPE = NORMAL requests a normal confirmation.
- TYPE = SEVER requests a confirmation that you may issue a SEVER.

MF = L
expands the APPCVM macro to generate the instructions necessary to initialize the APPC/VM parameter list as specified, but not to invoke the APPC/VM function.

PATHID = *label*/(*reg*)
lets you identify the path on which to send the confirmation request.

label
is the relocatable label of a halfword that contains the path id.

reg
is the register number that contains the path id in the low-order halfword.

WAIT = YES/NO
lets you specify when control is returned to your virtual machine.

- WAIT = YES returns control to your virtual machine when the SENDCNF is complete.
- WAIT = NO returns control to your virtual machine as soon as the SENDCNF request is initiated. When the SENDCNF completes, you are notified with a function complete interrupt.

SENDCNF Parameter List

The APPCVM SENDCNF parameter list has the following input format:

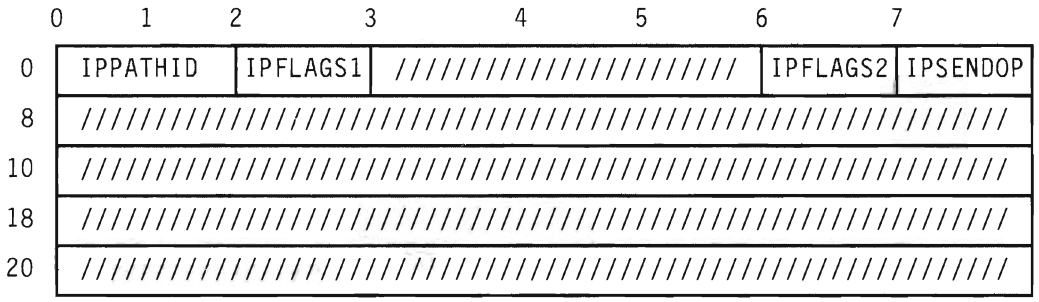


Figure 42. APPCVM SENDCNF Input Parameter List

The supplied parameters are:

IPPATHID
contains the path id on which the confirmation request is sent.

IPFLAGS1
contains the input flag:

IPAPPCSN (X'02')—An APPC SEND function was issued.

IPFLAGS2
contains the input flag:

IPWAIT (X'80')—A synchronous return was requested.

IPSENDOP
contains one of the following SEND option codes:

IPCNFRM (04)—You are requesting confirmation from your communication partner.
IPCNFSEV (05)—You are requesting confirmation from your communication partner that you can issue a SEVER.

Error Codes and Exceptions

Condition Codes

CC = 0	CC = 1	CC = 2	CC = 3
X	X	X	Not Possible

CC = 0
SENDCNF started successfully, but has not yet completed. When it completes, CP sends you a function complete interrupt. The function complete interrupt buffer has the same format as the SENDCNF output parameter list (see CC = 2).

Note: When WAIT = YES, CC = 0 is not possible.

CC = 1
An error occurred. The output parameter list is the same as the input shown in "SENDCNF Parameter List" on page 125, except that the return code is stored in IPRCODE.

You may get the following return codes (listed here in decimal):

01 You specified a path id that is not yet established.

03 A function is pending on this path.

30 You specified an APPC/VM function on a non-APPC path.

32 SENDCNF is an invalid function from CONNECT state.

34 SENDCNF is an invalid function from RECEIVE state.

35 SENDCNF is an invalid function from CONFIRM state.

36 SENDCNF is an invalid function from SEVER state.

37 The connection was established with SYNCLVL = NONE.

38 The IPSENDOP field contains an invalid value.

44 Before invoking SENDCNF, you started, but did not finish, sending a logical record.

CC = 2
SENDCNF completed (see section "SENDCNF Completion" on page 129), with no errors. When WAIT = NO, CC = 2 is not possible. The output parameter list when CC = 2 is:

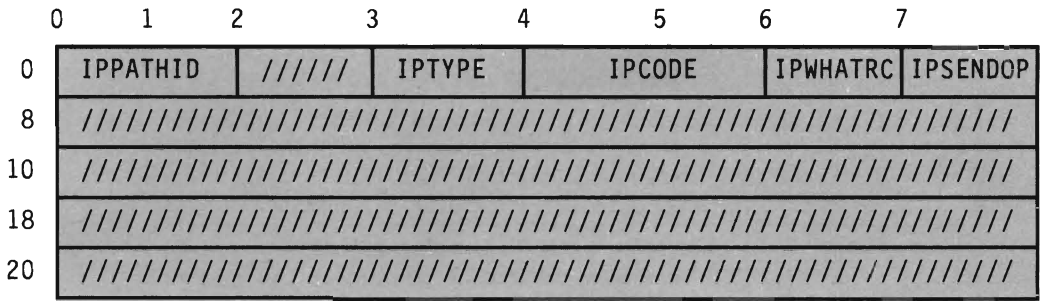


Figure 43. APPCVM SENDCNF Output Parameter List (Function Complete Interrupt)

The parameters are:

IPPATHID
contains the path id on which the function is complete.

IPTYPE
contains the function complete interrupt code (IPTYPFCA, X'87').

IPCODE
contains the error/SEVER code from the partner's SENDERR or SEVER. IPCODE is only valid when IPWHATRC = IPERROR or IPSABEND. See "APPC/VM Error/SEVER Codes" on page 183 for a description of the error/SEVER codes.

IPWHATRC
contains the what-received code:

IPCOMP (00)—Your partner's SENDCNFD completed the function.
IPERROR (03)—Your partner issued SENDERR.
IPSABEND (09)—Your partner issued a SEVER TYPE = ABEND.

IPSENDOP
contains one of the following SEND option codes:

IPCNFRM (04)—The SENDCNF TYPE = NORMAL is being completed.
IPCNFSEV (05)—The SENDCNF TYPE = SEVER is being completed.

SENDCNF Program Exceptions

The program exceptions for SENDCNF are:

Type	Description
Addressing	The parameter list address is outside of the virtual machine.
Operation	An external interrupt buffer has not been declared with the DCLBFR function, or the invoker is not in supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user.
Specification	The parameter list is not on a doubleword boundary.

State Changes

A state check occurs (IPRCODE = 32, 34, 35 or 36) if your virtual machine is not in SEND state on this path. A state check also occurs (IPRCODE = 44) if you started, but did not finish, sending a logical record on this path.

No state change occurs when CC = 1. State changes can occur when the function completes, and one of the following occurs:

- You regain control (CC = 2).
- You accept the function complete interrupt (CC = 0), or you use TESTCMPL to discover that the function was completed.

The state change depends on the IPWHATRC value:

IPWHATRC Value	State	Cause
IPCOMP	SEND	The **SENDCNF TYPE = NORMAL** was completed by the communication partner issuing a SENDCNFD.
IPCOMP	SEVER	The **SENDCNF TYPE = SEVER** was completed by the communication partner issuing a SENDCNFD.
IPERROR	RECEIVE	The SENDCNF was completed by the communication partner issuing a SENDERR.
IPSABEND	SEVER	The SENDCNF was completed by the communication partner issuing a SEVER TYPE = ABEND.

Note: When you issue SENDCNF, you have no way of telling if the SENDERR or SEVER indication received is in response to your confirmation request or if it was issued before your SENDCNF.

SENDCNF Completion

After issuing a SENDCNF, you cannot issue another SEND[2] , RECEIVE, or SEVER TYPE = NORMAL on that path until the outstanding SENDCNF is complete. SENDCNF is complete when the communication partner responds with a SENDCNFD, SENDERR or SEVER.

What Happens to Your Communication Partner

Your communication partner's outstanding function may complete, or your partner may get a message pending interrupt.

If your partner has a RECEIVE, SENDDATA RECEIVE = YES, or SENDERR outstanding on its half of the path, your partner's outstanding function is completed.

Your partner gets a message pending interrupt, if your partner has the following qualities:

- Has no function outstanding on its half of the path
- Is in RECEIVE state on its half of the path
- Is enabled for message pending interrupts.

See "Message Pending External Interrupt" on page 147.

---

2 SEND generally refers to all of the APPC/VM "SEND" functions: SENDCNF, SENDCNFD, SENDDATA, SENDERR, and SENDREQ.

APPCVM SENDCNFD

Function Code: X'04'

This function sends a confirmation response from your virtual machine to another virtual machine or to your own virtual machine. Invoke this as a positive response to your partner sending a SENDCNF. (For a negative response, invoke SENDERR.)

The APPCVM SENDCNFD syntax is:

label **APPCVM**	**SENDCNFD,**	Required
	PRMLIST = *label*/(*reg*),	Required
	MF = **L,**	Optional
	PATHID = *label*/(*reg*)	Optional

PRMLIST = *label*/(*reg*)
lets you specify the address of the APPC/VM parameter list. The address must be a guest real address (real to the virtual machine), and the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

reg
is register number that contains the address of the parameter list.

MF = **L**
expands the APPC/VM to generate the instructions necessary to initialize the APPC/VM parameter list as specified, but not to invoke the APPC/VM function.

PATHID = *label*/(*reg*)
lets you identify the path on which to send the confirmation.

label
is the relocatable label of a halfword that contains the path id.

reg
is the register number that contains the path id in the low-order halfword.

SENDCNFD Parameter List

The APPCVM SENDCNFD parameter list has the following input format:

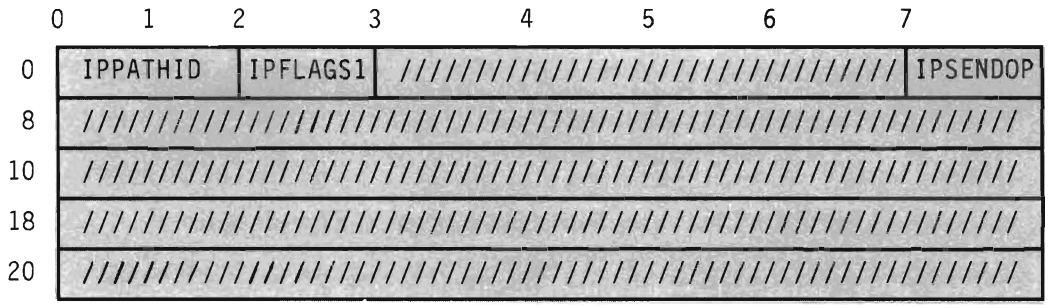


Figure 44. APPCVM SENDCNFD Input Parameter List

The supplied parameters are:

IPPATHID
: contains the path id on which the confirmation is sent.

IPFLAGS1
: contains the input flag:

  IPAPPCSN (X'02')—The APPC SEND function was issued.

IPSENDOP
: contains the SEND option code:

  IPCNFRMD (06)—Your communication partner is sending confirmation as requested.

Error Codes and Exceptions

Condition Codes

CC = 0	CC = 1	CC = 2	CC = 3
Not Possible	X	X	Not Possible

SENDCNFD always completes immediately.

CC = 1
An error occurred. The parameter list format is the same as the input shown in "SENDCNFD Parameter List," except that the return code is stored in IPRCODE.

You may get the following return codes (listed here in decimal):

01 You specified a path id that is not yet established.

30 You specified an APPC/VM function on a non-APPC path.

32 SENDCNFD is an invalid function from CONNECT state.

33 SENDCNFD is an invalid function from SEND state.

34 SENDCNFD is an invalid function from RECEIVE state.

36 SENDCNFD is an invalid function from SEVER state.

38 The IPSENDOP field contains an invalid value.

CC = 2
SENDCNFD completed (see "SENDCNFD Completion" on page 133).
The output parameter list when CC = 2 is:

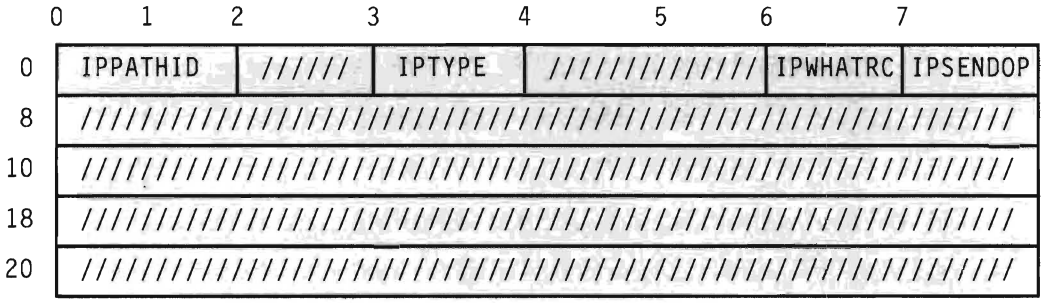


Figure 45. APPCVM SENDCNFD Output Parameter List

The parameters are:

IPPATHID
contains the path id on which the function is complete.

IPTYPE
contains the function complete interrupt code (IPTYPFCA, X'87').

IPWHATRC
contains the what-received code.

IPCOMP (00)—SENDCNFD completed in response to a SENDCNF TYPE = NORMAL.
IPSNORM (08)—SENDCNFD completed in response to a SENDCNF TYPE = SEVER.

IPSENDOP
contains the SEND option code:

IPCNFRMD (06)—The SENDCNFD is being completed.

SENDCNFD Program Exceptions

The program exceptions for SENDCNFD are:

Type	Description
Addressing	The parameter list address is outside of the virtual machine.
Operation	An external interrupt buffer has not been declared with the DCLBFR function, or the invoker is not in supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user.
Specification	The parameter list is not on a doubleword boundary.

State Changes

A state check will occur (IPRCODE = 32, 33, 34, or 36) if the virtual machine is not in CONFIRM state on this path.

No state change occurs when CC = 1. State changes do occur when the function completes; that is, when control is returned to the virtual machine (CC = 2). The state change depends on the value of IPWHATRC:

IPWHATRC Value	State	Cause
IPCOMP	RECEIVE	SENDCNFD was in response to SENDCNF TYPE = NORMAL.
IPSNORM	SEVER	SENDCNFD was in response to SENDCNF TYPE = SEVER.

SENDCNFD Completion

Because the SENDCNFD function completes immediately, you can issue another function on the same path as soon as your virtual machine regains control.

What Happens to Your Communication Partner

You can only issue SENDCNFD in response to a SENDCNF. SENDCNFD always causes the completion of the SENDCNF. If your communication partner issued SENDCNF with WAIT = NO, and your partner is enabled for function complete interrupts, then your partner will get a function complete interrupt.

APPCVM SENDDATA

Function Code: X'04'

This function sends data from your virtual machine to another virtual machine or to your own virtual machine. You can also use this function to switch the conversation state from SEND state to RECEIVE state and to define an answer area for your partner's SENDDATA.

The APPCVM SENDDATA syntax is:

label **APPCVM**	**SENDDATA,**	Required
	PRMLIST = *label*/(*reg*),	Required
	RECEIVE = **YES/NO,**	Required
	MF = **L,**	Optional
	PATHID = *label*/(*reg*),	Optional
	BUFLIST = **YES/NO,**	Optional
	BUFFER = *label*/(*reg*),	Optional
	BUFLEN = *label*/(*reg*),	Optional
	ANSLIST = **YES/NO,**	Optional
	ANSBUF = *label*/(*reg*),	Optional
	ANSLEN = *label*/(*reg*),	Optional
	WAIT = **YES/NO**	Optional

PRMLIST = *label*/(*reg*)
lets you specify the address of the APPC/VM parameter list. The address must be a guest real address (real to the virtual machine), and the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

reg
is the register number that contains the address of the parameter list.

RECEIVE = **YES/NO**
lets you specify whether or not to define an answer area.

- RECEIVE = YES defines an answer area.
- RECEIVE = NO does not define an answer area.

MF = **L**
expands the APPCVM macro to generate the instructions necessary to initialize the APPC/VM parameter list as specified, but not to invoke the APPC/VM function.

PATHID = *label*/(*reg*)
lets you specify the path id on which you send the data.

label
is the relocatable label of a halfword that contains the path id.

reg
is the register number that contains the path id in the low-order halfword.

BUFLIST = **YES/NO**
specifies the type of buffer address that the BUFFER parameter refers to. For more information, see "Specifying Buffers on SENDDATA" on page 137.

- BUFLIST = YES refers to a list of addresses.
- BUFLIST = NO refers to a single address.

BUFFER = *label*/(*reg*)
specifies the area(s) from which CP takes the data to be sent. For more information, see "Specifying Buffers on SENDDATA" on page 137.

label
is the relocatable label in storage where CP gets the data to send.

reg
is the register number that contains the address of the storage area. The storage area contains the data.

BUFLEN = *label*/(*reg*)
specifies the length of the area(s) from which APPC/VM takes the data to be sent. This length is not related to the length of a logical record. For more information, see "Specifying Buffers on SENDDATA" on page 137.

label
is the relocatable label of the fullword that contains the length.

reg
is the register number that contains the length.

ANSLIST = **YES/NO**
specifies the type of address that the ANSBUF parameter refers to. For more information, see "Specifying Buffers on SENDDATA" on page 137.

- ANSLIST = YES specifies that ANSBUF is referring to a list of addresses.
- ANSLIST = NO specifies that ANSBUF is referring to data.

ANSBUF = *label*/(*reg*)
specifies the address of the area(s) where APPC/VM places the data that is sent by your communication partner. For more information, see "Specifying Buffers on SENDDATA" on page 137.

label
is the relocatable label in storage that contains the data.

reg
is the register number that contains the address of the storage area. This area contains the data.

ANSLEN = *label*/(*reg*)
specifies the length of the area(s) into which APPC/VM places the data sent by the communication partner. For more information, refer to "Specifying Buffers on SENDDATA" on page 137.

label
is the relocatable label of the fullword that contains the length.

reg
is the register number that contains the length.

WAIT = **YES/NO**
lets you specify when control is returned to your virtual machine.

- WAIT = YES returns control to your virtual machine when the SENDDATA is complete.
- WAIT = NO returns control to your virtual machine when the SENDDATA request is initiated.

What the SENDDATA Data Looks Like

The SENDDATA data is made up of logical records. Each logical record has a 2-byte length field followed by a data field. The data field can range from 0 to 32,765 bytes long. The length field has the 15-bit length of the record, plus the high-order bit (APPC/VM does not examine the high-order bit). The length of the record includes the 2-byte length field; so, the length of the record is the data field length plus two. These logical record length values would be invalid, since the length must be greater than or equal to two and the high-order bit is ignored:

X'0000'
X'0001'
X'8000'
X'8001'

The logical record length does not depend on the length of data specified in a single SENDDATA BUFLEN = label/(reg). In other words, the data may consist of one or more complete records, the beginning of a record, the middle of a record, or the end of a record. You may specify any of the following combinations:

- One or more complete records, followed by the beginning of a record
- The end of a record, followed by one or more complete records
- The end of a record, followed by one or more complete records, followed by the beginning of a record
- The end of a record, followed by the beginning of a record.

A complete logical record has the 2-byte length field and all bytes of the data field, determined by the logical record length. If the data field has a length of zero, the complete logical record has only the 2-byte length field.

Specifying Buffers on SENDDATA

For APPCVM SENDDATA, you can specify the buffers with a single address and a single length (BUFLIST = NO or ANSLIST = NO) or with a list of addresses and lengths (BUFLIST = YES or ANSLIST = YES).

When you specify the buffer with a single address and a single length, BUFFER (or ANSBUF) specifies the address and BUFLEN (or ANSLEN) specifies the length. When you specify the buffer with a list of addresses and lengths, BUFFER (or ANSBUF) specifies the address of the list and BUFLEN (or ANSLEN) specifies the sum of the lengths of the buffers in the list.

You must follow these conventions when you use address lists (BUFLIST = YES or ANSLIST = YES):

- The list must begin on a doubleword boundary.
- Each list entry must be two fullwords:
  - The first fullword is the address of that portion of the list, and
  - The second fullword is the length of that portion of the list.

When you use an address list, the addresses and lengths in the address list are updated during APPC/VM processing. Do not alter them during processing or assume that they are unchanged when APPC/VM processing is complete. Also, APPC/VM assumes that there is another entry in the list until the sum of the lengths of the entries processed is equal to the total length specified (by BUFLEN or ANSLEN).

SENDDATA Parameter List

The APPCVM SENDDATA parameter list has the following input format:

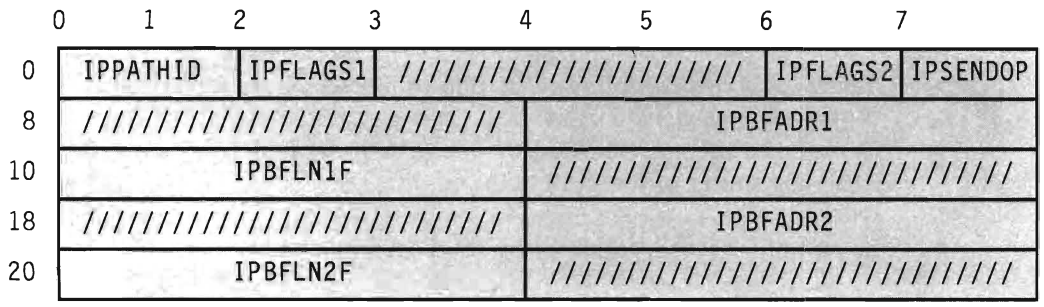


Figure 46. APPCVM SENDDATA Input Parameter List

The supplied parameters are:

IPPATHID
: contains the path id over which you send the data.

IPFLAGS1
: contains the flags:

  IPBUFLST (X'40')—You specified the buffer list option.
  IPANSLST (X'08')—You specified the answer list option.
  IPAPPCSN (X'02')—The APPC SEND function was issued.

IPFLAGS2
: contains the input flag:

  IPWAIT (X'80')—A synchronous return is desired.

IPSENDOP
: contains one of the following SEND option codes:

  IPDATA (01)—You are sending data.
  IPSNDRCV (02)—You are sending the data, the conversation is to be turned around, and an answer area is defined by IPBFADR2 and IPBFLN2F.

IPBFADR1
: contains the address of the area from which APPC/VM takes the message or the address of the address or length list. See "Specifying Buffers on SENDDATA" on page 137.

IPBFLN1F
: contains the length of the message being sent. This length is not related to the length of a logical record. It is used only to determine the length of the data to be moved by this SENDDATA. See "Specifying Buffers on SENDDATA" on page 137.

IPBFADR2
: contains the address of the area where APPC/VM places the answer from the communication partner, or the address of the address or length list (only valid when IPSENDOP=02). See "Specifying Buffers on SENDDATA" on page 137.

IPBFLN2F
: contains the length of the answer buffer (only valid when IPSENDOP=02). See "Specifying Buffers on SENDDATA" on page 137.

Error Codes and Exceptions

Condition Codes

CC=0	CC=1	CC=2	CC=3
X	X	X	X

CC=0
SENDDATA started successfully, but has not yet completed. If your virtual machine is enabled for function complete interrupts, one is sent to your virtual machine when SENDDATA completes. The function complete interrupt has the same format as the SENDDATA output parameter list (see CC=2,3). However, no condition code may be used to determine if a nonzero value is stored in IPAUDIT.

When control is returned to your virtual machine with CC=0, the parameter list may have been altered.

Note: When you specify WAIT=YES, CC=0 is not possible.

CC=1
An error occurred before the SENDDATA was initiated. The output parameter list is the same as the input shown in "SENDDATA Parameter List" on page 137, except that the return code is stored in IPRCODE. Other fields in the parameter list may also have been altered.

You may get the following return codes (listed here in decimal):

01 You specified a path id that is not yet established.

03 A function is pending on this path.

06 A protection exception occurred on your communication partner's predefined answer or RECEIVE area.

07 An addressing exception occurred on your communication partner's predefined answer or RECEIVE area.

10 Your buffer length or answer length is negative.

22 Your communication partner's predefined answer list or RECEIVE list is invalid.

23 A length specified in your SEND buffer list is negative.

24 The total length that you specified is not the total of the lengths in your SEND buffer list.

26 The buffer list address is not on a doubleword boundary.

27 The answer list address is not on a doubleword boundary.

30 You specified an APPC/VM function on a non-APPC path.

32 SENDDATA is an invalid function from CONNECT state.

34 SENDDATA is an invalid function from RECEIVE state.

35 SENDDATA is an invalid function from CONFIRM state.

36 SENDDATA is an invalid function from SEVER state.

38 There was an invalid value in IPSENDOP field.

42 There is an invalid logical record length in your data stream.

44 You started, but did not finish, sending a logical record. For SENDDATA, this can only occur if you specified RECEIVE = YES.

Return codes 6, 7, 22, 23, 24, 42, and 44 can only occur if your communication partner defined an answer area or RECEIVE area before you issued the SENDDATA. If your communication partner did not define an answer area or RECEIVE area before you issued the SENDDATA, CP reports those error conditions to you in the corresponding audit flags when your partner's RECEIVE completes. Your partner learns of the error through one of the following:

- A protection exception
- An addressing exception
- A return code on the RECEIVE
- In the audit flags when the RECEIVE completes.

For return codes 22, 23, 24, 42 or 44, data may have been copied to your communication partner's virtual machine before the error was detected. How much data was copied is unpredictable.

CC = 2 or
CC = 3

Function completed (also see "SENDDATA Completion" on page 146). When CC = 2, then the function completed with no error caused by the invoker. When CC = 3, there is some error information in IPAUDIT. When WAIT = NO, CC = 3 is not possible. The output parameter list when CC = 2 or 3 is:

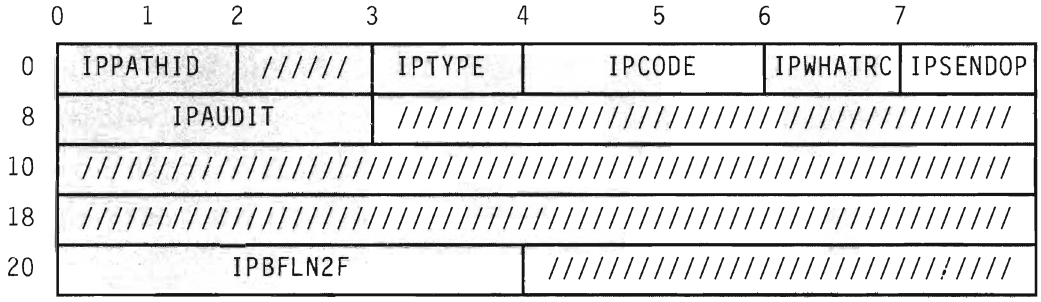


Figure 47. APPCVM SENDDATA Output Parameter List (Function Complete Interrupt)

The parameters are:

IPPATHID
: contains the path id on which the function is complete.

IPTYPE
: contains the function complete interrupt code (IPTYPFCA, X'87').

IPCODE
: contains the error/SEVER code from the partner's SENDERR or SEVER. IPCODE is only valid when IPWHATRC = IPERROR or IPSABEND. See "APPC/VM Error/SEVER Codes" on page 183 for a description of the error/SEVER codes.

IPWHATRC
: contains the what-received code. For RECEIVE = YES or RECEIVE = NO:

  IPCOMP (00)—Either of the following has occurred:
  - The SENDDATA RECEIVE = NO completed normally.
  - The SENDDATA RECEIVE = YES or NO completed with an error on your SEND buffer or on your partner's answer or RECEIVE buffer. See the IPAUDIT description.

  IPERROR
  (03)—Your partner issued SENDERR.

  IPSABEND
  (09)—Your partner issued a SEVER TYPE = ABEND.

  For RECEIVE = YES only:

IPDATA (01)—Only data was received.
IPSEND (02)—Your partner switched the conversation around and you are now in SEND state.
IPCNFRM
(04)—Your partner is requesting confirmation.
IPCNFSEV
(05)—Your partner is requesting confirmation that it can issue a SEVER.
IPSNORM
(08)—Your partner issued a SEVER TYPE = NORMAL.

IPSENDOP
contains one of the following SEND option codes:

IPDATA (01)—Your SENDDATA RECEIVE = NO is completing.
IPSNDRCV (02)—Your SENDDATA RECEIVE = YES is completing.

IPAUDIT
contains the following flags (if it's nonzero). If you specified WAIT = NO, IPAUDIT is in the function complete interrupt.

IPAUDIT1
IPADSNPX (X'40')—A protection exception occurred on your SEND buffer (IPBFADR1).
IPADSNAX (X'20')—An addressing exception occurred on your SEND buffer (IPBFADR1).
IPADANPX (X'10')—A protection exception occurred on your answer buffer (IPBFADR2).
IPADANAX (X'08')—An addressing exception occurred on your answer buffer (IPBFADR2).
IPAUDIT2
IPADRCPX (X'80')—A protection exception occurred on your communication partner's answer area or RECEIVE area.
IPADRCAX (X'40')—An addressing exception occurred on your communication partner's answer area or RECEIVE area.
IPADRPPX (X'20')—A protection exception occurred on your communication partner's SEND data area.
IPADRPAX (X'10')—An addressing exception occurred on your communication partner's SEND data area.
IPADRLST (X'04')—Your communication partner had an invalid SEND, answer or RECEIVE list.
IPAUDIT3
IPADBLEN (X'80')—A bad length is in your SEND buffer list.
IPADALEN (X'40')—A bad length is in your SEND answer list.
IPADBTOT (X'20')—Your total SEND buffer length is invalid.

IPADATOT (X'10')—Your total SEND answer length is invalid.
IPADTINV (X'08')—There is an invalid logical record length in your communication partner's data stream.
IPADIINV (X'04')—There is an invalid logical record length in your data stream.
IPADTTRN (X'02')—Your communication partner started, but did not finish, sending a logical record and tried to change to RECEIVE state.
IPADITRN (X'01')—You started, but did not finish, sending a logical record and tried to change to RECEIVE state.

CP reflects the exception to you, if your communication partner defined an answer or RECEIVE area before you issued the SENDDATA, and one of the following is present:

- Protection exception on your SEND buffer
- Addressing exception on your SEND buffer.

CP reflects the error to you in IPAUDIT1 when the SENDDATA completes, if your communication partner did not define an answer or RECEIVE area before you issued the SENDDATA, and one of the following is present:

- Protection exception on your SEND buffer
- Addressing exception on your SEND buffer.

IPBFLN2F
contains one of the following depending on the value of IPWHATRC.

- If IPWHATRC is equal to IPDATA, then IPBFLN2F contains the number of bytes that were sent by your communication partner, but did not fit into the defined answer area. This length is not the length of the APPC data stream being sent. Rather, it is the length of the data that has arrived and is ready to receive.

- If IPWHATRC is not equal to IPDATA, then IPBFLN2F contains the number of bytes left in your defined answer area.

Note: Nondata indications such as IPSEND do not appear in IPWHATRC until all data sent with or before the nondata function notice has been received. For example, IPWHATRC would be IPDATA if the following occurred:

1. You did a SENDDATA RECEIVE=YES with a 199-byte answer area, and

2. Your communication partner issued SENDDATA RECEIVE=YES with a data length of 200 bytes.

When you issue a RECEIVE for the 200th byte, then IPWHATRC would become IPSEND.

When RECEIVE = YES is specified, data may have been received for any value of IPWHATRC.

