z/OS
Version 2 Release 3

TSO/E Guide to the Server-Requester Programming Interface

SA32-0980-30

Note

Before using this information and the product it supports, read the information in "Notices" on page 71.

This edition applies to Version 2 Release 3 of z/OS (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2019-02-16

Contents

List of Figures....................vii
List of Tables....................ix

About this document....................xi
Who should use this document....................xi
How this document is organized....................xi
Where to find more information....................xii
How to send your comments to IBM....................xiii
If you have a technical problem....................xiii
Summary of changes....................xiv
Summary of changes for TSO/E for Version 2 Release 3 (V2R3) and its updates....................xiv
Summary of changes for TSO/E for Version 2 Release 2 (V2R2) and its updates....................xiv
z/OS Version 2 Release 1 summary of changes....................xiv

Chapter 1. Introduction....................1
Concepts of the TSO/E Enhanced Connectivity Facility....................1
What is an MVS Server?....................1
Service Functions....................2
Initialization/Termination Programs....................2
What is MVSSERV?....................2
The SRPI....................3
The CPRB Control Block....................3
The INITTERM Control Block....................4
The Sequence of Events in an MVSSERV Session....................4
What You Need to Do to Write Servers....................4
Writing Access Method Drivers for MVSSERV....................5

Chapter 2. Designing and Writing a Server....................7
Server Design....................7
Steps for Designing a Server....................7
Writing a Server....................7
Using the CPRB....................7
Receiving the Service Request....................8
Performing the Service....................10
Sending the Service Reply....................10
Sending a Service Request....................11
Receiving a Service Reply....................11
Issuing Messages....................12
The Server Recovery Routine....................12
Compiling or Assembling a Server....................12
Sample Servers....................12
Sample Server IBMABASE....................13
Sample Server IBMABAS1....................17
Sample Server IBMABAS2....................21

Chapter 3. Designing and Writing a Server Initialization/Termination Program......25
Program Design....................25
Steps for Designing an Initialization/Termination Program....................25
Writing an Initialization/Termination Program....................26
Initialization....................26

Input to the Initialization/Termination Program.......... 27
Loading the Servers.......... 28
Obtaining Resources.......... 28
Defining a Server.......... 28
Sending a Service Request.......... 29
Receiving a Service Reply.......... 29
Issuing Messages.......... 30
Recovery.......... 30
Ending Initialization.......... 30
Termination.......... 30
Freeing Resources.......... 31
Deleting the Servers.......... 31
Compiling or Assembling an Initialization/Termination Program.......... 31
Sample Initialization/Termination Program.......... 32

Chapter 4. Writing an Access Method Driver.......... 39
What is an Access Method Driver?.......... 39
Using the AMD Interface.......... 39
Writing an Access Method Driver.......... 40
Considerations for Writing Access Method Drivers.......... 40
Sending a Service Request.......... 41
Receiving a Service Reply.......... 41
Issuing Messages.......... 41
Sample Access Method Driver.......... 41

Chapter 5. Installing Programs and Data Sets for Use with MVSSERV.......... 43
Installing a Program.......... 43
In a STEPLIB.......... 43
In a System Library.......... 43
Using the Input Parameter Data Set.......... 43
Allocating the Input Parameter Data Set.......... 43
Initializing the Input Parameter Data Set.......... 44
Additional MVSSERV Data Sets.......... 44
Trace Data Set.......... 44
Dump Data Set.......... 45
Dump Suppression Data Set.......... 46

Chapter 6. Testing and Diagnosis.......... 49
Testing Servers.......... 49
Steps for Testing Servers.......... 49
Diagnosing Servers.......... 50
Reading the Trace Data Set.......... 50

Chapter 7. Macro Syntax and Parameters.......... 53
CHSDCPRB Macro.......... 53
Accessing the CPRB.......... 53
Creating a CPRB for the DEFSERV or SENDREQ Macro.......... 53
CHSCED Macro.......... 54
INITTERM Macro.......... 54
DEFSERV Macro.......... 55
Register Contents for DEFSERV.......... 55
DEFSERV Syntax and Parameters.......... 55
The DEFSERV CPRB.......... 56
SENDREQ Macro.......... 57
Register Contents for SENDREQ.......... 57
SENDREQ Syntax and Parameters.......... 57
The SENDREQ CPRB.......... 59

CHSTRACE Macro....59
CHSTRACE Considerations....59
CHSTRACE Syntax and Parameters.... 60

Chapter 8. MVSSERV Return Codes....63
Return Codes from the DEFSERV Macro.... 63
Return Codes from the DEFSERV CPRB....63
Return Codes from the SENDREQ Macro.... 64
Return Codes from the SENDREQ CPRB....64
Return Codes from the CHSTRACE Macro....65

