SC28-1309-0
File No. S370-39

Program Product

TSO Extensions Programmer's Guide to the Server-Requester Programming Interface for MVS/Extended Architecture

Program Number 5665-285

IBM

First Edition (September 1986)

This edition applies to TSO Extensions (TSO/E) Release 3, Program Number 5665-285, and all subsequent releases 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 Bibliography*, GC20-0001, for the editions that are applicable and current.

Preface

The server-requester programming interface (SRPI) of the TSO Extensions Enhanced Connectivity Facility lets you write *server* programs. The servers can provide MVS host computer services, data, and resources to *requester* programs on IBM personal computers.

The host computer must be running TSO Extensions (TSO/E) Release 3 with MVS/Extended Architecture (MVS/XA).

About this Book

This book tells you how to write an MVS server to receive a service request, process the request, and return a reply to the requester. The book includes a sample server, along with information on installing, testing, and debugging servers.

Audience

This book is for the following audience:

- Application programmers who design and write MVS servers and server initialization/termination programs.
- System programmers who allocate and initialize the data sets that make MVS servers and diagnosis information available to users.

The audience should be familiar with MVS programming conventions and the assembler language.

Organization

"Chapter 1. Introduction" describes MVS servers and how they provide MVS services, data, and resources to requester programs.

"Chapter 2. How to Write a Server" describes the input a server receives, the tools a server can use to process requests, and the output a server must provide.

"Chapter 3. How to Write a Server Initialization/Termination Program" describes how to write a program that initializes one or more servers, obtains resources for them, and terminates them.

"Chapter 4. How to Install a Server" describes how to allocate and initialize the data sets that give users access to servers and diagnosis information.

"Chapter 5. Testing and Diagnosis" tells how to use the MVSSERV command processor and test a server. This chapter also tells how to use the MVSSERV trace data set to diagnose any problems.

Related Publications

You may want the following books for help with writing servers in assembler:

- *OS/VS, DOS/VSE, and VM/370 Assembler Language*, GC33-4010
- *OS/VS, VM/370 Assembler Programmer's Guide*, GC33-4021

You may want the following books for related information about IBM Enhanced Connectivity Facilities and MVS:

- *Introduction to IBM System/370 to IBM Personal Computer Enhanced Connectivity Facilities*, GC23-0957
  - Introduces IBM Enhanced Connectivity Facilities, and the IBM host and personal computer hardware and software configurations supported.
- *IBM Programmer's Guide to the Server-Requester Programming Interface for the IBM Personal Computer and the IBM 3270 PC*, SC23-0959
  - Describes how to write requester programs for IBM personal computers; includes sample requesters.
- *TSO/E User's Guide*, SC28-1333
  - Describes how to assemble, link-edit, load, and execute programs in TSO/E.
- *TSO Extensions Command Processor Logic Volume IV*, LY28-1506
  - Contains diagnosis information related to the MVSSERV command processor and MVS servers.
- *TSO Extensions Command Language Reference*, GC28-1307
  - Gives syntax and reference information for the MVSSERV command.
- *TSO Messages*, GC28-1310
  - Lists and explains the MVSSERV messages.

- *MVS/Extended Architecture Message Library: System Codes*, GC28-1157
  - Lists MVS ABEND and reason codes.
- *MVS/Extended Architecture Message Library: System Messages, Volume 1*, GC28-1376
  - Lists messages associated with MVS ABEND and reason codes.
- *MVS/Extended Architecture Diagnostic Techniques System Programming Library*, GC28-0725
  - Contains information about system dump options.
- *OS/VS2 MVS Supervisor Services and Macro Instructions*, GC28-0683.
  - Contains information about the ESTAE macro for recovery routines.
- *TSO Extensions Guide to Writing a Terminal Monitor Program or a Command Processor* SC28-1136.
  - Contains information about the TSO command processor parameter list (CPPL).

Contents

Figures

Chapter 1. Introduction

This chapter introduces the TSO/E Enhanced Connectivity Facility, the server programs that you can write for it, and the MVSSERV command that manages TSO/E Enhanced Connectivity sessions on MVS/XA.

Concepts of the TSO/E Enhanced Connectivity Facility

The TSO/E Enhanced Connectivity Facility provides a standard way for application programs on different systems to share services.

With the TSO/E Enhanced Connectivity Facility, programs on properly-configured IBM Personal Computers (PCs) can obtain services from programs on IBM host computers running MVS/XA. The PC programs issue *service requests* and the host programs issue *service replies*, which the TSO/E Enhanced Connectivity Facility passes between the systems.

The PC programs that issue service requests are called *requesters*, and the host programs that issue replies are called *servers*. Servers and requesters together form enhanced connectivity applications.

Because the TSO/E Enhanced Connectivity Facility passes the requests and replies, you can write servers and requesters without concern for communications protocols. The requester simply specifies the server's name, the request input, and a reply buffer. The server receives the input, performs the service, and provides the reply. The TSO/E Enhanced Connectivity Facility passes the requests and replies in a standard, easily-referenced control block.

Host servers can give PC requesters access to host computer data, commands, and resources such as printers and storage. This book explains how to write an MVS host server, and includes a sample server that lets PC requesters process MVS data.

For information about PC hardware and software requirements, refer to *Introduction to IBM System/370 to IBM Personal Computer Enhanced Connectivity Facilities.*

What is an MVS Server?

MVS *servers* are programs that provide MVS host services, through the TSO/E Enhanced Connectivity Facility, to *requester* programs on a properly-configured IBM Personal Computer.

MVS servers are made up of *service functions.* The servers themselves are defined in *initialization/termination programs.*

Figure 1-1 shows the logical organization of servers, their service functions, and an initialization/termination program.

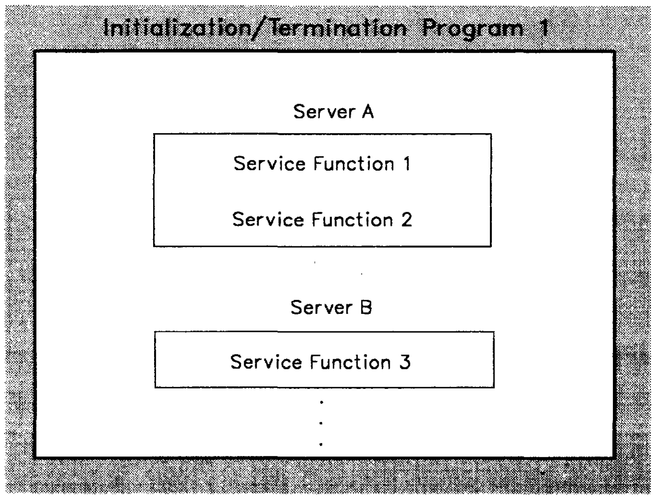


Figure 1-1. Logical Server Organization

Service Functions

A server can handle different service requests by having separate service functions, one for each request. Requests identify the service function as well as the server. The server receives the request and passes control to the specified service function. For details, see Chapter 2, "How to Write a Server."

Service functions can be related to the server in several ways: as subroutines of the server, as separate CSECTs, or as separate load modules.

Initialization/Termination Programs

An initialization/termination program defines one or more servers and provides a common work environment and resources for them. In particular, an initialization/termination program does the following:

- Defines its servers to the TSO/E Enhanced Connectivity manager, MVSSERV, so MVSSERV can route service requests to the servers.

- Isolates servers in a single MVS subtask, thus protecting the main task (MVSSERV) or other subtasks from server failures.

- Obtains and releases resources such as data sets, storage, and locks for the servers.

Servers and their initialization/termination programs can be physically packaged as separate load modules or as separate CSECTs in the same load module. Chapter 3, "How to Write a Server Initialization/Termination Program" describes factors to consider when packaging servers and initialization/termination programs.

What is MVSSERV?

MVSSERV is a TSO/E command processor that manages TSO/E Enhanced Connectivity sessions on the MVS host computer. Users issue MVSSERV on TSO/E to start an Enhanced Connectivity session. The users can then invoke requesters to issue service requests from a IBM PC that is running an Enhanced Connectivity program.

MVSSERV consists of a router and an interface to the servers. The server interface is called the *server-requester programming interface* (SRPI).

The router, through the SRPI, routes service requests to servers and routes service replies back to the requesters. Figure 1-2 shows the TSO/E Enhanced Connectivity environment during an MVSSERV session.

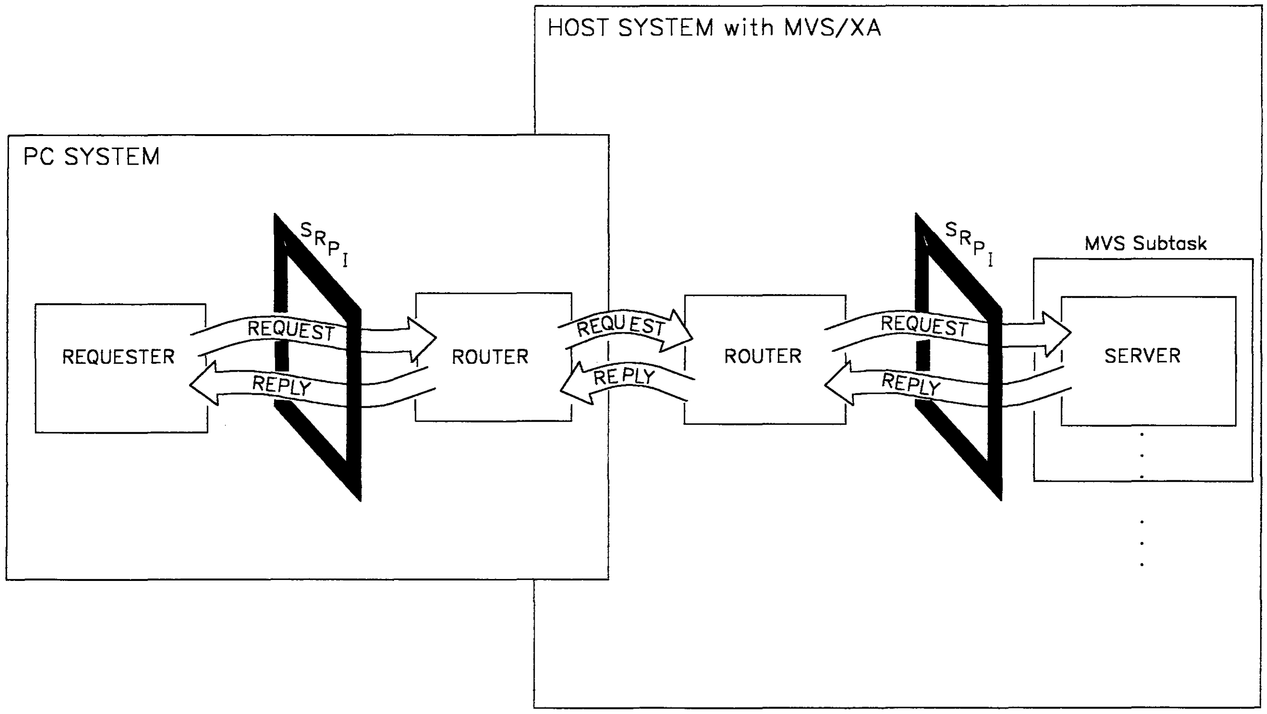


Figure 1-2. The MVSSERV Enhanced Connectivity Environment

The SRPI

MVSSERV's server-requester programming interface (SRPI) resembles the CALL/RETURN interface of most high-level programming languages. Through the SRPI, MVSSERV gives the server control along with input, a buffer for output, and a return address to which the server returns control. This interface allows you to write and use your own servers with MVSSERV.