SENDDATA Program Exceptions

The program exceptions for SENDDATA are:

Type	Description
Addressing	The parameter list address is outside of the virtual machine. An addressing exception also occurs for any of the following: • An invalid buffer address in the parameter list • An invalid buffer address in the buffer list • An invalid buffer list address.
Operation	An external interrupt buffer has not been declared with the DCLBFR function, or the invoker is not in supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user. A protection exception also occurs for any of the following: • Buffer address in the parameter list is protected • Buffer address in the buffer list is protected • Buffer list address is protected.
Specification	The parameter list is not on a doubleword boundary.

State Changes

A state check occurs (IPRCODE = 32, 34, 35, or 36) if your virtual machine is not in SEND state on this path. A state check also occurs (IPRCODE = 44) if you started, but did not finish, sending a logical record on this path at the completion of the SEND portion of your SENDDATA RECEIVE = YES.

When you issue SENDDATA RECEIVE = YES, your communication partner receives notice of this as if you had issued SEND followed by a RECEIVE. SENDDATA RECEIVE = YES is like the following sequence of functions:

1. SENDDATA RECEIVE = NO
2. RECEIVE.

When the "RECEIVE" part of the SENDDATA RECEIVE = YES begins, you should have completed sending any outstanding logical records. For example, the following sequence would cause an error:

1. You issue **SENDDATA RECEIVE=YES BUFLEN=999** to send a logical record with a logical record length of 1000 bytes.

2. Your partner **RECEIVE**s the 999 bytes.

3. The "RECEIVE" portion of your SENDDATA RECEIVE begins.

In this situation, the error is caused because you did not send all 1000 bytes; therefore, you did not complete sending the outstanding logical record.

No state change occurs when CC=1. State changes occur when:

- The function completes; that is, control returns to the virtual machine (CC=2 or 3).

- The function complete interrupt is accepted by the virtual machine (CC=0) or you complete the function using TESTCMPL.

The state change depends on the IPWHATRC value:

IPWHATRC Value	State	Cause
IPCOMP	No state change occurs.	Either of the following could be the cause: • The SENDDATA RECEIVE=NO has completed normally. • The SENDDATA RECEIVE=YES or NO completed with an error on your SEND buffer or on your partner's answer or RECEIVE buffer. See IPAUDIT for details.
IPDATA	RECEIVE	Either of the following could be the cause: • The SENDDATA RECEIVE=YES with a nonzero length answer area was completed by your partner sending data. • The SENDDATA RECEIVE=YES with a zero length answer area was completed by your partner receiving the data sent.
IPSEND	SEND	The SENDDATA RECEIVE=YES was completed by the communication partner issuing a RECEIVE or SENDDATA RECEIVE=YES.
IPERROR	RECEIVE	The SENDDATA was completed by the communication partner issuing a SENDERR.
IPCNFRM	CONFIRM	The SENDDATA RECEIVE=YES was completed by the communication partner issuing a SENDCNF TYPE=NORMAL.
IPCNFSEV	CONFIRM	The SENDDATA RECEIVE=YES was completed by the communication partner issuing a SENDCNF TYPE=SEVER.

IPWHATRC Value	State	Cause
IPSNORM	SEVER	The SENDDATA RECEIVE = YES was completed by the communication partner issuing a SEVER TYPE = NORMAL.
IPSABEND	SEVER	The SENDDATA was completed by the communication partner issuing a SEVER TYPE = ABEND.

SENDDATA Completion

After issuing a SENDDATA, you cannot issue another SEND[3] , RECEIVE, or SEVER TYPE = NORMAL on that path until the outstanding SENDDATA is complete. When the SENDDATA completes for a communicator depends on the value you give to the RECEIVE parameter of SENDDATA:

- When RECEIVE = NO, your SENDDATA is complete when all of the data is copied out of your SEND buffer, or when your communication partner issues a SENDERR or a SEVER.
- When RECEIVE = YES and you specify a zero answer area, your SENDDATA is complete when all of the data is copied out of your SEND buffer.
- When RECEIVE = YES and you specify a nonzero answer area, then the SENDDATA is complete when all of the data is copied out of your SEND buffer and your communication partner:
  - Sends a message(s) to your virtual machine to completely fill the answer area specified on your virtual machine's SENDDATA, or
  - Issues RECEIVE, SENDCNF, SENDDATA RECEIVE = YES, SENDERR, or SEVER.

Remember when you specify SENDDATA RECEIVE = YES with a nonzero answer area length, you get one function complete interrupt when your communication partner or an intermediate communication server issues a function. But, when you specify SENDDATA RECEIVE = NO followed by a RECEIVE, you receive two function complete interrupts. The first interrupt is a result of the data being copied out of your SEND buffer; the second interrupt is when your RECEIVE is completed.

---

[3] SEND generally refers to all of the APPC/VM "SEND" functions: SENDCNF, SENDCNFD, SENDDATA, SENDERR, and SENDREQ.

What Happens to Your Communication Partner

Your communication partner's outstanding function may complete, or your partner may get a message pending interrupt.

If your partner has a RECEIVE or SENDERR outstanding on its half of the path, your partner's function is completed. If your partner has a SENDDATA RECEIVE = YES outstanding on its half of the path, your partner's function is completed when you send enough data to fill your partner's predefined RECEIVE area if you specified RECEIVE = NO. If you specify RECEIVE = YES on your SENDDATA, then your partner's SENDDATA RECEIVE = YES completes in any case.

Your partner gets a message pending interrupt, if your partner has the following qualities:

- Has no function outstanding on its half of the path
- Is in RECEIVE state on its half of the path
- Is enabled for message pending interrupts.

Message Pending External Interrupt

Any of the following functions can cause a message pending interrupt to be sent to your communication partner:

RECEIVE
SENDCNF
SENDDATA
SENDERR.

For all functions, except SENDDATA, the IPBFLN1F field is zero. In fact, the IPBFLN1F could contain zeroes for SENDDATA as well.

APPC/VM always sends the message pending external interrupt data to the APPC/VM interrupt buffer defined by DCLBFR. The APPC/VM message pending external interrupt has the following format:

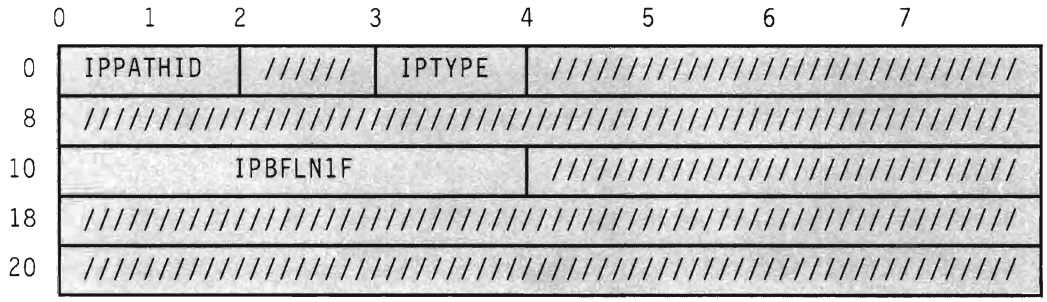


Figure 48. Message Pending External Interrupt

The parameters are:

IPPATHID
: contains the path id on which a message is pending.

IPTYPE
: contains the interrupt type for a message pending (IPTYPMPA, X'89').

IPBFLN1F
: contains the length of the pending message. This length is the length of the data that has arrived and is ready to receive.

  When the SENDDATA is sent by a virtual machine on the local system, the length in IPBFLN1F is the actual length of the data sent by the SENDDATA. On the other hand, when the SENDDATA is sent by a virtual machine on a remote system, the length in IPBFLN1F is the length of the data sent by TSAF's SENDDATA.

You get a message pending interrupt for a path only when your half of the path is in RECEIVE state. If your virtual machine is not in RECEIVE state on that path, the message pending interrupt is kept pending until your half of the path enters RECEIVE state. Then you get the message pending interrupt. When you receive the message, check the condition code or the IPWHATRC field to find what to do next. See "APPCVM RECEIVE" on page 111 for more information.

IUCV only reflects a message once. You cannot DESCRIBE a message again or get another message pending interrupt, if your virtual machine does any of the following:

- RECEIVEs the message
- DESCRIBEs the message
- Gets the message pending interrupt
- Is notified of the pending function on the completion of another function.

However, if you only partially received the message, you can issue additional RECEIVEs to receive the rest of the message.

APPCVM SENDERR

Function Code: X'04'

This function tells your communication partner that an error has occurred and causes a break in the usual SEND/RECEIVE sequence. This is done so your virtual machine can send error information.

The APPCVM SENDERR syntax is:

label **APPCVM**	**SENDERR,**	Required
	PRMLIST = *label*/(*reg*),	Required
	MF = **L,**	Optional
	PATHID = *label*/(*reg*),	Optional
	WAIT = **YES/NO,**	Optional
	CODE = *label*/(*reg*)	Restricted

PRMLIST = *label*/(*reg*)
lets you specify the address of the APPC/VM parameter list. The address must be a guest real address (real to the virtual machine), and the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

reg
is the register number that contains the address of the parameter list.

MF = **L**
expands the APPCVM macro to generate the instructions necessary to initialize the APPC/VM parameter list as specified, but not to invoke the APPC/VM function.

PATHID = *label*/(*reg*)
lets you specify the path id of the path on which you send the error notice.

label
is the relocatable label of a halfword that contains the path id.

reg
is the register number that contains the path id in the low-order halfword.

WAIT = YES/NO
lets you specify when control is returned to your virtual machine.

- WAIT = YES returns control to your virtual machine when the SENDERR is complete.
- WAIT = NO returns control to your virtual machine as soon as you issue the SENDERR request. When the SENDERR completes, you are notified with a function complete interrupt.

CODE = *label*/(*reg*)
specifies a 2-byte error code that your communication partner gets. IBM defines all the codes; applications may not define error/SEVER codes for their own use.

label
is a relocatable label in the storage area that contains the error code.

reg
is the register number that contains the error code in the low-order halfword.

Only communication servers (authorized by OPTION COMSRV in their directory entries) can specify CODE. When you do specify CODE, the APPCVM macro sets the IPCOMSRV flag.

For a complete list of the error/SEVER codes, see "APPC/VM Error/SEVER Codes" on page 183.

SENDERR Parameter List

The APPCVM SENDERR parameter list has the following input format:

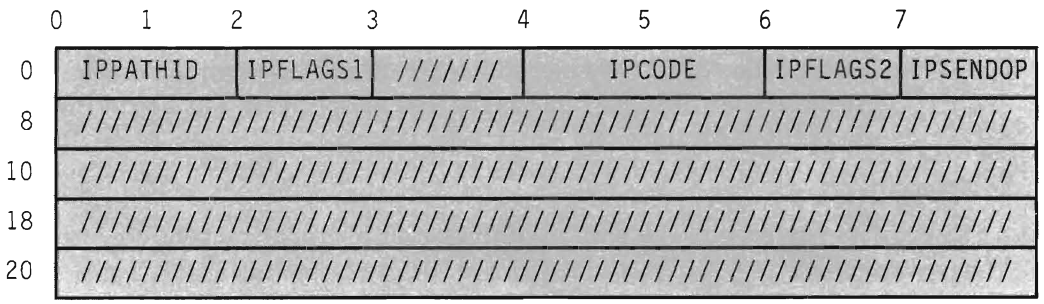


Figure 49. APPCVM SENDERR Input Parameter List

The supplied parameters are:

IPPATHID
contains the path id over which to send the SENDERR.

IPFLAGS1
contains the input flag:

IPAPPCSN (X'02')—The APPC SEND function was issued.

IPCODE
contains the SENDERR code. IPCODE is only valid when IPCOMSRV is set. For a complete list of the error/SEVER codes, see section "APPC/VM Error/SEVER Codes" on page 183.

IPFLAGS2
contains the input flags:

IPWAIT (X'80')—You specified a synchronous return.
IPCOMSRV
(X'20')—The SENDERR is being issued for another user. Only an authorized virtual machine (OPTION COMSRV in the directory entry) may specify IPCOMSRV. When you do specify IPCOMSRV, CP does not generate a SENDERR code, but, instead, uses the one that you provide. It is your responsibility to ensure that the code is appropriate.

IPSENDOP
contains the SEND option code:

IPERROR (03)—Indicates the SENDERR.

Error Codes and Exceptions

Condition Codes

CC = 0	CC = 1	CC = 2	CC = 3
X	X	X	Not Possible

CC = 0
SENDERR started successfully, but has not yet completed. When the function completes, a function complete interrupt is sent to your virtual machine. The function complete interrupt has the same format as the SENDERR output parameter list (see CC = 2).

Note: When you specify WAIT = YES, CC = 0 is not possible.

CC = 1
An error occurred. The output parameter list is the same as the input shown in "SENDERR Parameter List" on page 150, except that the return code is stored in IPRCODE.

You may get the following return codes (listed here in decimal):

01 You specified a path id that is not yet established.

03 A function is pending on this path.

29 You are not authorized to act for another user.

30 You specified an APPC/VM function on a non-APPC path.

32 SENDERR is an invalid function from CONNECT state.

36 SENDERR is an invalid function from SEVER state.

38 There is an invalid value in IPSENDOP field.

CC = 2

SENDERR completed (also see "SENDERR Completion" on page 154). The output parameter list when CC = 2 is:

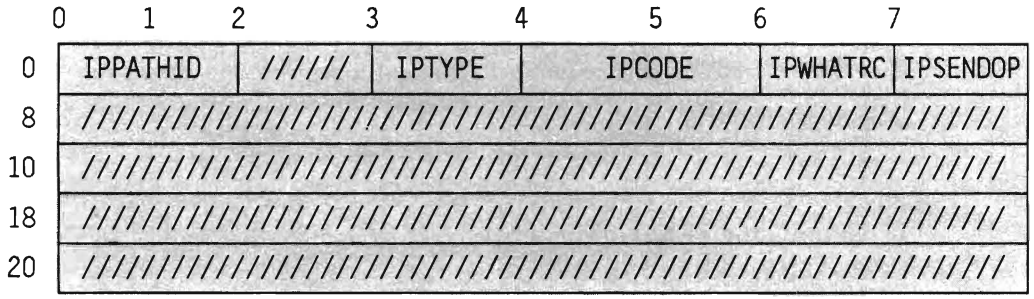


Figure 50. APPCVM SENDERR Output Parameter List (Function Complete Interrupt)

The parameters are:

IPPATHID
: contains the path id on which the function is complete.

IPTYPE
: contains the function complete interrupt code (IPTYPFCA, X'87').

IPCODE
: contains the error/SEVER code from the partner's SENDERR or SEVER. IPCODE is only valid when IPWHATRC = IPERROR or IPSABEND. For a complete list of the error/SEVER codes, see "APPC/VM Error/SEVER Codes" on page 183.

IPWHATRC
: contains the what-received code:

  IPCOMP (00)—indicates that a function completed with nothing received.
  IPERROR (03)—means that your partner issued SENDERR.
  IPSNORM (08)—means that your partner issued a SEVER TYPE = NORMAL.

IPSABEND (09)—means that your partner issued a SEVER TYPE = ABEND.

IPSENDOP
contains the SEND option code:

IPERROR (03)—means that the SENDERR is being completed.

SENDERR Program Exceptions

The program exceptions for SENDERR are:

Type	Description
Addressing	The parameter list address is outside of the virtual machine.
Operation	An external interrupt buffer has not been declared with the DCLBFR function, or the invoker is not in supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user.
Specification	The parameter list is not on a doubleword boundary.

State Changes

A state check occurs (IPRCODE = 32 or 36) if the virtual machine is not in SEND, RECEIVE or CONFIRM state on this path.

No state change occurs when CC = 1. State changes occur when:

- The function completes, that is, when control is returned to the virtual machine (CC = 2)
- The function complete interrupt is accepted by the virtual machine or you use TESTCMPL to discover that the function was completed.

The state change depends on the IPWHATRC value:

IPWHATRC Value	State	Cause
IPCOMP	SEND	The SENDERR has completed.
IPERROR	RECEIVE	The SENDERR was completed by the communication partner issuing a SENDERR from RECEIVE state.
IPSNORM	SEVER	The SENDERR was completed by the communication partner issuing a SEVER TYPE = NORMAL.

IPWHATRC Value	State	Cause
IPSABEND	SEVER	The SENDERR was completed by the communication partner issuing a SEVER TYPE = ABEND.

SENDERR Completion

After you issue a SENDERR, you cannot issue another SEND, RECEIVE, or SEVER TYPE = NORMAL on that path until the outstanding SENDERR is complete. SENDERR is complete when your communication partner or an intermediate communication server is notified of the SENDERR.

APPC/VM notifies your communication partner of the SENDERR when your partner's SENDDATA, SENDCNF, SENDERR, or RECEIVE completes. SENDERR causes your partner's outstanding functions to complete. If none of these functions are outstanding when your communication partner issues a function, that function completes immediately.

If your communication partner is in RECEIVE state and sends a SENDERR before it receives your SENDERR notice, your partner's SENDERR is invoked over yours. In this case, your partner would enter SEND state and you would be switched to RECEIVE state.

When SENDERR completes, CP resets your logical record counts to zero, as well as your communication partner's; that is, your next SENDDATA would be a new logical record.

SEVER Codes that You May Receive

The SEVER code you receive when your SENDERR completes depends on the state you are in when you issue SENDERR.

- If you are in SEND state, you may get any valid SEVER code.
- If you are in RECEIVE state, you may get any of the following SEVER codes:

Condition	IPCODE
RESOURCE_FAILURE_NO_RETRY	X'0610'
RESOURCE_FAILURE_RETRY	X'0620'

In addition, you get an indication of SEVER TYPE = NORMAL when your SENDERR completes, if your communication partner issued SEVER TYPE = NORMAL. You also get an indication of SEVER TYPE = NORMAL when your SENDERR completes, if your partner issued SEVER TYPE = ABEND with any of these SEVER codes:

- DEALLOCATE_ABEND_PROG (X'0210')
- DEALLOCATE_ABEND_SVC (X'0220')
- DEALLOCATE_ABEND_TIMER (X'0230')

You would receive notice of any other SEVER condition on a following verb (APPC function).

- If you are in CONFIRM state, you may get any of the following SEVER codes:

Condition	IPCODE
RESOURCE_FAILURE_NO_RETRY	X'0610'
RESOURCE_FAILURE_RETRY	X'0620'

You would receive notice of any other SEVER condition on a following verb.

What Happens to Your Communication Partner

Your communications partner's outstanding function may complete, or your partner may get a message pending interrupt.

If your partner has a RECEIVE, SENDDATA, SENDCNF, or SENDERR outstanding on its half of the path, your partner's function is completed.

Your partner gets a message pending interrupt, if your partner has these qualities:

- Has no function outstanding on its half of the path
- Is in RECEIVE state on its half of the path
- Is enabled for message pending interrupts.

See "Message Pending External Interrupt" on page 147.

If you do not specify "CODE=", the SENDERR code (IPCODE) that your communication partner gets depends on the state of the conversation and whether or not a logical record is being truncated. For communication servers, the SENDERR code that your communication partner gets depends on how you specified "CODE=". For a complete list of the error/SEVER codes, see "APPC/VM Error/SEVER Codes" on page 183.

APPCVM SENDREQ

Function Code: X'04'

This function signals your communication partner that you would like to send data. The communication partner can ignore your request.

The APPCVM SENDREQ syntax is:

label **APPCVM**	**SENDREQ,**	Required
	PRMLIST = *label*/(*reg*),	Required
	MF = **L**,	Optional
	PATHID = *label*/(*reg*)	Optional

PRMLIST = *label*/(*reg*)
lets you specify the address of the APPC/VM parameter list. The address must be a guest real address (real to the virtual machine), and the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

reg
is the register number that contains the address of the parameter list.

MF = **L**
expands the APPCVM macro to generate the instructions necessary to initialize the APPC/VM parameter list as specified, but not to invoke the APPC/VM function.

PATHID = *label*/(*reg*)
lets you specify the path id on which to send the request.

label
is the relocatable label of a halfword that contains the path id.

reg
is the register number that contains the path id in the low-order halfword.

SENDREQ Parameter List

The APPCVM SENDREQ parameter list has the following input format:

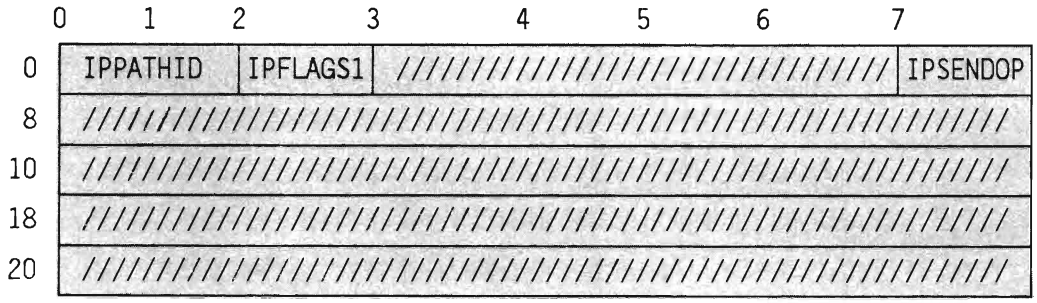


Figure 51. APPCVM SENDREQ Input Parameter List

The supplied parameters are:

IPPATHID
: contains the path id on which the request to send is to be sent.

IPFLAGS1
: contains the input flag:

  IPAPPCSN (X'02')—The APPC SEND function is issued.

IPSENDOP
: contains the SEND option code:

  IPREQSND (07)—Indicates the request to send.

Error Codes and Exceptions

Condition Codes

CC = 0	CC = 1	CC = 2	CC = 3
Not Possible	X	X	Not Possible

SENDREQ always completes immediately.

CC = 1
An error occurred. The output parameter list is the same as the input shown in "SENDREQ Parameter List," except that the return code is stored in IPRCODE.

You may get the following return codes (listed here in decimal):

01 You specified a path id that is not yet established.

30 You specified an APPC/VM function on a non-APPC path.

32 SENDREQ is an invalid function from CONNECT state.

36 SENDREQ is an invalid function from SEVER state.

38 There is an invalid value in the IPSENDOP field.

CC = 2
SENDREQ completed (also see "SENDREQ Completion" on page 159). The output parameter list when CC = 2 is:

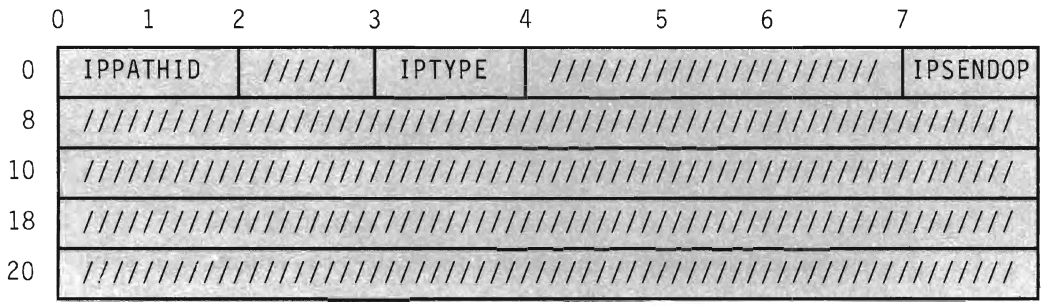


Figure 52. APPCVM SENDREQ Output Parameter List

The parameters are:

IPPATHID
contains the path id on which the function is complete.

IPTYPE
contains the function complete interrupt code (IPTYPFCA, X'87').

IPSENDOP
contains the SEND option code:

IPREQSND (07)—The SENDREQ is being completed.

SENDREQ Program Exceptions

The program exceptions for SENDREQ are:

Type	Description
Addressing	The parameter list address is outside of the virtual machine.
Operation	An external interrupt buffer has not been declared with the DCLBFR function, or the invoker is not in supervisor state.

Type	Description
Protection	The storage key of the parameter list address does not match the key of the user.
Specification	The parameter list is not on a doubleword boundary.

State Changes

A state check occurs (IPRCODE = 32 or 36) if the virtual machine is not in SEND, RECEIVE or CONFIRM state on this path.

No state change occurs.

SENDREQ Completion

Because the SENDREQ function completes immediately, you can issue another SEND or RECEIVE on the path when your virtual machine regains control.

You may issue more than one SENDREQ. Your communication partner does not get additional SENDREQs until it receives an indication of any preceding SENDREQs. Those SENDREQs sent before previous SENDREQs have been indicated to your partner are lost. CP does not notify you when your communication partner actually gets the SENDREQ interrupt. You also do not receive an error message if you issue another SENDREQ before your partner receives notification of previous SENDREQs.

You can issue SENDREQ even when another function is pending on the path. If the pending function is a SENDCNF TYPE = SEVER, then your partner may not be informed of your SENDREQ.

If you issue a SEVER before your communication partner learns of your SENDREQ, your communication partner may not be informed of your SENDREQ.

What Happens to Your Communication Partner

If your communication partner is enabled for SENDREQ interrupts, it gets the following SENDREQ interrupt:

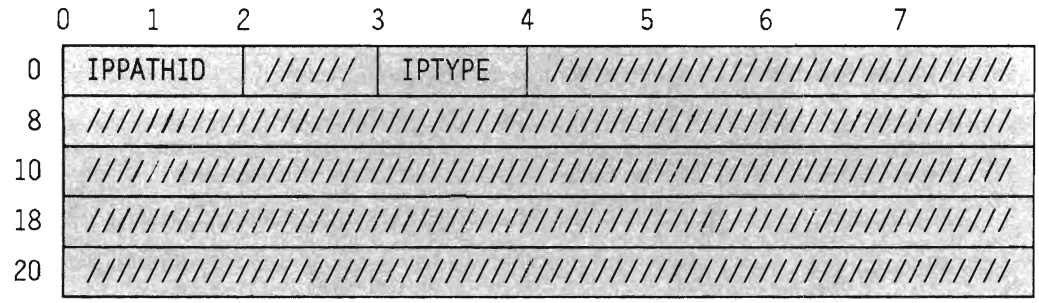


Figure 53. SENDREQ (Request-to-send) Interrupt

The parameters are:

IPPATHID
: is the path id on which you get the SENDREQ notice.

IPTYPE
: is the interrupt type for a SENDREQ notification (IPTYPSRA, X'88').

CP does not queue more than one SENDREQ interrupt on a single path for the communication partner at one time. So, the number of SENDREQ interrupts reflected to your communication partner may be less than the number of SENDREQs issued.

You cannot receive SENDREQ indicators with the RECEIVE function. They are only presented as an interrupt or with the DESCRIBE function.

IUCV SETCMASK

Function Code: X'11'

SETCMASK (Set Control Mask) enables or disables external interrupts for the following APPC/VM and IUCV control functions:

- Connection pending
- Connection complete
- Path severed
- Path quiesced (non-APPC only)
- Path resumed (non-APPC only).

You cannot use the SETCMASK function to disable interrupts on control paths.

To recognize this function, you must enable your virtual machine for external interrupts by setting the following bits to 1:

- Bit 7 in the virtual PSW
- Submask bit 30 in the control register 0.

You must also enable control interrupts with the SETMASK function. (Otherwise, APPC/VM ignores the SETCMASK settings.)

The *VM System Facilities for Programming* describes the IUCV version of this function unrelated to APPC/VM.

The IUCV SETCMASK syntax is:

label **IUCV**	**SETCMASK,**	Required
	PRMLIST = *label*/(*reg*),	Required
	MF = **L,**	Optional
	MASK = *label*/(*reg*)	Optional

PRMLIST = *label*/(*reg*)
lets you specify the address of the IUCV parameter list. The address must be a guest real address (real to the virtual machine), and the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

reg
is the register number that contains the address of the parameter list.

MF = L
expands the IUCV macro to generate the instructions necessary to initialize the IUCV parameter list as specified, but not to invoke the IUCV function.

MASK = *label*/(*reg*)
lets you specify the mask byte to determine which, if any, of the APPC/VM and IUCV external interrupts a virtual machine is to be enabled for.

label
is the relocatable label of a byte containing the mask.

reg
is the register number that contains the mask in its low-order byte.

SETCMASK Parameter List

The IUCV SETCMASK parameter list has the following input format:

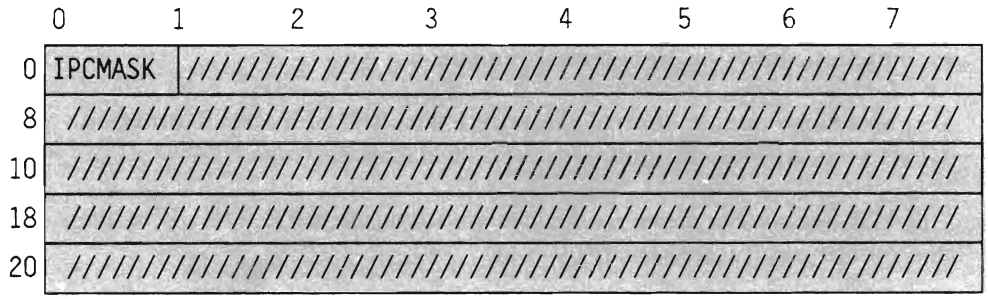


Figure 54. IUCV SETCMASK Input Parameter List

The supplied parameters are:

IPCMASK
specifies the mask byte to determine which of the IUCV control interrupts a virtual machine is to be enabled and disabled for. When a bit is off, the virtual machine is disabled for that interrupt.

For example, if IPCMASK contains X'C0', this means that the virtual machine is enabled for connection pending and connection complete interrupts, but disabled for all other interrupts.

IPCLPC (X'80')—You enabled for Type X'01' (non-APPC) and Type X'81' (APPC)— Pending connection.
IPCLCC (X'40')—You enabled for Type X'02' (non-APPC) and Type X'82' (APPC)— Connection complete.
IPCLPS (X'20')—You enabled for Type X'03' (non-APPC) and Type X'83' (APPC)—Path severed.

IPCLPQ (X'10')—You enabled for Type X'04' (non-APPC only)—Path quiesced.
IPCLPR (X'08')—You enabled for Type X'05' (non-APPC only)—Path resumed.
(X'04') — Reserved (Should be set to zero).
(X'02') — Reserved (Should be set to zero).
(X'01') — Reserved (Should be set to zero).

Error Codes and Exceptions

Condition Codes

CC = 0	CC = 1	CC = 2	CC = 3
X	Not Possible	Not Possible	Not Possible

CC = 0
Normal completion.

SETCMASK Program Exceptions

The program exceptions for SETCMASK are:

Type	Description
Addressing	The parameter list address is outside of the virtual machine.
Operation	A normal interrupt buffer has not been declared with the DCLBFR function, or the invoker is not in supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user.
Specification	The parameter list is not on a doubleword boundary.

State Changes

There are no states associated with the SETCMASK function.

SETCMASK Completion

The SETCMASK function completes immediately.

What Happens to Your Communication Partner

Not applicable.