Appendix A. Accessibility.... 67
Accessibility features....67
Consult assistive technologies.... 67
Keyboard navigation of the user interface.... 67
Dotted decimal syntax diagrams....67
Notices....71
Terms and conditions for product documentation.... 72
IBM Online Privacy Statement.... 73
Policy for unsupported hardware....73
Minimum supported hardware.... 74
Programming Interface Information.... 74
Trademarks.... 74
Index.... 75

List of Figures

1. Logical Server Organization....2
2. The MVSSERV Enhanced Connectivity Environment....3
3. Events in an MVSSERV Session....4
4. Overview of Service Request Handling....8
5. Registers Passed to the Server....9
6. MVSSERV Logical Task Structure....25
7. Overview of an Initialization/Termination Program's Processing....26
8. Registers Passed at Initialization....27
9. The Define Server Parameter Area....29
10. Registers Passed at Termination....30
11. The MVSSERV Enhanced Connectivity Environment....39
12. MVSSERV Input to an Access Method Driver....40
13. Sample Trace Data Set....51

List of Tables

1. CPRB Control Block on Entry to Server....9
2. CPRB Control Block on Exit from the Server.... 11
3. CPRB Control Block with Reply from Another Server.... 11
4. INITTERM Control Block with Initialization Input....27
5. CPRB Control Block Used to Define a Server.... 28
6. INITTERM Control Block with Termination Input.... 30
7. MVSSERV Macros.... 53
8. Connectivity Environment Descriptor (CED)....54
9. INITTERM Control Block....55
10. CPRB Control Block Used to Define a Server.... 57
11. CPRB Control Block for Sending a Request (SENDREQ)....59
12. Return Codes from the DEFSERV Macro....63
13. Return Codes in the DEFSERV CPRB.... 63
14. Return Codes from the SENDREQ Macro....64
15. Return Codes in the SENDREQ CPRB.... 64
16. Return Codes from the CHSTRACE Macro....65

About this document

This document supports z/OS® (5650-ZOS).

The server-requester programming interface (SRPI) of the TSO/E 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.

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

This document also includes information about how to write programs called *access method drivers*. Access method drivers allow the MVS host to manage server-requester communications across different hardware connections with the personal computer (PC).

Who should use this document

This document is intended for:

- Application programmers who design, write, and test 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.
- System programmers who write or install access method drivers for use with the TSO/E Enhanced Connectivity Facility.

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

How this document is organized

- Chapter 1, "Introduction," on page 1 describes MVS servers and how they provide MVS services, data, and resources to requester programs.
- Chapter 2, "Designing and Writing a Server," on page 7 describes the input a server receives, the tools a server can use to process requests, and the output a server must provide.
- Chapter 3, "Designing and Writing a Server Initialization/Termination Program," on page 25 describes how to write a program that initializes one or more servers, obtains resources for them, and terminates them.
- Chapter 4, "Writing an Access Method Driver," on page 39 describes how to write a program that can manage server-requester communications across specific PC-to-Host hardware connections.
- Chapter 5, "Installing Programs and Data Sets for Use with MVSSERV," on page 43 describes how to allocate and initialize the data sets that give users access to servers, initialization/termination programs, access method drivers, and diagnosis information.
- Chapter 6, "Testing and Diagnosis," on page 49 explains how to use the MVSSERV command to test a server. This chapter also tells how to use the MVSSERV trace data set to diagnose server problems.
- Chapter 7, "Macro Syntax and Parameters," on page 53 describes the syntax and parameters of the macros you can use in MVSSERV programming.
- Chapter 8, "MVSSERV Return Codes," on page 63 describes the return codes that you may receive from the MVSSERV macros.

Where to find more information

Please see *z/OS Information Roadmap* for an overview of the documentation associated with z/OS, including the documentation available for z/OS TSO/E.

How to send your comments to IBM

We invite you to submit comments about the z/OS product documentation. Your valuable feedback helps to ensure accurate and high-quality information.

Important: If your comment regards a technical question or problem, see instead "If you have a technical problem" on page xiii.

Submit your feedback by using the appropriate method for your type of comment or question:

Feedback on z/OS function
If your comment or question is about z/OS itself, submit a request through the IBM RFE Community (www.ibm.com/developerworks/rfe/).

Feedback on IBM Knowledge Center function
If your comment or question is about the IBM Knowledge Center functionality, for example search capabilities or how to arrange the browser view, send a detailed email to IBM Knowledge Center Support at ibmkc@us.ibm.com.

Feedback on the z/OS product documentation and content
If your comment is about the information that is provided in the z/OS product documentation library, send a detailed email to mhvrcfs@us.ibm.com. We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information.