Through the SRPI, MVSSERV calls servers and their initialization/termination programs for three phases of processing:

- Initialization -- setting up servers and their resources when MVSSERV begins, and defining the servers to MVSSERV.
- Handling service requests -- passing service requests to servers and sending back replies.
- Termination -- cleaning up servers and their resources when MVSSERV ends.

The CPRB Control Block

Service requests and replies pass through the SRPI in a control block called the *connectivity programming request block* (CPRB).

In MVSSERV, CPRBs have two purposes:

- The initialization/termination program uses a CPRB to define servers to MVSSERV.
- MVSSERV uses a CPRB to send service requests from the PC to the server, and to receive the server's reply.

The CPRB contains service request data such as the following:

- The name of the requested server and the service function ID
- The lengths and addresses of buffers containing input
- The lengths and addresses of reply buffers.

The INITTERM Control Block

When MVSSERV begins and ends, it passes the INITTERM control block to the initialization/termination programs. INITTERM indicates whether the call is for initialization *or* termination, and includes other input that the program needs.

Figure 1-3 shows the sequence of events in an MVSSERV session.

The Sequence of Events in an MVSSERV Session

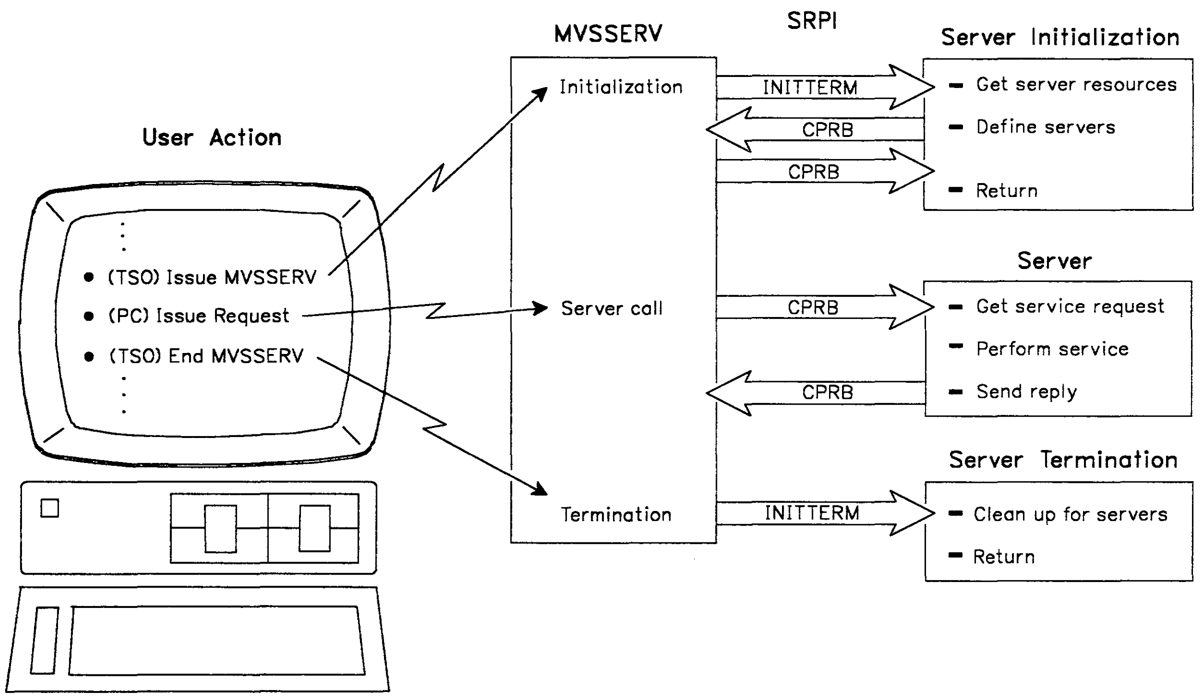


Figure 1-3. Events in an MVSSERV Session

What You Need to Do to Write Servers

The following is an overview of the steps you need to follow when writing servers for MVSSERV. Further details are contained in subsequent chapters of the book.

1. Select or create a load module to contain the executable code for the server and initialization/termination program. If they are in different load modules, the initialization/termination program must load the server (see Chapter 3 for details).

2. Write the server (see Chapter 2).

   - Access the service request input in the CPRB.
   - Call the service function.
   - Perform the service.
   - Indicate the reply length in the CPRB.
   - Set the return code in register 15.
   - Return control to MVSSERV.
   - Provide recovery (optional).
   - Compile or assemble the server and link it to a load module.

3. Write an initialization/termination program (see Chapter 3).

   - For initialization:

     - Load the server (if necessary).
     - Obtain resources (if necessary).
     - Define the server to MVSSERV and pass any parameters.

   - For termination:

     - Free any resources.
     - Delete the server (if loaded).

   - Compile or assemble the initialization/termination program and link it to a load module.

4. Install the server and initialization/termination program (see Chapter 4).

   - Install the programs in a STEPLIB or system library.
   - Define the initialization/termination program to MVSSERV in the input parameter data set.
   - Allocate diagnosis data sets (optional):
     - Trace data set
     - Dump data set
     - Dump suppression data set.

5. Invoke MVSSERV to test your server (see Chapter 5).

Writing Servers

Chapter 2, "How to Write a Server," describes how your servers and service functions must use the CPRB to communicate with a requester. The chapter also contains a sample server to illustrate use of the CPRB. As long as your servers use the CPRB correctly, they can do any processing and provide any services available on MVS. Your servers are restricted only by MVS and PC resources, and by any conventions used in the requester program.

You can write servers in any programming language, as long as the server can access the CPRB fields to retrieve input from, and send replies to, the requester. IBM provides a CPRB mapping macro, CHSDCPRB, for use with the assembler language.

Writing Server Initialization/Termination Programs

Chapter 3, "How to Write a Server Initialization/Termination Program" describes how to write a program to initialize your servers, define them to MVSSERV, and terminate them. The chapter explains how to use the DEFSERV macro to define a server to MVSSERV, and includes a sample initialization/termination program.

Note: Servers, as well as initialization/termination programs, can issue the DEFSERV macro to define other servers.

Installing Servers

After you have written a server and defined it in an initialization/termination program, you need to:

- Install the server and initialization/termination program in a private STEPLIB for testing or in a system library for general use.
- Name the initialization/termination program in an *input parameter data set* for MVSSERV. The input parameter data set must be available to the programs or users who invoke MVSSERV and issue requests.

Chapter 4, "How to Install a Server" describes how to install the server and how to allocate and initialize the input parameter data set.

If you want diagnostic information, additional data sets are needed. The diagnostic data sets receive MVSSERV trace data and dump data, and allow you to specify any ABENDs for which you do not want dumps to be taken. Chapter 4, "How to Install a Server" describes how to allocate and use the diagnostic data sets.

Testing and Diagnosing Servers

After you have written and installed servers and their initialization/termination programs, you will want to test them with the corresponding requesters. Chapter 5, "Testing and Diagnosis" describes how to invoke MVSSERV on TSO/E to test a server. The chapter also tells how to read diagnostic messages in the MVSSERV trace data set.

Chapter 2. How to Write a Server

This chapter describes the steps to follow when designing and writing servers.

Server Design

Servers provide MVS services, data, and resources to requester programs. Therefore, before you write a server, you need to consider what output it will provide, and what requester input it will receive.

Servers and requesters work in pairs. Each service request must name the corresponding server and service function, and must include any input that the server needs. The server must use the input and produce output that the requester can use.

For information about writing requesters, refer to *IBM Programmer's Guide to the Server-Requester Programming Interface for the IBM Personal Computer and the IBM 3270 PC.*

Steps for Designing a Server

Follow these steps when designing a server:

1. Decide what service request (or requests) your server will handle. If your server handles more than one service request, your server needs a service function for each request. The service functions can be:

   - Subroutines of the server
   - CSECTs of the server
   - Load modules that are separate from the server.

2. Decide what AMODE and RMODE the server should execute in. Servers can execute in any AMODE or RMODE.

3. Select a name for the server. Server names must conform to MVS program naming conventions. They can have up to eight characters, including the characters A-Z, 0-9, @, #, and $. The first character cannot be 0-9.

Writing a Server

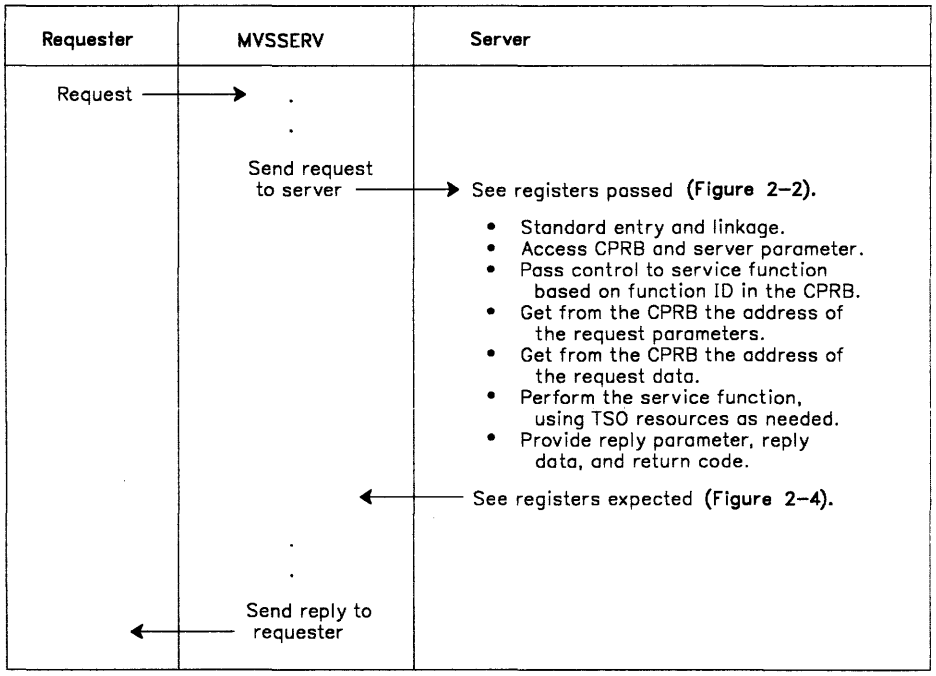
Your server must follow certain rules to receive service requests and reply to them successfully. The rules apply to using the connectivity programming request block (CPRB).

Using the CPRB

To respond to a service request, the server must:

- Receive the service request input in the CPRB
- Perform the service
- Send a service reply in the CPRB.

Figure 2-1 shows the process for handling service requests.



- Standard entry and linkage.
- Access CPRB and server parameter.
- Pass control to service function based on function ID in the CPRB.
- Get from the CPRB the address of the request parameters.
- Get from the CPRB the address of the request data.
- Perform the service function, using TSO resources as needed.
- Provide reply parameter, reply data, and return code.

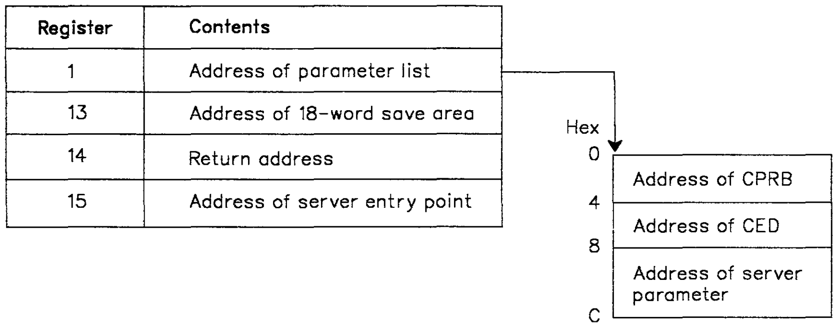
Figure 2-1. Overview of Service Request Handling

Receiving the Service Request

MVSSERV passes control to the server in key 8, problem program state, with the register contents shown in Figure 2-2.