IUCV SETMASK

Function Code: X'10'

SETMASK (Set Mask) enables or disables external interrupts for the following APPC/VM and IUCV functions:

- Message pending interrupts
- SENDREQ (request-to-send) interrupts
- Function complete interrupts
- APPC/VM and IUCV control interrupts.

You cannot use the SETMASK function to disable interrupts on control paths.

To recognize this function, you must enable your virtual machine for external interrupts by setting the following bits to 1:

- Bit 7 in the virtual PSW
- Submask bit 30 in control register 0.

The IUCV SETMASK function specifies a byte of selective masks. This lets you selectively mask APPC/VM and IUCV external interrupts and, as a group, lets you mask APPC/VM and IUCV control external interrupts.

The *VM System Facilities for Programming* describes the IUCV version of this function unrelated to APPC/VM.

The IUCV SETMASK syntax is:

label **IUCV**	**SETMASK,**	Required
	PRMLIST = *label*/(*reg*),	Required
	MF = **L,**	Optional
	MASK = *label*/(*reg*)	Optional

PRMLIST = *label*/(*reg*)
lets you specify the address of the IUCV parameter list. The address must be a guest real address (real to the virtual machine), and the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

reg
is the register number that contains the address of the parameter list.

MF = L
expands the IUCV macro to generate the instructions necessary to initialize the IUCV parameter list as specified, but not to invoke the IUCV function.

MASK = *label*/(*reg*)
lets you specify the mask byte to determine which, if any, of the APPC/VM and IUCV external interrupts a virtual machine is to be enabled for.

label
is the relocatable label of a byte containing the mask.

reg
is the register number that contains the mask in its low-order byte.

SETMASK Parameter List

The APPCVM SETMASK parameter list has the following input format:

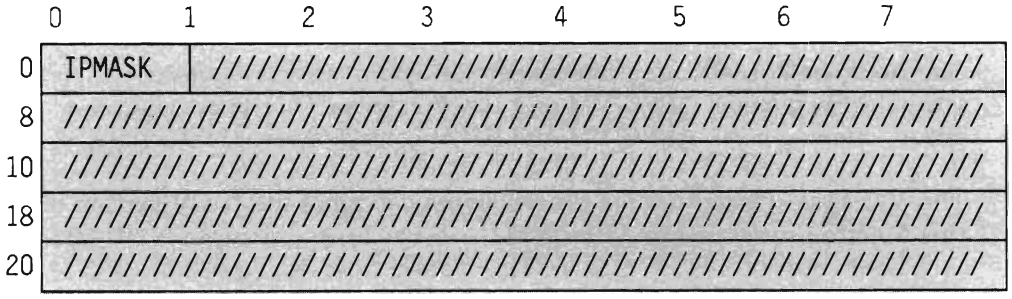


Figure 55. IUCV SETMASK Input Parameter List

IPMASK
specifies the mask byte to determine which of the APPC/VM and IUCV interrupts a virtual machine is to be enabled for.

For example, if IPMASK contains X'C0', this means that the virtual machine is enabled for APPC message interrupts and SENDREQ interrupts, but disabled for all other interrupts.

IPSNDN (X'80')—You enabled for Type X'09', nonpriority message interrupts (non-APPC), and Type X'89', message pending interrupts (APPC).
IPSNDP (X'40')—You enabled for Type X'08', priority message interrupts (non-APPC), and Type X'88', SENDREQ interrupts (APPC).
IPRPYN (X'20')—You enabled for Type X'07', nonpriority reply interrupts (non-APPC), and Type X'87', function complete interrupts (APPC).

IPRPYP (X'10')—You enabled for Type X'06', priority reply interrupts (non-APPC only).
IPCTRL (X'08')—You enabled for control interrupts (non-APPC and APPC).
(X'04') — Reserved (Should be set to zero).
(X'02') — Reserved (Should be set to zero).
(X'01') — Reserved (Should be set to zero).

Error Codes and Exceptions

Condition Codes

CC = 0	CC = 1	CC = 2	CC = 3
X	Not Possible	Not Possible	Not Possible

CC = 0
Normal completion.

SETMASK Program Exceptions

The program exceptions for SETMASK are:

Type	Description
Addressing	The parameter list address is outside of the virtual machine.
Operation	A normal interrupt buffer has not been declared with the the DCLBFR function, or the invoker is not in supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user.
Specification	The parameter list is not on a doubleword boundary.

State Changes

No states are associated with the SETMASK function.

SETMASK Completion

The SETMASK function completes immediately.

What Happens to Your Communication Partner

Not applicable.

APPCVM SEVER

Function Code: X'0F'

This function breaks a communications path with another virtual machine or your own virtual machine. After severing the connection with the other virtual machine, you cannot send or receive any other messages on that connection. Remember that your communication partner cannot receive any of the data that has not yet been copied out of your storage.

The APPCVM SEVER syntax is:

label **APPCVM**	**SEVER,**	Required
	PRMLIST = *label*/(*reg*),	Required
	TYPE = NORMAL/ABEND,	Required
	CODE = *label*/(*reg*),	Optional
	MF = L,	Optional
	PATHID = *label*/(*reg*)	Optional

PRMLIST = *label*/(*reg*)
lets you specify the address of the APPC/VM parameter list. The address must be a guest real address (real to the virtual machine), and the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

reg
is the register number that contains the address of the parameter list.

TYPE = NORMAL/ABEND
indicates the type of SEVER that is to be performed.

- TYPE = NORMAL severs the path normally. You can only sever the path normally if you are in SEND state and not in the middle of sending a logical record, or if you are in SEVER state.
- TYPE = ABEND severs the path abnormally. APPC/VM invokes SEVER TYPE = ABEND from SEND, RECEIVE, or CONFIRM state, even if there is a function that still has not completed on the specified path.

CODE = *label*/(*reg*)
specifies a 2-byte SEVER code. CODE is only valid when you specify TYPE = ABEND. IBM defines all the codes; applications may not define error or SEVER codes for their own use.

label
is the relocatable label in storage of the 2-byte SEVER code.

reg
is the register number that contains the address of the SEVER code.

When CP issues a SEVER, or your communication partner issues an IUCV SEVER or RTRVBFR, CP determines the SEVER code to reflect. This code is X'0610'. See "APPC/VM Error/SEVER Codes" on page 183 for a complete list of the SEVER codes.

MF=L
expands the APPCVM macro to generate the instructions necessary to initialize the APPC/VM parameter list as specified, but not to invoke the APPC/VM function.

PATHID=*label*/(*reg*)
lets you specify the path id that is to be severed.

label
is the relocatable label of a halfword that contains the path id.

reg
is the register number that contains the path id in the low-order halfword.

SEVER Codes That You Can Issue

When you can specify each SEVER code depends on the state of your path. However, you can issue IUCV SEVER on any APPC path at any time.

- If you are in CONNECT state, you can only issue IUCV SEVER.
- After you ACCEPT the connection, and before you issue any other function on the path, you can issue any of the following APPCVM SEVERs in addition to IUCV SEVER:

APPC Error Condition	APPC/VM Code
CONVERSATION_TYPE_MISMATCH	X'0120'
SYNC_LEVEL_NOT_SUPPORTED_BY_PGM	X'0130'
TRANS_PGM_NOT_AVAIL_NO_RETRY	X'0140'
TRANS_PGM_NOT_AVAIL_RETRY	X'0141'
TPN_NOT_RECOGNIZED	X'0142'

APPC Error Condition	APPC/VM Code
PIP_NOT_SPECIFIED_CORRECTLY	X'0150'
DEALLOCATE_ABEND_PROG	X'0210'

- After the path is established (that is, the CONNECT/ACCEPT sequence is complete), you can issue the following APPCVM SEVER in addition to IUCV SEVER:

APPC Error Condition	APPC/VM Code
DEALLOCATE_ABEND_PROG	X'0210'

The SEVER type and code presented to your partner may not always be the SEVER type and code that you specified. For example, if your partner issues a SENDERR from RECEIVE state, a SEVER code of DEALLOCATE_ABEND_PROG is presented to your partner as DEALLOCATE_NORMAL in the completion data of your partner's SENDERR. See "APPCVM SENDERR" on page 149 for more information.

See "APPC/VM Error/SEVER Codes" on page 183 for a complete list of the SEVER codes.

SEVER Parameter List

The APPCVM SEVER parameter list has the following input format:

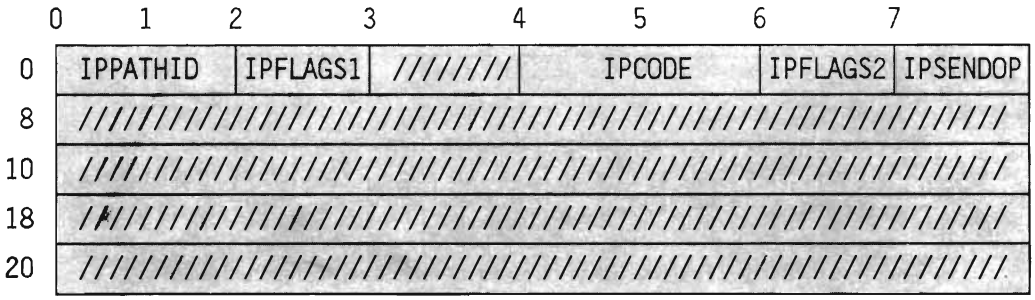


Figure 56. APPCVM SEVER Input Parameter List

The supplied parameters are:

IPPATHID
: contains the path id being severed.

IPFLAGS1
: contains the input flag:

  IPAPPC (X'08')—The APPC SEVER function was issued.

IPCODE
: contains the SEVER code. IPCODE is only valid when IPSENDOP = IPSABEND. See "APPC/VM Error/SEVER Codes" on page 183 for a description of the SEVER codes.

IPFLAGS2
: contains the input flag:

  IPCOMSRV (X'20')—means that the SEVER is on behalf of another user. Only an authorized user (OPTION COMSRV in directory) can specify IPCOMSRV. When you do specify IPCOMSRV, CP does not verify the SEVER code. It is your responsibility to ensure that the code is valid.

IPSENDOP
: contains one of the following SEND option codes:

  IPSNORM (08)—You requested that the path be severed normally.
  IPSABEND (09)—You requested that the path be severed abnormally.

Error Codes and Exceptions

Condition Codes

CC = 0	CC = 1	CC = 2	CC = 3
Not Possible	X	X	Not Possible

CC = 1
An error occurred. The parameter list format is the same as the input shown in "SEVER Parameter List" on page 171, except that the return code is stored in IPRCODE.

You may get the following return codes (listed here in decimal):

01 You specified a path id that is not yet established.

03 A function is pending on this path.

29 You are not authorized to act for another user.

30 You specified an APPC/VM function on a non-APPC path.

32 APPC/VM SEVER is an invalid function from CONNECT state.

34 SEVER TYPE = NORMAL is an invalid function from RECEIVE state.

35 SEVER TYPE = NORMAL is an invalid function from CONFIRM state.

36 SEVER TYPE = ABEND is an invalid function from SEVER state.

38 There is an invalid value in the IPSENDOP field.

44 Before invoking SEVER TYPE = NORMAL, you started, but did not finish, sending a logical record.

46 You specified an invalid SEVER code.

CC = 2
SEVER completed (see "SEVER Completion" on page 174). The output parameter list is the same as the input shown in "SEVER Parameter List" on page 171.

SEVER Program Exceptions

The program exceptions for SEVER are:

Type	Description
Addressing	The parameter list address is outside of the virtual machine.
Operation	An external interrupt buffer has not been declared with the DCLBFR function, or the invoker is not in supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user.
Specification	The parameter list is not on a doubleword boundary.

State Changes

State checks depend on the TYPE you specify:

- When TYPE = NORMAL, a state check occurs (IPRCODE = 32, 34, or 35) if the virtual machine is not in SEND or SEVER state on this path.
- When TYPE = NORMAL, a state check occurs (IPRCODE = 44) if the virtual machine is in SEND state on this path and started, but did not finish, sending a logical record.
- When TYPE = ABEND, a state check occurs (IPRCODE = 32 or 36) if the virtual machine is in CONNECT or SEVER state on this path.

When your virtual machine regains control after successfully completing the SEVER (CC = 2), you enter the RESET state.

No state change occurs when CC = 1.

SEVER Completion

The SEVER function completes immediately. After SEVER completes, you cannot issue any other functions on that path.

You cannot issue SEVER TYPE = NORMAL if there is another function outstanding on the path. You can, however, issue SEVER TYPE = ABEND even if there is an outstanding function on a path. CP may not present the outstanding function to your communication partner. For example, if you issue the following sequence of commands, your communication partner is notified of the SEVER, but not the SENDERR:

1. **SENDERR**
2. **SEVER TYPE = ABEND** (before your partner receives the SENDERR)

Also, for example, in the following sequence of commands, your communication partner cannot receive more than the amount of data specified in the RECEIVE.

1. You issue a **SENDDATA BUFLEN = 200.**
2. Your communication partner issues a **RECEIVE BUFLEN = 100.**
3. You issue a **SEVER TYPE = ABEND.**

CP notifies your communication partner of the SEVER with a SEVER interrupt. In addition, CP notifies your partner the next time your partner issues a function on which CP can report the SEVER.

What Happens to Your Communication Partner

Your communication partner may be affected different ways, depending on the sequence of functions that you issue. Any of the following conditions can occur:

- You issue CONNECT, and then issue IUCV SEVER before your partner gets the connection pending interrupt. In this case, your partner does not get a connection pending interrupt or a SEVER interrupt.
- You issue CONNECT, and then issue IUCV SEVER after your partner gets the connection pending interrupt, but before your partner issues ACCEPT. In this case, if your partner is enabled for SEVER interrupts, your partner gets a SEVER interrupt.
- You issue an IUCV SEVER after receiving a connection pending interrupt, instead of issuing an ACCEPT. If your partner issued:

- CONNECT with WAIT = NO and is enabled for SEVER interrupts, your partner gets a SEVER interrupt.

- CONNECT with WAIT = YES, your partner's CONNECT completes with a SEVER indication.

- You issue a SEVER TYPE = NORMAL or SEVER TYPE = ABEND (whichever is appropriate) anytime after you and your partner have established a path (that is, after the CONNECT/ACCEPT sequence is complete). In this case, if your partner is enabled for SEVER interrupts, your partner gets a SEVER interrupt.

  In addition to the SEVER interrupt, if your partner has a RECEIVE, SENDDATA, SENDCNF, or SENDERR outstanding on its half of the path, the function completes.

  If your partner issues a RECEIVE, SENDDATA, SENDCNF, or SENDERR after you have issued the SEVER, your partner's function completes immediately. Your partner gets an indication of IPWHATRC = IPSNORM or IPWHATRC = IPSABEND.

When your partner gets the SEVER interrupt, its state does not change to SEVER state. Your partner only enters SEVER state after a function completes (IPTYPE = X'87') with IPWHATRC = IPSNORM or IPWHATRC = IPSABEND.

SEVER External Interrupt

The SEVER external interrupt on the APPC/VM path has the following format:

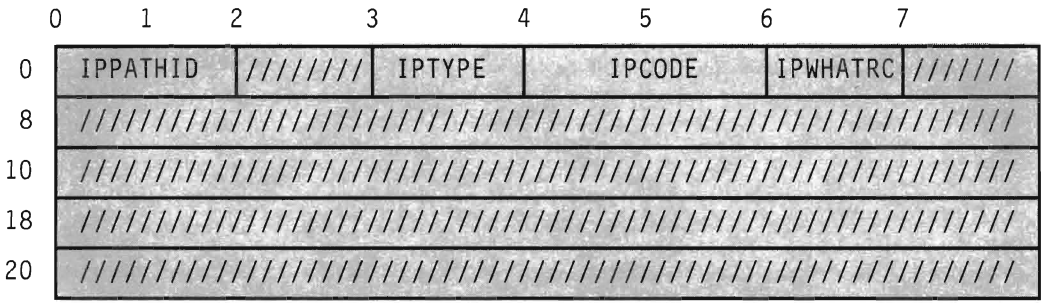


Figure 57. SEVER External Interrupt

The parameters are:

IPPATHID
: contains the path id being severed.

IPTYPE
: contains the interrupt type for SEVER (IPTYPSVA, X'83').

IPCODE
: contains the SEVER code from the partner's SEVER. See "APPC/VM Error/SEVER Codes" on page 183 for a description of the error/SEVER codes.

IPWHATRC
: contains the what-received code:

  IPSNORM (08)—Your partner issued a SEVER TYPE = NORMAL.
  IPSABEND (09)—Your partner issued SEVER TYPE = ABEND.

Non-APPC SEVERs, with ALL specified as YES or NO, function on APPC paths. CP ignores the user data field and reflects a SEVER TYPE = ABEND to your communication partner. The SEVER code is X'0610'.

IUCV TESTCMPL

Function Code: X'07'

TESTCMPL determines if any messages or functions have been completed. You can identify a specific path when you invoke this function. If you do not specify a path, the next function on the queue of completed functions (if there is a function) is displayed.

TESTCMPL does not present functions completed on control paths.

The *VM System Facilities for Programming* describes the IUCV version of this function unrelated to APPC/VM. Parameters other than those listed here are available, but have no meaning on APPC/VM paths and are ignored.

The IUCV TESTCMPL syntax is:

label	**IUCV**	**TESTCMPL,**	Required
		PRMLIST = *label*/**(***reg***),**	Required
		MF = **L,**	Optional
		PATHID = *label*/**(***reg***)**	Optional

PRMLIST = *label*/**(***reg***)**
lets you specify the address of the IUCV parameter list. The address must be a guest real address (real to the virtual machine), and the parameter list must be on a doubleword boundary.

label
is the relocatable label of the parameter list.

reg
is the register number that contains the address of the parameter list.

MF = **L**
expands the IUCV macro to generate the instructions necessary to initialize the IUCV parameter list as specified, but not to invoke the IUCV function.

PATHID = *label*/**(***reg***)**
lets you identify the path id to do the test completion.

label
is the relocatable label of a halfword that contains the path id.

reg
is the register number that contains the path id in the low-order halfword.

TESTCMPL Parameter List

The IUCV TESTCMPL parameter list has the following input format:

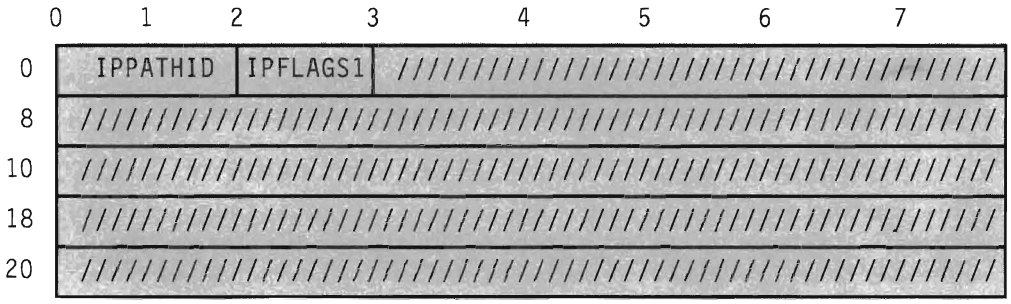


Figure 58. IUCV TESTCMPL Input Parameter List

The supplied parameters are:

IPPATHID
contains the path id on which you want to complete the function. This parameter is only valid when the IPFGPID flag is set.

IPFLAGS1
contains the following input flag:

IPFGPID (X'02')—means that you specified path id.

Error Codes and Exceptions

Condition Codes

CC = 0	CC = 1	CC = 2	CC = 3
X	X	X	X

CC = 0
Normal completion.

The output parameter list is:

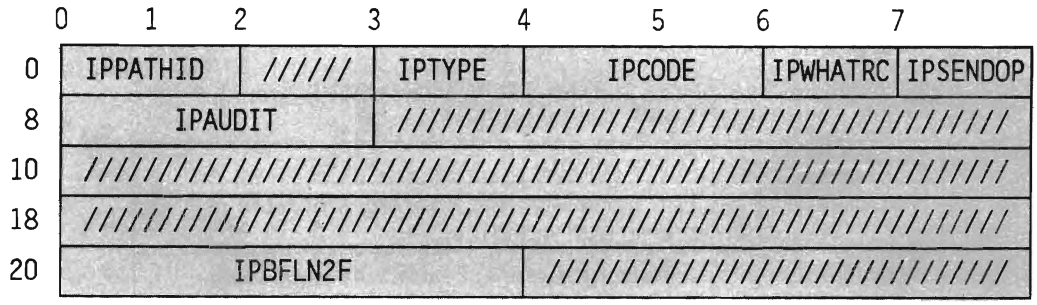


Figure 59. IUCV TESTCMPL Output Parameter List

The parameters are:

IPSENDOP
contains one of the following SEND option codes:

IPDATA (01)—SENDDATA RECEIVE = NO is being completed.
IPSNDRCV (02)—SENDDATA RECEIVE = YES is being completed.
IPERROR (03)—SENDERR is being completed.
IPCNFRM (04)—SENDCNF TYPE = NORMAL is being completed.
IPCNFSEV (05)—SENDCNF TYPE = SEVER is being completed.
IPRECV (10)—RECEIVE is being completed.

The contents of the other fields in this parameter list depend on what function has completed. The function that has completed is indicated in IPSENDOP. See the CC = 2 description under the specific function being completed for more information.

CC = 1
An error occurred. The parameter list format is the same as the input shown in "TESTCMPL Parameter List" on page 178, except the return code is stored in IPRCODE.

You may get this return code (listed here in decimal):

01 You specified a path id that is not yet established.

CC = 2
IUCV did not find any APPC/VM function completes or IUCV message completes.

CC = 3
Nonzero IPAUDIT stored.

TESTCMPL Program Exceptions

The program exceptions for the IUCV TESTCMPL are:

Type	Description
Addressing	The parameter list address is outside of the virtual machine.
Operation	A normal interrupt buffer has not been declared with the DCLBFR function, or the invoker is not in supervisor state.
Protection	The storage key of the parameter list address does not match the key of the user.
Specification	The parameter list is not on a doubleword boundary.

State Changes

The state of the path on which the function is being completed is set according to the function. See each function description for details.

TESTCMPL Completion

The TESTCMPL function completes immediately.

What Happens to Your Communication Partner

Not applicable.

IUCV TESTMSG

Function Code: X'01'

TESTMSG (Test Message) lets you avoid using external interrupt handling. When you invoke the TESTMSG function, your virtual machine enters a WAIT state if none of the following are pending:

- SENDREQ interrupts (APPC)
- Function complete interrupts (APPC)
- Message pending interrupts (non-APPC and APPC)
- Message complete interrupts (non-APPC).

If any of these interrupts become pending while your virtual machine is in the WAIT state, the virtual machine re-executes the TESTMSG function. TESTMSG then returns a condition code. TESTMSG ignores APPC/VM message pending interrupts unless the path corresponding to the message pending is in RECEIVE state.

TESTMSG does not receive or describe the interrupt. You must use RECEIVE, DESCRIBE or TESTCMPL, or enable for interrupts to clear the interrupt.

TESTMSG ignores interrupts pending on control paths.

The *VM System Facilities for Programming* describes the IUCV version of this function unrelated to APPC/VM.

The IUCV TESTMSG syntax is:

label **IUCV TESTMSG**		Required

TESTMSG Parameter List

The TESTMSG function does not use a parameter list.

Error Codes and Exceptions

Condition Codes

CC = 0	CC = 1	CC = 2	CC = 3
Not Possible	X	X	X

CC = 1
A message or SENDREQ indication is pending.

CC = 2
A message completion or function completion is pending.

CC = 3
One or more conditions causing a condition code 1 and one or more conditions causing a condition code 2 are pending.

TESTMSG Program Exceptions

The program exceptions for TESTMSG are:

Type	Description
Operation	A normal interrupt buffer has not been declared with the DCLBFR function, or the invoker is not in supervisor state.

State Changes

No states are associated with the TESTMSG function.

TESTMSG Completion

The TESTMSG function completes when control is returned to your virtual machine.

What Happens to Your Communication Partner

Not applicable.

APPC/VM Error/SEVER Codes

APPC/VM defines 2-byte error and SEVER codes. Applications may not define any APPC/VM error or SEVER codes for their own use.

When CP does a SEVER, or when your communication partner issues IUCV SEVER or RTRVBFR, CP determines the SEVER code to reflect. This code is X'0610'.

The following is a list of the defined APPC/VM error/SEVER codes. See the *SNA Transaction Programmer's Reference Manual For LU Type 6.2* for a description of the meaning of each error condition. This manual also describes when to use each error/SEVER code.

"SEVER Codes That You Can Issue" on page 170 lists the SEVER codes that you can specify in your applications. Communication servers can issue any valid SEVER code at any time. For example, after the local path is established, the communication server may have to report an allocation failure to the local program.

Note: Code your applications to be adaptable to change if any additional SEVER codes are defined.

The following three tables list all of the currently defined SEVER codes:

APPC Error Condition	APPC/VM Code
ALLOCATION_FAILURE_NO_RETRY	X'0110'
ALLOCATION_FAILURE_RETRY	X'0111'
CONVERSATION_TYPE_MISMATCH	X'0120'
SYNC_LEVEL_NOT_SUPPORTED_BY_PGM	X'0130'
SYNC_LEVEL_NOT_SUPPORTED_BY_LU	X'0131'
TRANS_PGM_NOT_AVAIL_NO_RETRY	X'0140'
TRANS_PGM_NOT_AVAIL_RETRY	X'0141'
TPN_NOT_RECOGNIZED	X'0142'
PIP_NOT_SPECIFIED_CORRECTLY	X'0150'
DEALLOCATE_ABEND_PROG	X'0210'
DEALLOCATE_ABEND_SVC	X'0220'
DEALLOCATE_ABEND_TIMER	X'0230'
INVALID_LU_NAME	X'0301'
INVALID_MODE_NAME	X'0302'

Figure 60 (Part 1 of 2). APPC/VM Error Codes

APPC Error Condition	APPC/VM Code
RESOURCE_FAILURE_NO_RETRY[4] ALLOCATION_FAILURE_NO_RETRY	X'0610'
RESOURCE_FAILURE_RETRY	X'0620'

Figure 60 (Part 2 of 2). APPC/VM Error Codes

The following table lists the codes that are possible when your communication partner issues SENDERR from within the VM collection:

APPC Error Condition	APPC/VM Code
PROG_ERROR_NO_TRUNC	X'0410'
PROG_ERROR_TRUNC	X'0420'
PROG_ERROR_PURGING	X'0430'

Figure 61. APPC/VM SENDERR Codes from Within the TSAF Collection

The following table lists the SENDERR codes that are currently defined:

APPC Error Condition	APPC/VM Code
PROG_ERROR_NO_TRUNC	X'0410'
PROG_ERROR_TRUNC	X'0420'
PROG_ERROR_PURGING	X'0430'
SVC_ERROR_NO_TRUNC	X'0510'
SVC_ERROR_TRUNC	X'0520'
SVC_ERROR_PURGING	X'0530'

Figure 62. APPC/VM-Defined SENDERR Codes

[4] X'0610' on the CONNECT functions means an ALLOCATION_FAILURE_NO_RETRY. On all other functions X'0610' means a RESOURCE_FAILURE_NO_RETRY.

Chapter 8. APPC Verbs Mapped with APPC/VM Functions

APPC/VM is a means of communication between two virtual machines. This APPC/VM interface provides a limited set of the SNA LU 6.2 base communication functions. See the *Systems Network Architecture Transaction Programmer's Reference Manual for LU Type 6.2* for more specific information on the LU 6.2 protocol. This chapter maps APPC/VM functions with the APPC functions provided with the SNA LU 6.2 protocol.

Appendix B, "APPC - APPC/VM Mapping Summary" on page 233 contains a summary of the mapping between APPC and APPC/VM.

Conversations with APPC

Programs that connect to a resource and the resource manager must follow the rules of an APPC conversation. APPC/VM supports and enforces these rules, as described in the following sections.

Starting a Conversation

If your virtual machine manages a resource, and another virtual machine is trying to establish a path to the resource (you receive a connection pending interrupt), check to be sure that the connection pending interrupt is for an APPC connection. Do not assume that the virtual machine trying to connect is on the local TSAF collection, or that the virtual machine is a VM program.

The resource manager virtual machine is responsible for invoking the transaction program and verifying the contents of the FMH5. CP recognizes nothing smaller than the resource manager virtual machine. CMS and GCS recognize nothing smaller than a program. In general, in the CMS and GCS environments, each inbound connection does not cause the resource manager to create another instance of the transaction program. Instead, the program is notified that another path is being established.

It is the program's responsiblity to receive the Attach FMH5 (optionally) and save its relevant contents. In addition, the program must issue an IUCV ACCEPT before communicating on the APPC/VM path. ACCEPT is not part of the APPC architecture. If there is something wrong in the Attach FMH5 data (for example, the program does not support the synchronization level specified), then it is the program's responsibility to SEVER the connection with the appropriate SEVER code.

After the CONNECT/ACCEPT sequence has been successfully completed on both sides, the two programs can exchange data using the half-duplex protocol of an APPC conversation. APPC/VM fully supports the basic functions of this communication.

APPC Functions Not Supported

APPC/VM supports the base set of APPC basic conversation functions. APPC/VM does not provide support for mapped conversations or operator control verbs. Although not explicitly provided, you can use the following APPC option sets by providing a general purpose application running on top of APPC/VM:

- The GET_ATTRIBUTES verb
- The GET_TYPE verb
- The FILL(LL) option of RECEIVE_AND_WAIT.

APPC Return Codes

CP reports errors that it finds in the IPRCODE or IPAUDIT field. The application or communication servers report errors that they find in the IPCODE field on SEVER or SENDERR functions. This same condition may be reported in IPRCODE, IPAUDIT, or IPCODE depending on the following:

- Whether or not the path goes through an intermediate communication server (for example, TSAF), and
- Where CP detected the error along the path.

CP does not report the error in both IPRCODE/IPAUDIT and IPCODE at the same time.

The tables in the following sections have entries for the return code (or condition), and the corresponding IPRCODE and/or IPCODE.

When an application receives an error via an IPRCODE, the application should sever. The application can sever the path using the APPCVM SEVER function with a SEVER code that indicates DEALLOCATE_ABEND_PROG.

There are some APPC/VM return codes that do not correspond to defined APPC return codes. The IUCV return codes that do not correspond to any defined APPC return code are not discussed in this chapter.

APPC/VM Interrupts

APPC/VM uses external interrupts to signal certain events. The interrupts are:

- Those that allow applications to process other paths while waiting for input or a function to complete on other paths:
  - Message pending interrupts
  - Connection pending interrupts
  - Function complete interrupts
  - Connection complete interrupts.

  This asynchronous capability of APPC/VM is not based on the APPC architecture functions (POST_ON_RECEIPT, WAIT, and TEST), but, instead, is a VM-unique asynchronous implementation.

- Those that asynchronously indicate your partner has issued a SENDREQ or a SEVER:
  - SENDREQ interrupts
  - SEVER interrupts.