To help us better process your submission, include the following information:

- Your name, company/university/institution name, and email address
- The following deliverable title and order number: z/OS TSO/E Guide to the Server-Requester Programming Interface, SA32-0980-30
- The section title of the specific information to which your comment relates
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute the comments in any way appropriate without incurring any obligation to you.

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

If you have a technical problem or question, do not use the feedback methods that are provided for sending documentation comments. Instead, take one or more of the following actions:

- Go to the IBM Support Portal (support.ibm.com).
- Contact your IBM service representative.
- Call IBM technical support.

Summary of changes

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line to the left of the change.

Summary of changes for TSO/E for Version 2 Release 3 (V2R3) and its updates

This information contains no technical changes for this release.

Summary of changes for TSO/E for Version 2 Release 2 (V2R2) and its updates

This information contains no technical changes for this release.

z/OS Version 2 Release 1 summary of changes

See the Version 2 Release 1 (V2R1) versions of the following publications for all enhancements related to z/OS V2R1:

- *z/OS Migration*
- *z/OS Planning for Installation*
- *z/OS Summary of Message and Interface Changes*
- *z/OS Introduction and Release Guide*

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 Facility sessions on MVS.

Concepts of the TSO/E Enhanced Connectivity Facility

The TSO/E Enhanced Connectivity Facility provides a standard way for 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. 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 users access to host computer data and resources such as printers and storage. This document explains how to write an MVS host server and includes a sample server that lets a PC requester process MVS data.

For information about PC hardware and software requirements, refer to *Enhanced Connectivity Facilities Introduction*.

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 on page 2 shows the logical organization of servers, their service functions, and an initialization/termination program.

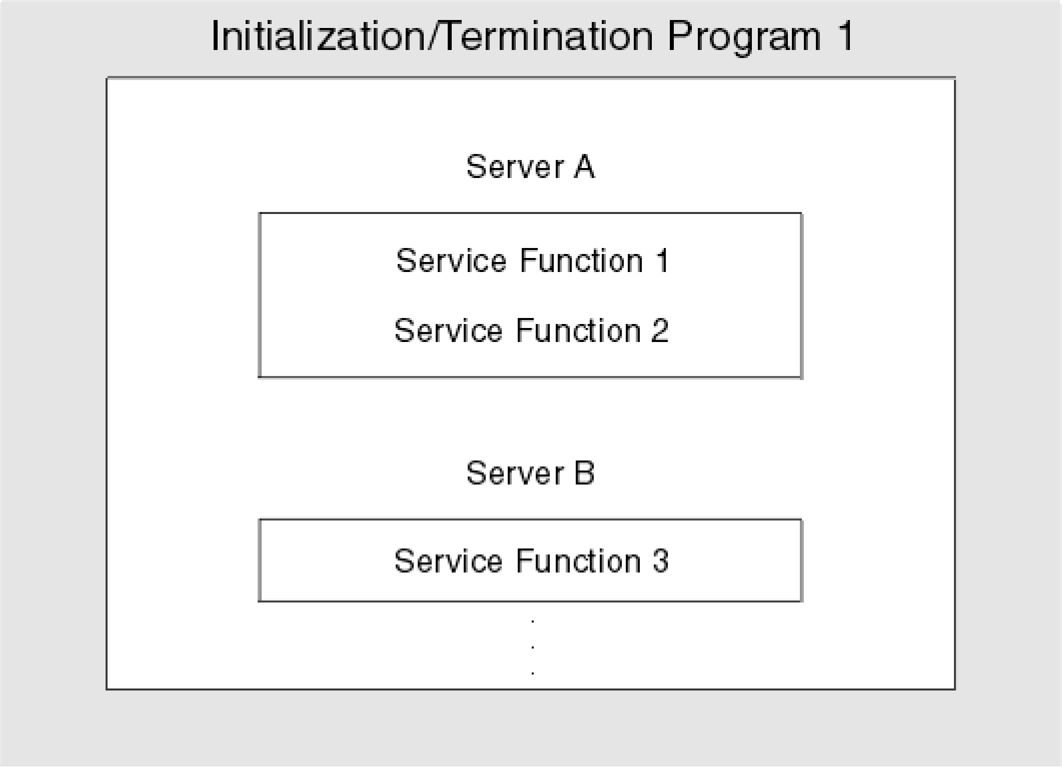


Figure 1: Logical Server Organization

Service Functions

A service function is the part of a server that satisfies a particular service request.

A server can handle different service requests by having a service function for each request. Requests identify the service function as well as the server. The server receives the request and passes control to the requested service function. For details, see Chapter 2, “Designing and Writing a Server,” on page 7.