Register 1 points to a parameter list that contains addresses of the CPRB, the connectivity environment descriptor (CED), and a parameter from the server initialization/termination program. Of the three:

- The CPRB contains the service request.
- The CED is for system use only. (If the server issues the DEFSERV macro to define another server, it must pass the CED address.)
- The server parameter can point to data sets or other resources for the server. (For details about creating the server parameter, see Chapter 3, "How to Write a Server Initialization/Termination Program.")



Register	Contents
1	Address of parameter list
13	Address of 18-word save area
14	Return address
15	Address of server entry point

Figure 2-2. Registers Passed to the Server

Mapping to the CPRB Fields

Your server can use the CHSDCPRB mapping macro to access the fields of the CPRB. The CHSDCPRB macro has the following assembler syntax:

[label] CHSDCPRB [DSECT=YES|NO]

Code the macro with DSECT=YES (or omit the DSECT parameter) to build a DSECT for the CPRB fields. You can use the label CHSDCPRB to address the CPRB with an assembler USING statement. For an example of using the CHSDCPRB macro, see "Sample Server" on page 2-6.

Figure 2-3 shows the CPRB with the fields that pertain to the server.

The Receive Request CPRB (Entry to Server)

Field Label	Byte Offset	Byte Length	Contents
CRBF1	0(0)	1	The control block's version number (first four bits) and modification level number (last four bits).
	1(1)	2	Reserved
CRBF4	3(3)	1	The type of request (X'01' indicates a service request).
CRBCPRB	4(4)	4	The value of C'CPRB'.
	8(8)	8	Reserved
CRBSNAME	16(10)	8	The name of the requested server.
	24(18)	2	Reserved
CRBFID❶	26(1A)	2	The ID of the requested service function.
	28(1C)	12	Reserved
CRBRQDLN❶	40(28)	4	The length of the request data.
CRBRQDAT❶	44(2C)	4	The address of the request data.
CRBRPDLN❷	48(30)	4	The length of the reply data (maximum length allowed by the requester).
CRBRPDAT❸	52(34)	4	The address of the buffer for reply data.
CRBRQPLN❶	56(38)	4	The length of the request parameters.
CRBRQPRM❶	60(3C)	4	The address of the request parameters.
CRBRPPLN❷	64(40)	4	The length of the reply parameters (maximum length allowed by the requester).
CRBRPPRM❸	68(44)	4	The address of the buffer for reply parameters.
	72(48)	40	Reserved

Figure 2-3. CPRB Control Block on Entry to Server

Notes:

❶ **Request** field. Use but do not alter.
❷ **Request/Reply** field. The requester initializes these fields. The server may modify the contents of these fields.
❸ **Address of Reply** field. Use but do not alter. The server may return information in a buffer located at this address. Do not return more information than will fit in the buffer (as indicated in the associated length field).
Do not modify any fields other than those marked with a ❷.

Sending the Service Reply

After performing the requested service function, the server must:

- Move the reply data, if any, to the reply data buffer using CPRB field CRBRPDAT.
- Move the reply parameters, if any, to the reply parameter buffer using CPRB field CRBRPPRM.
- Put the actual reply data length in CPRB field CRBRPDLN.
- Put the actual reply parameter length in CPRB field CRBRPPLN.

- Put the return code expected by the requester in register 15.

- Return the reply CPRB to the requester (branch to the return address that was in register 14 on entry to the server).

  The registers should have the following contents when the server ends:

Register	Contents
13	Address of 18-word save area
14	Return address
15	Server return code

Figure 2-4. Registers Expected from the Server

Figure 2-5 shows the CPRB fields that the server uses in its reply.

The Send Reply CPRB (Exit from Server)

Field Label	Byte Offset	Byte Length	Contents
	0(0)	48	Reserved
CRBRPDLN❶	48(30)	4	Specify the actual length of the reply data.
	52(34)	12	Reserved
CRBRPPLN❶	64(40)	4	Specify the actual length of the reply parameters.
	68(44)	44	Reserved

Figure 2-5. CPRB Control Block on Exit from the Server

Note:

❶ The actual length cannot exceed the initial value (maximum allowed by the requester).

The Server Recovery Routine

Your servers can have their own recovery routines. If a server fails and does not recover, MVSSERV traps the error, provides a dump, and marks the server's subtask as inactive, preventing further requests for that server or any other servers defined by the same initialization/termination program.

Your servers can issue the ESTAE macro to establish a recovery routine. The server recovery routine should do the following:

- Record pertinent diagnostic information in the SDWA and VRA, such as the caller, the current module in control, and the input parameters.

- Optionally, specify a dump (if not, MVSSERV provides one).

- If the failure is recoverable, set return parameters specifying that a retry is to be made. The retry routine must return control to MVSSERV with the server's return code.

- If the failure is not recoverable, percolate to MVSSERV.

For more information about using the ESTAE macro and retry routines, refer to *OS/VS2 MVS Supervisor Services and Macro Instructions.*

Compiling or Assembling a Server

After writing a server, you need to compile or assemble it and link it to a load module. For information about preparing and running a program in TSO/E, refer to Section III of the *TSO/E User's Guide.*

Sample Server

The sample server in Figure 2-6 corresponds to the sample assembler requester in the *IBM Programmer's Guide to the Server-Requester Programming Interface for the IBM Personal Computer and the IBM 3270 PC.* The server, IBMABASE, has two service functions:

- Function 1 retrieves a record from a customer records data set on MVS, translates the record into ASCII, and sends the record to the requester for processing.

- Function 2 receives a record with a positive balance from the requester, translates the record back into EBCDIC, and puts the record into an accounts receivable data set on MVS.

The initialization/termination program for this server is in Chapter 3, "How to Write a Server Initialization/Termination Program."

```
**********************************************************************
* TITLE: IBMABASE MAINLINE
*
* LOGIC: Determine the function requested, and perform that function.
*
* OPERATION:
* 1. Establish addressability to the CPRB.
* 2. Establish addressability to the server parameters.
* 3. If the data sets are not open:
*       - Open them.
* 4. Determine the function requested.
* 5. If function 1 is requested:
*       - Issue the GET macro to read in a record.
*       - If the end of file was encountered:
*           a. Close the data sets.
*           b. Set end of file return code
*       - Else, no end of file encountered:
*           a. If the transaction should be logged:
*                - Issue the PUT macro to output the log message.
*           b. Translate the reply data into ASCII.
* 6. If function 2 is requested:
*       - Translate the request data into EBCDIC.
*       - Issue the PUT macro to write the record.
*       - If the transaction should be logged:
*           a. Issue the PUT macro to output the log message.
* 7. Else set invalid function return code.
* 8. Return to the caller with return code.
**********************************************************************
IBMABASE CSECT
IBMABASE AMODE 24
IBMABASE RMODE 24
         STM   14,12,12(13)         Save the caller's registers.
         LR    12,15                Establish addressability within
         USING IBMABASE,12          this CSECT.
         ST    13,SAVEAREA+4        Save the caller's save area address.
         LA    15,SAVEAREA          Obtain our save area address.
         ST    15,8(,13)            Chain it in the caller's save area.
         LR    13,15                Point register 13 to our save area.

         L     2,0(,1)              Obtain the CPRB address.
         USING CHSDCPRB,2           Establish addressability to it.
         L     3,8(,1)              Obtain server parameter address.
         USING PARAMETERS,3         Establish addressability to them.
         SPACE
         CLI   STATUS,OPENED        Are the data sets opened?
         BE    OPEN                 Yes, then don't try to open them.
         L     5,DCBIN_ADDR         Load the INPUT DCB address.
         L     6,DCBOUT_ADDR        Load the OUTPUT DCB address.
         L     7,DCBLOG_ADDR        Load the LOG DCB address.
         L     8,OPEN_ADDR          Load the list form address.
         OPEN  ((5),,(6),,(7)),MF=(E,(8)) Open the data sets.
         MVI   STATUS,OPENED        Indicate that they are open.
OPEN     DS    0H
         LA    4,1                  Load the first function ID.
         CH    4,CRBFID             Is function one requested?
         BE    FUNCTION_1           Yes, branch to the function.
         LA    4,2                  Load the second function ID.
         CH    4,CRBFID             Is function two requested?
         BE    FUNCTION_2           Yes, branch to the function.
         LA    15,8                 Else, bad function id.
         B     EXIT                 Exit the server.
```

Figure 2-6 (Part 1 of 4). Sample Server

```
FUNCTION_1 DS  0H
         L     4,CRBRPDAT
         USING REPLY_BUFFER,4
         L     5,DCBIN_ADDR           Obtain INPUT DCB address.
         USING IHADCB,5
         MVC   DCBEODA,=AL3(END_OF_FILE) Set end of file exit.
         GET   (5),REPLY              Get the record.
         DROP  5
         L     5,CRBRQPRM             Load request parameter address.
         CLI   0(5),X'01'             Should we log the transaction?
         BNE   NO_INPUT_LOG           No, branch around logging.
         L     5,DCBLOG_ADDR          Obtain LOG DCB address.
         PUT   (5),INPUT_LOG          Output the log message and
         PUT   (5),REPLY              the record.
         PUT   (5),BLANK              Insert a blank line.
NO_INPUT_LOG DS 0H
         LA    15,0                   Set the return code.
         TR    TRANS_PART,TRANS_ASCII Translate the record to ASCII.
         CLI   CUST_BAL,X'60'         Check for a minus sign.
         BNE   DO_PACK
         NI    CUST_BAL+3,X'DF'       Allow CVB to make it negative.
DO_PACK  DS    0H
         PACK  WORKAREA(8),CUST_BAL(4) Convert balance to decimal.
         CVB   6,WORKAREA             Convert balance to binary.
         ST    6,BINARY_BAL           Save the balance.
**********************************************************************
* Move the balance into the reply area, taking into account the PC's
*      method of reverse byte retrieval.
**********************************************************************
         MVC   CUST_BAL(1),BINARY_BAL+3   Place it into the reply.
         MVC   CUST_BAL+1(1),BINARY_BAL+2 Place it into the reply.
         MVC   CUST_BAL+2(1),BINARY_BAL+1 Place it into the reply.
         MVC   CUST_BAL+3(1),BINARY_BAL   Place it into the reply.
         LA    5,REPLY_LEN            Get the length of the reply,
         ST    5,CRBRPDLN             and store it into the CPRB.
         LA    5,0                    Set the reply parameter length,
         ST    5,CRBRPPLN             and store it into the CPRB.
         B     EXIT
END_OF_FILE DS 0H
         L     5,DCBIN_ADDR           Load the INPUT DCB address.
         L     6,DCBOUT_ADDR          Load the OUTPUT DCB address.
         L     7,DCBLOG_ADDR          Load the LOG DCB address.
         L     8,CLOSE_ADDR           Load the list form address.
         CLOSE ((5),,(6),,(7)),MF=(E,(8)) Close the data sets.
         MVI   STATUS,CLOSED          Indicate that they are closed.
         LA    15,4                   Set end of file return code.
         B     EXIT
```