Programs must take care when taking advantage of the asynchronous nature of the SENDREQ and SEVER interrupts. For example, deadlock may result if an application waits for a SENDREQ interrupt before continuing processing. Though SENDREQ and SEVER interrupts are asynchronous, they may not always travel from the invoker to the target without the target issuing the appropriate verbs. The APPC architecture specifies the verbs on which the SENDREQ may be reported to the target.

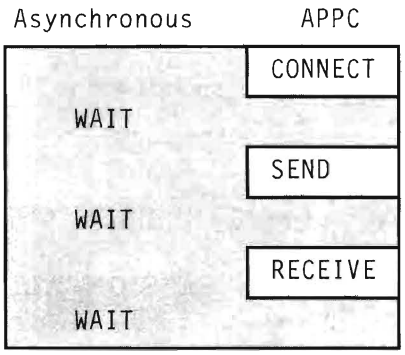


Figure 63. An APPC/VM Program

In Figure 63, the white area indicates the APPC/VM verbs that correspond to APPC. The shaded area indicates the asynchronous functions of APPC/VM that do not correspond to the APPC architecture. These APPC/VM functions do not correspond to any function of APPC. A program that wants to avoid non-APPC functions should:

- Only use the APPC/VM functions and IUCV DCLBFR and RTRVBFR
- Be enabled only for connection pending interrupts.

APPC ALLOCATE

The APPC ALLOCATE verb maps to the APPC/VM function, CONNECT.

Do not make assumptions about the target of the CONNECT when the CONNECT completes. Your CONNECT may complete before the target program is even invoked.

Parameters

The following list maps APPC parameters to APPC/VM parameters. Each entry lists the APPC parameter first, followed by the APPC/VM parameter in italics.

LU_NAME - *first 8 bytes of the connection extension*
Each TSAF collection is an SNA LU. The first eight bytes of the connection extension must be zero in APPC/VM to specify LU_NAME (OWN). The parameter value LU_NAME (OWN) is an optional parameter. It is not part of the base set.

MODE_NAME - *mode name in the connection extension*
The APPC MODE_NAME(variable) corresponds to the mode name field in the connect parameter list extension. If you want APPC/VM to use the default mode name for the LU, specify a mode name of zeros in APPC/VM.

Mode names are 8 bytes in APPC/VM.

TPN - *RESID*
APPC TPN(variable) corresponds to the IPRESID field in the connect parameter list.

TYPE - *no parameter*
APPC/VM only supports TYPE(BASIC_CONVERSATION).

RETURN_CONTROL(WHEN_SESSION_ALLOCATED) - *no parameter*
APPC/VM only supports RETURN_CONTROL(WHEN_SESSION_ALLOCATED), so you do not need a parameter in APPC/VM. APPC/VM does not support the option sets:

- RETURN_CONTROL(DELAYED_ALLOCATION_PERMITTED)
- RETURN_CONTROL(IMMEDIATE).

Note: The program may use WAIT = NO and get control back before the ALLOCATE sequence completes. However, the program may not use the path because the CONNECT has not yet completed.

Therefore, this maps to
RETURN_CONTROL(WHEN_SESSION_ALLOCATED) on the path.

SYNC_LEVEL - *SYNCLVL*
APPC/VM supports the APPC options:

- SYNC_LEVEL(NONE) as SYNCLVL=NONE
- SYNC_LEVEL(CONFIRM) as SYNCLVL=CONFIRM.

APPC/VM does not support the option set SYNCLVL(SYNCPT).

SECURITY - *no parameter*
APPC/VM does not support the security option set.

PIP - *no parameter*
APPC/VM does not support the PIP option set.

RESOURCE - *IPPATHID*
The resource id returned in APPC/VM is a path id. The path id is a halfword number.

RETURN_CODE - *IPRCODE* **and** *IPCODE*
The APPC RETURN_CODE variable corresponds to:

- The APPC/VM IPRCODE of CONNECT
- The APPC/VM IPCODE in the SEVER indication.

The connecting program must look at the IPRCODE when it receives a CC=1 on CONNECT. Also, if your partner rejects the connection with SEVER, then the connecting program must look at IPCODE to determine the allocation error.

RETURN_CODE	IPRCODE	IPCODE
OK	0	Not applicable
ALLOCATION_ERROR ALLOCATION_FAILURE_RETRY ALLOCATION_FAILURE_NO_RETRY	 11,12,13,14 15	 X'0111' X'0110' X'0610'
PARAMETER_ERROR Invalid LU name Invalid mode name	 40 41	 Not applicable

State Changes

For either of the following:

- APPC ALLOCATE
- APPC/VM CONNECT,

you, the invoker, are in SEND state when the function completes successfully.

ABEND Conditions

The ABEND conditions are:

Parameter Check Condition	IPRCODE
MODE_NAME(SNASVCMG) is not supported.	41
The program is not allowed to specify MODENAME(SNASVCMG).	41

APPC CONFIRM

The APPC CONFIRM verb maps to the APPC/VM function, SENDCNF TYPE = NORMAL.

Parameters

The following list maps APPC parameters to APPC/VM parameters. Each entry lists the APPC parameter first, followed by the APPC/VM parameter in italics.

RESOURCE - *PATHID*
The resource id returned in APPC/VM is a path id. The path id is a halfword number.

REQUEST_TO_SEND_RECEIVED
APPC/VM indicates that the partner issued REQUEST_TO_SEND by reflecting a SENDREQ interrupt.

RETURN_CODE - *IPCODE*
The APPC RETURN_CODE variable corresponds to:

- The APPC/VM IPCODE of SENDERR
- The APPC/VM IPCODE of SEVER.

If the SENDCNF completes with a SENDERR or SEVER, then the virtual machine that invoked SENDCNF should look at the IPCODE field to determine the error.

RETURN_CODE	IPCODE
OK	X'0000'
ALLOCATION_ERROR	Any alloc error code
DEALLOCATE_ABEND_PROG	X'0210'
DEALLOCATE_ABEND_SVC	X'0220'
DEALLOCATE_ABEND_TIMER	X'0230'
PROG_ERROR_PURGING	X'0430'
RESOURCE_FAILURE_NO_RETRY	X'0610'
RESOURCE_FAILURE_RETRY	X'0620'
SVC_ERROR_PURGING	X'0530'

State Changes

No state changes occur. APPC/VM does not support the DEFER state.

ABEND Conditions

The parameter check conditions follow:

Parameter Check Condition	IPRCODE
SYNC_LEVEL(NONE)	37
Invalid resource id	1

The state check conditions follow:

State Check Condition	IPRCODE
Conversation not in SEND state.	32, 34, 35, 36
Conversation started but did not finish sending a logical record	44

APPC CONFIRMED

The APPC CONFIRMED verb maps to the APPC/VM function, SENDCNFD.

Parameters

The following maps the APPC parameter to the APPC/VM parameter. The entry lists the APPC parameter first in blue, followed by the APPC/VM parameter in italics.

RESOURCE - *PATHID*
The resource id returned in APPC/VM is a path id. The path id is a halfword number.

State Changes

Your program may be in either of the following states:

- RECEIVE, if the SENDCNFD is in response to a SENDCNF TYPE = NORMAL
- SEVER, if the SENDCNFD is in response to a SENDCNF TYPE = SEVER. SEVER state is the APPC/VM equivalent of DEALLOCATE state.

ABEND Conditions

The parameter check conditions follow:

Parameter Check Condition	IPRCODE
Invalid resource id	1

The state check conditions follow:

State Check Condition	IPRCODE
Conversation not in CONFIRM state.	32, 33, 34, 36

APPC DEALLOCATE

The APPC DEALLOCATE verb maps to the APPC/VM functions, SEVER and SENDCNF TYPE = SEVER.

Parameters

The following list maps APPC parameters to APPC/VM parameters. Each entry lists the APPC parameter first in blue, followed by the APPC/VM parameter in italics.

RESOURCE - *PATHID*
The resource id returned in APPC/VM is a path id. The path id is a halfword number.

TYPE - *TYPE* **and** *CODE*

TYPE(SYNC_LEVEL)
Use the following APPC/VM functions:

- SEVER TYPE = NORMAL, to do a SYNC_LEVEL(NONE).
- SENDCNF TYPE = SEVER, followed by SEVER TYPE = NORMAL, to do a SYNC_LEVEL(CONFIRM).

TYPE(FLUSH)
Use APPC/VM SEVER TYPE = NORMAL.

TYPE(CONFIRM)
Use the SENDCNF TYPE = SEVER, followed by SEVER TYPE = NORMAL, to do a TYPE(CONFIRM).

TYPE(ABEND_PROG)
Use APPC/VM SEVER with the appropriate SEVER code (CODE = X'210').

TYPE(ABEND_SVC) and TYPE(ABEND_TIMER)
APPC/VM does not support these for general applications.

TYPE(LOCAL)
Use APPC/VM SEVER TYPE = NORMAL after receiving a SEVER from your partner.

LOG_DATA - *no support*
APPC/VM does not support the APPC log data function.

RETURN_CODE - *IPRCODE* and *IPCODE*
For all types of DEALLOCATE, except SYNC_LEVEL(CONFIRM), the only possible return code is OK. For SYNC_LEVEL(CONFIRM), the same mapping exists as for the return codes from CONFIRM (see "APPC CONFIRM" on page 191 for details).

State Changes

After the SEVER completes, your program is in RESET state.

ABEND Conditions

The parameter check conditions follow:

Parameter Check Condition	IPRCODE
Invalid resource id	1

The state check conditions follow:

State Check Condition	IPRCODE
Issued TYPE(FLUSH), TYPE(CONFIRM), TYPE(LOCAL) or TYPE(SYNC_LEVEL) from the wrong state.	32, 34, 35, 36
Issued TYPE(FLUSH) or TYPE(SYNC_LEVEL), and the conversation started but did not finish sending a logical record.	44

APPC GET_ATTRIBUTES

APPC/VM provides the function of the GET_ATTRIBUTES verb, but does not provide a specific APPC/VM function.

The transaction program is responsible for remembering:

- The MODE_NAME from the VM area of the initial receive of the FMH5
- The PARTNER_LU_NAME from the VM area of the initial RECEIVE of the FMH5
- The SYNC_LEVEL in the connection pending interrupt and/or FMH5 ATTACH.

APPC/VM does not provide:

- OWN_FULLY_QUALIFIED_LU_NAME
- PARTNER_FULLY_QUALIFIED_LU_NAME.

APPC RECEIVE_AND_WAIT

The APPC RECEIVE_AND_WAIT verb maps to the APPC/VM function, RECEIVE.

Parameters

The following list maps APPC parameters to APPC/VM parameters. Each entry lists the APPC parameter first, followed by the APPC/VM parameter in italics.

RESOURCE - *PATHID*
The resource id returned in APPC/VM is a path id. The path id is a halfword number.

FILL - *no parameter*
APPC/VM supports FILL(BUFFER), but does not support FILL(LL).

DATA - *BUFFER*
The APPC parameter, DATA(variable), and the APPC/VM parameter, BUFFER=, specifies the address of the buffer to place the data being received.

LENGTH - *BUFLEN*
The APPC parameter, LENGTH(variable), and the APPC/VM parameter, BUFLEN=, define the RECEIVE area length.

When control is returned to the program at the completion of RECEIVE_AND_WAIT, the LENGTH variable contains the length of data received. For APPC/VM, the length variable, IPBFLN2F, contains one of the following:

- The amount of space left in the buffer
- A count of how much data is pending that did not fit into the buffer.

When you are in SEND state and specify a zero length, the RECEIVE completes before the target responds. This maps to PREPARE_TO_RECEIVE TYPE (FLUSH). When you issue RECEIVE with a zero length from RECEIVE state, it completes immediately even if nothing is pending on the path. In APPC/VM, you can use RECEIVE and TESTMSG to do the equivalent of an APPC RECEIVE_AND_WAIT with a zero length.

REQUEST_TO_SEND_RECEIVED - *SENDREQ interrupt*
APPC/VM indicates that the partner issued REQUEST_TO_SEND by reflecting a SENDREQ interrupt.

WHAT_RECEIVED - *IPWHATRC*
You can receive data along with other indicators.

WHAT_RECEIVED(DATA)
In APPC/VM, this indication is presented by either IPWHATRC = IPDATA or IPWHATRC ≠ IPDATA with the length field, IPBFLN2F, less than IPBFLN1F when you issued RECEIVE.

WHAT_RECEIVED(DATA_COMPLETE,DATA_INCOMPLETE, LL_TRUNCATED)
Does not occur in APPC/VM, because APPC/VM does not support FILL(LL).

WHAT_RECEIVED(SEND)
In APPC/VM, this indication is IPWHATRC = IPSEND.

WHAT_RECEIVED(CONFIRM)
In APPC/VM, this indication is IPWHATRC = IPCNFRM.

WHAT_RECEIVED(CONFIRM_SEND)
Does not occur in APPC/VM. APPC/VM does not provide remote support of PREPARE-TO-RECEIVE.

WHAT_RECEIVED(CONFIRM_DEALLOCATE)
In APPC/VM, this function is IPWHATRC = IPCNFSEV.

WHAT_RECEIVED(TAKE_SYNCPT, TAKE_SYNCPT_SEND, TAKE_SYNCPT_DEALLOCATE)
Does not occur in APPC/VM, because APPC/VM does not support the SYNCPT option set.

RETURN_CODE - *IPCODE*
The APPC RETURN_CODE variable corresponds to:

- The APPC/VM IPCODE of SENDERR or SEVER TYPE = ABEND
- The IPWHATRC field of the function.

If the RECEIVE completes with a SENDERR or SEVER TYPE = ABEND, then the virtual machine that issued RECEIVE should look at the IPCODE field to determine the error. IPWHATRC reports the DEALLOCATE_NORMAL indication as IPSNORM.

RETURN_CODE	IPCODE
OK	X'0000'

RETURN_CODE	IPCODE
ALLOCATION_ERROR	Any alloc error code
DEALLOCATE_ABEND_PROG	X'0210'
DEALLOCATE_ABEND_SVC	X'0220'
DEALLOCATE_ABEND_TIMER	X'0230'
PROG_ERROR_NO_TRUNC	X'0410'
PROG_ERROR_TRUNC	X'0420'
PROG_ERROR_PURGING	X'0430'
SVC_ERROR_NO_TRUNC	X'0510'
SVC_ERROR_TRUNC	X'0520'
SVC_ERROR_PURGING	X'0530'
RESOURCE_FAILURE_NO_RETRY	X'0610'
RESOURCE_FAILURE_RETRY	X'0620'

State Changes

When RECEIVE completes, your program may be in any of the following states:

- RECEIVE state, when WHAT_RECEIVED is DATA
- SEND state, when WHAT_RECEIVED is SEND
- CONFIRM state, when WHAT_RECEIVED is CONFIRM or CONFIRM_DEALLOCATE.

APPC/VM does not support the optional SYNCPT state. Also, no state change occurs when the verb is issued in RECEIVE state and WHAT_RECEIVED is DATA.

ABEND Conditions

The parameter check conditions follow:

Parameter Check Condition	IPRCODE
Invalid resource id	1

The state check conditions follow:

State Check Condition	IPRCODE
Conversation not in SEND or RECEIVE state	32, 35, 36
Conversation started but did not finish sending a logical record	44[5]

[5] This condition may also be reported in IPAUDIT by the IPADITRN flag.

APPC REQUEST_TO_SEND

The APPC REQUEST_TO_SEND verb maps to the APPC/VM function, SENDREQ.

Parameters

The following maps the APPC parameter to the APPC/VM parameter. The entry lists the APPC parameter first, followed by the APPC/VM parameter in italics.

RESOURCE - *PATHID*
The resource id returned in APPC/VM is a path id. The path id is a halfword number.

State Changes

No state changes occur.

ABEND Conditions

The parameter check conditions follow:

Parameter Check Condition	IPRCODE
Invalid resource id	1

The state check conditions follow:

State Check Condition	IPRCODE
Conversation not in SEND, RECEIVE, or CONFIRM state	32, 33, 36

APPC SEND_DATA

The APPC SEND_DATA verb maps to the APPC/VM function SENDDATA RECEIVE=NO.

Parameters

The following list maps APPC parameters to APPC/VM parameters. Each entry lists the APPC parameter first, followed by the APPC/VM parameter in italics.

RESOURCE - *PATHID*
The resource id returned in APPC/VM is a path id. The path id is a halfword number.

DATA - *BUFFER*
The APPC parameter, DATA(variable), and the APPC/VM parameter, BUFFER=, specify the address of the data to send.

LENGTH - *BUFLEN*
The APPC parameter, LENGTH(variable), and the APPC/VM parameter, BUFLEN=, specify the length of the data to send.

REQUEST_TO_SEND_RECEIVED - *SENDREQ interrupt*
APPC/VM indicates that the partner issued REQUEST_TO_SEND by reflecting a SENDREQ interrupt.

RETURN_CODE - and *IPCODE*
The APPC RETURN_CODE variable corresponds to:

- The APPC/VM IPCODE of SENDERR
- The APPC/VM IPCODE of SEVER.

If the SENDDATA completes with a SENDERR or SEVER, the virtual machine that issued the SENDDATA should look at the IPCODE field to determine the error.

RETURN_CODE	IPCODE
OK	X'0000'
ALLOCATION_ERROR	Any alloc error code
DEALLOCATE_ABEND_PROG	X'0210'
DEALLOCATE_ABEND_SVC	X'0220'

RETURN_CODE	IPCODE
DEALLOCATE_ABEND_TIMER	X'0230'
PROG_ERROR_PURGING	X'0430'
SVC_ERROR_PURGING	X'0530'
RESOURCE_FAILURE_NO_RETRY	X'0610'
RESOURCE_FAILURE_RETRY	X'0620'

State Changes

No state changes occur. APPC/VM does not support the DEFER state.

ABEND Conditions

The parameter check conditions follow:

Parameter Check Condition	IPRCODE
Invalid resource id	1
Invalid logical record length	42[6]

The state check conditions follow:

State Check Condition	IPRCODE
Conversation not in SEND state	32, 34, 35, 36

[6] This condition may also be reported in IPAUDIT by the IPADIINV flag.

APPC SEND_ERROR

The APPC SEND_ERROR verb maps to the APPC/VM function, SENDERR.

Parameters

The following list maps APPC parameters to APPC/VM parameters. Each entry lists the APPC parameter first, followed by the APPC/VM parameter in italics.

RESOURCE - *PATHID*
The resource id returned in APPC/VM is a path id. The path id is a halfword number.

TYPE - *no parameter*
APPC/VM only supports the APPC TYPE (PROG) option. APPC/VM does not support the APPC TYPE(SVC) option for general applications.

LOG_DATA - *no support*
APPC/VM does not support the APPC log data function.

REQUEST_TO_SEND_RECEIVED - *SENDREQ interrupt*
APPC/VM indicates that the partner issued REQUEST_TO_SEND by reflecting a SENDREQ interrupt.

RETURN_CODE - *IPCODE*
The APPC RETURN_CODE variable corresponds to:

- The APPC/VM IPCODE of SENDERR or SEVER TYPE=ABEND
- The IPWHATRC field of the function.

If the SENDERR completes with an indication that the communication partner issued a SENDERR or SEVER, the virtual machine should look at the IPCODE field to determine the error.

If you issue SEND_ERROR from the SEND state, the following return codes are possible. You may also get an indication of DEALLOCATE_NORMAL in IPWHATRC as IPSNORM.

RETURN_CODE	IPCODE
OK	X'0000'

RETURN_CODE	IPCODE
ALLOCATION_ERROR	Any alloc error code
DEALLOCATE_ABEND_PROG	X'0210'
DEALLOCATE_ABEND_SVC	X'0220'
DEALLOCATE_ABEND_TIMER	X'0230'
PROG_ERROR_PURGING	X'0430'
SVC_ERROR_PURGING	X'0530'
RESOURCE_FAILURE_NO_RETRY	X'0610'
RESOURCE_FAILURE_RETRY	X'0620'

If you issue SEND_ERROR from the RECEIVE state, the following return codes are possible. You may also get an indication of DEALLOCATE_NORMAL in IPWHATRC.

RETURN_CODE	IPCODE
OK	X'0000'
RESOURCE_FAILURE_NO_RETRY	X'0610'
RESOURCE_FAILURE_RETRY	X'0620'

If you issue SEND_ERROR from the CONFIRM state, the following return codes are possible:

RETURN_CODE	IPCODE
OK	X'0000'
RESOURCE_FAILURE_NO_RETRY	X'0610'
RESOURCE_FAILURE_RETRY	X'0620'

State Changes

If you issue the SENDERR, you remain in or are put into SEND state.

ABEND Conditions

The parameter check conditions follow:

Parameter Check Condition	IPRCODE
LOG_DATA not supported	Not supported
Invalid resource id	1

The state check conditions follow:

State Check Condition	IPRCODE
Conversation not in SEND, RECEIVE, or CONFIRM state	32, 36

Part Three: CP System Services for TSAF

This part describes the two CP system services that let TSAF communicate with CP. It introduces and describes how the system services contribute to the TSAF facility.

- "Chapter 9. Collection Resource Management (*CRM) System Service" on page 209 describes what the Collection Resource Management System Service (*CRM) is responsible for and how the intended TSAF virtual machine connects to and severs from the Collection Resource Management System Service. This chapter also describes the two types of messages that the Collection Resource Management System Service expects from the TSAF virtual machine.

- "Chapter 10. Identify (*IDENT) System Service" on page 215 describes what the Identify System Service (*IDENT) is responsible for, how to connect to it, and how to sever from it. This chapter also describes how the Identify System Service lets different virtual machines revoke resources.

Chapter 9. Collection Resource Management (*CRM) System Service

The Collection Resource Management System Service is a CP system service, known as *CRM. It lets an authorized virtual machine connect to it and become the TSAF virtual machine.

Authorizing Virtual Machines to Connect to *CRM

You, the system administrator, can authorize a virtual machine to connect to *CRM, and thus become the TSAF virtual machine, with the *CRM parameter of the IUCV control statement. This parameter is described in "Setting Up the TSAF Virtual Machine" on page 11.

What *CRM Does

*CRM provides a communications path between CP and the TSAF virtual machine. The messages sent back and forth on the *CRM path keep the collection resource table and the CP system resource table current.

The TSAF virtual machine uses the *CRM path to:

- Get the names of the global resources listed in the local system resource table. The TSAF virtual machine receives this information only while it is connected to *CRM.
- Notify CP of a revoke for the loser of a global resource when two disjoint collections merge.
- Notify CP of a revoke, initiated by a virtual machine on another system, for a global resource managed on the local system.

The Identify System Service (*IDENT) uses the *CRM path to:

- Send an Identify request to the TSAF virtual machine when a virtual machine tries to connect to *IDENT to identify a global resource. The TSAF virtual machine verifies if another system in the collection manages the resource. Then the TSAF virtual machine replies yes or no to the Identify request.
- Send the TSAF virtual machine a revoke request when a resource manager gives up managing a global resource. The resource manager

gives up managing by severing its connection to the Identify System Service.

- Send the TSAF virtual machine a revoke request when an authorized virtual machine connects to the Identify System Service to revoke a global resource.

  When another system in the collection manages the global resource, the TSAF virtual machine on that system tells CP, through its *CRM path, that a virtual machine revoked the resource.

Connecting to *CRM - Becoming the TSAF Virtual Machine

The intended TSAF virtual machine does an IUCV CONNECT to *CRM during its initialization. This establishes an IUCV path, called the *CRM path. Only one virtual machine can connect to *CRM, and the virtual machine must be authorized in its directory entry.

When a virtual machine tries to connect to *CRM, *CRM severs the pending connection for the following conditions:

- A virtual machine is already connected to *CRM.
- The virtual machine trying to connect to *CRM specified PRMDATA = YES or QUIESCE = YES on the connect request.
- The virtual machine trying to connect to *CRM specified PRTY = YES on the connect request and is authorized to send priority messages to *CRM. In some cases, however, *CRM accepts the connection, but the virtual machine (now the TSAF virtual machine) is not allowed to send priority messages on the path.

After *CRM accepts the connection to the virtual machine, if the TSAF virtual machine,

- Issues a SEVER on the path to *CRM, *CRM severs the path to the TSAF virtual machine.
- Issues a QUIESCE on the path to *CRM, *CRM severs the path to the TSAF virtual machine.
- Issues a RESUME on the path to *CRM, IUCV returns without taking any action.

*CRM Communications

When the TSAF virtual machine is connected to *CRM, *CRM expects either of these two types of messages from the TSAF virtual machine:

1. A request that *CRM send the global resources in the system resource table to the TSAF virtual machine.

2. A REVOKE request. The request could be remote from a virtual machine on another system. Or, the revoke could be one that the TSAF virtual machine requests because another collection is joining this collection. The local system manager may lose management of the resource when there are duplicate resources in the merging collections. See "When Two Collections Merge to Form One" on page 30 for details on merging collections.

Requesting System Resource Table Information

The TSAF virtual machine issues a one-way IUCV SEND to send messages on the *CRM path. For the format of the IUCV SEND function, see the *VM System Facilities for Programming*.

For a request to send the global resources in the system resource table, the TSAF virtual machine sends the data in the following format:

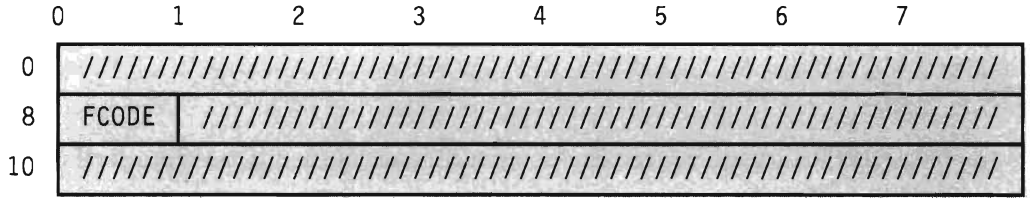


Figure 64. SEND Data Format from TSAF Virtual Machine

FCODE
: is the function code for the request. FCODE = 3 indicates that the TSAF virtual machine wants *CRM to send the global resources in the system resource table.

*CRM responds to a request to send the global resources in the system resource table by issuing a one-way SEND on the *CRM path. The SEND data has the following format:

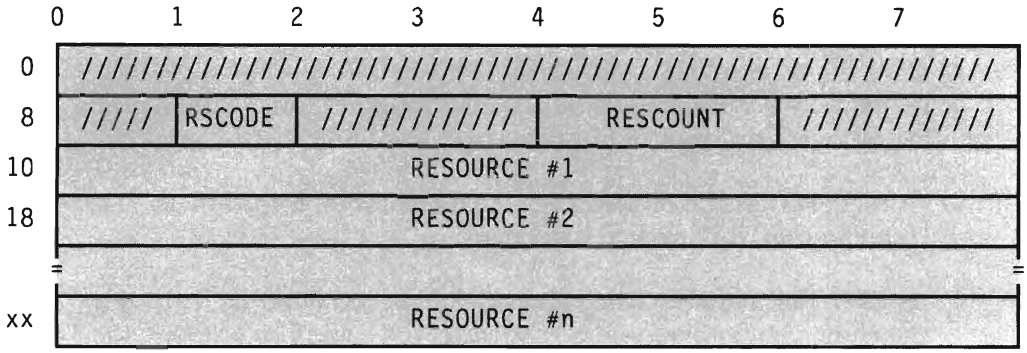


Figure 65. SEND Data Format from *CRM

RSCODE = 3
is the response to a request to send the global resources in the system resource table.

RESCOUNT
is the number of resource names that follow, starting in byte 16. If there are no global resources in the table, RESCOUNT is 0 and the SEND data is 16 bytes in length.

*CRM only includes nonpending resources in the SEND data, because the TSAF virtual machine already has information about pending resources. See "What *IDENT Does" on page 215 for information about pending resources.

Revoking a Resource

For a REVOKE request, the TSAF virtual machine sends the data in the following format:

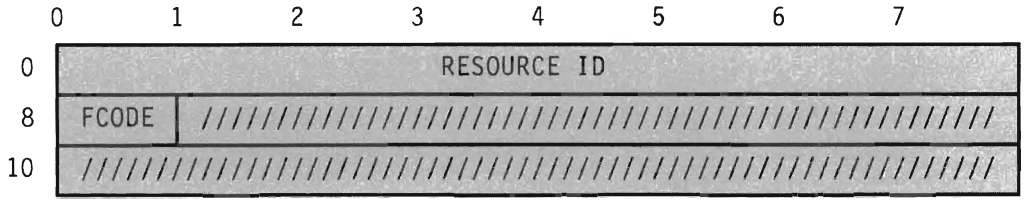


Figure 66. Revoke Data from the TSAF Virtual Machine

RESOURCE ID
is the resource that the TSAF virtual machine is revoking.

FCODE
is the function code for the request. FCODE = 2 indicates a revoke request.

When the TSAF virtual machine sends a revoke request for a global resource managed on the local system, *CRM invokes a routine in *IDENT to:

1. Delete the resource from the system resource table.
2. Sever the path between *IDENT and the resource manager virtual machine.

Severing the *CRM Connection

If the TSAF virtual machine severs its connection to *CRM, it gives up its status as the TSAF virtual machine. The sever does not directly affect any existing APPC/VM paths. However, you cannot establish new remote connections until a virtual machine, which can handle remote connections, connects to *CRM to become the TSAF virtual machine.

The condition of the TSAF virtual machine can affect any remote paths that the TSAF virtual machine is supporting. For example, if you re-IPL the TSAF virtual machine, not only does CP sever the path to *CRM, but CP severs any other IUCV connections that the TSAF virtual machine has. Also, if the TSAF virtual machine that you re-IPLed has any cross-system paths through it (meaning that the TSAF virtual machine is not on the source or target system for the path, that is, it is an intermediate TSAF virtual machine), the TSAF virtual machines on the other systems in the collection may sever the cross-system paths. The TSAF virtual machines in the collection would sever the paths if they could not find another way to route them.

*CRM SEVER Reason Codes

When CP severs the *CRM path, it stores a return code in byte 0 of the IPUSER field in the IUCV SEVER interrupt buffer. This indicates the reason for the SEVER. The following lists the possible Collection Resource Management System Service return codes.

*CRM issues return codes 0 through 6, and 1xx. The Identify System Service (*IDENT) issues return codes 6, 7, and 8.

Code Description

0 The SEVER is a response to a SEVER that the TSAF virtual machine issued.

1 CP cannot make the connection because a virtual machine is already connected to *CRM. The virtual machine trying to connect to *CRM could be the same virtual machine that is already connected.

2 The virtual machine specified one of the following in the pending connection parameter list: PRMDATA = YES, PRTY = YES, or QUIESCE = YES. When PRTY = YES is specified, this return code is issued only if the virtual machine has directory authorization to send priority messages to *CRM.

3 *CRM received a message with an invalid FCODE from the TSAF virtual machine.

4 The TSAF virtual machine sent a message to *CRM that was not TYPE=1WAY.

5 The TSAF virtual machine is not allowed to issue IUCV QUIESCE on the *CRM path.

6 CP severed the *CRM path because the message limit was exceeded on the path. The TSAF virtual machine was not receiving the messages sent by CP on the *CRM path.