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 and storage 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, “Designing and Writing a Server Initialization/Termination Program,” on page 25 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 switch to PC mode and invoke requesters from an 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 2 on page 3 shows the TSO/E Enhanced Connectivity environment during an MVSSERV session.

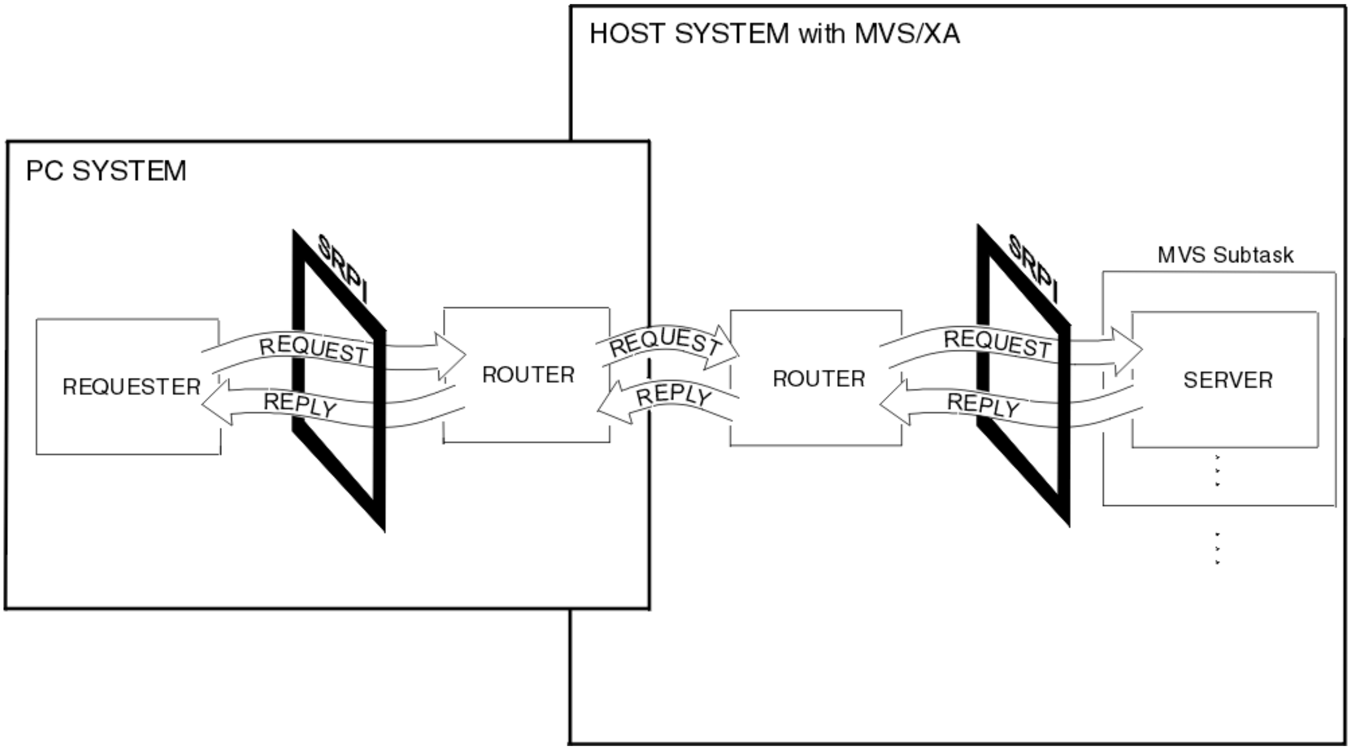


Figure 2: The MVSSERV Enhanced Connectivity Environment

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 pointers to input, a buffer for output, and a return address. 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).

CPRBs have several purposes:

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

- Servers can send requests to other servers in a CPRB.

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.

The Sequence of Events in an MVSSERV Session

Figure 3 on page 4 shows the sequence of events in an MVSSERV session.