Figure 2-6 (Part 2 of 4). Sample Server

```
FUNCTION_2 DS  0H
         L     4,CRBRQDAT
         USING REPLY_BUFFER,4
         TR    TRANS_PART,TRANS_EBCDIC Translate the record to EBCDIC.
***********************************************************************
* Move the reply balance into the work area, taking into account the
*      PC's method of reverse byte retrieval.
***********************************************************************
         MVC   BINARY_BAL(1),CUST_BAL+3   Obtain customer balance.
         MVC   BINARY_BAL+1(1),CUST_BAL+2 Obtain customer balance.
         MVC   BINARY_BAL+2(1),CUST_BAL+1 Obtain customer balance.
         MVC   BINARY_BAL+3(1),CUST_BAL   Obtain customer balance.
         L     6,BINARY_BAL          Prepare for CVD.
         CVD   6,WORKAREA            Convert the balance to decimal.
         UNPK  CUST_BAL(4),WORKAREA(8) Convert to zoned decimal.
         MVZ   CUST_BAL+3(1),CUST_BAL+2  Fix zone format.
         L     5,DCBOUT_ADDR         Obtain OUTPUT DCB address.
         PUT   (5),REPLY             Output the record.
         L     5,CRBRQPRM            Load request parameter address.
         CLI   0(5),X'01'            Should we log the transaction?
         BNE   NO_OUTPUT_LOG         No, branch around logging.
         L     5,DCBLOG_ADDR         Obtain LOG DCB address.
         PUT   (5),OUTPUT_LOG        Output the log message and
         PUT   (5),REPLY             the record.
         PUT   (5),BLANK             Insert a blank line.
NO_OUTPUT_LOG DS 0H
         LA    15,0                  Set the return code.
EXIT     DS    0H
         L     13,SAVEAREA+4         Restore caller's save area address.
         L     14,12(,13)            Restore the caller's registers
         LM    0,12,20(13)           except for 15 (return code).
         BR    14                    Return to caller with return code.
***********************************************************************
* Data section.
***********************************************************************
* EBCDIC to ASCII translate table.
***********************************************************************
TRANS_ASCII DS 0CL256
         DC    X'00010203CF09D37FD4D5C30B0C0D0E0F'
         DC    X'10111213C7B408C91819CCCD831DD21F'
         DC    X'81821C84860A171B89919295A2050607'
         DC    X'E0EE16E5D01EEA048AF6C6C21415C11A'
         DC    X'20A6E180EB909FE2AB8B9B2E3C282B7C'
         DC    X'26A9AA9CDBA599E3A89E21242A293B5E'
         DC    X'2D2FDFDC9ADDDE989DACBA2C255F3E3F'
         DC    X'D78894B0B1B2FCD6FB603A2340273D22'
         DC    X'F861626364656667686996A4F3AFAEC5'
         DC    X'8C6A6B6C6D6E6F7071729787CE93F1FE'
         DC    X'C87E737475767778797AEFC0DA5BF2F9'
         DC    X'B5B6FDB7B8B9E6BBBCBD8DD9BF5DD8C4'
         DC    X'7B414243444546474849CBCABEE8ECED'
         DC    X'7D4A4B4C4D4E4F505152A1ADF5F4A38F'
         DC    X'5CE7535455565758595AA0858EE9E4D1'
         DC    X'30313233343536373839B3F7F0FAA7FF'
***********************************************************************
* ASCII to EBCDIC translate table.
***********************************************************************
TRANS_EBCDIC DS 0CL256
         DC    X'00010203372D2E2F1605250B0C0D0E0F'
         DC    X'101112133C3D322618193F27221D351F'
```

Figure 2-6 (Part 3 of 4). Sample Server

```
         DC    X'405A7F7B5B6C507D4D5D5C4E6B604B61'
         DC    X'F0F1F2F3F4F5F6F7F8F97A5E4C7E6E6F'
         DC    X'7CC1C2C3C4C5C6C7C8C9D1D2D3D4D5D6'
         DC    X'D7D8D9E2E3E4E5E6E7E8E9ADE0BD5F6D'
         DC    X'79818283848586878889919293949596'
         DC    X'979899A2A3A4A5A6A7A8A9C04FD0A107'
         DC    X'4320211C23EB249B7128384990BAECDF'
         DC    X'45292A9D722B8A9A6756644A53685946'
         DC    X'EADA2CDE8B5541FE5851524869DB8E8D'
         DC    X'737475FA15B0B1B3B4B56AB7B8B9CCBC'
         DC    X'AB3E3B0ABF8F3A14A017CBCA1A1B9C04'
         DC    X'34EF1E0608097770BEBBAC5463656662'
         DC    X'30424757EE33B6E1CDED3644CECF31AA'
         DC    X'FC9EAE8CDDDC39FB80AFFD7876B29FFF'
SAVEAREA DC    18F'0'                IBMABASE's save area.
SUBSAVE  DC    18F'0'                IBMABASE subroutine's save area.
WORKAREA DS    D                     Work area for CVB and CVD.
BINARY_BAL DS  F                     Holds binary form of the balance.
STATUS   DC    X'0'                  Status word.
INPUT_LOG DS   0CL109                Input log message.
         DC    CL109'The following customer record was read from the cu*
               stomer files.'

OUTPUT_LOG DS  0CL109                Output log message.
         DC    CL109'The following customer record was written to the b*
               illing file.'

BLANK    DC    CL109' '
OPENED   EQU   X'00'                 Data sets are opened.
CLOSED   EQU   X'01'                 Data sets are closed.
***********************************************************************
* Server parameter list mapping.
***********************************************************************
PARAMETERS DSECT
DCBIN_ADDR DS  A                     INPUT DCB address.
DCBOUT_ADDR DS A                     OUTPUT DCB address.
DCBLOG_ADDR DS A                     LOG DCB address.
OPEN_ADDR DS   A                     OPEN list form address.
CLOSE_ADDR DS  A                     CLOSE list form address.
***********************************************************************
* CPRB reply buffer mapping.
***********************************************************************
REPLY_BUFFER DSECT
REPLY    DS    0CL109
TRANS_PART DS  0CL105
CUST_NAME DS   CL25
CUST_ADDR DS   CL25
CUST_CITY DS   CL15
CUST_STATE DS  CL15
CUST_ZIP  DS   CL9
CUST_ACCT DS   CL16
CUST_BAL  DS   CL4
REPLY_LEN EQU  *-REPLY
***********************************************************************
* CPRB mapping
***********************************************************************
         CHSDCPRB DSECT=YES
***********************************************************************
* DCB mapping
***********************************************************************
         DCBD  DSORG=PS
         END   IBMABASE
```

Figure 2-6 (Part 4 of 4). Sample Server

Chapter 3. How to Write a Server Initialization/Termination Program

This chapter describes the steps to follow when designing and writing server initialization/termination programs.

Program Design

When MVSSERV gets control, it invokes your server initialization/termination programs in separate subtasks. The initialization/termination programs define one or more servers to MVSSERV, and optionally load the servers and provide resources for them. When MVSSERV ends, it re-invokes your initialization/termination programs to free any server resources and terminate the servers.

Figure 3-1 shows the position of initialization/termination programs in the MVSSERV task structure.

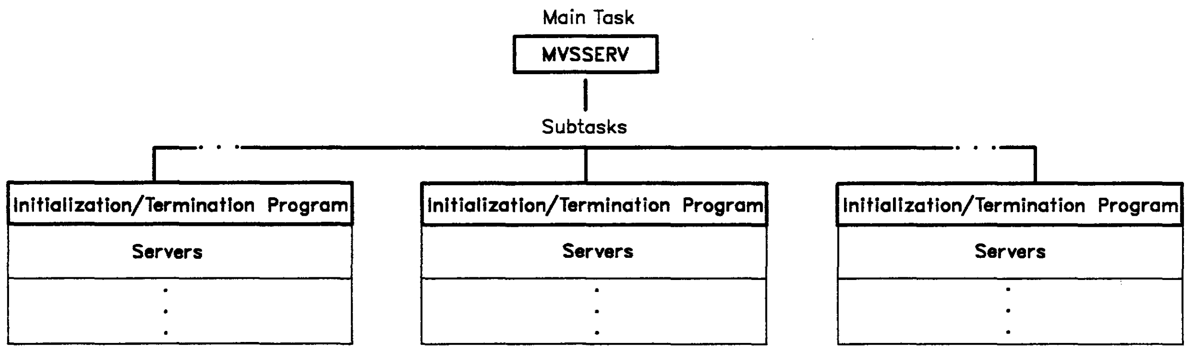

Main Task
MVSSERV
Subtasks
Initialization/Termination Program
Servers
Initialization/Termination Program
Servers
Initialization/Termination Program
Servers

Figure 3-1. MVSSERV Logical Task Structure

When you design an initialization/termination program, you need to consider what servers it will define, what resources the servers require, and how to package the initialization/termination program in relation to the servers.

Steps for Designing an Initialization/Termination Program

Follow these steps when designing an initialization/termination program:

1. Decide what servers the initialization/termination program will define. The main considerations are server resources and recovery.

   - Resources -- The initialization/termination program can obtain and release resources such as storage, data sets, and locks for its servers. If servers share resources, you can increase their efficiency by having a single initialization/termination program define the servers and obtain and release the resources for them.

   - Recovery -- If a server fails and cannot recover, MVSSERV calls the server's initialization/termination program to terminate all the servers it defined. Therefore, you might want to define interdependent servers in the same initialization/termination program, and define unrelated servers in different initialization/termination programs.

2. Decide how to package the initialization/termination program in relation to the servers.

   You can package servers and their initialization/termination program as CSECTs of the same load module or as different load modules. The main consideration is server loading:

   - If you want the server and its initialization/termination program to be ***in the same*** load module, the initialization/termination program does not have to load the server. The initialization/termination program can use a constant server address to define the server to MVSSERV.

   - If you want the server and its initialization/termination program to be in ***different*** load modules, the initialization/termination program must load the server and get the server address from the LOAD macro to use when defining the server to MVSSERV.

3. Decide what AMODE and RMODE the initialization/termination program should execute in. Initialization/termination programs can execute in any AMODE or RMODE.

4. Select a name for the initialization/termination program. The name must conform to MVS program naming conventions. It can have up to eight characters, including the characters A-Z, 0-9, @, #, and $. The first character cannot be 0-9.

5. Put the name of the initialization/termination program in the input parameter data set (see Chapter 4, "How to Install a Server").

Writing an Initialization/Termination Program

Figure 3-2 gives an overview of an initialization/termination program's processing.

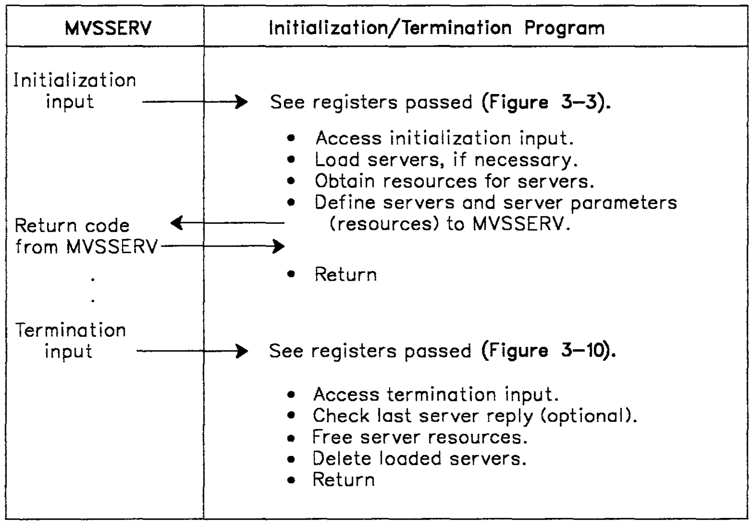

MVSSERV
Initialization/Termination Program
Initialization input
See registers passed (Figure 3–3).
• Access initialization input.
• Load servers, if necessary.
• Obtain resources for servers.
• Define servers and server parameters (resources) to MVSSERV.
Return code from MVSSERV
• Return
Termination input
See registers passed (Figure 3–10).
• Access termination input.
• Check last server reply (optional).
• Free server resources.
• Delete loaded servers.
• Return

Figure 3-2. Overview of an Initialization/Termination Program's Processing

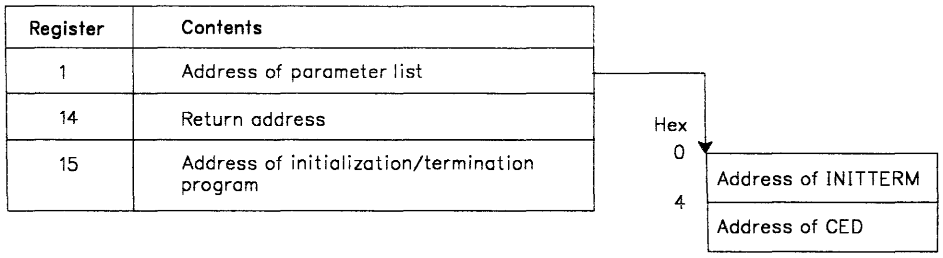
Initialization

When MVSSERV receives control, it invokes the server initialization/termination programs in separate subtasks. MVSSERV gets the names of the initialization/termination programs from the input parameter data set described in Chapter 4, "How to Install a Server."