7 The reply to a "2WAY" message sent by *IDENT to the TSAF virtual machine on the *CRM path contained an invalid return code. RCODE is not 0, 1, or 2.

8 The message complete interrupt for a two-way message contained a nonzero IPAUDIT trail with bits other than IPADSVRD set. *IDENT sent this message on the *CRM path to the TSAF virtual machine.

1*xx* The TSAF virtual machine sent a message that caused an IPRCODE of *xx* when *CRM received it.

Chapter 10. Identify (*IDENT) System Service

The Identify System Service is a CP system service, known as *IDENT, that lets authorized virtual machines connect to it and identify themselves as resource managers. *IDENT also lets authorized virtual machines revoke ownership of a resource.

Authorizing Virtual Machines to Connect to *IDENT

You, the system administrator, can authorize virtual machines to connect to *IDENT, and thus become resource managers, with the *IDENT parameter of the IUCV control statement, explained in "IUCV Directory Control Statement for *IDENT Authorization" on page 19. The parameters on the IUCV *IDENT control statement follow:

1. The first parameter following *IDENT authorizes for:
   - A specific resource name (resid)
   - Any resource names (RESANY).
2. The second parameter authorizes for:
   - Local resources only (LOCAL)
   - Global and local resources (GLOBAL).
3. The third parameter authorizes a virtual machine to REVOKE the current management of the resource.

What *IDENT Does

*IDENT maintains a local system resource table. Each entry contains:

- Pointers to the next and previous entries in the table
- Resource name and the VMBLOK address of the virtual machine that manages the resource
- Path id of the IUCV path between *IDENT and the virtual machine that manages the resource
- Local and/or global indicator
- Other indicators for pending functions related to the resource:

- Identify is pending.
- Sever by the resource manager is pending.
- Revoke is pending.
- Remote revoke is pending; that is, the TSAF virtual machine or a virtual machine on another system is trying to revoke the global resource.

*IDENT adds an entry to the table each time it accepts a virtual machine connection. *IDENT deletes the entry when it severs the associated connection. A virtual machine manages a resource only while it is connected to *IDENT.

*IDENT also uses the *CRM path, described in "What *CRM Does" on page 209.

Some Rules about Resources

Only the local system knows about local resources. However, a global resource is known to all the TSAF virtual machines within the collection. Only one virtual machine can manage a global resource at a time. A resource can have a local manager and a global manager on the same system.

The general rule is that the first virtual machine to request to be the resource manager will manage the resource. No more than 200 resources, both local and global, can be defined on the local system at one time.

CP passes identify and revoke requests for global resources to the TSAF virtual machine. This is so all TSAF virtual machines in the collection can reach an agreement. CP marks the resource table entry as *pending* until the TSAF virtual machine responds. The resource becomes unavailable; a virtual machine cannot connect to, identify, or revoke a resource that is marked as pending in the system resource table.

*IDENT Communications - Connecting to *IDENT

To use the Identify System Service, issue an IUCV CONNECT. You must specify the userid as *IDENT, and the user data field must have the following format:

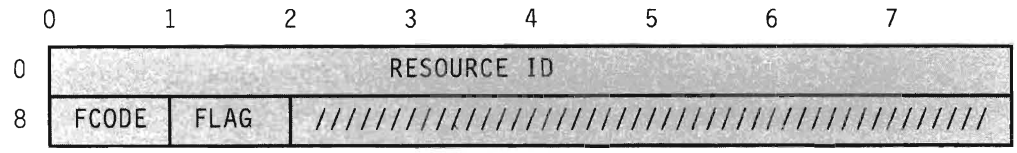


Figure 67. User Data Field for CONNECT

RESOURCE ID
: is the name of the resource that you are requesting to manage. The first byte of the resource id must be alphanumeric. (IBM reserves names beginning with the remaining characters for its own use.) The resource id that you specify cannot be blank, hex zeroes, or "ANY", "ALLOW", or "SYSTEM".

FCODE
: is the function code.

  FCODE = 1
  : indicates an Identify request.

  FCODE = 2
  : indicates a REVOKE request.

FLAG is a flag byte that indicates the following:

: For Identify requests:

  Bit 0 on
  : defines the resource as global, known to all TSAF virtual machines in the collection.

  Bit 0 off
  : defines the resource as local, known only to the local system.

  For REVOKE requests:

  Bit 0 on
  : tells CP to revoke the global resource, known to all TSAF virtual machines in the collection.

  Bit 0 off
  : tells CP to revoke the local resource, known only to the local system.

When you try to connect to *IDENT to manage or revoke a resource, *IDENT checks the validity of the pending connection parameter list. *IDENT severs the connection for any of the following:

- The function code is not equal to 1 (to manage a resource) or 2 (to revoke a resource).
- PRMDATA = YES is specified.
- The IUCV control statements in the CP directory do not show you authorized for the connection.
- The resource id is invalid.

How *IDENT Processes Requests to Manage a Resource

If you are requesting to manage a resource and the parameter list is valid, *IDENT checks its system resource table. If it finds the resource in its table, it severs the connection to you, because that resource already exists. If *IDENT does not find the resource in the table, and you request to manage:

- A local resource, *IDENT adds the resource to the table and accepts your connection.
- A global resource, *IDENT:
  1. Adds the resource name to the system resource table
  2. Marks the table entry as a pending Identify
  3. Passes the request to the TSAF virtual machine.

So that you do not send any messages over the path, CP accepts connections to *IDENT by specifying QUIESCE = YES. Because *IDENT quiesces the path to you, the resource manager, on the ACCEPT, *IDENT can never receive an incoming message on the path. If you issue a QUIESCE or a RESUME on the path, IUCV returns with no action taken.

How CP Passes Requests to the TSAF Virtual Machine

*IDENT issues a two-way IUCV SEND. This passes the request to manage the global resource on the *CRM path to the TSAF virtual machine. For the complete format of the IUCV SEND function, see the *VM System Facilities for Programming*.

The SEND data has the following format:

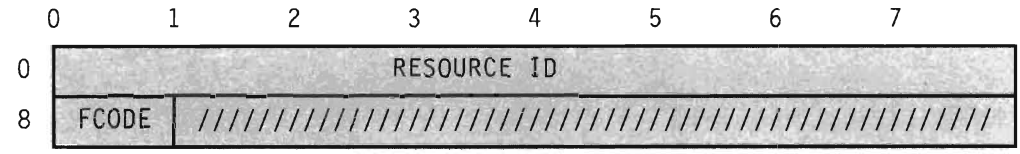


Figure 68. SEND Data Format from *IDENT

RESOURCE ID
contains the resource that you are requesting to manage.

FCODE
contains the function code for the request. FCODE = 1 is an Identify request to add the resource.

Answer Data from the TSAF Virtual Machine

Because the Identify message to the TSAF virtual machine from *IDENT is two-way, *IDENT provides an answer area. The answer data from the TSAF virtual machine is in the following format:

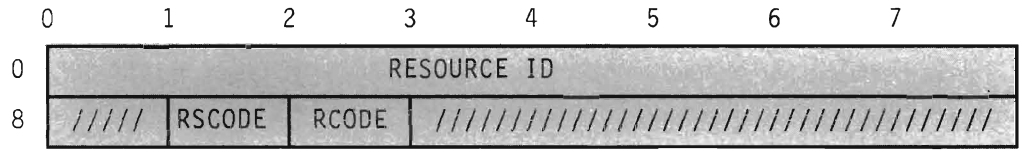


Figure 69. Answer Data Format from the TSAF Virtual Machine

RESOURCE ID
contains the resource that you are requesting to manage.

RSCODE
contains the response code. RSCODE = 1 is an Identify response.

RCODE
contains the TSAF virtual machine's return code for the response.

RCODE = 0
means to *IDENT that the Identify was successful, and TSAF lets your virtual machine manage the global resource. The TSAF virtual machine sends this reply if no other virtual machine in the collection already manages the resource. *IDENT marks the system resource table entry as *not pending* and accepts the connection.

If, when you request to manage a global resource, there is no virtual machine connected to *CRM, then your system is not part of a collection. *IDENT acts as if the TSAF virtual machine told it to accept the connection. If CP or the TSAF virtual machine severs the *CRM path before the TSAF virtual machine sends the reply to the Identify request, *IDENT acts as if the TSAF virtual machine told it to accept the connection.

RCODE = 1
means that the Identify was unsuccessful. *IDENT severs the pending connection and deletes the pending resource table entry.

RCODE = 2
means that the TSAF virtual machine was busy, and *IDENT should retry the SEND.

How Virtual Machines Connect to a Resource Manager

If your virtual machine becomes a resource manager (establishes a connection to *IDENT), APPC/VM lets authorized virtual machines connect to your virtual machine. Virtual machines can connect to your virtual machine by specifying the resource name. If a virtual machine on your local system tries to connect to your resource, CP handles the connection; TSAF does not get involved with local connections. If the resource is defined as both local and global, CP connects the requesting virtual machine to the local resource manager.

However, if a virtual machine on a different system tries to connect to your resource, the following occurs:

1. The other system's CP passes the connect request to the TSAF virtual machine on the other system.
2. The TSAF virtual machine checks its table, and, as a result, passes the request on to your local TSAF virtual machine.
3. Your local TSAF virtual machine issues a CONNECT to you on behalf of the original requesting virtual machine.

You, as the resource manager, can either ACCEPT or SEVER the connection. The TSAF virtual machine on your system reflects your answer back to the TSAF virtual machine on the other system, who passes the answer to CP and then to the requesting virtual machine.

Severing the *IDENT Connection - Revoking a Resource

The following can revoke a resource:

- A virtual machine authorized to revoke the resource
- The resource manager virtual machine (by severing its path to *IDENT)
- The TSAF virtual machine.

Connecting to *IDENT to Revoke a Resource

If you are authorized, you can revoke a resource. Your system administrator can authorize you to revoke a particular resource (*resid*) by specifying the following in your CP directory entry:

```
IUCV *IDENT resid LOCAL/GLOBAL REVOKE
```

This is also explained in "Chapter 1. Preparing to Use TSAF" on page 11.

You can connect to *IDENT (function code = 2) to revoke a local or a global resource. See "*IDENT Communications - Connecting to *IDENT" on page 216 for the CONNECT data. If the resource is local, *IDENT:

1. Deletes the resource from the system resource table
2. Severs the connection to the former resource manager
3. Severs the connection to your virtual machine.

If the resource is global, and managed on the local system, *IDENT:

1. Marks the resource as a pending revoke in the system resource table
2. Passes the request to the TSAF virtual machine through the *CRM connection.

If the resource is not defined on the local system as a global resource, *IDENT passes the request on to the TSAF virtual machine.

How CP Passes Revoke Requests to the TSAF Virtual Machine

*IDENT issues a two-way IUCV SEND to pass the request on the *CRM path to the TSAF virtual machine. For the complete format of the IUCV SEND function, see the *VM System Facilities for Programming*.

The SEND data has the following format:

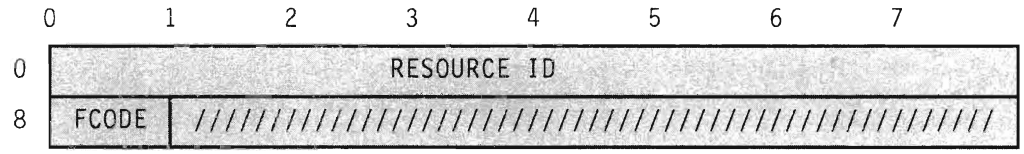


Figure 70. SEND Data Format from *IDENT

RESOURCE ID
: contains the resource that you are requesting to revoke.

FCODE
: contains the function code for the request. FCODE = 2 is a REVOKE request to delete the global resource.

The TSAF virtual machine notifies the other TSAF virtual machines in the collection of the revoke. If the resource that you revoked was not on your system, the TSAF virtual machine on the system where the resource was tells CP that the resource was revoked remotely. (See "Chapter 9. Collection Resource Management (*CRM) System Service" on page 209 for the format of the revoke message that the TSAF virtual machine sends to *CRM). *IDENT deletes the resource on that system from the system resource table and severs the connection between *IDENT and the virtual machine that managed the resource.

After notifying the other TSAF virtual machines in the collection, the TSAF virtual machine on your system tells *IDENT the results of the revoke request.

Answer Data from the TSAF Virtual Machine

Because the SEND to the TSAF virtual machine is 2-WAY, *IDENT provides an answer area. The answer data from the TSAF virtual machine is in the following format:

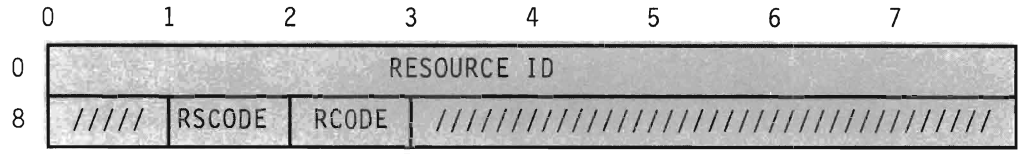


Figure 71. Answer Data Format from the TSAF Virtual Machine

RESOURCE ID
: contains the resource that you are revoking.

RSCODE
: contains the response code. RSCODE = 2 is a REVOKE response.

RCODE
: contains the TSAF virtual machine's return code for the response.

  RCODE = 0
  : means that TSAF revoked the resource.

  RCODE = 1
  : means that TSAF could not find the resource. *IDENT severs the pending connection with your virtual machine.

  RCODE = 2
  : means that the TSAF virtual machine was busy, and *IDENT should retry the SEND.

When *IDENT gets the reply with RCODE = 0, and the global resource that you are revoking is managed on the local system, *IDENT:

1. Deletes the resource from the system resource table.
2. Severs the connection to the resource manager virtual machine.
3. Severs the pending connection with your virtual machine.

When *IDENT gets the reply with RCODE = 0, and the global resource that you are revoking is not managed on the local system, *IDENT severs the pending connection with your virtual machine.

Revoking Your Own Resources

To stop managing a resource on your virtual machine, issue a SEVER on the path to *IDENT. If you defined the resource as a local resource, *IDENT:

1. Deletes the local resource from the system resource table.
2. Severs its half of the path.

If you defined the resource as a global resource, *IDENT marks the resource as a pending revoke (due to the manager's SEVER) in the system resource table. *IDENT then sends a revoke notification to the TSAF virtual machine on the *CRM path. See the format on page 221 for revoke requests sent to the TSAF virtual machine. The TSAF virtual machine notifies all the other TSAF virtual machines in the collection. When notified of the reply from the TSAF virtual machine, *IDENT:

1. Deletes the global resource from the system resource table.
2. Severs its half of the path.

The SEVER does not affect existing APPC/VM paths to your virtual machine. However, CP does not establish any new paths to you. If another virtual machine connects to *IDENT to manage the resource that you revoked, requests to connect to the resource go to that virtual machine.

Note: If a virtual machine initiates a connection request to a resource that you manage before your revoke completes, the path may be established.

Revoking Resources in Merging Collections

When two disjoint collections merge, and the same resource name is specified on both collections, the TSAF virtual machine issues a revoke for the virtual machine that loses management responsibility. TSAF does this through the *CRM connection. *CRM passes the revoke to *IDENT, which severs the path between *IDENT and the former resource manager virtual machine.

See "When Two Collections Merge to Form One" on page 30 for information on how TSAF determines what collection wins management responsibility of duplicate resources.

*IDENT Sever Reason Codes

When *IDENT severs one of its paths, it stores a return code in byte 10 of the IPUSER field in the IUCV SEVER parameter list. This indicates the reason for the SEVER. The *IDENT return code is zero for SEVERs that are in response to a SEVER by the virtual machine. The following lists the possible *IDENT return codes:

Code	Description
0	*IDENT revoked the resource as requested.
1	An I/O error occurred while CP was reading the CP directory. *IDENT was checking for authorization to identify or revoke the resource.
2	The CONNECT parameter list has an invalid function code in FCODE of IPUSER; it is not equal to 1 for Identify or 2 for Revoke.
3	The CONNECT parameter list has an invalid parameter PRMDATA.
4	The virtual machine is not authorized to connect to *IDENT for the specified resource.
5	The virtual machine is not authorized to identify the resource as a global resource.
6	The virtual machine is not authorized to revoke the specified resource.
7	The virtual machine is not authorized to revoke the specified resource globally.
8	CP cannot identify the resource because the system resource table currently contains the maximum of 200 entries owned on the local system. (These resources can be any combination of local and global resources).
9	A virtual machine already manages the resource being identified. The virtual machine trying to identify a resource could be the same virtual machine that already manages the resource.
10	A virtual machine revoked the resource. The resource may have been revoked by the virtual machine that managed the resource.
11	The resource to be revoked does not exist.
12	The resource is pending identification by a virtual machine and is not available to be identified or revoked.
13	The resource is pending a revoke by a virtual machine and is not available to be identified or revoked. An authorized virtual machine is revoking the resource, or the resource manager virtual machine is severing its path to *IDENT.
14	The connect parameter list has an invalid resource name specified.

Appendix A. APPC/VM and IUCV Condition Codes and Return Codes

This appendix summarizes the return codes and conditions codes that you may get upon execution of the APPC/VM or IUCV functions available with the TSAF support.

There are four possible values for condition codes (0,1,2,3). Condition codes vary in meaning depending on the function on the APPCVM macro.

With the APPC/VM support, there are four types of return codes. The types of return codes are meaningful depending on the condition code value. The types of return codes (and what-received indications) follow:

IPRCODE
reports error conditions that CP detects when the function is initiated.

There is no corrective action for this type of error. You should sever the path when you get nonzero in this field.

IPAUDIT
reports error conditions that CP detects between the time that the function is initiated and the time that the function completes. IPAUDIT codes are only issued for RECEIVE and SENDDATA.

Like IPRCODE, there is no corrective action for this type of error. You should sever the path when you get a nonzero IPAUDIT.

IPCODE
contains the SEVER or error return code caused by your partner application or intermediate communication server. If IPWHATRC is:

- X'09', the return code is a SEVER code.
- X'03', the return code is an error code.

IPWHATRC
contains the return code or what-received indication caused by your partner application or intermediate communication server.

IPCODE and IPWHATRC are contiguous fields. You can think of them as a logical 3-byte entity.

When IPWHATRC is a what-received indication, IPCODE contains zero and serves no purpose. IPWHATRC represents a what-received indication when it contains one of the following:

- X'01'—DATA

- X'02'—SEND
- X'04'—CONFIRM
- X'05'—CONFIRM_DEALLOCATE

On the other hand, IPWHATRC and IPCODE represent a return code when IPWHATRC contains one of the following:

- X'03'—SENDERR
- X'08'—SEVER TYPE = NORMAL
- X'09'—SEVER TYPE = ABEND

Figure 72 summarizes the conditions under which each of the four return code fields is meaningful. In the figure, an "X" indicates that the field is meaningful for the corresponding condition code. For WAIT = YES, CC = 0 is not possible. For WAIT = NO, CC = 3 is not possible.

CC =	IPRCODE (IUCV)	IPAUDIT (IUCV)	IPCODE (APPC/VM)	IPWHATRC (APPC/VM)
CC = 0		X	X	X
CC = 1	X			
CC = 2			X	X
CC = 3		X		

Figure 72. Meaningful Codes Based on CC =

Note: CC = 0 (WAIT = NO), IPAUDIT, IPCODE, and IPWHATRC are only meaningful in the function complete interrupt. The interrupt signals the completion of the function that started with CC = 0.

The remainder of this appendix contains two figures:

1. Figure 73 shows condition codes and IPRCODE return code values for each APPC/VM and IUCV function.

2. Figure 74 on page 231 lists the IPAUDIT fields for the appropriate APPC/VM functions (RECEIVE, SENDDATA, TESTCMPL).

Condition Codes and IPRCODE Values

Function	Condition Codes (CC =)	Return Codes (Stored in IPRCODE) for CC = 1
ACCEPT (IUCV)	**1** Nonzero value stored in IPRCODE. **2** ACCEPT is complete.	**01** Invalid path id - not a pending connection. **20** Connection cannot be completed - originator has severed the path.

Figure 73 (Part 1 of 5). APPC/VM and IUCV Condition Codes and IPRCODE Values

Function	Condition Codes (CC =)	Return Codes (Stored in IPRCODE) for CC = 1
CONNECT (APPC/VM)	**0** Started successfully, but not yet complete. **1** Nonzero value stored in IPRCODE. **2** CONNECT completed.	**11** TSAF could not find the resource, or it is not available. **12** Your partner has not invoked the DCLBFR function. **13** Your virtual machine already has the maximum number of connections. **14** Your partner already has the maximum number of connections. **15** You are not authorized to connect to the resource. **28** You specified CONTROL = YES, but no control buffer exists. **29** Not authorized to act for another user. **39** Invalid connection parameter list extension length. **40** Invalid LU-NAME. **41** Invalid mode name.
DCLBFR (IUCV)	**0** Normal completion. **1** Nonzero value stored in IPRCODE. **3** Errors while reading directory.	**19** Buffer already declared.
DESCRIBE (IUCV)	**0** Normal completion. **2** No message found.	None.
QUERY (IUCV)	**0** Normal return. **2** User not in CP directory. **3** Errors when reading CP directory.	None

Figure 73 (Part 2 of 5). APPC/VM and IUCV Condition Codes and IPRCODE Values

Function	Condition Codes (CC=)	Return Codes (Stored in IPRCODE) for CC=1
RECEIVE (APPC/VM)	**0** Started successfully, but not yet complete. **1** Nonzero value in IPRCODE. **2** RECEIVE completed. **3** RECEIVE completed, with errors. The type of error is reported in the IPAUDIT field, listed in Figure 74 on page 231.	**01** Invalid path id. **03** Pending function on this path. **06** Storage protection exception on partner's SEND buffer. **07** Addressing exception on your partner's SEND buffer. **10** Negative buffer length. **22** Partner's SEND list is invalid. **23** Negative length in RECEIVE list. **24** Incorrect total length. **26** Buffer list address is not on a doubleword boundary. **30** APPC/VM function on a non-APPC path. **32** Invalid function from CONNECT state. **35** Invalid function from CONFIRM state. **36** Invalid function from SEVER state. **43** Invalid record length in partner's data stream. **44** Did not finish sending a logical record. **45** Your partner did not finish sending a logical record.
RTRVBFR (IUCV)	**0** Normal completion.	None.
SENDCNF (APPC/VM)	**0** Started, but not yet completed. **1** Nonzero value stored in IPRCODE. **2** SENDCNF completed.	**01** Invalid path id. **03** Pending function on this path. **30** APPC/VM function on a non-APPC path. **32** Invalid function from CONNECT state. **34** Invalid function from RECEIVE state. **35** Invalid function from CONFIRM state. **36** Invalid function from SEVER state. **37** Connection established with SYNCLVL=NONE. **38** Invalid value in IPSENDOP. **44** Did not finish sending a logical record.
SENDCNFD (APPC/VM)	**1** Nonzero value stored in IPRCODE. **2** SENDCNFD completed.	**01** Invalid path id **30** APPC/VM function on a non-APPC path. **32** Invalid function from CONNECT state. **33** Invalid function from SEND state. **34** Invalid function from RECEIVE state. **36** Invalid function from SEVER state. **38** Invalid value in IPSENDOP.

Figure 73 (Part 3 of 5). APPC/VM and IUCV Condition Codes and IPRCODE Values

Function	Condition Codes (CC=)	Return Codes (Stored in IPRCODE) for CC=1
SENDDATA (APPC/VM)	**0** Started successfully, but not yet complete. **1** Nonzero value in IPRCODE. **2** SENDDATA completed. **3** SENDDATA completed, with errors. The type of error is reported in the IPAUDIT field, listed in Figure 74 on page 231.	**01** Invalid path id. **03** Pending function on this path. **06** Protection exception on partner's answer or RECEIVE area. **07** Addressing exception on partner's answer or RECEIVE area. **10** Negative buffer or answer length. **22** Partner's answer or RECEIVE list is invalid. **23** Negative length in SEND buffer list. **24** Incorrect total length. **26** Buffer list address is not on a doubleword boundary. **27** Buffer list or answer list is not on a doubleword boundary. **30** APPC/VM function on a non-APPC path. **32** Invalid function from CONNECT state. **34** Invalid function from RECEIVE state. **35** Invalid function from CONFIRM state. **36** Invalid function from SEVER state. **38** Invalid value in IPSENDOP. **42** Invalid logical record length in data stream. **44** Did not finish sending a logical record.
SENDERR (APPC/VM)	**0** Started successfully, but not yet complete. **1** Nonzero value in IPRCODE. **2** SENDERR completed.	**01** Invalid path id. **03** Pending function on this path. **29** Not authorized to act for another user. **30** APPC/VM function on a non-APPC path. **32** Invalid function from CONNECT state. **36** Invalid function from SEVER state. **38** Invalid value in IPSENDOP.
SENDREQ (APPC/VM)	**1** Nonzero value in IPRCODE. **2** SENDREQ completed.	**01** Invalid path id. **30** APPC/VM function on a non-APPC path. **32** Invalid function from CONNECT state. **36** Invalid function from SEVER state. **38** Invalid value in IPSENDOP.
SETCMASK (IUCV)	**0** Normal completion.	None
SETMASK (IUCV)	**0** Normal completion.	None

Figure 73 (Part 4 of 5). APPC/VM and IUCV Condition Codes and IPRCODE Values

Function	Condition Codes (CC=)	Return Codes (Stored in IPRCODE) for CC=1
SEVER (APPC/VM)	**1** Nonzero value in IPRCODE. **2** SEVER completed.	**01** Invalid path id. **03** Pending function on this path. **29** Not authorized to act for another user. **30** APPC/VM function on a non-APPC path. **32** Invalid function from CONNECT state. **34** Invalid function from RECEIVE state. **35** Invalid function from CONFIRM state. **36** Invalid function from SEVER state. **38** Invalid value in IPSENDOP. **44** Did not finish sending a logical record. **46** Invalid SEVER code.
TESTCMPL (IUCV)	**0** Normal completion. **1** Nonzero value in IPRCODE. **2** No message or function was found. **3** Function completed with errors. The type of error is reported in the IPAUDIT field, listed in Figure 74 on page 231.	**01** Invalid path id.
TESTMSG (IUCV)	**1** Pending message or SENDREQ indication. **2** Pending message or function completion. **3** CC=1 and CC=2 occurred.	None.

Figure 73 (Part 5 of 5). APPC/VM and IUCV Condition Codes and IPRCODE Values

IPAUDIT Values

The following table lists the IPAUDIT codes for the appropriate functions. The IPAUDIT field in the TESTCMPL output parameter list can contain any of the following values, depending on the function TESTCMPL is completing.

Function	IPAUDIT Code (Stored in IPAUDIT)
RECEIVE (APPC/VM)	**X'100000'** Protection exception on your RECEIVE buffer. **X'080000'** Addressing exception on your RECEIVE buffer. **X'002000'** Protection exception on your partner's SEND buffer. **X'001000'** Addressing exception on your partner's SEND buffer. **X'000400'** Your partner had an invalid SEND list. **X'000040'** Bad length in your RECEIVE buffer list. **X'000010'** RECEIVE buffer length is invalid. **X'000008'** Partner's data stream has invalid logical record length. **X'000002'** Partner did not finish sending logical record and tried to change to RECEIVE state.
SENDDATA (APPC/VM)	**X'400000'** Protection exception on your SEND buffer. **X'200000'** Addressing exception on your SEND buffer. **X'100000'** Protection exception on your answer buffer. **X'080000'** Addressing exception on your answer buffer. **X'008000'** Protection exception on your partner's answer or RECEIVE area. **X'004000'** Addressing exception on your partner's answer or RECEIVE area. **X'002000'** Protection exception on your partner's SEND data area. **X'001000'** Addressing exception on your partner's SEND data area. **X'000400'** Partner had an invalid SEND, answer or RECEIVE list. **X'000080'** Bad length in your SEND buffer list. **X'000040'** Bad length in your SEND answer list. **X'000020'** Total SEND buffer length is invalid. **X'000010'** Total SEND answer length is invalid. **X'000008'** Invalid logical record length in partner's data stream. **X'000004'** Invalid logical record length in your data stream. **X'000002'** Partner did not finish sending a logical record and tried to change to RECEIVE state. **X'000001'** You did not finish sending a logical record and tried to change to RECEIVE state.

Figure 74. IPAUDIT Values

Appendix B. APPC - APPC/VM Mapping Summary

This appendix summarizes the mapping of SNA LU 6.2 (APPC architecture) verb functions to APPCVM and IUCV macro functions. Figure 75 contains a list of the APPC verbs and their equivalents in VM. The tables on pages 235 through 248 summarize the APPC to APPC/VM mapping in a detailed manner. The mapping is organized in tables by APPC verb names and APPC/VM (and IUCV) macro names. Both inputs and outputs are shown.

To make the APPC/VM to APPC/SNA mapping throughout this appendix more readable, the following name values are used in places where the program interface specifies numeric values.

CC = started for CC = 0.

CC = error for CC = 1.

CC = completed
for CC = 2.

CC = 3 indicates that the function completed with error information available in the IPAUDIT fields of the interrupt buffer.

APPC Verb Name to APPC/VM Macro Parameter Name

The following chart lists the APPC verbs and their APPC/VM equivalents and gives an overall view of how APPC and APPC/VM functions are related.

Note: The IUCV functions related to APPC/VM are unique to VM and have no equivalents in APPC.

APPC Verb	APPC/VM Equivalent
ACTIVATE_SESSION	*No equivalent*
ALLOCATE	CONNECT
CHANGE_SESSION_LIMIT	*No equivalent*

Figure 75 (Part 1 of 2). APPC Verbs and APPC/VM Functions

APPC Verb	APPC/VM Equivalent
CONFIRM	SENDCNF TYPE = NORMAL
CONFIRMED	SENDCNFD
DEACTIVATE_SESSION	*No equivalent*
DEALLOCATE	SEVER and SENDCNF TYPE = SEVER
FLUSH (remote)	*No equivalent*
GET_ATTRIBUTES	*No direct support*
INITIALIZE_SESSION_LIMIT	*No equivalent*
PREPARE_TO_RECEIVE (remote)	*No equivalent*
PROCESS_SESSION_LIMIT	*No equivalent*
RECEIVE_AND_WAIT	RECEIVE
REQUEST_TO_SEND	SENDREQ
RESET_SESSION_LIMIT	*No equivalent*
SEND_DATA	SENDDATA RECEIVE = NO
SEND_DATA and RECEIVE_AND_WAIT	SENDATA RECEIVE = YES
SEND_ERROR	SENDERR

Figure 75 (Part 2 of 2). APPC Verbs and APPC/VM Functions

All VM programs that communicate by APPC/VM reside within a single TSAF collection, the logical equivalent of a single SNA Logical Unit (LU). Therefore, as shown in Figure 75, the LU-to-LU session control (CNOS) verbs are not supported by VM. For the same reason, the remote LU support of the optional APPC FLUSH and PREPARE_TO_RECEIVE verbs are not provided by VM.