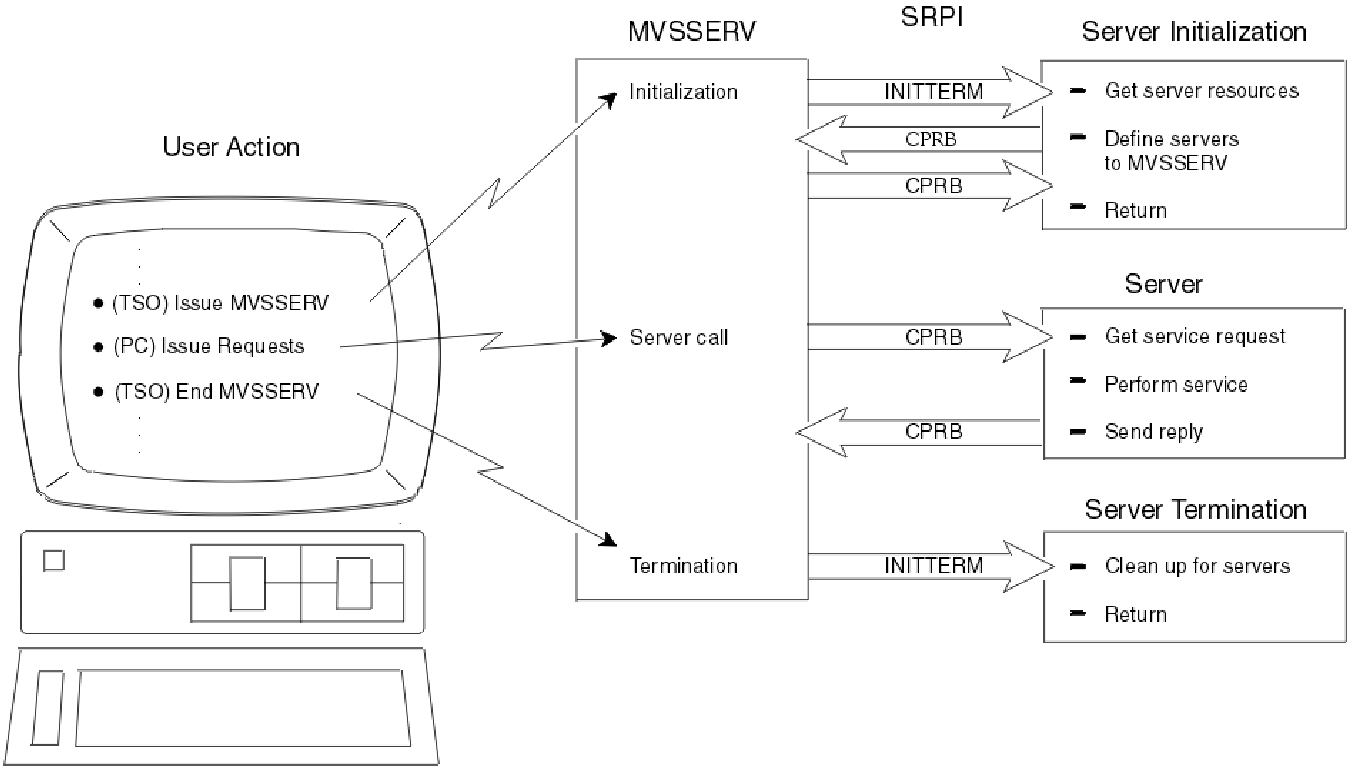


Figure 3: Events in an MVSSERV Session

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. Subsequent chapters of the document give further details.

1. Select or create a load module data set to contain the executable code for the server and initialization/termination program. If the server and initialization/termination program are in different load modules, the initialization/termination program must load the server (see Chapter 3, “Designing and Writing a Server Initialization/Termination Program,” on page 25 for details).
2. Write the server (see Chapter 2, “Designing and Writing a Server,” on page 7).
   - The server must:
     - Access the service request input in the CPRB.

  - Call the requested service function.
  - Perform the service, calling other servers if necessary.
  - 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, "Designing and Writing a Server Initialization/Termination Program," on page 25).
   - For initialization, the program must:
     - Load the server (if necessary).
     - Obtain resources (if necessary).
     - Define the server to MVSSERV and pass a parameter list (parmlist) pointing to any resources.
4. For termination, the program must:
   - Free any resources.
   - Delete the server (if loaded).
   - Compile or assemble the initialization/termination program and link it to a load module.
5. Install the server and initialization/termination program (see Chapter 5, "Installing Programs and Data Sets for Use with MVSSERV," on page 43).
   - 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
6. Invoke MVSSERV to test your server (see Chapter 6, "Testing and Diagnosis," on page 49).

Writing Access Method Drivers for MVSSERV

MVSSERV includes programs called access method drivers (AMDs), which manage Host-to-PC communications across certain hardware connections. Specifically, the MVSSERV AMDs communicate with PCs that have Distributed Function Terminal (DFT) and Control Unit Terminal (CUT) mode attachment to the host through the IBM 3174 or 3274 control unit. In addition, MVSSERV allows installations to write and install their own AMDs to manage other communication methods. Chapter 4, "Writing an Access Method Driver," on page 39 describes MVSSERV's AMD interface and special considerations for writing your own AMDs.