Input to the Initialization/Termination Program

Figure 3-3 and Figure 3-4 show the input that MVSSERV makes available to the initialization/termination programs.

As shown in Figure 3-3, Register 1 points to a two-word area. The first word contains the address of the INITTERM control block; the second word contains the address of data (the CED) for MVSSERV use only. Your initialization/termination program needs the CED address when defining servers to MVSSERV.



Register	Contents
1	Address of parameter list
14	Return address
15	Address of initialization/termination program

Figure 3-3. Registers Passed at Initialization

You can use the INITTERM mapping macro to obtain input from the INITTERM control block. The INITTERM macro has the following assembler syntax:

[label] INITTERM [DSECT=YES|NO]

Code the macro with DSECT=YES (or omit the DSECT parameter) to build a DSECT for the control block fields. You can use the label INITTERM to address the control block with an assembler USING statement. For an example of using the assembler INITTERM macro, see "Sample Initialization/Termination Program" on page 3-11.

Field Label	Byte Offset	Byte Length	Contents
INTINIT❶	0(0)	4	Initialization or termination indicator. X'0000' indicates the call is for initialization. (X'0001' indicates termination.)
INTWALEN❷	4(4)	4	Work area length. Specify the length of a work area that the program can use at termination time.
INTWAPTR❷	8(8)	4	Work area address. Specify the address of a work area that the program can use at termination time.
	12(C)	16	Reserved
INTENVRN	28(1C)	4	Address of the TSO CPPL (command processor parameter list). The CPPL is for system use only; its address must be in register 1 if a server or initialization/termination program invokes a TSO command processor or uses TSO services such as SCAN or PARSE. For more information about the CPPL, see *TSO/E Guide to Writing a Terminal Monitor Program or a Command Processor.*
	32(20)	4	Reserved

Figure 3-4. INITTERM Control Block with Initialization Input

Notes:

❶ Check for initialization or termination indicator.
❷ Specify a work area (optional).

Loading the Servers

If the servers are not in the same load module as their initialization/termination program, the initialization/termination program needs to load the servers.

The following assembler language example shows how an initialization/termination program can load a server that is not in the same load module.

```
LOAD EP=server name        Load the server
LR   5,0                   Get server address from LOAD macro
  .                        for use in the DEFSERV macro
  .
```

Obtaining Resources

Your initialization/termination program can obtain any resources that the servers require or share. For example, the initialization/termination program can:

- Open data sets that the servers will need to use.
- Obtain storage using the GETMAIN macro, such as a work area to be shared among the servers.
- Obtain locks for any resources to which the servers require exclusive access.

The initialization/termination program makes resources available to the server by placing them in a server parameter list as part of the server definition process. When MVSSERV passes a service request to the server, it passes the server parameter list as well.

Defining a Server

The initialization/termination program must define its servers to MVSSERV. The definition must include the names and addresses of the servers, and the addresses of any parameter lists to be passed to the servers along with service requests. MVSSERV makes a table of the names and addresses; the MVSSERV router obtains the addresses of requested servers from the table.

You can define the servers using the IBM-supplied DEFSERV macro. The coded DEFSERV macro does the following:

- Fills in fields of a connectivity programming request block (CPRB) that defines the server to MVSSERV.
- Creates a parameter list for the server.

Before issuing the DEFSERV macro, the initialization/termination program must define storage for the DEFSERV CPRB. To define storage, the program can use the CHSDCPRB macro with the following assembler syntax:

[label] CHSDCPRB DSECT=NO

For an example of using the CHSDCPRB macro with DEFSERV, see "Sample Initialization/Termination Program" on page 3-11.

The initialization/termination program can use the same CPRB to define each of its servers to MVSSERV.

Register Contents for DEFSERV

Before issuing the DEFSERV macro, an initialization/termination program must set Register 13 to the specified contents:

Register	Contents
13	Address of a standard 18-word save area

There are no requirements for the other registers. However, the DEFSERV macro may change the contents of the following registers: 0, 1, 14, 15.

Syntax and Parameters in Assembler

Figure 3-5 shows the syntax of the DEFSERV macro for assembler language. For the execute form, all parameters are required keyword parameters. For the list form, only a label and the MF keyword are required.

```
EXECUTE FORM:

  label          DEFSERV    CPRB=address,
                            CED=address,
                            SERVNAME=server name,
                            SERVEPA=server address,
                            SERVPARM=parmlist address,
                            MF=(E,parmlist_name)
LIST FORM:

  parmlist_name  DEFSERV    MF=L
```

Note: The addresses can be any address valid in an RX instruction, or the number of a general register (2-12) enclosed in parentheses.

Figure 3-5. DEFSERV Macro in Assembler Language

CPRB = address
Specify the address of the DEFSERV CPRB. The CPRB must begin on a fullword boundary. The address must be in the same addressing mode (AMODE) as the initialization/termination program. For 24-bit addresses (AMODE 24), the high-order byte in the address must be 0; for 31-bit addresses (AMODE 31), the high-order bit must be set to 1.

CED = address
Specify the address of the CED that was passed as input to the initialization/termination program.

SERVNAME = server name
Specify the name of the server being defined. You can also specify a general register (2-12) that points to an eight-byte field containing the server name. To do so, enclose the register number in parentheses. This name is passed to MVSSERV in the CRBSNAME field of the DEFSERV CPRB.

SERVEPA = server address
Specify the address of the server being defined. If the initialization/termination program loaded the server, obtain the address from the LOAD macro. If you do not obtain the address from the LOAD macro, and the server is AMODE 31, be sure to specify the address with the high-order bit set to 1.

SERVPARM = parmlist address
Specify the address of a server parameter list. The address must be in the same AMODE as the server address. The server parameter should point to any resources that the initialization/termination program obtained for the server, such as shared storage, data sets, and locks. MVSSERV passes this parameter to the server when it calls the server to handle a service request.

MF = (E,parmlist name)
Specify the name of a five-word area that will contain:

- The addresses of the CPRB and CED (two words)
- The server entry point address and server parameter address (the *define server parameter area*-- three words).

parmlist name MF = L
Generates storage for five words (20 bytes) of storage to contain the addresses of the CPRB and CED (two words) and the define server parameter area (three words). The DEFSERV macro fills in this storage. The label on this statement must match the **parmlist name** used in the MF keyword of the execute form of the macro.

Note: If the issuing program is reentrant, it must use the GETMAIN macro to allocate the five words of storage.

Results of the DEFSERV Macro

The DEFSERV macro fills in fields of a CPRB that MVSSERV uses to identify the server name with the server's address and parameter list. The CPRB and its significant fields are shown in Figure 3-6.

The DEFSERV Request CPRB

Field Label	Byte Offset	Byte Length	Contents
CRBF1	0(0)	1	The control block's version number (first four bits) and modification level number (last four bits).
	1(1)	2	Reserved
CRBF4	3(3)	1	The type of request (Hex '03' indicates a Define Server request).
CRBCPRB	4(4)	4	The value of C'CPRB'.
	8(8)	8	Reserved
CRBSNAME	16(10)	8	The server name specified in the DEFSERV parameter SERVNAME.
	24(18)	32	Reserved
CRBRQPLN	56(38)	4	The value x'0003', indicating the length of the define server parameter area.
CRBRQPRM	60(3C)	4	The address of the define server parameter area.
	64(40)	48	Reserved

Figure 3-6. CPRB Control Block used to Define a Server

Note: All fields shown are set by the DEFSERV macro.

The Define Server Parameter Area

The field CRBRQPRM of the DEFSERV CPRB points to the *define server parameter area.* This area, created by the DEFSERV MACRO, points to the following:

- The server entry point.
- The server parameter - resources passed to the server when it is called.

Figure 3-7 shows the layout of the define server parameter area.

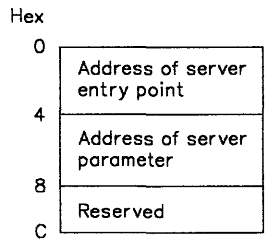


Figure 3-7. The Define Server Parameter Area

Return Codes from the DEFSERV Macro

When a program resumes control after issuing the DEFSERV macro, the program must inspect register 15 for a return code from MVSSERV. The possible return codes are shown in Figure 3-8.

Return Code Dec(Hex)	Meaning
0(0)	The DEFSERV request was successful.
4(4)	The DEFSERV request was unsuccessful. The program must inspect the MVSSERV return code in the CPRB (field CRBCRTNC) to determine the cause of the failure. See "Return Codes from the DEFSERV CPRB."
8(8)	The CPRB is not valid. Data fields in the CPRB, such as CPRBF4, contained information that was not valid.
12(0C)	The CPRB is not valid. 24-bit addresses in the CPRB were not valid (the high-order byte of the addresses was not 0).
16(10)	The CPRB is not valid. The address of the CPRB or addresses within the CPRB were checked and found to be not valid, causing the router to ABEND.

Figure 3-8. Return Codes from the DEFSERV Macro

Return Codes from the DEFSERV CPRB

If the return code in register 15 is 0, the program must check for an additional return code in the DEFSERV CPRB, which MVSSERV returns after it is finished with the DEFSERV macro. The additional return code, if any, is in field CRBCRTNC, as shown in Figure 3-9.

The DEFSERV Reply CPRB

Field Label	Byte Offset	Byte Length	Contents
	0(0)	12	Reserved
CRBCRTNC	12(0C)	4	The return code from MVSSERV in response to the DEFSERV request CPRB. Contains one of the following return codes: 0000 Processing was successful. 0148 Request failed; another defined server has the same name. 0152 Request failed; MVSSERV error.
	16(10)	96	Reserved

Figure 3-9. CPRB Control Block for the DEFSERV Reply

Recovery

Like the server, the initialization/termination program can have its own recovery routine. If the initialization/termination program fails and does not recover, MVSSERV traps the error and marks all the servers in the subtask as unavailable.

If the initialization/termination program provides recovery, it must use the ESTAE 0 option to delete its recovery environment before returning control to MVSSERV after initialization and after termination.

Ending Initialization

When the initialization/termination program is finished with initialization, it must return control to MVSSERV with a return code of 0 (successful) or 4 (unsuccessful) in register 15. If the return code is 4, MVSSERV marks all the servers in the subtask as unavailable.

Termination

Before MVSSERV ends, it calls the initialization/termination program again to end the servers and free any resources obtained for them. The termination input to the initialization/termination program is shown in Figure 3-10 and Figure 3-11, with the significant fields indicated.