APPC Verb Parameters Mapped with APPC/VM Macro Parameters

The tables in this section provide, for each supported APPC function, a detailed view of how APPC verb parameters and APPC/VM macro parameters are related.

APPC ALLOCATE to APPC/VM CONNECT

APPC ALLOCATE parameters and the APPC/VM equivalents are listed below:

APPC ALLOCATE Input	APPC/VM CONNECT Input
LU_NAME (OWN)[7] LU_NAME (OTHER (variable)) MODE_NAME (variable)	BUFFER = [VM architected data] bytes 0- 7 = 0 = > OWN bytes 8-15 = mode name
TPN (variable)	RESID = variable
TYPE (BASIC_CONVERSATION)	Default, no parameter
RETURN_CONTROL (WHEN_SESSION_ALLOCATED)	Default, no parameter
SYNC_LEVEL (NONE) (CONFIRM)	SYNCLVL = NONE = CONFIRM
RECOVERY_LEVEL (NONE)	Default, no parameter
SECURITY (NONE)	Default, no parameter
PIP (NO)	Default, no parameter
Not Available[8]	ALTID = variable

APPC ALLOCATE Output	APPC/VM CONNECT Output
RESOURCE (variable)	IPPATHID
RETURN_CODE (OK)	CC = error[9] CC = {started|complete} IPTYPE = '82'
RETURN_CODE (ALLOCATION_ERROR {ALLOCATION_FAILURE_RETRY, ALLOCATION_FAILURE_NO_RETRY, SYNC_LEVEL_NOT_SUPPORTED_BY_LU	 IPTYPE = '83', CC = error IPCODE = X'0111', IPRCODE = 11,12,13,14 IPCODE = X'0110',X'0610', IPRCODE = 15 N/A
RETURN_CODE (PARAMETER_ERROR) {invalid LU name invalid mode name}	 IPRCODE = 40 IPRCODE = 41

The APPC/VM output appears in the connection complete interrupt buffer.

---

7 Option set supported for communication within the collection.

8 This function is provided for communication servers only, and is not part of the general API.

9 Possible IPRCODEs when CC = error are 28,29,39,40,41.

APPC CONFIRM to APPC/VM SENDCNF TYPE = NORMAL

APPC CONFIRM parameters and the APPC/VM equivalents are listed below:

APPC CONFIRM Input	APPC/VM SENDCNF TYPE = NORMAL Input
RESOURCE (variable)	PATHID = variable
Default, no parameter No equivalent	WAIT = YES WAIT = NO

APPC CONFIRM Output	APPC/VM SENDCNF TYPE = NORMAL Output
RETURN_CODE	CC = {started | complete} and (IPCODE + IPWHATRC)
(OK)	X'0000' + X'00'
(ALLOCATION_ERROR)	{X'0110',X'0111',X'0130'} + X'09'
(DEALLOCATE_ABEND_PROG)	X'0210' + X'09'
(DEALLOCATE_ABEND_SVC)	X'0220' + X'09'
(DEALLOCATE_ABEND_TIMER)	X'0230' + X'09'
(PROG_ERROR_PURGING)	X'0430' + X'03'
(RESOURCE_FAILURE_NO_RETRY)	X'0610' + X'09'
(RESOURCE_FAILURE_RETRY)	X'0620' + X'09'
(SVC_ERROR_PURGING)	X'0530' + X'03'
REQUEST_TO_SEND_RECEIVED (YES) (NO)	(SENDREQ interrupt presented for YES condition)

Possible IPRCODEs when CC = error are: 1, 3, 30, 32, 34, 35, 36, 37, 38, 44.

APPC CONFIRMED to APPC/VM SENDCNFD

APPC CONFIRMED parameters and the APPC/VM equivalents are listed below:

APPC CONFIRMED Input	APPC/VM SENDCNFD Input
RESOURCE (variable)	PATHID = variable

Possible IPRCODEs when CC = error are: 0, 1, 30, 32, 33, 34, 36, 38.

APPC DEALLOCATE to APPC/VM SEVER or SENDCNF TYPE = SEVER

APPC DEALLOCATE parameters and the APPC/VM equivalents are listed below:

APPC DEALLOCATE Input	APPC/VM SEVER and SENDCNF TYPE = SEVER Input
RESOURCE (variable)	PATHID = variable
TYPE (SYNC_LEVEL) "if SYNC_LEVEL = NONE" "if SYNC_LEVEL = CONFIRM"	SEVER TYPE = NORMAL (2 steps) SENDCNF TYPE = SEVER and "if (CC = {started |complete} & IPCODE = 0)" "then" SEVER TYPE = NORMAL
TYPE (FLUSH)	SEVER TYPE = NORMAL
TYPE (CONFIRM)	SENDCNF TYPE = SEVER and "if CC = {started | complete} & IPCODE = 0)" "then" SEVER TYPE = NORMAL
TYPE (ABEND_PROG)	SEVER TYPE = ABEND CODE = {DEALLOCATE_ABEND_PROG}
TYPE (ABEND_SVC)[10]	SEVER TYPE = ABEND CODE = {DEALLOCATE_ABEND_SVC}
TYPE (ABEND_TIMER)[10]	SEVER TYPE = ABEND CODE = {DEALLOCATE_ABEND_TIMER}
TYPE (LOCAL)	SEVER TYPE = NORMAL
LOG_DATA (NO) LOG_DATA (YES (variable))[10]	Default, no parameter Not supported

APPC DEALLOCATE Output	APPC/VM SEVER or SENDCNF TYPE = SEVER Output
RETURN_CODE "if TYPE(¬CONFIRM)" (OK)	CC = {started} and (IPCODE + IPWHATRC) X'0000' + X'00'

10 For remote support only.

APPC DEALLOCATE Output	APPC/VM SEVER or SENDCNF TYPE=SEVER Output
"if TYPE(CONFIRM)"	
(OK)	X'0000' + X'00'
(ALLOCATION_ERROR)	X'01xx' + X'09'
(DEALLOCATE_ABEND_PROG)	X'0210' + X'09'
(DEALLOCATE_ABEND_SVC)	X'0220' + X'09'
(DEALLOCATE_ABEND_TIMER)	X'0230' + X'09'
(PROG_ERROR_PURGING)	X'0430' + X'03'
(RESOURCE_FAILURE_NO_RETRY)	X'0610' + X'09'
(RESOURCE_FAILURE_RETRY)	X'0620' + X'09'
(SVC_ERROR_PURGING)	X'0530' + X'03'

Possible IPRCODEs when CC=error are: 1, 3, 29, 30, 32, 34, 35, 36, 38, 44, 46.

APPC GET_ATTRIBUTES to Indirect APPC/VM Support

APPC GET_ATTRIBUTES and the APPC/VM equivalents are listed below:

APPC GET_ATTRIBUTES Input	APPC/VM Input
RESOURCE (variable)	No direct support.

APPC GET_ATTRIBUTES Output	APPC/VM Output
OWN_FULLY_QUALIFIED_LU_NAME (variable)	Not available; a null value is valid if the name is not known.
PARTNER_LU_NAME (variable)	From VM architected data in the first RECEIVE.
PARTNER_FULLY_QUALIFIED_LU_NAME (variable)	Not available; a null value is valid if the name is not known.
MODE_NAME (variable)	From VM architected data in the first RECEIVE.
SYNC_LEVEL (variable)	From connection pending interrupt buffer or FMH5 in first RECEIVE.

APPC RECEIVE_AND_WAIT to APPC/VM RECEIVE

The RECEIVE function is used by the transaction program to receive the FMH5. This use of RECEIVE is unique to APPC/VM.

APPC RECEIVE_AND_WAIT parameters and the APPC/VM equivalents are listed below:

APPC RECEIVE_AND_WAIT Input	APPC/VM PRECEIVE Input
RESOURCE (variable)	PATHID = variable
FILL (LL)	Not supported (not required if FILL (BUFFER) supported)
FILL (BUFFER)	Default, no parameter

APPC RECEIVE_AND_WAIT Input/Output	APPC/VM RECEIVE Input/Output
LENGTH (variable) "input"	BUFLEN = variable[11]
LENGTH (variable) "output"	If IPWHATRC = DATA then IPBFLN2F = #arrived − BUFLEN else IPBFLN2F = BUFLEN − #received

APPC RECEIVE_AND_WAIT Output	APPC/VM RECEIVE Output
DATA (variable)	BUFFER = variable
"If verb issued in SEND state" RETURN_CODE (OK) (ALLOCATION_ERROR) (DEALLOCATE_ABEND_PROG) (DEALLOCATE_ABEND_SVC) (DEALLOCATE_ABEND_TIMER) (PROG_ERROR_PURGING) (RESOURCE_FAILURE_NO_RETRY) (RESOURCE_FAILURE_RETRY) (SVC_ERROR_PURGING)	CC = {started | complete} and (IPCODE + IPWHATRC) X'0610' + X'09' X'0000' + X'00' X'01xx' + X'09' X'0210' + X'09' X'0220' + X'09' X'0230' + X'09' X'0430' + X'03' X'0620' + X'09' X'0530' + X'03'

[11] If the length specified is zero, the APPC/VM verb could complete before the partner responds.

APPC RECEIVE_AND_WAIT Output	APPC/VM RECEIVE Output
"If verb issued in RECEIVE state" RETURN_CODE (OK) (ALLOCATION_ERROR) (DEALLOCATE_ABEND_PROG) (DEALLOCATE_ABEND_SVC) (DEALLOCATE_ABEND_TIMER) (DEALLOCATE_NORMAL) (PROG_ERROR_NO_TRUNC) (PROG_ERROR_PURGING) (PROG_ERROR_TRUNC) (RESOURCE_FAILURE_NO_RETRY) (RESOURCE_FAILURE_RETRY) (SVC_ERROR_NO_TRUNC) (SVC_ERROR_PURGING) (SVC_ERROR_TRUNC)	CC = {started | complete} and (IPCODE + IPWHATRC)[12] X'0000' + X'00' X'01xx' + X'09' X'0210' + X'09' X'0220' + X'09' X'0230' + X'09' X'0000' + X'08' X'0410' + X'03' X'0430' + X'03' X'0420' + X'03' X'0610' + X'09' X'0620' + X'09' X'0510' + X'03' X'0530' + X'03' X'0520' + X'03'
REQUEST_TO_SEND_RECEIVED (YES) (NO)	(SENDREQ interrupt presented for YES condition)
WHAT_RECEIVED (DATA) (DATA_COMPLETE) (DATA_INCOMPLETE) (LL_TRUNCATED) (SEND) (CONFIRM) (CONFIRM_SEND) (CONFIRM_DEALLOCATE)	IPWHATRC = X'01' or ¬ = X'01' & IPBFLN2F < BUFLEN Cannot occur, no FILL(LL) Cannot occur, no FILL(LL) Cannot occur, no FILL(LL) = X'02'[12] = X'04'[12] Cannot occur = X'05'[12]

The APPC/VM output appears in the function complete interrupt buffer, except for BUFFER = variable.

Possible IPRCODEs when CC = error are: 1, 3, 6, 7, 10, 22, 23, 24, 26, 30, 32, 35, 36, 43, 44, 45.

---

12 Data may also have been RECEIVEd. This represents combined architecture functions implemented for efficiency.

APPC REQUEST_TO_SEND to APPC/VM SENDREQ

APPC REQUEST_TO_SEND parameters and the APPC/VM equivalents are listed below:

APPC REQUEST_TO_SEND Input	APPC/VM SENDREQ Input
RESOURCE (variable)	PATHID = variable

Possible IPRCODEs when CC = error are: 1, 30, 32, 36, 38.

APPC SEND_DATA to APPC/VM SENDDATA RECEIVE = NO

APPC SEND_DATA parameters and the APPC/VM equivalents are listed below:

APPC SEND_DATA Input	APPC/VM SENDDATA RECEIVE = NO Input
RESOURCE (variable)	PATHID = variable
DATA (variable)	BUFFER = variable
LENGTH (variable)	BUFLEN = variable

APPC SEND_DATA Output	APPC/VM SENDDATA RECEIVE = NO Output
RETURN_CODE	CC = {started | complete} and (IPCODE + IPWHATRC)
(OK)	X'0000' + X'00'
(ALLOCATION_ERROR)	X'01xx' + X'09'
(DEALLOCATE_ABEND_PROG)	X'0210' + X'09'
(DEALLOCATE_ABEND_SVC)	X'0220' + X'09'
(DEALLOCATE_ABEND_TIMER)	X'0230' + X'09'
(PROG_ERROR_PURGING)	X'0430' + X'03'
(RESOURCE_FAILURE_NO_RETRY)	X'0610' + X'09'
(RESOURCE_FAILURE_RETRY)	X'0620' + X'09'
(SVC_ERROR_PURGING)	X'0530' + X'03'
REQUEST_TO_SEND_RECEIVED	(SENDREQ interrupt presented
(YES)	for YES condition)
(NO)	

APPC/VM output appears in the function complete interrupt buffer.

The possible IPRCODEs when CC = error are: 1, 3, 6, 7, 10, 22, 23, 24, 26, 27, 30, 32, 34, 35, 36, 38, 42.

APPC SEND_DATA and RECEIVE_AND_WAIT to APPC/VM SENDDATA RECEIVE = YES

The APPC architecture-defined equivalent of the SENDDATA RECEIVE = YES is the following APPC verbs immediately following one another:

1. SEND_DATA
2. RECEIVE_AND_WAIT

The SENDDATA RECEIVE = YES fields for input are:

PATHID = variable
BUFFER = variable
BUFLEN = variable
WAIT = YES
ANSBUF = variable
ANSLEN = variable

Figure 76. APPC/VM SENDDATA RECEIVE = YES Input Fields

The APPC/VM output is in the function complete interrupt buffer. The SENDDATA RECEIVE = YES fields for output are:

CC = {started | complete} and (IPCODE + IPWHATRC) X'0000' + X'00' X'01xx' + X'09' X'0210' + X'09' X'0220' + X'09' X'0230' + X'09' X'0430' + X'03' X'0610' + X'09' X'0620' + X'09' X'0530' + X'03'
WHATRC[13] = {X'01', X'02', X'08', X'04', and X'05'}

Figure 77. APPC/VM SENDDATA RECEIVE = YES Output Fields

Possible IPRCODEs when CC = error are 1, 3, 6, 7, 10, 22, 23, 24, 26, 27, 30, 32, 34, 35, 36, 38, 42, 44.

---

[13] A consequence of the RECEIVE = YES option; IPCODE is zero.

APPC SEND_ERROR to APPC/VM SENDERR

APPC SEND_ERROR parameters and the APPC/VM equivalents are listed below:

APPC SEND_ERROR Input	APPC/VM SENDERR Input
RESOURCE (variable)	PATHID = variable
TYPE (PROG) (SVC)[14]	Default, no parameter CODE = { 510, 520, 530}[15]
LOG_DATA (NO) LOG_DATA (YES (variable))[15]	Default, no parameter Not supported
Default, no parameter No equivalent	WAIT = YES WAIT = NO

APPC SEND_ERROR Output	APPC/VM SENDERR Output
RETURN_CODE	CC = {started | complete} and (IPCODE + IPWHATRC)
"if verb issued in SEND state" (OK) (ALLOCATION_ERROR) (DEALLOCATE_ABEND_PROG) (DEALLOCATE_ABEND_SVC) (DEALLOCATE_ABEND_TIMER) (PROG_ERROR_PURGING) (RESOURCE_FAILURE_NO_RETRY) (RESOURCE_FAILURE_RETRY) (SVC_ERROR_PURGING)	 X'0000' + X'00' X'01xx' + X'09' X'0210' + X'09' X'0220' + X'09' X'0230' + X'09' X'0430' + X'03' X'0610' + X'09' X'0620' + X'09' X'0530' + X'03'
"if verb issued in RECEIVE state" (OK) (DEALLOCATE_NORMAL) (RESOURCE_FAILURE_NO_RETRY) (RESOURCE_FAILURE_RETRY)	 X'0000' + X'00' X'0000' + X'08' X'0610' + X'09' X'0620' + X'09'
"if verb issued in CONFIRM state" (OK) (RESOURCE_FAILURE_NO_RETRY) (RESOURCE_FAILURE_RETRY)	 X'0000' + X'00' X'0610' + X'09' X'0620' + X'09'
REQUEST_TO_SEND_RECEIVED (YES) (NO)	(SENDREQ interrupt presented for YES condition)

The SENDERR output is in the function complete interrupt buffer.

Possible IPRCODEs when CC=error are 1, 3, 29, 30, 32, 36, 38.

---

[14] Remote support only.

[15] Use of CODE parameter is restricted to communication servers.

APPC LU-Generated Responses to APPC/VM SEVER TYPE = ABEND

APPC LU Generated Responses	APPC/VM SEVER TYPE = ABEND Codes
CONVERSATION_TYPE_MISMATCH	X'0120'
SYNC_LEVEL_NOT_SUPPORTED_BY_PGM	X'0130'
TRANS_PGM_NOT_AVAIL_NO_RETRY	X'0140'
TRANS_PGM_NOT_AVAIL_RETRY	X'0141'
TPN_NOT_RECOGNIZED	X'0142'
PIP_NOT_SPECIFIED_CORRECTLY	X'0150'
DEALLOCATE_ABEND_PROG	X'0210' or X'0610'

In a VM application, this SEVER can only be issued between the ACCEPT and the "next" APPC/VM operation. In an APPC/SNA implementation, these "deallocate" conversation return codes would be issued by the LU before starting the TPN.

No APPC Function to IUCV ACCEPT

(APPC Equivalent)	IUCV ACCEPT
ACCEPT causes a Connection Complete interrupt for CONNECT WAIT = NO. For CONNECT WAIT = YES, ACCEPT causes IPTYPE = Connection Complete.	PATHID = variable

CONNECT completes when your communication partner issues ACCEPT. Your communication partner may not be the actual target of your CONNECT, but, instead, may be an intermediate communication server. Do not make assumptions about the target of the CONNECT when the CONNECT completes. Your CONNECT may complete before the target program is even invoked. ACCEPT is part of the IUCV interface, and you should not use it for special significance such as synchronization.

Appendix C. Sample TSAF User Program

This section contains a sample TSAF user application. The program shows how to use APPC/VM to connect to and request a service from a resource. When reviewing this program, note the following:

- This sample demonstrates the APPC/VM functions. It does not show how to use APPC/VM efficiently or how to write a user application.
- Though this program could be used as a CMS application, it does not use the CMS IUCV support; rather it uses the CP IUCV support directly.
- For simplicity, this program only uses synchronous support. This is not recommended for use by general applications. When you use synchronous support, if the resource that you are communicating with becomes hung up, you must either "LOGOFF" or reset your virtual machine to get out of the APPC/VM WAIT.

The amount of data being sent and received by this program and the sample resource program (in Appendix D, "Sample TSAF Resource Manager Program" on page 257) is transparent to the other side. However, the two communicating programs must use the same format for data that is being exchanged, and they both must follow the APPC/VM protocol.

After assembling this program, you may invoke it by entering:

START USER *resid filename filetype*

where:

resid
is the resource id that has the requested file.

filename
is the file name of the requested file.

filetype
is the file type of the requested file.

This program can then connect to the resource and request the file. After the resource sends the contents of the specified file, this program types out the file.

```
         PRINT NOGEN               Don't expand macro calls
*
*--------------------------------------------------------------------
* Basic housekeeping
*--------------------------------------------------------------------
USER     START X'20000'            Start program counter at X'20000'
         USING USER,R12            Establish base register 12
         STM   R14,R12,12(R13)     Save system's registers
         ST    R13,SAVEMAIN+4      Save pointer to system's save area
         LA    R13,SAVEMAIN        R13 points to our save area
         LA    R2,PLIST            Get address of APPC/VM parm list
         USING IPARML,R2           Establish parameter list mapping
*
*--------------------------------------------------------------------
*   When invoking this program, you must specify a resource id
*   and the file name and file type of the requested file.
*   We get the resource id, file name and file type out of the
*   parameter list; CMS passes the address of the parameter list to us
*   in register 1.  You can find more information about this interface
*   in the VM/SP CMS Command Reference under the START command.
*--------------------------------------------------------------------
         MVC   RESIDSP,8(R1)       Get the resource id
         MVC   FILENAME,16(R1)     Get the file name
         MVC   FILETYPE,24(R1)     Get the file type
*
*--------------------------------------------------------------------
*   For the purposes of this example, we keep track of the state of
*   the path.  Before we establish the path, it is in RESET state.
*--------------------------------------------------------------------
         MVI   STATE,RESET         Show that no path is established yet
*
*--------------------------------------------------------------------
*   Before using APPC/VM, we must define an interrupt buffer for our
*   program to use, using the DCLBFR function.  The interrupt buffer
*   that we use is at label INTBUF.  Even though we may not get any
*   interrupts, we must still issue DCLBFR to establish ourselves as
*   an APPC/VM user before we can issue any other APPC/VM functions.
*--------------------------------------------------------------------
         XC    PLIST,PLIST         Start with a clean parameter list
         IUCV  DCLBFR,             DCLBFR function                    $
               PRMLIST=PLIST,      Use PLIST as the parameter list    $
               BUFFER=INTBUF,      Use INTBUF as the interrupt buffer $
               CONTROL=NO          Declare INTBUF an application buffer
         BC    CC1,DCLFAIL         Go to DCLFAIL, if a user error
         BC    CC3,DIRERROR        Go to DIRERROR, if a directory error
*
*--------------------------------------------------------------------
*   After establishing our virtual machine as an APPC/VM user, we try
*   to CONNECT to the resource (specified with resource id).  Because
*   this program isn't interested in LU_NAME and mode name, we specify
*   a connection extension length of zero.  CP uses the default
*   LU_NAME and mode name when creating the allocation data for our
*   target.  Whether or not the target is interested in this
*   information is not a concern of our program.
*
*   Because we specify WAIT=YES, our virtual machine won't regain
*   control until the target resource accepts or rejects (SEVERs) our
*   connection.  The possible condition codes are:
*     CC=1  An error.
*     CC=2  Connection completes successfully (partner ACCEPTs), or
*           Connection completes unsuccessfully (partner SEVERs).
*--------------------------------------------------------------------
         XC    PLIST,PLIST         Start with a clean parameter list
```

```
         APPCVM CONNECT,           CONNECT function                    $
               PRMLIST=PLIST,      Use PLIST as the parameter list     $
               RESID=RESIDSP,      RESIDSP contains the target resource$
               SYNCLVL=NONE,       Don't allow SENDCNF or SENDCNFD     $
               BUFLEN=ZERO,        No connection extension             $
               WAIT=YES            Wait for the ACCEPT or SEVER
         BC    CC1,CONERROR        Go to CONERROR, if error
*
*--------------------------------------------------------------------
*   The CONNECT is complete.  We examine the connection complete data
*   to see if the target ACCEPTed or SEVERed.  If the target severed,
*   we issue a RTRVBFR and exit the program.
*--------------------------------------------------------------------
         MVC   USERPATH,IPPATHID   Save the path id
         CLI   IPTYPE,IPTYPSVA     Was the CONNECT rejected?
         BE    CONERROR            If CONNECT was rejected, exit
         MVI   STATE,SEND          Otherwise, change to SEND state
*
*--------------------------------------------------------------------
*   We have established an APPC/VM path between our program and the
*   resource. We are in SEND state, and the resource is in RECEIVE
*   state.
*
*   We send the file request to the resource.  We use SENDDATA
*   RECEIVE=YES to do the following:
*     1) Send the request.
*     2) Switch the resource to SEND state so the resource can respond.
*     3) Define a receive area for the initial SEND by the resource.
*
*   The length of the SEND is the length of the file name and file type
*   plus the 2-byte length field sent with the data.
*     - The length is the length of the data being sent (including
*       the 2-byte header).
*
*   Because we specify WAIT=YES on this SENDDATA, we do not get
*   control back until the resource sends a response.  We define our
*   response area just large enough for the response header, which
*   contains the length of the response.
*--------------------------------------------------------------------
         LA    R7,SENDBLEN         Get length of header and data
         STH   R7,SENDLEN          Store the length in the header
         LA    R3,RECVHLEN         Receive just the header
         XC    PLIST,PLIST         Start with a clean parameter list
         APPCVM SENDDATA,          SENDDATA function                   $
               PRMLIST=PLIST,      Use PLIST as the parameter list     $
               PATHID=USERPATH,    Send the data on the user path      $
               BUFFER=SENDBUF,     Take the data out of SENDBUF        $
               BUFLEN=(R7),        Get the length of SENDBUF           $
               ANSBUF=RECVBUF,     Predefine an answer area            $
               ANSLEN=(R3),        Receive just the header             $
               RECEIVE=YES,        Switch to RECEIVE state             $
               WAIT=YES            Get control after the SEND starts
         BC    CC1+CC3,SENDERR     Go to SENDERR, if error
         MVI   STATE,RECEIVE       Assume we end up in RECEIVE state
         CLI   IPWHATRC,IPDATA     Is there more data coming?
         BE    SENDOK              If so, go to SENDOK
         MVI   STATE,SEND          Maybe it's SEND state
         CLI   IPWHATRC,IPSEND     Is that the end of the data?
         BE    SENDOK              Yes, just the header was sent
         MVI   STATE,RECEIVE       Maybe there was a SENDERR
         CLI   IPWHATRC,IPERROR    Was there a SENDERR?
         BE    SENDERR             Yes, go to SENDERR
         MVI   STATE,SEVER         Otherwise, sever the path
```

```
         B     SENDERR             Go to SENDERR
*
*-------------------------------------------------------------------
*   The SENDDATA RECEIVE=YES completed.  The resource sent us the
*   response to our request.  We check the return code sent by the
*   resource to see if our request was successful.
*-------------------------------------------------------------------
SENDOK   DS    0H
         CLC   RECVRC,ZERO         Was the file read correctly?
         BNE   READERR             If not, go to READERR
*
*-------------------------------------------------------------------
*   We now display the file contents by receiving and displaying
*   80 bytes at a time.  We could have easily defined a larger buffer
*   and received all of the data at once on the SENDDATA RECEIVE=YES.
*   However, allocating a large buffer would be unnecessary for some
*   file requests.  We've written this program to work no matter how
*   much data the resource sent in response (up to 32K) or how many
*   SENDDATAs the resource used to send the response.
*
*   In the following set of instructions, R3 contains the number of
*   bytes being received, and R4 contains the number of bytes left
*   to receive.
*-------------------------------------------------------------------
         LH    R4,RECVLEN          Get the logical record length
         LA    R3,RECVHLEN         Get the length of the header
         SR    R4,R3               Get the length of the data
DISPLAY  DS    0H
         LA    R3,RECVDLEN         Get the 80-byte typical length
         CR    R4,R3               Are there 80 bytes left?
         BNL   DISPIT              If so, continue
         LR    R3,R4               If not, display what's left
DISPIT   DS    0H
         XC    PLIST,PLIST         Start with a clean parameter list
         APPCVM RECEIVE,           RECEIVE function                  $
               PRMLIST=PLIST,      Use PLIST as the parameter list   $
               PATHID=USERPATH,    Use the user path                 $
               BUFFER=RECVDATA,    Use data area--no more headers sent $
               BUFLEN=(R3),        RECEIVE more data                 $
               WAIT=YES            Wait for the RECEIVE to complete
         BC    CC1+CC3,RECVERR     If there is an error, go to RECVERR
         MVI   STATE,RECEIVE       Assume we're in RECEIVE state
         CLI   IPWHATRC,IPDATA     Is there more data to come?
         BE    RECVOK              If so, go to RECVOK
         MVI   STATE,SEND          Maybe it's SEND state
         CLI   IPWHATRC,IPSEND     Is that the end of the data?
         BE    RECVOK              If so, go to RECVOK
         MVI   STATE,RECEIVE       Maybe there was a SENDERR
         CLI   IPWHATRC,IPERROR    Was there a SENDERR?
         BE    RECVERR             If so, go to RECVERR
         MVI   STATE,SEVER         Sever the path
         B     RECVERR             Go to RECVERR
*
*-------------------------------------------------------------------
*   The RECEIVE was successful.  Display the next line of the file.
*-------------------------------------------------------------------
RECVOK   DS    0H
         WRTERM RECVDATA,(R3)      Display the next line
*
*-------------------------------------------------------------------
*   Subtract the amount of data just received from the total amount
*   left.  If there is data left to receive and display, then go back
```

```
*   to DISPLAY to display it.
*-------------------------------------------------------------------
         SR    R4,R3               Subtract the amount received
         BH    DISPLAY             Go to DISPLAY, if more to receive
*
*-------------------------------------------------------------------
*   At this point, we have displayed the entire contents of the file.
*   For simplicity, this program is designed to request a single file
*   and then sever the path.  Our last RECEIVE should have indicated
*   that the resource switched us back to SEND state (IPWHATRC=IPSEND).
*
*   If we are in SEND state, we issue a SEVER TYPE=NORMAL to end
*   our conversation normally.  If we aren't in SEND state, we issue
*   a SEVER TYPE=ABEND to end our conversation in an abnormal way.
*-------------------------------------------------------------------
         SR    R15,R15             Indicate normal completion
         CLI   STATE,SEND          Are we in SEND state?
         BE    NRMSEVER            If so, sever normally (NRMSEVER)
         B     ERRSEVER            If not, sever abnormally (ERRSEVER)
*
*-------------------------------------------------------------------
*   The following sets of instructions cover the possible errors that
*   could occur while running this program.  These routines set
*   the appropriate return codes, sever the path, and exit the program.
*-------------------------------------------------------------------
*
*
*-------------------------------------------------------------------
*   There was an error on the DCLBFR.  The only IPRCODE defined for
*   DCLBFR is 19, which means that an IUCV interrupt buffer has already
*   been defined by our virtual machine.  Because the buffer was not
*   declared by our program, we should not do a RTRVBFR.
*-------------------------------------------------------------------
DCLFAIL  DS    0H
         LA    R15,DCLBFAIL        Indicate the DCLBFR failed
         B     EXIT                Go to the EXIT
*
*-------------------------------------------------------------------
*   Condition code 3 from DCLBFR means that CP encountered an I/O
*   error while reading the directory entry for this virtual machine.
*   Because the buffer was never declared, we cannot issue a SEVER or
*   a RTRVBFR.
*-------------------------------------------------------------------
DIRERROR DS    0H
         LA    R15,DCLBFRIO        Indicate an I/O error on DCLBFR
         B     EXIT                Go to the EXIT
*
*-------------------------------------------------------------------
*   There was an error on the CONNECT.  We do not have a path, and,
*   therefore, cannot invoke SEVER.  We must do the RTRVBFR.
*-------------------------------------------------------------------
CONERROR DS    0H
         LA    R15,CONFAIL         Indicate error on CONNECT to *IDENT
         B     RTREXIT             Do a RTRVBFR and exit
*
*-------------------------------------------------------------------
*   An error occurred on the SENDDATA that sent the request to the
*   resource.
*-------------------------------------------------------------------
SENDERR  DS    0H
         LA    R15,SENDFAIL        Indicate SEND of the request failed
         B     SEVERERR            Sever and exit
*
*-------------------------------------------------------------------
```

```
*   The resource reported an error.  We add 100 to the code returned
*   and exit.
*-------------------------------------------------------------------
READERR  DS    0H
         ICM   R15,B'1111',RECVRC  Get the return code
         LA    R15,100(,R15)       Add 100
         B     SEVERERR            Sever and exit
*
*-------------------------------------------------------------------
*   An error occurred while receiving the data sent.
*-------------------------------------------------------------------
RECVERR  DS    0H
         LA    R15,RECVFAIL        Indicate a RECEIVE failed
*
*-------------------------------------------------------------------
*   An error occurred.  We want to sever the path abnormally.  However,
*   if the error that occurred was that the path was severed, then we
*   are in SEVER state and must issue SEVER TYPE=NORMAL.
*-------------------------------------------------------------------
SEVERERR DS    0H
         CLI   STATE,SEVER         Are we in SEVER state?
         BE    NRMSEVER            If so, sever normally (NRMSEVER)
*
*-------------------------------------------------------------------
*   Sever the path normally or abnormally.
*-------------------------------------------------------------------
ERRSEVER DS    0H
         XC    PLIST,PLIST         Clear the parameter list
         APPCVM SEVER,             SEVER function                    $
               PRMLIST=PLIST,      Use PLIST as the parameter list   $
               PATHID=USERPATH,    Sever the user's path             $
               TYPE=ABEND,         SEVER type is ABEND               $
               CODE=DEALPROG       SEVER code for DEALLOCATE_ABEND_PROG
         B     RTREXIT             Exit now
NRMSEVER DS    0H
         XC    PLIST,PLIST         Clear the parameter list
         APPCVM SEVER,             SEVER function                    $
               PRMLIST=PLIST,      Use PLIST as the parameter list   $
               PATHID=USERPATH,    Sever the user's path             $
               TYPE=NORMAL         SEVER type is normal
*
*-------------------------------------------------------------------
*   EXITS:
*   We use this exit to leave the program anytime after we have
*   issued the DCLBFR successfully.
*-------------------------------------------------------------------
RTREXIT  DS    0H
         IUCV  RTRVBFR             Do the RTRVBFR
EXIT     DS    0H
         ST    R15,SAVERC          Save the return code
*
*-------------------------------------------------------------------
* Basic housekeeping to exit
*-------------------------------------------------------------------
         L     R13,SAVEMAIN+4      Restore ptr. to system's save area
         LM    R14,R12,12(R13)     Restore the system registers
         L     R15,SAVERC          Restore the return code
         BR    R14                 Return control to the system
         DROP  R2                  End parameter list addressability
*
*-------------------------------------------------------------------
*   Program storage areas
*-------------------------------------------------------------------
```

```
SAVEMAIN DS    18F                 Save area for the user program
RESIDSP  DS    CL8                 Resource id
USERPATH DS    H                   Path id for the user path
SAVERC   DS    F                   Save area for the return code
*
*-------------------------------------------------------------------
*   Program constants and equates
*-------------------------------------------------------------------
ZERO     DC    F'0'                Zero
*
*-------------------------------------------------------------------
*   SEVER Codes
*-------------------------------------------------------------------
DEALPROG DC    X'0210'             SEVER code for DEALLOCATE_ABEND_PROG
*
*-------------------------------------------------------------------
*   Return Codes
*-------------------------------------------------------------------
DCLBFAIL EQU   1                   DCLBFR failed
DCLBFRIO EQU   2                   I/O error on DCLBFR
CONFAIL  EQU   3                   Error on CONNECT to resource
SENDFAIL EQU   4                   Error on SEND of request
RECVFAIL EQU   5                   Error on RECEIVE of response
*
*-------------------------------------------------------------------
*   Storage for the IUCV parameter list
*-------------------------------------------------------------------
         DS    0D                  Force doubleword boundary
PLIST    DS    XL40                Parameter list for IUCV functions
INTBUF   DS    XL40                Buffer for DCLBFR function
*
*-------------------------------------------------------------------
*   State flag
*-------------------------------------------------------------------
STATE    DC    CL1'                State of the path indication
RESET    EQU   C'P'                RESET state
SEND     EQU   C'S'                SEND state
RECEIVE  EQU   C'R'                RECEIVE state
SEVER    EQU   C'V'                SEVER state
*
*-------------------------------------------------------------------
*   The SEND buffer for the file request to the resource consists of
*   the following:
*     - The first two bytes contain the length of the data being sent
*       (including the 2-byte length field).
*     - Following that is the file name and file type that we want
*       from the resource.
*-------------------------------------------------------------------
SENDBUF  DS    0H                  Buffer for SEND function
SENDLEN  DS    XL2                 SENDDATA data length
FILENAME DS    CL8                 File name
FILETYPE DS    CL8                 File type
SENDBLEN EQU   *-SENDBUF           Length of the SEND header and data
*
*-------------------------------------------------------------------
*   When receiving the response from the resource, the RECEIVE buffer
*   for the data sent consists of the following:
*     - The first two bytes contain the length of the data sent
*       (including the 2-byte length field).
*     - The next four bytes contain the return code.
*     - Following that is the response.
*-------------------------------------------------------------------
RECVBUF  DS    0CL88               Buffer for RECEIVE function
RECVLEN  DS    XL2                 Length of data sent
```

```
RECVRC   DS    XL4                 Return code from resource
RECVHLEN EQU   *-RECVBUF           Length of RECEIVE header
RECVDATA DS    XL80                The data portion of the data stream
RECVDLEN EQU   *-RECVDATA          Length of RECEIVE data area
*
*-------------------------------------------------------------------
*   DSECTS
*-------------------------------------------------------------------
         COPY  EQU                 Include the register equates
         COPY  IPARML              Include IUCV parameter list DSECT
         END   USER
```