Chapter 2. Designing and Writing 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 define 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 provide 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:

   - Server subroutines
   - Server CSECTs
   - Load modules that are separate from the server

   If a service function fails, all other service functions of the same server are disabled. For recovery purposes, you might want to handle unrelated requests in separate servers rather than in functions of the same server. You could then isolate the servers by defining them in different initialization/termination programs (for details, see "Steps for Designing an Initialization/Termination Program" on page 25).
2. Decide whether the server should use 24- or 31-bit addressing. Servers can execute in AMODE 24 or 31, and in RMODE 24 or ANY.
3. Select a name for the server. Names can have up to eight characters, including the characters A-Z, 0-9, @, #, and $. The first character cannot be 0-9.

Writing a Server

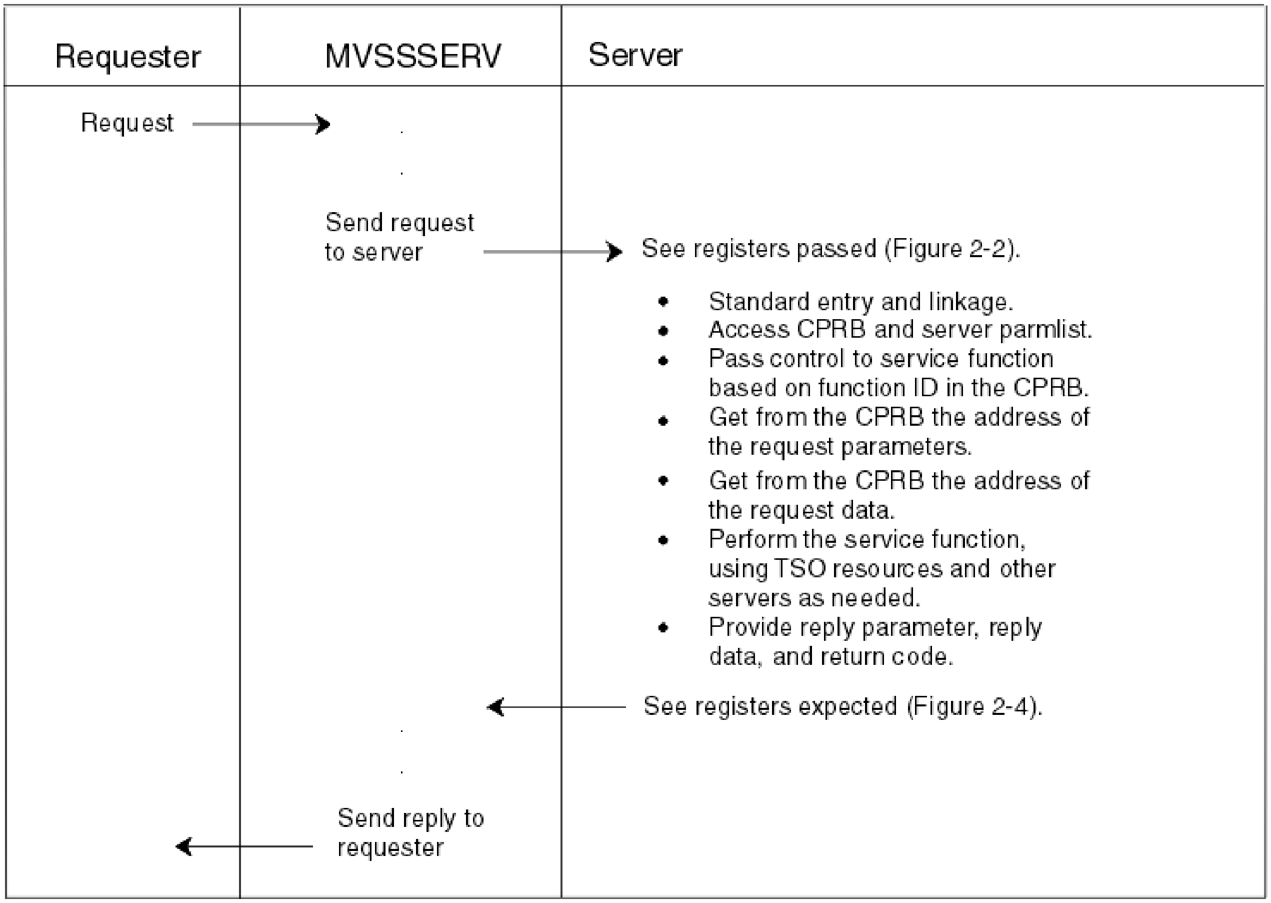
Your server must follow certain rules to receive service requests and reply to them successfully. The rules apply to using the connectivity programming request block (CPRB).

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 4 on page 8 shows the process for handling service requests.



- Standard entry and linkage.
- Access CPRB and server parmlist.
- 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 and other servers as needed.
- Provide reply parameter, reply data, and return code.