Register	Contents
1	Address of parameter list
14	Return address
15	Address of initialization/termination program

Hex	
0	Address of INITTERM
4	Address of CED

Figure 3-10. Registers Passed at Termination

Field Label	Byte Offset	Byte Length	Contents
INTINIT❶	0(0)	4	Initialization or termination indicator. X'0001' indicates that the call is for termination.
INTWALEN	4(4)	4	Work area length. The length of a work area, if any, specified at initialization.
INTWAPTR	8(8)	4	Work area address. The address of a work area, if any, specified at initialization.
INTSNAME❷	12(C)	8	Name of last server to send a reply. If the initialization/termination program defined this server and the last reply was not received successfully (see INTRSN), the initialization/termination program may take appropriate action; for example, cancelling the last service performed.
INTRSN❸	20(14)	4	Return code for last reply. Contains one of the following return codes: 0(0) Processing was successful. 4(4) The reply *may not* have been successfully received by the requester. 8(8) The reply was not successfully received by the requester. 10(A) The reply CPRB from the server was not valid.
	24(18)	4	Reserved
INTENVRN	28(1C)	4	CPPL address (See Figure 3-4).
	32(20)	4	Reserved

Figure 3-11. INITTERM Control Block with Termination Input

Notes:

❶ Check for initialization or termination.
❷ Check the name of the last server to send a reply (optional).
❸ If the last server was defined by the initialization/termination program, check the status of the last reply (optional).

Freeing Resources

Your initialization/termination program must release any resources that it obtained. The program must:

- Release any locks that were obtained.
- Free any storage that was obtained by GETMAIN.
- Close any open data sets.

Deleting the Servers

The initialization/termination program must delete any servers that it loaded. The following assembler language example shows how to delete a server.

```
DELETE EP=server name               Delete the server
  .
  .
```

When finished, the initialization/termination program must return control to MVSSERV with a return code of 0 (successful) or 4 (unsuccessful) in register 15.

Compiling or Assembling an Initialization/Termination Program

After writing a server, you need to compile or assemble it and link it to a load module. For information about preparing and running a program in TSO/E, refer to Section III of the *TSO/E User's Guide*.

Sample Initialization/Termination Program

The initialization/termination program in Figure 3-12 corresponds to the sample server in Figure 2-6. The initialization/termination program does the following:

- Loads the server.
- Opens two data sets (one for customer records and one for accounts receivable).
- Points to the two data sets in a server parameter list.
- Issues the DEFSERV macro.
- Cleans up at termination.

Note that this program is not reentrant.

```
***********************************************************************
* TITLE: IBMINTRM MAINLINE
*
* LOGIC: Perform server initialization/termination.
*
* OPERATION:
* 1. Determine if the program is in initialization or termination.
* 2. If initialization:
*    - Call INIT_SERVER to load and define the server to MVSSERV.
* 3. Else, termination:
*    - Call CLEAN_UP to delete the server.
* 4. Return to caller with return code.
***********************************************************************
IBMINTRM CSECT
IBMINTRM AMODE 24
IBMINTRM RMODE 24
         STM   14,12,12(13)         Save the caller's registers.
         LR    12,15                Establish addressability within
         USING IBMINTRM,12          this CSECT.
         ST    13,SAVEAREA+4        Save the caller's save area address.
         LA    15,SAVEAREA          Obtain our save area address.
         ST    15,8(,13)            Chain it in the caller's save area.
         LR    13,15                Point register 13 to our save area.
         L     2,0(,1)              Load the init/term area address.
         USING INITTERM,2           Establish addressability to it.
         L     3,4(,1)              Load the CED address.
         LA    4,0                  Load the initialization value (0).
         C     4,INTINIT            Are we in initialization?
         BNE   TERMINATE            No, then we must terminate.
         BAL   14,INIT_SERVER       Yes, Call INIT_SERVER.
         B     EXIT                 Yes, leave the init/term program.
TERMINATE DS   0H
         BAL   14,CLEAN_UP          Call CLEAN_UP.
EXIT     L     13,SAVEAREA+4        Restore caller's save area address.
         L     14,12(,13)           Restore the caller's registers
         LM    0,12,20(13)          except for 15 (return code).
         BR    14                   Return to caller with return code.
***********************************************************************
* TITLE: INIT_SERVER
*
* LOGIC: Load the server and define it to MVSSERV.
*
* OPERATION:
* 1. Load the server.
* 2. Open the necessary data sets.
* 3. If the data sets are not open:
*    - Set a bad return code.
* 4. Else:
*    - Issue the DEFSERV macro to attempt to define
*      the server to MVSSERV.
* 5. Save the return codes.
* 6. Return to the mainline.
***********************************************************************
INIT_SERVER DS 0H
         STM   14,12,12(13)         Save the caller's registers.
         ST    13,SUBSAVE+4         Save the caller's save area address.
         LA    15,SUBSAVE           Obtain our save area address.
         ST    15,8(,13)            Chain it in the caller's save area.
         LR    13,15                Point register 13 to our save area.
```

Figure 3-12 (Part 1 of 4). Sample Initialization/Termination Program

```
************************************************************************
* Issue the LOAD macro to load the server into storage.
**********************************************************************
         LOAD  EP=IBMABASE
         LR    5,0                     Save IBMABASE's address.
**********************************************************************
* Open the customer record files.
**********************************************************************
         OPEN  (DCBIN,(INPUT),DCBOUT,(OUTPUT),DCBLOG,(OUTPUT)),        *
               MF=(E,OPEN_LIST)
         LA    15,0                    Initialize the return code.
         LA    4,DCBIN                 Load the address of the DCB.
         USING IHADCB,4                Establish addressability to the
*                                      fields in the DCB mapping.
         TM    DCBOFLGS,DCBOFOPN       Test for successful open.
         BO    INPUT_OK                If open don't set bad return code.
         LA    15,4                    Bad open, set bad return code.
INPUT_OK DS    0H
         LA    4,DCBOUT                Load the address of the DCB.
         TM    DCBOFLGS,DCBOFOPN       Test for successful open.
         BO    OUTPUT_OK               If open don't set bad return code.
         LA    15,4                    Bad open, set bad return code.
OUTPUT_OK DS   0H
         LA    4,DCBLOG                Load the address of the DCB.
         TM    DCBOFLGS,DCBOFOPN       Test for successful open.
         BO    LOG_OK                  If open don't set bad return code.
         LA    15,4                    Bad open, set bad return code.
LOG_OK   DS    0H
         DROP  4
         LTR   15,15                   Did the opens work?
         BNZ   LEAVE                   No, then don't define the server.
**********************************************************************
* Issue the DEFSERV macro to define the server to MVSSERV.
**********************************************************************
         LA    4,CHSDCPRB              Load the address of the CPRB.
         LA    6,DCBIN                 Get INPUT DCB address.
         ST    6,SERVER_PARM           Place it in the server parameter.
         LA    6,DCBOUT                Get OUTPUT DCB address.
         ST    6,SERVER_PARM+4         Place it in the server parameter.
         LA    6,DCBLOG                Get OUTPUT DCB address.
         ST    6,SERVER_PARM+8         Place it in the server parameter.
         LA    6,OPEN_LIST             Get address of OPEN list form.
         ST    6,SERVER_PARM+12        Place it in the server parameter.
         LA    6,CLOSE_LIST            Get address of CLOSE list form.
         ST    6,SERVER_PARM+16        Place it in the server parameter.
         LA    6,SERVER_PARM           Load the address of the server
*                                      parameter.
         DEFSERV CPRB=(4),CED=(3),SERVNAME=SERVER_NAME,               *
               SERVEPA=(5),SERVPARM=(6),MF=(E,PARMLIST)
         LTR   15,15                   Check the return code.
         BZ    CPRB_RET                If it is zero, check the CPRB's.
         LA    15,4                    Otherwise, set a bad return code,
         B     LEAVE                   and leave.
CPRB_RET DS    0H
         L     15,CRBCRTNC             Obtain the CPRB's return code.
         LTR   15,15                   Check the return code.
         BZ    LEAVE                   If it is zero, then leave.
         LA    15,4                    Otherwise, set a bad return code.
```

Figure 3-12 (Part 2 of 4). Sample Initialization/Termination Program

```
LEAVE    DS    0H
         L     13,SUBSAVE+4          Restore caller's save area address.
         L     14,12(,13)            Restore the caller's registers
         LM    0,12,20(13)           except for 15 (return code).
         BR    14                    Return to caller with return code.
**********************************************************************
* TITLE: CLEAN_UP
*
* LOGIC: Delete the server from storage.
*
* OPERATION:
* 1. Delete the server.
* 2. Close the necessary data sets.
* 3. Return to the mainline.
**********************************************************************
CLEAN_UP DS 0H
         STM   14,12,12(13)          Save the caller's registers.
         ST    13,SUBSAVE+4          Save the caller's save area address.
         LA    15,SUBSAVE            Obtain our save area address.
         ST    15,8(,13)             Chain it in the caller's save area.
         LR    13,15                 Point register 13 to our save area.
**********************************************************************
* Issue the DELETE macro to delete the server from storage.
**********************************************************************
         DELETE EP=IBMABASE
**********************************************************************
* Close the customer record files.
**********************************************************************
         CLOSE (DCBIN,,DCBOUT,,DCBLOG),MF=(E,CLOSE_LIST)
         LA    15,0
         L     13,SUBSAVE+4          Restore caller's save area address.
         L     14,12(,13)            Restore the caller's registers
         LM    0,12,20(13)           except for 15 (return code).
         BR    14                    Return to caller with return code.
**********************************************************************
* Data section.
**********************************************************************

SERVER_NAME DC CL8'IBMABASE'         Server name.

**********************************************************************
* OPEN macro (list form).
**********************************************************************

OPEN_LIST OPEN (,(INPUT),,(OUTPUT),,(OUTPUT)),MF=L

**********************************************************************
* CLOSE macro (list form).
**********************************************************************

CLOSE_LIST CLOSE (,,,,,),MF=L
```

Figure 3-12 (Part 3 of 4). Sample Initialization/Termination Program

```
***********************************************************************
* DCB macro (input).
***********************************************************************

DCBIN    DCB  DDNAME=CUSTRECS,DSORG=PS,MACRF=GM

***********************************************************************
* DCB macro (output).
***********************************************************************

DCBOUT   DCB DDNAME=ACCTRECS,DSORG=PS,MACRF=PM

***********************************************************************
* DCB macro (log).
***********************************************************************

DCBLOG   DCB DDNAME=LOGTRANS,DSORG=PS,MACRF=PM
SAVEAREA DC    18F'0'                IBMINTRM's save area.
SUBSAVE  DC    18F'0'                IBMINTRM subroutine's save area.

***********************************************************************
* Server parameter list, contains the addresses of:
*        The INPUT DCB
*        The OUTPUT DCB
*        The LOG DCB
*        The OPEN LIST FORM
*        The CLOSE LIST FORM
***********************************************************************

SERVER_PARM DC 5A(0)

***********************************************************************
* Issue the DEFSERV macro list form to supply a parameter list.
***********************************************************************

PARMLIST DEFSERV  MF=L

***********************************************************************
* CPRB
***********************************************************************

         CHSDCPRB DSECT=NO

***********************************************************************
* INIT/TERM area mapping.
***********************************************************************

         INITTERM DSECT=YES

***********************************************************************
* DCB macro mapping.
***********************************************************************
         DCBD  DSORG=PS
         END   IBMINTRM
```

Figure 3-12 (Part 4 of 4). Sample Initialization/Termination Program

Chapter 4. How to Install a Server

After a server has been written, compiled or assembled, and linked to a load module, you must install the server to make it available to users and to MVSSERV.

Installation is a two-step process. The steps are:

1. Installing the server (and its initialization/termination program).

2. Naming the server's initialization/termination program in an input parameter data set.

Installing the Server

You can install a server in one of two ways:

- In a STEPLIB.
- In a system library in the linklist concatenation.