Appendix D. Sample TSAF Resource Manager Program

This section contains a sample TSAF resource application. The program shows how to use the *IDENT System Service and APPC/VM. When reviewing this program, note the following:

- This sample demonstrates the IUCV and APPC/VM functions. It does not show how to use IUCV or APPC/VM efficiently or how to write a good server virtual machine application.

- Though this program could be used as a CMS application, it does not use the CMS IUCV support; rather it uses the CP IUCV and APPC/VM support directly. However, for simplicity, this program does use various CMS macros (for example, the WAITECB macro). Descriptions of these macros can be found in the *VM/SP CMS Macros and Functions Reference*.

- To highlight the IUCV and APPC/VM functions,, we have simplified the program logic in the following ways:

  - The program only accepts one user connection at any given time.

  - The response to most error conditions is to sever the corresponding path.

    It is a good practice to display the IPAUDIT or IPRCODE fields when an error occurs, so system support personnel can take appropriate action. In general, programs cannot correct errors that are reported in IPAUDIT or IPRCODE.

  - This program does not check all error conditions. For example, the program does not check to verify that you, the invoker, specified a resource id.

  - This program can only handle fixed length files with 80-byte records. The maximum number of records this program can read is 400.

This program runs disabled for IUCV interrupts. The PSW that we are given control with is enabled for external interrupts. However, the IUCV interrupt mask in control register 0 is not set. The IUCVWAIT routine sets the mask before entering a WAIT state. The interrupt routine disables our program again by turning off the IUCV mask in control register 0 after each interrupt has been received.

After assembling this program, you may invoke it by entering:

START RESOURCE *resid*

resid
is the resource id that manages the files.

This program can then identify itself to CP as the manager of the specified resource and wait to handle user requests for specific files.

```
         PRINT NOGEN               Don't expand macro calls
*-------------------------------------------------------------------
* Basic housekeeping
*-------------------------------------------------------------------
RESOURCE START X'20000'            Start program counter at X'20000'
         USING RESOURCE,R12        Establish base register 12
         STM   R14,R12,12(R13      Save system's registers
         ST    R13,SAVEMAIN+4      Save pointer to system's save area
         LA    R13,SAVEMAIN        R13 points to our save area
         LA    R2,PLIST            Get address of APPC/VM parm list
         USING IPARML,R2           Establish parameter list mapping
         LA    R3,INTBUF           Get address of the interrupt buffer
*
*-------------------------------------------------------------------
*   When invoking this program, you must specify a resource id.  We get
*   the resource id out of the parameter list.  CMS passes the address
*   of the parameter list to us in register 1.  You can find more
*   information about this interface in the VM/SP CMS Command Reference
*   under the START command.
*-------------------------------------------------------------------
         MVC   RESID,8(R1)         Get the resource id
*
*-------------------------------------------------------------------
*   Before using IUCV or APPC/VM, we must define an interrupt buffer
*   for our program to use, using the DCLBFR function.  The interrupt
*   buffer that we use is at label INTBUF.
*-------------------------------------------------------------------
         XC    PLIST,PLIST         Start with a clean parameter list
         IUCV  DCLBFR,             DCLBFR function                    $
               PRMLIST=PLIST,      Use PLIST as the parameter list    $
               BUFFER=INTBUF,      Use INTBUF as the interrupt buffer $
               CONTROL=NO          Declare INTBUF an application buffer
         BC    CC1,DCLERR          Go to DCLERR, if DCLBFR failed
         BC    CC3,DIRERROR        Go to DIRERROR, if a directory error
*
*-------------------------------------------------------------------
*   After establishing our virtual machine as an IUCV user, we try to
*   CONNECT to the *IDENT System Service and identify our virtual
*   machine as the manager of the specified resource.  This connection
*   is non-APPC, because we cannot use APPC/VM to connect to CP
*   system services.  The VM/SP System Reference for Programmers
*   contains a description of the non-APPC IUCV CONNECT.
*-------------------------------------------------------------------
         XC    PLIST,PLIST         Start with a clean parameter list
         MVC   IDRESID,RESID       Move resource id into user data
         IUCV  CONNECT,            CONNECT function                   $
               PRMLIST=PLIST,      Use PLIST as the parameter list    $
               USERID=IDENT,       *IDENT is the target userid        $
               USERDTA=IDDATA      User data for identify
         BC    CC1,CONERROR        Go to CONERROR, if error
*
*-------------------------------------------------------------------
*   We have successfully started the connection to *IDENT.  IUCV
*   returns the path id for our half of the path in the parameter list.
```

```
*   We keep track of this path over which we are connected to *IDENT.
*-------------------------------------------------------------------
         MVC   IDENTPTH,IPPATHID   Save the path to *IDENT
*
*-------------------------------------------------------------------
*   We are interested in the following interrupts:
*      - Connection complete interrupts - These indicates that
*        *IDENT accepted our connection and we now manage the resource
*        that we specified.
*      - SEVER interrupts - These indicates that *IDENT rejected
*        our connection and that we do not manage the specified
*        resource.
*
*   We issue a SETMASK to disable all noncontrol interrupts.
*   We also must issue a SETCMASK to specifically disable all control
*   interrupts except connection complete and SEVER.
*-------------------------------------------------------------------
         XC    PLIST,PLIST         Start with a clean parameter list
         LA    R1,CNTRLMSK         Load the enable mask
         IUCV  SETMASK,            SETMASK function                  $
               PRMLIST=PLIST,      PLIST is the parameter list       $
               MASK=(R1)           The mask is in R1
         XC    PLIST,PLIST         Start with a clean parameter list
         LA    R1,CCOMPMSK+SEVERMSK Load the enable mask
         IUCV  SETCMASK,           SETCMASK function                 $
               PRMLIST=PLIST,      PLIST is the parameter list       $
               MASK=(R1)           The mask is in R1
*
*-------------------------------------------------------------------
*   Subroutine IUCVWAIT enables for IUCV interrupts and then waits
*   until an IUCV interrupt occurs.
*-------------------------------------------------------------------
         BAL   R9,IUCVWAIT         Go to IUCVWAIT, wait for interrupt
*
*-------------------------------------------------------------------
*   If the path is severed now it means that we never became the
*   resource manager.  In this case, we issue RTRVBFR and exit.  We do
*   not need to do a SEVER, because RTRVBFR cleans up the path.
*-------------------------------------------------------------------
         DROP  R2                  End parameter list addressability
         USING IPARML,R3           Interrupt buffer addressability
         CLI   IPTYPE,IPTYPSV      Was the identify rejected via SEVER?
         BE    IDNTERR             If so, go to IDNTERR
         DROP  R3                  Done with interrupt buffer for now
         USING IPARML,R2           Back to the parameter list
*
*-------------------------------------------------------------------
*   At this point, we have established ourselves as the resource
*   manager.
*-------------------------------------------------------------------
*
*
*-------------------------------------------------------------------
*   Now we wait for either a connection pending to indicate that
*   a user has work for us or a SEVER to indicate that we no longer
*   manage the resource.  We issue SETMASK to enable for control
*   interrupts only and SETCMASK to enable for connection pending and
*   SEVER interrupts.
*
*   NOTE:  The SEVER interrupt tells us that we no longer manage the
*   resource, and we will not be getting any connection routed to us.
*   Since we only handle one user at a time and currently do not have
*   anyone connected to us, we would exit.  However, if we did have
*   a user connected to us, we could continue processing even after
```

```
*   we lose management responsibility of a particular resource.
*-------------------------------------------------------------------
NEXTUSER DS    0H
         XC    PLIST,PLIST         Start with a clean parameter list
         LA    R1,CNTRLMSK         Load the enable mask
         IUCV  SETMASK,            SETMASK function                  $
               PRMLIST=PLIST,      PLIST is the parameter list       $
               MASK=(R1)           The mask is in R1
         XC    PLIST,PLIST         Start with a clean parameter list
         LA    R1,CPENDMSK+SEVERMSK Load the enable mask
         IUCV  SETCMASK,           SETCMASK function                 $
               PRMLIST=PLIST,      PLIST is the parameter list       $
               MASK=(R1)           The mask is in R1
         BAL   R9,IUCVWAIT         Go to IUCVWAIT, wait for interrupt
         DROP  R2                  End parameter list addressability
         USING IPARML,R3           Interrupt buffer addressability
         CLI   IPTYPE,IPTYPSV      Do we still manage the resource?
         BE    IDREVOKE            If not, do a RTRVBFR and exit
         DROP  R3                  Done with interrupt buffer for now
         USING IPARML,R2           Back to the parameter list
*
*-------------------------------------------------------------------
*   A new user has connected to us.  This program lets anyone
*   authorized in the CP directory connect to it.  However, if the
*   program was concerned with security here is where we would check
*   the user's userid.
*-------------------------------------------------------------------
         DROP  R2                  End parameter list addressability
         USING IPARML,R3           Establish interrupt buffer address
         MVC   USERPATH,IPPATHID   Save the path id of the user path
         DROP  R3                  End interrupt buffer addressability
*
*-------------------------------------------------------------------
*   We accept the connection.
*-------------------------------------------------------------------
         USING IPARML,R2           Establish parameter list address
         XC    PLIST,PLIST         Start with a clean parameter list
         IUCV  ACCEPT,             ACCEPT function                   $
               PRMLIST=PLIST,      Use PLIST as the parameter list   $
               PATHID=USERPATH     Accept the user path
         BC    CC2,ACCEPTOK        Go to ACCEPTOK, if successful
*
*-------------------------------------------------------------------
*   The ACCEPT failed.  The user who connected must have severed the
*   path or logged off.  We sever our half of the path and wait
*   for someone else to connect to us.  We use IUCV SEVER since we
*   are in CONNECT state and APPCVM SEVER is not valid from this
*   state.
*-------------------------------------------------------------------
         XC    PLIST,PLIST         Start with a clean parameter list
         IUCV  SEVER,              Non-IUCV SEVER                    $
               PRMLIST=PLIST,      Use PLIST as the parameter list   $
               PATHID=USERPATH     Sever the user's path
         BC    CC1,SEVERR          Indicate SEVER error
         B     NEXTUSER            Go get the next user
*
*-------------------------------------------------------------------
*   The ACCEPT was successful.  We now issue a RECEIVE to define a
*   buffer to get the user's request.
*-------------------------------------------------------------------
ACCEPTOK DS    0H
         XC    PLIST,PLIST         Start with a clean parameter list
         APPCVM RECEIVE,           RECEIVE function                  $
```

```
                PRMLIST=PLIST,      Use PLIST as the parameter list     $
                PATHID=USERPATH,    RECEIVE data on user path           $
                BUFFER=RECVBUF,     Put the data into the RECEIVE buffer$
                BUFLEN=RECVBUFL,    Length of the RECEIVE buffer        $
                WAIT=NO             Return after RECEIVE is started
          BC    CC1,RECVERR         Go to RECVERR, if there is an error
          BC    CC2,RECVCOMP        Go to RECVCOMP, if receive completed
*
*--------------------------------------------------------------------
*   When the RECEIVE does not complete immediately, we must wait for
*   it to complete.  We could enable for IUCV function complete
*   interrupts and wait for an interrupt.  However, so we can show the
*   the use of more IUCV functions in this program, we use the
*   TESTMSG and TESTCMPL functions.
*
*   TESTMSG puts the virtual machine in a WAIT state until a message
*   become pending (CC=1), a SENDREQ indication is posted (CC=1), or a
*   function completes (CC=2).  If both function completes and message
*   pendings or SENDREQs are posted, TESTMSG completes with CC=3.
*
*   For CC=1, we issue a DESCRIBE to drain the message pending or
*   SENDREQ interrupt.  For CC=2 or 3, we issue a TESTCMPL to complete
*   the function and get the function complete data.
*--------------------------------------------------------------------
TESTMSG   DS    0H
          IUCV TESTMSG              Wait for the function to complete
          BC    CC2+CC3,TESTCMPL    Function completed, issue TESTCMPL
          XC    PLIST,PLIST         Start with a clean parameter list
          IUCV DESCRIBE,            DESCRIBE function                   $
                PRMLIST=PLIST       Use PLIST as the parameter list
          B     TESTMSG             Wait for what we want
TESTCMPL  DS    0H
          XC    PLIST,PLIST         Start with a clean parameter list
          IUCV TESTCMPL,            TESTCMPL function                   $
                PRMLIST=PLIST       Use PLIST as the parameter list
RECVCOMP  DS    0H
          LA    R4,PLIST            Function complete data in PLIST
          BAL   R9,CHKFCOMP         Check for SEVER, etc.
          BNZ   NEXTUSER            Go to NEXTUSER, if path was severed
*
*--------------------------------------------------------------------
*   The data has been received.  Read the file on behalf of the user.
*   See the VM/SP Macros and Functions Reference for descriptions of
*   the FSOPEN, FSREAD and FSCLOSE macros.
*--------------------------------------------------------------------
DATARCVD  DS    0H
          MVC   RECVDATA+16(2),A1   Add file mode to input
          LA    R4,RECVDATA         Address of file name and file type
          LA    R5,SENDDATA         Get address I/O buffer
          FSOPEN (R4),FSCB=FILEFSCB,ERROR=READERR
          LA    R6,400              Read maximum of 400 records
READLOOP  DS    0H
          XC    0(80,R5),0(R5)      Clear the next 80 bytes
          FSREAD (R4),FSCB=FILEFSCB,ERROR=CHECKEOF,BUFFER=(R5)
          LA    R5,80(,R5)          Bump to next record position
          BCT   R6,READLOOP         Read the next record
          LH    R15,EOF             Pretend it was end of file
CHECKEOF  DS    0H
          LR    R6,R15              Save the return code from FSREAD
          FSCLOSE (R4),FSCB=FILEFSCB
          LR    R15,R6              Restore the FSREAD return code
          CH    R15,EOF             Was this the end of the file?
          BNE   READERR             If not, go to READERR
```

```
         SR    R15,R15             Zero the return code
*
*--------------------------------------------------------------------
*   We send the file back to the user in the following format:
*     - The first 2 bytes contain the length of the variable
*       (including the 2 byte length field).
*     - The next 4 bytes contain the return code from the FSREAD.
*     - Following these are the file contents.
*
*   Note that the data is sent as a single SEND.  The sample user
*   program reads it in 80 byte records.  Neither end of the
*   conversation needs to know what the sizes are of the SENDs and
*   RECEIVEs being issued.  Also note that we specify the RECEIVE=YES
*   to put the user back in SEND state and allocate a buffer for the
*   next request.  The sample user program does not make a second
*   request during this connection, but this is irrelevant to our
*   program.
*--------------------------------------------------------------------
READERR  DS    0H
         LA    R4,SENDBUF          Get address of start of SEND area
         SR    R5,R4               Get length of SEND
         STH   R5,SENDLEN          Length of data plus header
         STCM  R15,B'1111',SENDRC  Save the return code
         LA    R4,PLIST            Function complete data in PLIST
         XC    PLIST,PLIST         Start with a clean parameter list
         APPCVM SENDDATA,          SENDDATA function                  $
               PRMLIST=PLIST,      Use PLIST as the parameter list    $
               PATHID=USERPATH,    Send the data on the user path     $
               BUFFER=SENDBUF,     Take the data out of SENDBUF       $
               BUFLEN=(R5),        The length of the data to send     $
               RECEIVE=YES,        Switch to RECEIVE state            $
               ANSBUF=RECVBUF,     Next request into RECEIVE buffer   $
               ANSLEN=RECVBUFL,    Length of the RECEIVE buffer       $
               WAIT=NO             Return after the SEND is started
         BC    CC1,SENDERR         Go to SENDERR, if there is an error
         BC    CC2,SENDCOMP        Go to SENDCOMP, SENDDATA completed
*
*--------------------------------------------------------------------
*   If the SENDDATA doesn't complete immediately, we enable for IUCV
*   function complete interrupts and wait for the interrupt.  R4 is
*   set to point to the interrupt buffer for CHKFCOMP.
*--------------------------------------------------------------------
         SPACE
         XC    PLIST,PLIST         Start with a clean parameter list
         LA    R1,FCOMPMSK         Enable for function complete
         IUCV  SETMASK,            SETMASK function                   $
               PRMLIST=PLIST,      PLIST is the parameter list        $
               MASK=(R1)           The mask is in R1
         BAL   R9,IUCVWAIT         Wait for the IUCV interrupt
         LA    R4,INTBUF           Function complete data in INTBUF
SENDCOMP DS    0H
         BAL   R9,CHKFCOMP         Check for SEVER, etc.
         BNZ   NEXTUSER            Go to NEXTUSER, if path was severed
*
*--------------------------------------------------------------------
*   The function is complete, and we receive a second request.  Go to
*   DATARCVD to handle it.
*--------------------------------------------------------------------
         B     DATARCVD            Go to DATARCVD, handle the request
*
*
*--------------------------------------------------------------------
*   The following sets of instructions cover the possible errors that
```

```
*   could occur while running this program.  These routines set the
*   appropriate return codes, sever the path, and exit the program.
*---------------------------------------------------------------------
*
*
*---------------------------------------------------------------------
*   There was an error on the DCLBFR.  The only IPRCODE defined for
*   DCLBFR is 19 which means that an IUCV interrupt buffer has already
*   been defined by our virtual machine.  Because the buffer was not
*   declared by our program, we should not do a RTRVBFR.
*---------------------------------------------------------------------
DCLERR   DS    0H
         LA    R15,DCLFAIL         Indicate the DCLBFR failed
         B     EXIT                Exit
*
*---------------------------------------------------------------------
*   Condition code 3 from DCLBFR means that CP encountered an I/O
*   error while reading the directory entry for this virtual machine.
*   Since the buffer was never declared, we cannot SEVER or do a
*   RTRVBFR.
*---------------------------------------------------------------------
DIRERROR DS    0H
         LA    R15,DCLBFRIO        Indicate an I/O error on DCLBFR
         B     EXIT                Exit
*
*---------------------------------------------------------------------
*   There was an error on the CONNECT.  We do not have a path and,
*   therefore, cannot invoke SEVER.  We must do the RTRVBFR.
*---------------------------------------------------------------------
CONERROR DS    0H
         LA    R15,CONFAIL         Indicate error on CONNECT to *IDENT
         B     RTREXIT             Do a RTRVBFR and exit
*
*---------------------------------------------------------------------
*   An error occurred on the Identify.  We do a RTRVBFR and exit.
*---------------------------------------------------------------------
IDNTERR  DS    0H
         LA    R15,IDNTFAIL        Indicate error on Identify
         B     RTREXIT             Do a RTRVBFR and exit
*
*---------------------------------------------------------------------
*   We no longer manage the resource.  We do a RTRVBFR and exit.
*---------------------------------------------------------------------
IDREVOKE DS    0H
         LA    R15,REVOKE          Indicate ownership revoked
         B     RTREXIT             Do a RTRVBFR and exit
*
*---------------------------------------------------------------------
*   An error occurred on SEVER.  We do a RTRVBFR and exit.
*---------------------------------------------------------------------
SEVERR   DS    0H
         LA    R15,SEVFAIL         Indicate error on SEVER
         B     RTREXIT             Do a RTRVBFR and exit
*
*---------------------------------------------------------------------
*   An error occurred on RECEIVE.  We do a SEVER, then a RTRVBFR.
*---------------------------------------------------------------------
RECVERR  DS    0H
         LA    R15,RECFAIL         Indicate error on RECEIVE
         B     ERRSEVER            SEVER and exit
*
*---------------------------------------------------------------------
*   An error occurred on SENDDATA.  We do a SEVER, then a RTRVBFR.
```

```
*-------------------------------------------------------------------
SENDERR  DS    0H
         LA    R15,SENDFAIL        Indicate error on RECEIVE
*
*-------------------------------------------------------------------
*   SEVER the user path abnormally and exit.
*-------------------------------------------------------------------
ERRSEVER DS    0H
         XC    PLIST,PLIST         Clear the parameter list
         APPCVM SEVER,             SEVER function                  $
               PRMLIST=PLIST,      Use PLIST as the parameter list $
               PATHID=USERPATH,    Sever the user's path           $
               TYPE=ABEND,         SEVER type is abend             $
               CODE=DEALPROG       SEVER code for DEALLOCATEPROG
*
*-------------------------------------------------------------------
*   EXITS:
*   We use this exit to leave the program any time after we have
*   successfully issued the DCLBFR.
*-------------------------------------------------------------------
RTREXIT  DS    0H
         IUCV RTRVBFR              Do the RTRVBFR
EXIT     DS    0H
         ST    R15,SAVERC          Save the return code
*
*-------------------------------------------------------------------
* Basic housekeeping to leave
*-------------------------------------------------------------------
         L     R13,SAVEMAIN+4      Restore pointer to system's savearea
         LM    R14,R12,12(R13)     Restore the system's registers
         L     R15,SAVERC          Restore the return code
         BR    R14                 Return control to the system
*
*-------------------------------------------------------------------
*   The following subroutine checks the function complete data.
*   On input, R4 should point to the function complete data.  On
*   output, if function complete data is:
*
*      - OK, the condition code is 0.
*      - Not OK, we sever the user path and set condition code to
*        non-zero.
*-------------------------------------------------------------------
         USING IPARML,R4           Parameter list addressability
CHKFCOMP DS    0H
*
*-------------------------------------------------------------------
*   Check for IPSEND.  If IPSEND is present, then the user has turned
*   the conversation around.  This marks a successful completion of all
*   SENDDATAs and RECEIVEs performed by this program.
*-------------------------------------------------------------------
CHKSEND  DS    0H
         CLI   IPWHATRC,IPSEND     Did user turn around conversation?
         BER   R9                  If so, return
*
*-------------------------------------------------------------------
*   Any indication other than the user turned around the conversation
*   is abnormal, and we sever the path.  If the user issued
*   SEVER, we sever the path with SEVER TYPE=NORMAL.  If the user
*   did not issue SEVER, we sever the path with SEVER TYPE=ABEND.
*-------------------------------------------------------------------
         CLI   IPWHATRC,IPSNORM    Did the user sever normally?
         BE    NRMSEVER            If so, sever the path normally
         CLI   IPWHATRC,IPSABEND   Did the user sever abnormally?
```

```
         BE    NRMSEVER            If so, sever the path normally
         DROP  R4                  Done with function complete data
         USING IPARML,R2           Use the parameter list
         XC    PLIST,PLIST         Start with a clean parameter list
         APPCVM SEVER,             User didn't sever, but we do        $
               PRMLIST=PLIST,      Use PLIST as the parameter list     $
               PATHID=USERPATH,    Sever the user's path               $
               TYPE=ABEND,         SEVER type is ABEND                 $
               CODE=DEALPROG       SEVER code is DEALLOCATE PROG
         BC    CC1,SEVERR          Go to SEVERR, if the SEVER fails
         CLI   *,X'FF'             Set non-zero condition code
         BR    R9                  Return
NRMSEVER DS    0H
         XC    PLIST,PLIST         Start with a clean parameter list
         APPCVM SEVER,             User severed, we sever normally     $
               PRMLIST=PLIST,      Use PLIST as the parameter list     $
               PATHID=USERPATH,    Sever the user's path               $
               TYPE=NORMAL         SEVER type is NORMAL
         BC    CC1,SEVERR          Go to SEVERR, if the SEVER fails
         CLI   *,X'FF'             Set non-zero condition code
         BR    R9                  Return
         DROP  R2                  End parameter list addressability
*
*-------------------------------------------------------------------
*   The following routine enables for IUCV interrupts, and then waits
*   for any IUCV interrupt.  So our program can specify an address to
*   gain control when an external interrupt occurs, we use the
*   HNDEXT CMS macro.  We can set bit 30 of control register 0,
*   which is the IUCV interrupt mask, to enable for IUCV external
*   interrupts.
*
*   WAITECB is a CMS macro that gives control to CMS until the
*   corresponding ECB is posted.  Our interrupt routine (at label
*   EXTINT) posts the ECB when an IUCV interrupt is reflected to
*   the virtual machine.
*   NOTE: Both the HNDEXT and WAITECB MACROS are explained in the
*   VM/SP Macros and Functions Reference.
*-------------------------------------------------------------------
IUCVWAIT DS    0H
         MVI   ECBIUCV,X'00'       Clear previous ECB flags
         HNDEXT SET,EXTINT         Handle external interrupts
         STCTL C0,C0,C0SAVE        Get control register 0
         OI    C0SAVE+3,IUCVENAB   Enable for IUCV interrupts
         LCTL  C0,C0,C0SAVE        Put new value back
         WAITECB ECB=ECBIUCV       Wait for the interrupt
         BR    R9                  Return
*
*-------------------------------------------------------------------
*   The following routine is the external interrupt handler for IUCV
*   interrupts.  Since we enabled for particular types of
*   interrupts, we only need to verify that the interrupt is IUCV.
*   However, in this program we check that the interrupt type
*   received is the type that we want.  For example, when we're
*   enabled for connection pending interrupts we may get both APPC and
*   non-APPC connection pendings, even if we're only interested in the
*   APPC type.
*-------------------------------------------------------------------
         DROP  R12                 Drop the base register for awhile
         USING EXTINT,R15          Temporary addressability
EXTINT   DS    0H
         STM   R14,R12,12(R13)     Save system's registers
         ST    R13,SAVEEXT+4       Save pointer to system's save area
         LA    R13,SAVEEXT         R13 points to our save area
         LR    R12,R15             Restore the base register
```

```
         SL    R12,=A(EXTINT-RESOURCE) ...
         DROP  R15                 Get back to base register 12
         USING RESOURCE,R12        Establish base register 12
         CLC   X'62'(2,R1),IUCVTYPE Is this an IUCV interrupt?
         BNE   EXTEXIT             If not, ignore the interrupt
         LA    R3,INTBUF           Get address of the interrupt buffer
         USING IPARML,R3           Interrupt buffer addressability
*
*--------------------------------------------------------------------
*   The following routine checks for connection pending interrupts.
*   Our program gets a connection pending interrupt when a user
*   issues an APPCVM CONNECT to the resource that our program
*   manages.  This program only checks for APPC connection pending
*   interrupts, not non-APPC connection pending interrupts.
*--------------------------------------------------------------------
         CLI   IPTYPE,IPTYPPCA     Is this an APPC connection pending?
         BE    EXTPOST             If so, post the ECB
*
*--------------------------------------------------------------------
*   This routine checks for connection complete interrupts.  Our
*   program gets connection complete interrupts when our
*   communication partner accepts our CONNECT.  This program only
*   initiates one CONNECT, the connection to *IDENT.
*--------------------------------------------------------------------
         CLI   IPTYPE,IPTYPCC      Is this a non-APPC connection comp?
         BE    EXTPOST             If so, post the ECB
*
*--------------------------------------------------------------------
*   This routine checks for a SEVER interrupt.
*--------------------------------------------------------------------
         CLI   IPTYPE,IPTYPSVA     Is this an APPC SEVER?
         BE    EXTEXIT             If so, wait for function complete
         CLI   IPTYPE,IPTYPSV      Is this a non-APPC SEVER?
         BE    EXTPOST             If so, post the ECB
*--------------------------------------------------------------------
*   This routine checks for a function complete interrupt.  Our
*   program gets a function complete interrupt when a function
*   completes.  In our case, the function is SENDDATA RECEIVE=YES or
*   RECEIVE.
*--------------------------------------------------------------------
         CLI   IPTYPE,IPTYPFCA     Is this a function completion?
         BE    EXTPOST             If so, post the ECB
*--------------------------------------------------------------------
*   This routine takes care of when we get an IUCV interrupt type that
*   we don't want (for example, a non-APPC connection pending).  We
*   SEVER the path shown in the interrupt buffer, using a non-APPC
*   SEVER since it works on either type of path.  If the path is
*   non-APPC, our communication partner gets a SEVER interrupt with
*   user data set to zero.  If the path is APPC, our partner gets
*   a SEVER indication with TYPE=ABEND and SEVER CODE=X'0610'.
*--------------------------------------------------------------------
         LH    R0,IPPATHID         Save the pathid
         DROP  R3                  Done with interrupt buffer for now
         USING IPARML,R2           Establish parameter list mapping
         XC    PLIST,PLIST         Zero the parameter list
         IUCV  SEVER,              Non-APPC SEVER function            $
               PRMLIST=PLIST,      Use PLIST as the parameter list    $
               PATHID=(R0)         Use the interrupt's path
         BC    CC1,SEVERR          Go to SEVERR, if the SEVER fails
         B     EXTEXIT             Otherwise, exit
         DROP  R2                  Done with parameter list for now
         USING IPARML,R3           Back to interrupt buffer mapping
*
```

```
*---------------------------------------------------------------------
*   We post the ECB so CMS returns control to the main program.
*   We also disable for IUCV interrupt by resetting bit 30 in control
*   register 0.  We issue the HNDEXT macro to give control over
*   external interrupts back to CMS.
*---------------------------------------------------------------------
EXTPOST  DS    0H
         MVI   ECBIUCV,EVENTCMP    Indicate IUCV interrupt received
         STCTL C0,C0,C0SAVE        Get control register 0
         NI    C0SAVE+3,X'FF'-IUCVENAB Disable for IUCV interrupts
         LCTL C0,C0,C0SAVE         Put new value back
         HNDEXT CLR                Give interrupts back to CMS
EXTEXIT  DS    0H
         LA    R2,PLIST            Use the PLIST
         L     R13,SAVEEXT+4       Restore R13 pointer
         LM    R14,R12,12(R13)     Restore the system's registers
         BR    R14                 Return to CMS
         DROP  R3                  Done with interrupt buffer for now
*
*---------------------------------------------------------------------
*   Program storage areas
*---------------------------------------------------------------------
SAVEMAIN DS    18F            Save area for the resource program
SAVEEXT  DS    18F            Save area for external interrupt handler
RESID    DS    CL8            Resource id
C0SAVE   DS    F              Save area for control register 0
IDENTPTH DS    H              Path id for *IDENT connection
USERPATH DS    H              Path id for user path
SAVERC   DS    F              Save area for the return code
*
*---------------------------------------------------------------------
*   Program constants and equates
*---------------------------------------------------------------------
IDENT    DC    CL8'*IDENT'    Userid of *IDENT System Service
ZERO     DC    F'0'           Zero
IUCVTYPE DC    X'4000'        External interrupt code for IUCV
RECVBUFL DC    A(RECVBLEN)    Length of RECEIVE buffer
SENDBUFL DC    A(SENDBLEN)    Length of SEND buffer
EOF      DC    H'12'          End of file indicator from FSREAD
A1       DC    C'A1'          File mode
IUCVENAB EQU   X'02'          CR 0 bit to enable IUCV interrupts
*
*---------------------------------------------------------------------
*   SEVER Codes
*---------------------------------------------------------------------
DEALPROG DC    X'0210'        SEVER code for DEALLOCATE_ABEND_PROG
*
*---------------------------------------------------------------------
*   Return Codes
*---------------------------------------------------------------------
DCLFAIL  EQU   1              DCLBFR failed
DCLBFRIO EQU   2              I/O error on DCLBFR
CONFAIL  EQU   3              Error on CONNECT to *IDENT
IDNTFAIL EQU   4              Connection to *IDENT was rejected
REVOKE   EQU   5              Connection to *IDENT was severed
SEVFAIL  EQU   6              SEVER failed
RECFAIL  EQU   7              RECEIVE on user path failed
SENDFAIL EQU   8              SENDDATA on user path failed
*
*---------------------------------------------------------------------
*   SETMASK mask values
*---------------------------------------------------------------------
FCOMPMSK EQU   X'20'          Enable for function complete interrupts
CNTRLMSK EQU   X'08'          Enable for control interrupts
```

```
*
*---------------------------------------------------------------------
*   SETCMASK MASK VALUES
*---------------------------------------------------------------------
CPENDMSK EQU   X'80'           Enable for connect pending interrupts
CCOMPMSK EQU   X'40'           Enable for connect complete interrupts
SEVERMSK EQU   X'20'           Enable for SEVER interrupts
*
*---------------------------------------------------------------------
*   User data for the CONNECT to *IDENT
*---------------------------------------------------------------------
IDDATA   DS    0CL16           User data for CONNECT to *IDENT
IDRESID  DC    CL8' '          Resource id for *IDENT
         DC    X'01'           Indicate an identify request
         DC    X'80'           Declare a global resource
         DC    XL6'00'         Unused
*
*---------------------------------------------------------------------
*   Storage for the IUCV parameter list and interrupt buffer
*---------------------------------------------------------------------
         DS    0D              Force doubleword boundary
PLIST    DS    XL40            Parameter list for IUCV functions
INTBUF   DS    XL40            Buffer of IUCV interrupt data
*
*---------------------------------------------------------------------
*   This set of instructions is the ECB used to wait for IUCV external
*   interrupts.
*---------------------------------------------------------------------
ECBIUCV  DC    F'0'            ECB for WAIT function
EVENTCMP EQU   X'40'           Event complete ECB flag
*
*---------------------------------------------------------------------
*   FSCB for the FSOPEN, FSREAD and FSCLOSE macros.  The FSCB macro
*   is described in the VM/SP Macros and Functions Reference.
*---------------------------------------------------------------------
FILEFSCB FSCB  'FILENAME FILETYPE FM',RECFM=F
*
*---------------------------------------------------------------------
*   Force the literals to come out here.  Because the SEND buffer is so
*   large, we don't have addressability to the end of it.
*---------------------------------------------------------------------
         LTORG
*
*---------------------------------------------------------------------
*   The following is the RECEIVE buffer for the data sent from the
*   user.  When receiving the file, we receive the following:
*     - The first two bytes contain the length of the data sent
*       (including the 2-byte length field).
*     - The next bytes are the file name (8 bytes) and the file type
*       (8 bytes).
*---------------------------------------------------------------------
RECVBUF  DS    0CL244          Buffer for RECEIVE function
RECVLEN  DS    XL2             Length of data sent
RECVDATA DS    XL240           The data portion of the data stream
RECVBLEN EQU   *-RECVBUF       Length of RECEIVE header and data
*
*---------------------------------------------------------------------
*   The following is the SEND buffer for the response back to the user.
*     - The first two bytes contain the length of the data being sent
*       (including the 2-byte length field).
*     - The next four bytes contain the return code.
*     - Following that are the file contents.  The maximum size of
*       the response is 400, 80-byte records. (This number has no
```

```
*       significance).
*-------------------------------------------------------------------
SENDBUF  DS    0XL32008       Buffer for SEND function
SENDLEN  DS    XL2            SEND data length
SENDRC   DS    XL4            Return code
SENDDATA DS    XL32000        The data being sent
SENDBLEN EQU   *-SENDBUF      Length of SEND header and SEND data
*
*-------------------------------------------------------------------
*   DSECTS
*-------------------------------------------------------------------
         COPY  EQU            Include the register equates
         COPY  IPARML         Include the IUCV parameter list DSECT
         END   RESOURCE
```