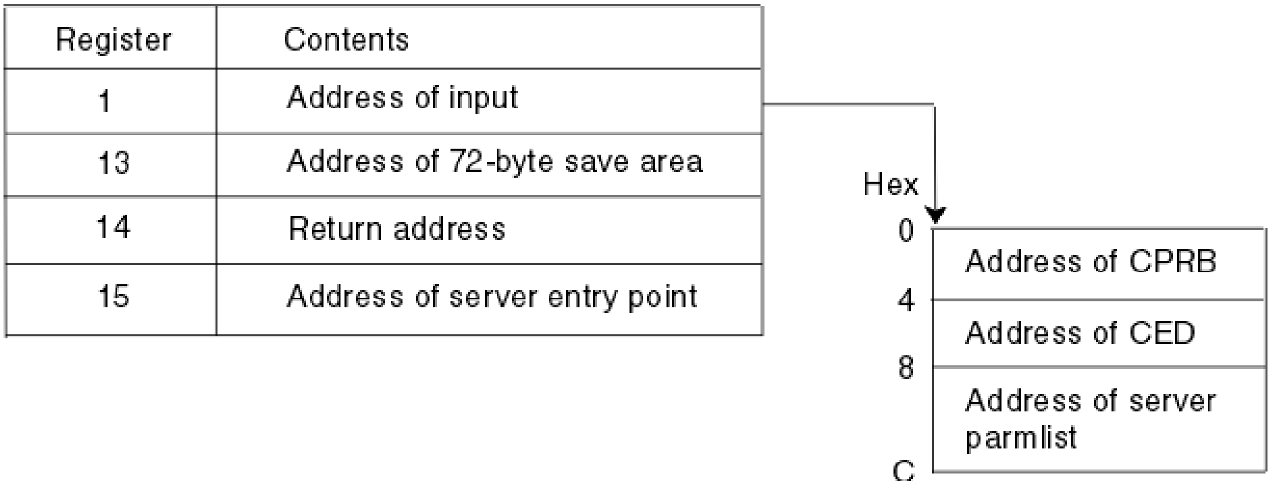
Figure 4: Overview of Service Request Handling

Receiving the Service Request

MVSSERV passes control to the server in key 8, problem program state, with the register contents shown in Figure 5 on page 9.

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

- The CPRB contains the service request.
- The CED is for MVSSERV use only. (If the server issues the DEFSERV, SENDREQ, or CHSTRACE macros, it must pass the CED address.)
- The server parmlist can point to resources such as data sets for the server to use. (For details about creating the server parmlist, see Chapter 3, “Designing and Writing a Server Initialization/Termination Program,” on page 25.)



Register	Contents
1	Address of input
13	Address of 72-byte save area
14	Return address
15	Address of server entry point

Hex	
0	Address of CPRB
4	Address of CED
8	Address of server parmlist
C	

Figure 5: 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. For details, see "CHSDCPRB Macro" on page 53.

Table 1 on page 9 shows the CPRB with the fields that pertain to the server.

The Receive Request CPRB (Entry to Server)

Table 1: CPRB Control Block on Entry to Server

Offset Dec(Hex)	Number of Bytes	Field Name	Contents
0(0)	1	CRBF1	The control block's version number (first four bits) and modification level number (last four bits).
1(1)	2		Reserved
3(3)	1	CRBF4	The type of request. X'01' indicates a service request. (X'03' indicates a define server (DEFSERV) request.)
4(4)	4	CRBCPRB	Control block identifier (character string 'CPRB').
8(8)	8		Reserved
16(10)	8	CRBSNAME	The name of the requested server.
24(18)	2		Reserved
26(1A)	2	CRBFID[1]	The ID of the requested service function (1-99)
28(1C)	12		Reserved
40(28)	4	CRBRQDLN[1]	The length of the request data.
44(2C)	4	CRBRQDAT[1]	The address of the request data.
48(30)	4	CRBRPDLN[2]	The length of the reply data (maximum length allowed by the requester).
52(34)	4	CRBRPDAT[3]	The address of the buffer for reply data.
56(38)	4	CRBRQPLN[1]	The length of the request parameters.
60(3C)	4	CRBRQPRM[1]	The address of the request parameters.
64(40)	4	CRBRPPLN[2]	The length of the reply parameters (maximum length allowed by the requester).
68(44)	4	CRBRPPRM[3]	The address of the buffer for reply parameters.
72(48)	40		Reserved

Notes:

1
: **Request** field. Use but do not alter.

2
: **Request/Reply** field. The requester initializes these fields. The server may modify the contents of these fields.

3
: **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 **2**.

Performing the Service

To perform a service, the server can:

- Use any MVS facilities available to a problem program.
- Define other servers to MVSSERV, using the DEFSERV macro.
- Send requests, using the SENDREQ macro, to other servers that have previously been defined in the current MVSSERV session.
- Issue messages to the terminal, to the MVSSERV trace data set, or to both, using the CHSTRACE macro.