In a STEPLIB

You can install a server in a STEPLIB in a user's logon procedure. This method of installation lets you restrict the server to specific users, and is especially useful when testing a new server.

To install an existing server in a STEPLIB, add the following JCL in the user's logon procedure:

```
//STEPLIB DD DSN=data.set.name,DISP=SHR
```

If the server's initialization/termination program is not in the same data set and is also being tested, install it in the STEPLIB in the same way.

In a System Library

To make a server available to all system users, copy the server and its initialization/termination program to a member or members of a system library in the linklist concatenation, such as data set SYS1.LPALIB.

Input Parameter Data Set

Before issuing the MVSSERV command, you must name the initialization/termination programs in the input parameter data set. From this input, MVSSERV invokes the initialization/termination programs, which define the servers to MVSSERV.

Allocating the Input Parameter Data Set

The input parameter data set must have the following characteristics:

- ddname -- CHSPARM
- logical record length -- 80
- format -- fixed or fixed block.

You can create the input parameter data set with the following command:

```
ALLOCATE F(CHSPARM) DA('data.set.name') NEW LRECL(80) RECFM(F)
```

To make the input parameter data set available to an MVSSERV user, install the existing data set in the user's logon procedure, or in a CLIST or ISPF dialog that issues MVSSERV for the user.

- In a logon procedure, you can use the following JCL:

  ```
  //CHSPARM DD DSN=data.set.name,DISP=SHR
  ```

- In a CLIST or ISPF dialog, you can use the following command:

  ```
  ALLOCATE F(CHSPARM) DA('data.set.name') SHR
  ```

Be sure that the user has authority to access the input parameter data set.

Initializing the Input Parameter Data Set

Each record of the input parameter data set must contain the name of an initialization/termination program, starting in column 1. The name must conform to MVS program naming conventions; it can have up to eight characters, including the characters A-Z, 0-9, @, #, and $. The first character cannot be 0-9.

For example, type the name of your initialization/termination program in the following position on a line of the input parameter data set:

```
----+----1----+----2----+----3----+----4----+----5---
pgm-name
```

Additional MVSSERV Data Sets

Along with the input parameter data set, you can allocate optional data sets to contain MVSSERV diagnosis information. These diagnostic data sets can also be allocated in a user's logon procedure, in a CLIST or ISPF dialog that invokes MVSSERV, or in ready mode TSO. The diagnostic data sets and their functions are as follows:

- Trace data set -- receives trace data and messages
- Dump data set -- receives system dump data
- Dump suppression data set -- lets you specify ABEND codes for which you do not want dumps to be taken.

Trace Data Set

You can specify a data set to receive trace data from an MVSSERV session. The level of trace data depends on the parameter with which MVSSERV is invoked (see Chapter 5, "Testing and Diagnosis" for a list of these parameters):

- TRACE parameter -- records events in the MVSSERV session, including any internal problems.
- IOTRACE parameter -- records the TRACE information and communications with the PC, including data transmissions and the contents of the CPRB.

Allocating the Trace Data Set

The trace data set must have the following characteristics:

- ddname -- CHSTRACE
- logical record length -- 80
- format -- fixed or fixed block.

You can create the trace data set with the following command:

```
ALLOCATE F(CHSTRACE) DA('data.set.name') NEW LRECL(80) RECFM(F)
```

To make the trace data set available to an MVSSERV user, install the existing data set in the user's logon procedure, or in a CLIST or ISPF dialog that issues MVSSERV for the user. Users must have their own trace data sets.

- In a logon procedure, you can use the following JCL:

  ```
  //CHSTRACE DD DSN=data.set.name,DISP=OLD
  ```

- In a CLIST or ISPF dialog, you can use the following command:

  ```
  ALLOCATE F(CHSTRACE) DA('data.set.name') OLD
  ```

Refer to Chapter 5, "Testing and Diagnosis" for more information about the MVSSERV trace parameters and syntax.

Note: Use of the trace parameters may affect MVSSERV performance. Therefore, your installation may decide not to use the MVSSERV trace parameters for regular production work. However, for testing or diagnosing servers, or requesting diagnosis help from IBM service personnel, use MVSSERV with the trace data set and the parameter TRACE or IOTRACE.

Dump Data Set

You can allocate a data set to receive dump data from an MVSSERV session. If you allocate a dump data set, MVSSERV provides a dump at the first occurrence of an ABEND.

Allocating the Dump Data Set

The dump data set must be associated with one of the following ddnames:

- SYSUDUMP, for a formatted dump of the MVSSERV storage area
- SYSMDUMP, for an unformatted dump of the MVSSERV storage area and the system nucleus
- SYSABEND, for a formatted dump of the MVSSERV storage area including the local system queue area and IOS control blocks.

The exact contents of a dump depend on the default options specified in your SYS1.PARMLIB members SYSUDUMP, SYSMDUMP, and SYSABEND. These system default options can be changed using the CHNGDUMP command. For further information about the possible dump data sets and how to read them, refer to *MVS/XA Diagnostic Techniques*.

To make a dump data set available to an MVSSERV user, install the existing data set in the user's logon procedure, or in a CLIST or ISPF dialog that issues MVSSERV for the user. Users must have their own dump data sets.

- In a logon procedure, you can use the following JCL:

  ```
  //SYSUDUMP DD DSN=data.set.name,DISP=OLD
  ```

- In a CLIST or ISPF dialog, you can use the following command:

  ```
  ALLOCATE F(SYSUDUMP) DA('data.set.name') OLD
  ```

Dump Suppression Data Set

If you use a dump data set, you can eliminate unnecessary dumps by using the MVSSERV dump suppression data set. The dump suppression data set lets you specify ABEND codes for which you do not want to receive dumps from MVSSERV. For example, you can specify ABEND code 913 to avoid dumps caused by unsuccessful OPEN macro requests.

Allocating the Dump Suppression Data Set

The dump suppression data set must have the following characteristics:

- ddname -- CHSABEND
- logical record length -- 80
- format -- fixed or fixed block.

You can create the dump suppression data set with the following command:

```
ALLOCATE F(CHSABEND) DA('data.set.name') NEW LRECL(80) RECFM(F)
```

To make the dump suppression data set available to an MVSSERV user, allocate the existing data set in the user's logon procedure, or in a CLIST or ISPF dialog that issues MVSSERV for the user.

- In a user's logon procedure, you can use the following JCL:

  ```
  //CHSABEND DD DSN=data.set.name,DISP=SHR
  ```

- In a CLIST or ISPF dialog, you can use the following command:

  ```
  ALLOCATE F(CHSABEND) DA('data.set.name') SHR
  ```

Initializing the Dump Suppression Data Set

Each 80-byte record of the dump suppression data set must be in the following format:

```
OFFSET      LENGTH      DESCRIPTION

+0           3          EBCDIC ABEND code
                              in hex. for system ABENDs
                              in decimal for user ABENDs
+3          1           Reserved
+4          4           EBCDIC REASON code (hex)
+8          1           Reserved
+9          1           EBCDIC dump action field:
                              0 = Do not dump
                              1 = SNAP Dump
+10         70          Reserved
```

Use leading zeros for ABEND and reason codes as needed. For example, to suppress dumps from ABENDs of the OPEN macro (ABEND code 913) caused by RACF authorization failure (reason code 38), type the following on a line of the dump suppression data set:

```
----+----1----+----2----+----3----+----4----+----5---

913 0038 0
```

You can replace the first character of the ABEND code and the entire reason code with X's, to signify all values. For example, to suppress dumps from all reason codes of the 913 macro, type the following:

```
----+----1----+----2----+----3----+----4----+----5---

913 XXXX 0
```

And to suppress dumps for all ABEND codes ending in 13, type the following:

```
----+----1----+----2----+----3----+----4----+----5---

X13 XXXX 0
```

For a list of ABEND and reason codes, refer to *MVS/XA Message Library: System Codes* and *MVS/XA Message Library: System Messages.*

Chapter 5. Testing and Diagnosis

Testing Servers

After you have written and installed a server, you will want to test it. You can first test the server as a member of a STEPLIB. When you are satisfied that the server works correctly, you can then re-install and test the server again for general use in a system library.

When testing a server, you need to start an MVSSERV session on TSO/E and, on the PC, invoke the requester program that requests the server. The requester must name the server and service function, and pass any data and parameters that the service function needs.

Steps for Testing Servers

Use the following steps to test a server.

1. Make sure that you have the necessary data sets available for your MVSSERV session. For information about allocating the data sets, refer to Chapter 4, "How to Install a Server." You should have the following data sets available:

   - The server and its initialization/termination program, installed in a STEPLIB in your logon procedure.

   - An input parameter data set, containing the name of the initialization/termination program.

   - A trace data set, to receive MVSSERV messages.

   You may also want to have the dump data set and the dump suppression data set as described in Chapter 4, "How to Install a Server."

2. To start the MVSSERV session, logon to TSO/E and issue the MVSSERV command.

   MVSSERV has the following syntax, with the default underscored:

   ```
   MVSSERV        ┌ NOTRACE ┐
                  │ TRACE   │
                  └ IOTRACE ┘
   ```

   For a test, use the TRACE parameter. TRACE produces messages in the trace data set about internal MVSSERV events, including server failures.

3. Switch to the PC session. (If you are using a PC other than the 3270 PC, issue the appropriate TSO/E Enhanced Connectivity Facility command for the PC.)

4. Invoke the requester that corresponds to the server you want to test.

5. Note any messages from the requester. The requester should issue messages about any non-zero return codes from the server.

6. Verify that the request was satisfied.

7. Switch back to the host session and press the END PF key to end MVSSERV.

8. Note any messages that appear on your TSO/E screen. In TSO/E, you can obtain online help for MVSSERV terminal messages by typing the message ID (CHSxxxxxxx) in the following command:

   ```
   HELP MVSSERV MSG(message ID)
   ```

9. Access the trace data set. Because you used the TRACE parameter when invoking MVSSERV, the trace data set should have recorded informational and error messages about events in the session and any errors that may have occurred.

   See the following section, "Diagnosing Servers," for information about reading the trace data set messages.

10. When the server works properly, you may want to copy it to LPALIB to make it available to other users. Make sure that the other users allocate the input parameter data set in their logon procedures, in a CLIST or ISPF dialog, or in line mode TSO. Then test the server again in its new installation.

Diagnosing Servers

Reading the Trace Data Set

You can edit, browse, or print the trace data set to see messages about your most recent MVSSERV session. For explanations of the messages, refer to *TSO Messages*. The message explanations include information about what action, if any, you must take when you see a message.

The messages are preceded by message IDs beginning with the letters CHS. The last character of the message ID indicates the type of message: I for informational messages, and E for error messages.

Informational Messages

Informational messages provide information about the status of the MVSSERV session and data transmissions. Informational messages also describe exception conditions, such as server failures, which do not cause MVSSERV to end.

Error Messages

Error messages describe conditions that cause MVSSERV to end abnormally. The conditions may be internal MVSSERV errors, system errors, or input errors. Possible input errors include incorrect syntax of the MVSSERV command, a missing input parameter data set, or an invalid CPRB address.

Internal and system errors often require help from IBM service personnel, but you may be able to correct input errors by following directions in the message explanations.

The Internal Execution Path Trace Table

The last message in the trace data set, CHSTTP02I, displays MVSSERV's internal execution path trace table. MVSSERV makes an entry in the table whenever one MVSSERV module calls another. Thus, the table provides a history of MVSSERV module calls and makes it possible to track internal MVSSERV errors. For information about how to read the internal execution path trace table, refer to *TSO/E Command Processor Logic Volume IV*.