Glossary of Terms and Abbreviations

This section explains or defines the terms, acronyms, and abbreviations that appear in this manual. For a complete list of terms used in VM/SP refer to the *VM/SP Library Guide, Glossary, and Master Index*. You may also want to refer to the *IBM Vocabulary for Data Processing, Telecommunications, and Office Systems*, GC20-1699.

***IDENT.** The Identify System Service. This CP system service allows authorized virtual machines to connect to it and identify themselves as resource managers.

***CRM.** The Collection Resource Management System Service. This CP system service allows an authorized virtual machine to connect to it and become the TSAF virtual machine.

A

accept. The act of allowing a connection to your virtual machine from another virtual machine or from your own virtual machine.

Advanced Program-to-Program Communication (APPC). The inter-program communication service within Systems Network Architecture LU 6.2 (SNA LU 6.2) on which the APPC/VM interface is based.

Advanced Program-to-Program Communication/VM (APPC/VM). An application program interface (API) for communicating between two virtual machines that is mappable to the SNA LU 6.2 APPC interface and is based on IUCV functions. Along with the TSAF virtual machine, APPC/VM provides this communication within a single system and throughout a collection of systems.

API. Application Program Interface.

APPC. Advanced Program-to-Program Communication.

APPC/VM. Advanced Program-to-Program Communication/VM (APPC/VM)

asynchronous communication. Communication when you have specified WAIT = NO on the particular APPC/VM function. Communicating asynchronously means that you can issue APPC/VM functions to other paths, while waiting for the asynchronous function to complete.

B

BSC. Binary synchronous communication.

C

collection. A group of VM/SP systems that can share resources. Each system within the collection must have the TSAF virtual machine installed and running.

communication. See asynchronous communication and/or synchronous communication.

communication partner. The virtual machine on the other end of the local APPC/VM path, not necessarily the target of the communication.

communication server. A server, such as the TSAF virtual machine, that provides communications between systems.

connect. The act of establishing a path to communicate with another virtual machine or with your own virtual machine.

conversation state. A state that is associated with an APPC/VM program at each end of an APPC/VM path to define which communicator can issue which functions at any given time. The possible states are: CONFIRM, CONNECT, RECEIVE, RESET, SEND, SEVER.

CTC. Channel-to-channel (links).

D

data stream. A set of logical records sent one after the other.

dynamic configuration. The act of configuring a collection, or reconfiguring a collection when a system enters or leaves the collection after a link goes down within the collection.

G

GCS. Group Control System component of VM/SP.

I

interrupt. In APPC/VM, a way in which you may receive notification of pending functions.

IPCS. Interactive Problem Control System component of VM/SP.

L

link. In TSAF, the physical connection between two systems.

LL. Logical record length.

logical record. A formatted record that consists of a 2-byte logical record length field and a data field of variable length.

LU. Logical unit.

M

message. The data sent by a single APPC/VM SENDDATA function.

P

path. In TSAF, any connection between two virtual machines either on the same or different systems.

R

receive. The act of bringing into the specified buffer that data sent to your virtual machine from another virtual machine or from your own virtual machine.

record. See logical record.

resource. A program, a data file, a specific set of files, a device or any other entity or set of entities that you might want to uniquely identify for purposes of application program processing in a VM system. A resource can be identified by up to eight characters.

resource id. A one-to-eight character name used to identify a resource.

resource manager. A program or set of programs executing in a virtual machine and managing access to one or more VM resources; also called a server.

route. In TSAF, a number of links and possible intermediate systems that allow the connection of one system to another.

S

server. A program or set of programs executing in a virtual machine and managing access to one or more VM resources; also called a resource manager.

sever. The act of ending communication with another virtual machine or with your own virtual machine.

SNA. Systems Network Architecture.

state. See conversation state.

synchronous communication. Communication when you have specified WAIT = YES on the particular APPC/VM function. Communicating synchronously means that your virtual machine is put in a WAIT state, and you cannot issue any

APPC/VM functions to any paths until the function completes.

T

TSAF. Transparent Services Access Facility.

Transparent Services Access Facility. A facility that lets users connect to and communicate with local or remote virtual machines within a collection of systems. With TSAF, a user can connect to a program by specifying a name that the program has made known, instead of specifying a userid and nodeid.

TSAF virtual machine component. A component within VM/SP that handles communication between systems by letting APPC/VM paths span more than one system.

Bibliography

For general VM/SP reference:

VM/SP Planning Guide and Reference, SC19-6201

VM/SP Operator's Guide, SC19-6202

VM/SP CP Command Reference, SC19-6211

VM/SP Installation Guide, SC24-5237

VM/SP CMS Macros and Functions Reference, SC24-5284

VM Diagnosis Guide, LY24-5241

For IUCV programming and reference:

VM/SP Group Control System Command and Macro Reference, SC24-5250

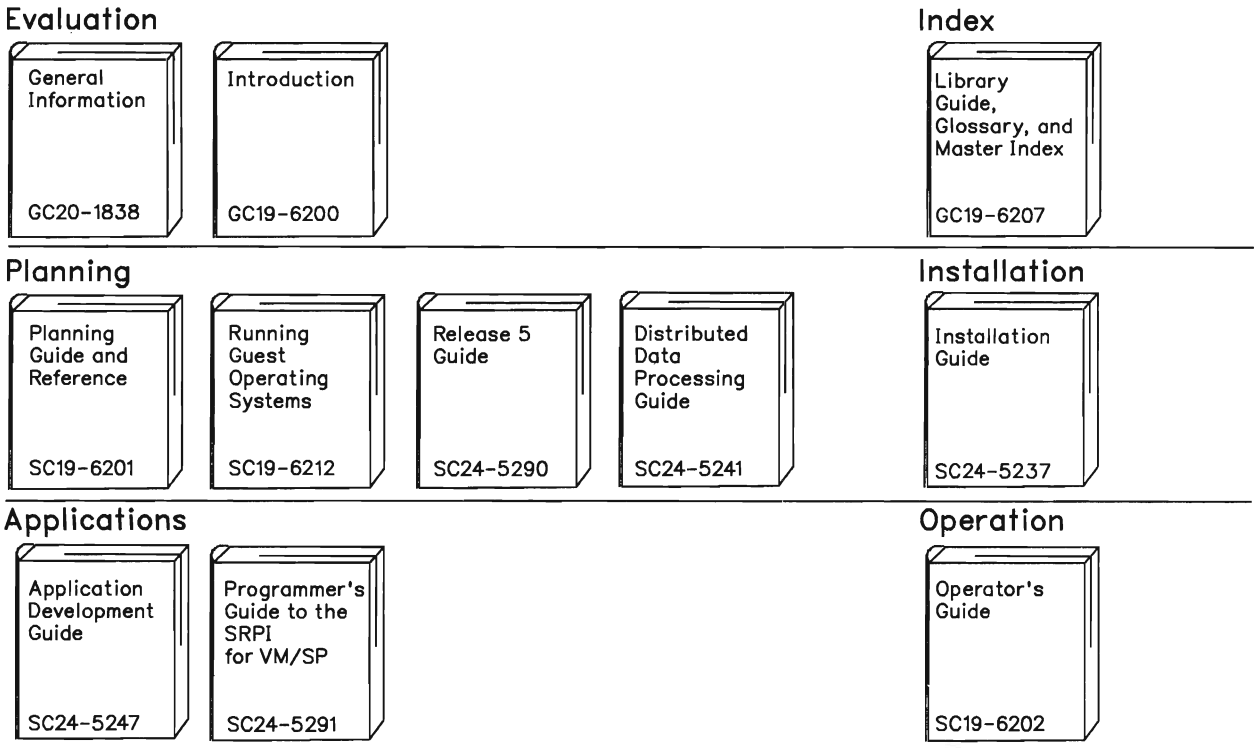
VM System Facilities for Programming, SC24-5288

For SNA LU Type 6.2 APPC architecture reference:

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

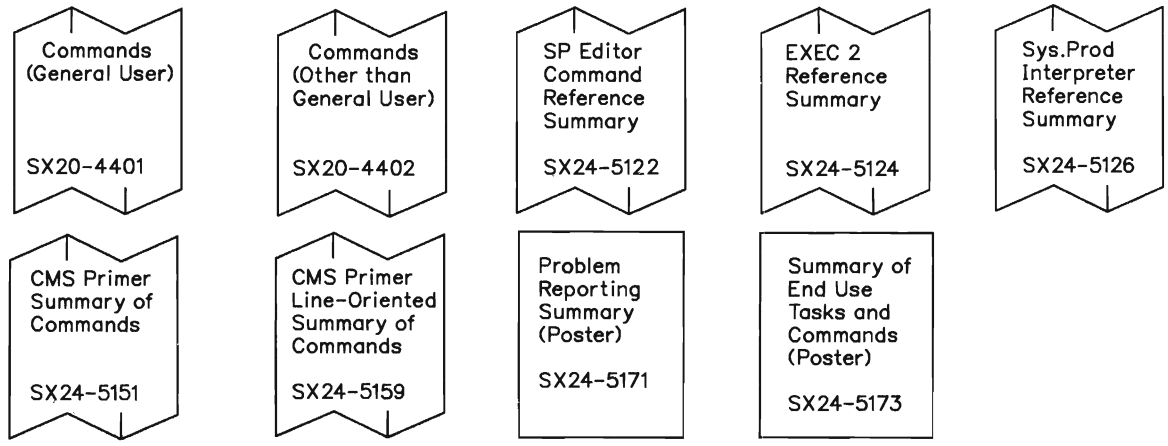
Systems Network Architecture Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2, SC30-3112

The VM/SP Library (Part 1 of 3)



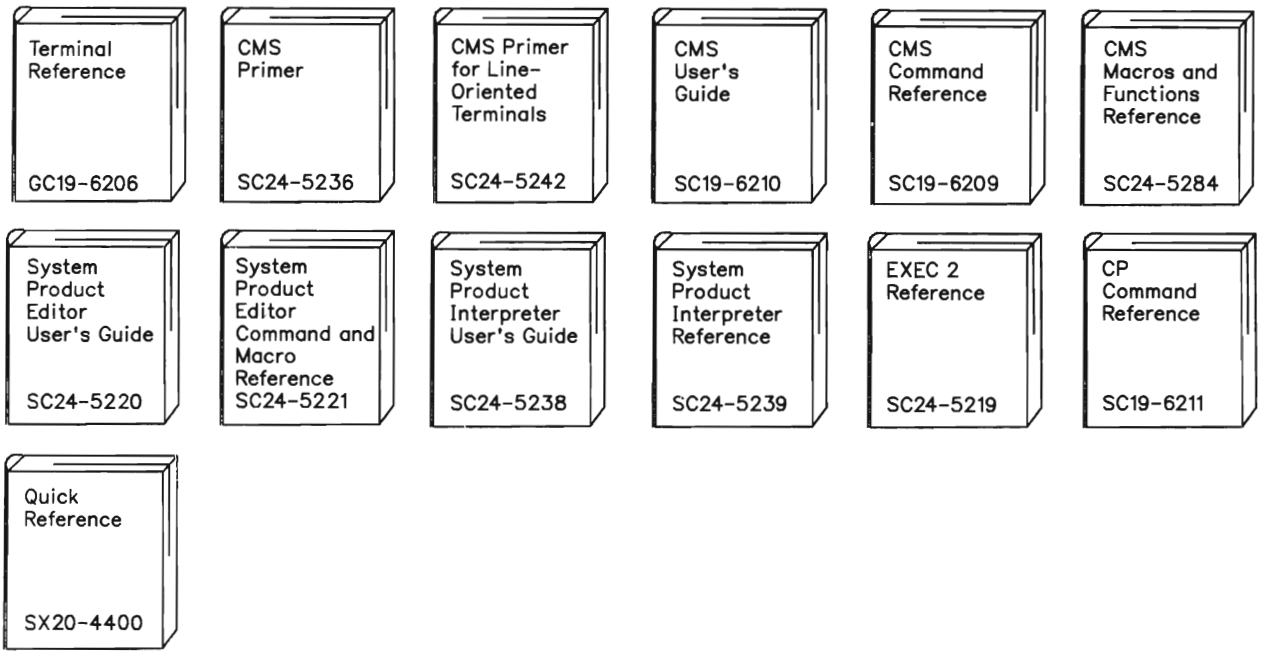
Reference Summaries

To order all of the Reference Summaries, use order number SBOF-3242

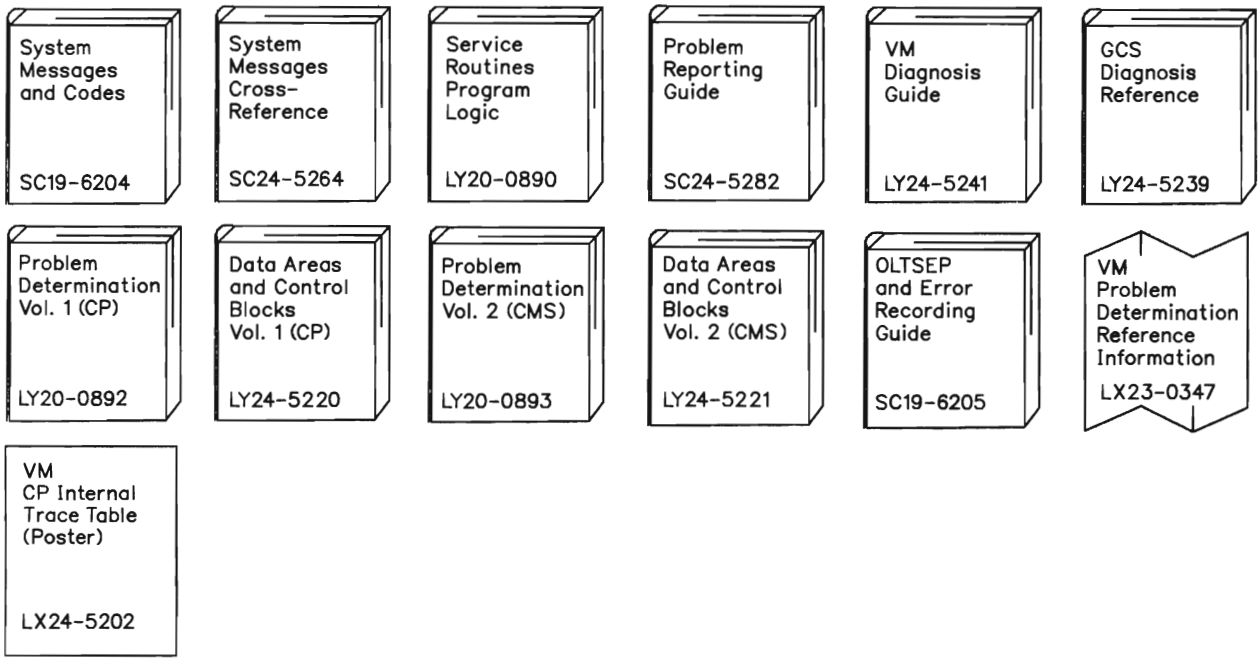


The VM/SP Library (Part 2 of 3)

End Use

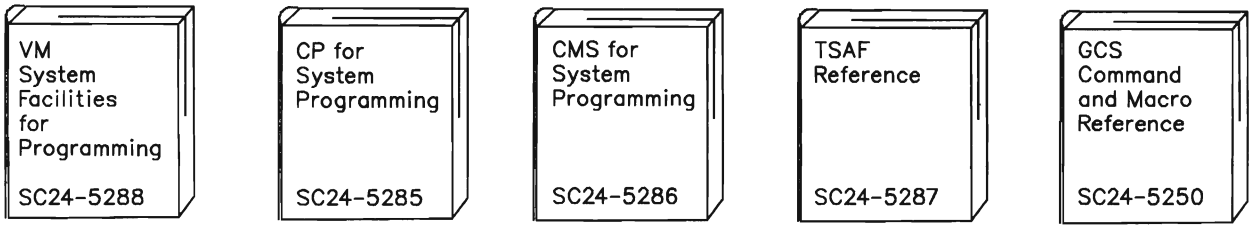


Diagnosis

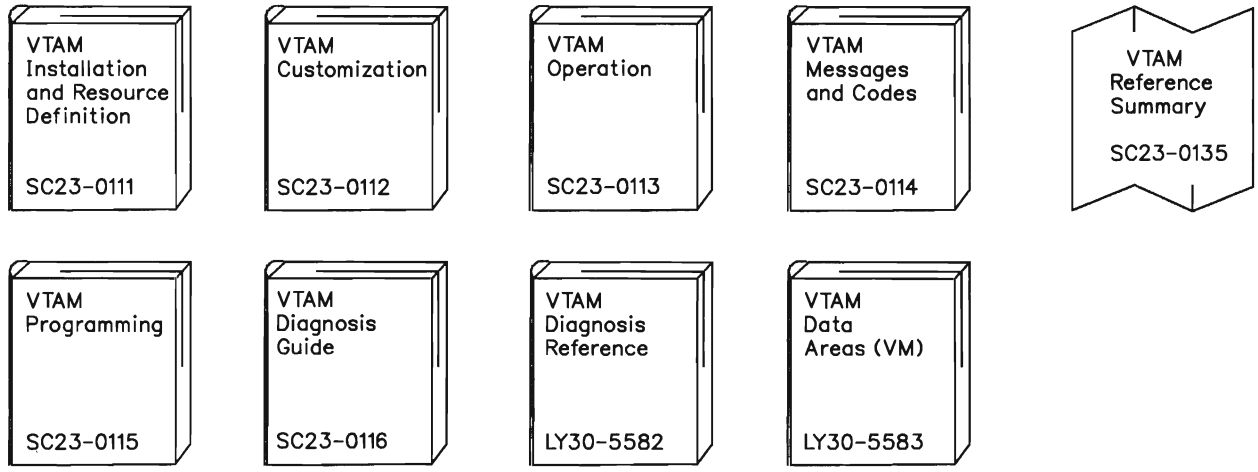


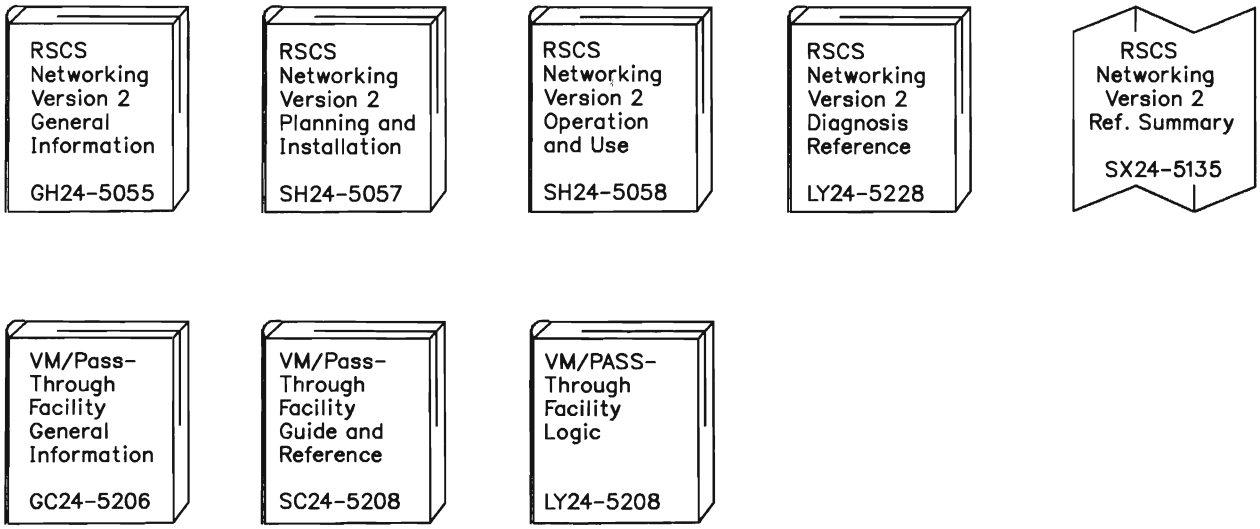
The VM/SP Library (Part 3 of 3)

Administration



Auxiliary Communication Support





Special Characters

A

B

D

J

L

M

N

O

P

Q

R

S

T

U

V

W

Numerics

International Business
Machines Corporation
P.O. Box 6
Endicott, New York 13760

File No. S370/4300-34
Printed in U.S.A.

SC24-5287-0

VM/SP Transparent Services
Access Facility Reference
Order No. SC24-5287-0

READER'S COMMENT FORM

Is there anything you especially like or dislike about this book? Feel free to comment on specific errors or omissions, accuracy, organization, or completeness of this book.

If you use this form to comment on the online HELP facility, please copy the top line of the HELP screen.

_____ _____ _____ **Help Information line ____ of ____**

IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you, and all such information will be considered nonconfidential.

Note: Do not use this form to report system problems or to request copies of publications. Instead, contact your IBM representative or the IBM branch office serving you.

Would you like a reply? ___YES ___NO

Please print your name, company name, and address:

_____

_____

_____

_____

IBM Branch Office serving you: _____

Thank you for your cooperation. You can either mail this form directly to us or give this form to an IBM representative who will forward it to us.

SC24-5287-0

Reader's Comment Form

CUT
OR
FOLD
ALONG
LINE

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NY

POSTAGE WILL BE PAID BY ADDRESSEE:

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
DEPARTMENT G60
PO BOX 6
ENDICOTT NY 13760-9987

Fold and tape | Please Do Not Staple | Fold and tape

IBM®