Figure 5-1 shows a sample of a trace data set obtained using the TRACE parameter. The message IDs are in the left-hand column of the figure.

```
CHSCMI02I The control unit supports Read Partitioned Queries.
CHSTCA13I DFT access method driver is active.
CHSTRR01I CPRB request at 12:37:07 server=SERVER2  function=0001:
CHSRUTR06I Server request failed; SERVER2 is in an inactive task.
CHSDCOM09I User pressed the PF3 key, requesting termination.
CHSCPS08I MVSSERV is ending.
CHSTTP01I Internal trace table follows.  Last entry is 019:
CHSTTP02I 000 TIOR     001 TIOR     002 TIOR     003 TIOR
CHSTTP02I 004 TIOR     005 TIOR     006 TIOR     007 TIOR
CHSTTP02I 008 TIOR     009 TIOR     010 TIOR     011 TIOR
CHSTTP02I 012 TIOR     013 TIOR     014 TIOR     015 TIPM
CHSTTP02I 016 TIOR     017 TIOR     018 TIOR     019 TTTP
CHSTTP02I 020 TSRV     021 TRUTR    022 TRUTR    023 TRUTR
CHSTTP02I 024 TRUTR    025 TCMI     026 TLMP     027 TIOR
CHSTTP02I 028 TDCA     029 HRES     030 TDCOM    031 TCH7
CHSTTP02I 032 TC7H     033 PACK     034 TINF     035 TTRL
CHSTTP02I 036 TLMP     037 TIOR     038 HQNL     039 TDCOM
    .
    .
    .
```

Figure 5-1. Sample Trace Data Set

For explanations of the messages that appear in your MVSSERV trace data set, refer to *TSO/E Messages.* The message explanations include information about what action, if any, you must take when you see a message.

Glossary

This glossary defines important terms and abbreviations used in this manual. If a term is not defined here, refer to the index or *IBM Vocabulary for Data Processing, Telecommunications, and Office Systems*, GC20-1699.

A

ABEND. Abnormal end of task.

address. A character or group of characters that identify a location in storage, a device in a system or network, or some other data source.

addressing. (1) In data communications, the way that the sending or control station selects the station to which it is sending data. (2) A means of identifying storage locations. (3) Specifying an address or location within a file.

allocate. To assign a resource, such as a disk file or a diskette.

American National Standard Code for Information Interchange (ASCII). The code developed by ANSI for information interchange among data processing systems, data communications systems, and associated equipment. The ASCII character set consists of 7-bit control characters and symbolic characters.

application program (m). The instructions to a computer to accomplish processing tasks for a user.

ASCII. See *American National Standard Code for Information Interchange.*

assembler language. A source language that includes symbolic machine language statements in which there is a one-to-one correspondence with instruction formats and data formats of the computer.

B

buffer. An area of storage, temporarily reserved for performing input or output, into which data is read, or from which data is written.

C

compile. To translate a program written in a high-level programming language into a machine language program.

configuration. The arrangement of a computer network as defined by the nature, number, and characteristics of the computers and other machines that are attached to the network.

connectivity programming request block (CPRB). An interface control block used by requesters and servers to communicate information.

constant. A value that does not change. Contrast with *variable.*

control program. (1) A computer program designed to schedule and supervise the execution of programs in a computer system. (2) The set of functions in the IBM 3270 Personal Computer allowing you to create windows, change their size and position, and hide and enlarge them; use autokeying and copying; save and restore autokey recordings, notepad contents, and screen profiles; and transfer files between System/370 and PC sessions.

CPRB. See *connectivity programming request block.*

D

data set. The major unit of data storage and retrieval, consisting of a collection of data in one of several prescribed arrangements and described by control information to which the system has access.

default. A value that is used when nothing is specified by the user.

device. Equipment that is designed for a specific purpose and that attaches to your computer, for example, a printer, disk drive, or display station.

DFT. See *distributed function terminal.*

distributed function terminal (DFT). (1) An operational mode. (2) A hardware/software protocol used to communicate between a terminal and a 3274 control unit.

E

EBCDIC. See *extended binary-coded decimal interchange code.*

end user. (1) The ultimate source or destination of information flowing through a system. (2) A person, process, program, device, or system that employs a user application network for the purpose of data processing and information exchange. See also *user.*

Enhanced Connectivity Facility. The strategy for sharing services and resources in a heterogeneous network.

Enhanced Connectivity Facility management services. Any service provided through the Server-Requester Programming Interface.

Enhanced Connectivity Facility base component. The base code for an Enhanced Connectivity Facility environment, including a router.

entry point. An address or label of an instruction performed upon entering a computer program, a routine, or a subroutine. A program may have several different entry points, each corresponding to a different function or purpose.

extended binary-coded decimal interchange code (EBCDIC). A set of 256 characters, each represented by 8 bits.

F

field. (1) An area in a record or panel used to contain a particular category of data. (2) The smallest component of a record that can be referred to by a name. (3) An area in a structured file defined in the form used to enter and display data. Fields are defined using either text data paths or tree data paths. See *protected field* and *unprotected field.*

file. A collection of related data that is stored and retrieved by an assigned name.

file name. The name used by a program to identify a file.

file naming. Assigning a specific name to a data file on the System/370 or the PC.

format. (1) A defined arrangement of such things as characters, fields, and lines, usually used for displays, printouts, or files. (2) The pattern which determines how data is recorded.

H

hardware. The equipment, as opposed to the programming, of a computer system.

hex. See *hexadecimal.*

hexadecimal. Pertaining to a system of numbers to the base sixteen; hexadecimal digits range from 0 (zero) through 9 (nine) and A (ten) through F (fifteen).

host. A computer that receives requests for services from another personal computer in the Enhanced Connectivity Facility environment.

host computer. The primary and controlling computer in a network; usually provides services such as computation, data base access, and advanced programming functions. Sometimes referred to as a host processor or mainframe.

I

ID. Identification.

initialize. To set counters, switches, addresses, or contents of storage to starting values.

interface. A shared boundary between two or more entities. An interface might be a hardware or software component that links two devices or programs together.

invoke. To start a command, procedure, or program.

K

keyword. One of the predefined words of a programming language; a reserved word.

L

load. (1) To move data or programs into storage. (2) To place a diskette into a diskette drive. (3) To insert paper into a printer.

load module. (1) A program unit that is suitable for loading into main storage for execution; is usually the output of a linkage editor.

LOGOFF. A command for ending a host session.

LOGON. A command for beginning a host session.

M

macro. (1) A single instruction representing a set of instructions. (2) The name of a "pseudo command" that performs the functions of many commands, by combining those commands under the common label described above.

main task. (1) The main program within a partition in a multiprogramming environment. See also *subtask*.

message. (1) A response from a program to inform you of a condition that may affect further processing of a current program. (2) Information sent from one user in a multi-user operating system to another user.

module. A discrete programming unit that usually performs a specific task or set of tasks. Modules are subroutines and calling programs that are assembled separately, then linked to make a complete program.

MVS router. A program running under TSO/E that uses the Server-Requester Programming Interface (SRPI) to route requests from the PC to the corresponding server on the host. The MVS router is part of the MVSSERV command processor in TSO/E Release 3.

MVSSERV. (1) A program that provides the Server-Requester Programming Interface (SRPI) and a service request manager on an IBM System/370 using the TSO/E (time sharing option/extensions) on MVS/XA. (2) A command processor in TSO/E Release 3. It initializes, terminates, and provides recovery for an Enhanced Connectivity Facility session between a PC and a host system. It also establishes communication and routes requests from the PC user to the corresponding server on the host.

O

object module. A set of instructions in machine language. The object module is produced by a compiler or assembler from a subroutine or source module and can be input to the linking program. The object module consists of object code. See *module*.

operating environment. The operating environment at the node, generally referred to as the operating system. It provides services to the Enhanced Connectivity Facility implementation, requesters, and servers.

operating system. Software that controls the running of programs; in addition, an operating system can provide services such as resource allocation, scheduling, input/output control, and data management.

P

parameter. (1) Information that the user supplies to a panel, command, macro, or function. (2) In Enhanced Connectivity Facility, information that a requester or server passes to a send_request or send_reply function.

personal computer. A properly-configured IBM Personal Computer or 3270 Personal Computer that allows communication between IBM requesters/servers programs. These communicating programs reside on both a personal computer and host system.

presentation space. In the 3270 PC environment, a region in computer memory (either host, PC, or notepad) that can be displayed, in whole or part, in a window on the screen. For example, a spreadsheet consisting of 4,096 rows and 4,096 columns is a presentation space that cannot be viewed in its entirety on one screen. However, it can be viewed in sequence, part-by-part. Sometimes "presentation space" is used synonymously with session, although not all presentation spaces are sessions. (Technically, some internal components are presentation spaces.)

program. A file containing a set of instructions conforming to a particular programming language syntax.

prompt. A displayed request for information or user action.

protocol. In data communications, the rules for transferring data.

Q

query. The action of searching data for desired information.

queue. A line or list formed by items waiting to be processed.

R

record. A collection of fields treated as a unit.

register. A storage area, in a computer, capable of storing a specified amount of data such as a bit or an address. Each register is 32 bits long.

reply. The answer to a service request that came from the server.

request. The requirement for service that came from the requester.

requester. The program that relays a request to another computer through the server-requester programming interface (SRPI). Contrast with *server.*

required parameter. A parameter that must have a defined option. The user must provide a value if no default is supplied.

return code. A value that is returned by a subroutine or function to indicate the results of an operation of the program.

router. An Enhanced Connectivity Facility program which interprets requests for services and directs them to the applicable server. See also *MVS router, Server-Requester Programming Interface.*

S

server (m). The program that responds to a request from another computer through the Server-Requester Programming Interface (SRPI). Contrast with *requester.*

server return code. A 4-byte return code presented to the server's Enhanced Connectivity Facility Management Services. The content and format of the return status are defined by the individual server.

server-requester programming interface (SRPI). (1) A protocol between requesters and servers in an Enhanced Connectivity Facility network; includes the protocol to define Enhanced Connectivity Facility subsystem. (2) The interface that enables Enhanced Connectivity Facility between requesters and servers in a network.

session. A connection between two stations that allows them to communicate.

software. Programs, procedures, rules, and any associated documentation pertaining to the operation of a computer system. Contrast with *hardware.*

SRPI. See *server-requester programming interface.*

storage. (1) The location of saved information. (2) In contrast to memory, the saving of information on physical devices such as disk or tape. See *memory.*

subtask. A task that is initiated and terminated by a higher order task. Contrast with *main task.*

subroutine. (1) A sequential set of statements that can be used in one or more computer programs and at one or more points in a computer program. (2) A routine that can be part of another routine.

syntax. The rules for the construction of a command or program.

T

trace. To record data that provides a history of events occurring in the system.

transparent. In data transmission, pertaining to information that is not recognized by the receiving program or device as transmission control characters.

U

user. Anyone who requires the services of a computer system. See also *end user*.

V

variable. A name used to represent a data item with a value that can change while the program is running. Contrast with *constant*.

Index

TSO Extensions Programmer's Guide to the Server-Requester Programming Interface for MVS/Extended Architecture

SC28-1309-0 S370-39

Printed in U.S.A.

TSO Extensions
Programmer's Guide
to the Server-Requester
Programming Interface for
MVS/Extended Architecture

READER'S
COMMENT
FORM

SC28-1309-0

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note: ***Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.***

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, company, mailing address, and date:

____

____

____

____

What is your occupation? ______________________________

How do you use this publication? ______________________________

Number of latest Newsletter associated with this publication: ____________________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

TSO/E Programmer's Guide to the Server-Requester Programming Interface for MVS/XA

SC28-1309-0 S370-39

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 PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department D58, Building 921-2
PO Box 390
Poughkeepsie, New York 12602

Fold and tape Please Do Not Staple Fold and tape

Printed in U.S.A.

IBM®