Using Request and Reply buffers

Servers and requesters can use request and reply buffers to pass any agreed-upon information. The CPRB lets you specify separate buffers for data and parameters, but their use is unrestricted. For example, an application might use parameter buffers to pass instructions to the server and data buffers to pass the results. MVSSERV does not verify or modify the buffer contents.

To share data and parameters with a PC requester, the MVS server might need to convert request data and parameters from ASCII to EBCDIC, and convert reply data and parameters from EBCDIC to ASCII. The sample servers in “Sample Server IBMABAS1” on page 17 and “Sample Server IBMABAS2” on page 21 demonstrate how to perform such data conversion.

Sending the Service Reply

If the server can perform the requested service function, the server should:

- Move reply data, if any, to the reply data buffer pointed to by CPRB field CRBRPDAT.
- Move reply parameters, if any, to the reply parameter buffer pointed to by CPRB field CRBRPPRM.
- Set the actual reply data length (number of bytes) in CPRB field CRBRPDLN (the actual length must be less than or equal to the reply data length passed from the requester).
- Set the actual reply parameter length (number of bytes) in CPRB field CRBRPPLN (the actual length must be less than or equal to the reply parameter length passed from the requester).

Whether or not the server can perform the requested service function, it must:

- 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 13
Register Address of 72-byte save area

Register 14
Register Return address

Register 15
Register Server return code

Table 2 on page 11 shows the CPRB fields that the server uses in its reply.

The Send Reply CPRB (Exit from Server)

Table 2: CPRB Control Block on Exit from the Server

Offset Dec(Hex)	Number of Bytes	Field Name	Contents
0(0)	48		Reserved
48(30)	4	CRBRPDLN 1	Specify the actual length of the reply data.
52(34)	12		Reserved
64(40)	4	CRBRPPLN 1	Specify the actual length of the reply parameters.
68(44)	44		Reserved

Note:

1
The actual length cannot exceed the initial value (maximum allowed by the requester).

Sending a Service Request

In the process of handling a service request, a server can issue its own service requests to another MVS server defined in the same MVSSERV session. A server can use the results of its request in its reply.

To send a service request from a server, use the CHSDCPRB macro to create a CPRB and the SENDREQ macro to initialize and send the CPRB. For details, see "SENDREQ Macro" on page 57. The SENDREQ macro sends a service request to another server in a CPRB identical to the one shown in Table 1 on page 9.

Receiving a Service Reply

On return from the SENDREQ macro, an updated CPRB and reply buffers are returned, indicating the status of the requested service. Table 3 on page 11 shows the CPRB on return from issuing a service request.

The Receive Reply CPRB (Entry to Server)

Table 3: CPRB Control Block with Reply from Another Server

Offset Dec(Hex)	Number of Bytes	Field Name	Contents
0(0)	8		Reserved
8(8)	4	CRBSRTNC 1	The server return code from Register 15. (Filled in by MVSSERV.)
12(C)	4	CRBCRTNC 2	The return code from MVSSERV. For a list of return codes, see Chapter 8, "MVSSERV Return Codes," on page 63.
16(10)	32		Reserved
48(30)	4	CRBRPDLN 3	The length of the reply data.
52(34)	12		Reserved
64(40)	4	CRBRPPLN 3	The length of the reply parameters.
68(44)	44		Reserved

Notes®:

1
Check for a return code from the server.

2
Check for a return code from MVSSERV.

3
Use to obtain reply data and parameters from their buffers.

Issuing Messages

Your servers can issue messages to the terminal, to the MVSSERV trace data set, or to both. To issue a message and specify its destination, use the CHSTRACE macro.

For details of the CHSTRACE macro, see Chapter 7, "Macro Syntax and Parameters," on page 53. For information about the MVSSERV trace data set, see "Trace Data Set" on page 44.

The Server Recovery Routine

Servers can have their own recovery routines. If a server fails and does not recover, MVSSERV traps the error, provides a dump, and prevents that server or any other servers defined by the same initialization/termination program from processing further requests during that MVSSERV session.

To establish a recovery routine, servers must issue the ESTAE macro. 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 recovery routines, refer to *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN*.

For an example of a server recovery routine, see "Sample Server IBMABASE" on page 13.

Compiling or Assembling a Server

After writing a server, you must compile or assemble it and link-edit it. For information about preparing and running a program in TSO/E, refer to *z/OS TSO/E Programming Guide*.

Sample Servers

The sample server in "Sample Server IBMABASE" on page 13corresponds 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 sends a request to server IBMABAS1 in "Sample Server IBMABAS1" on page 17 to:
  - Retrieve a record from a customer records data set on MVS
  - Translate the record into ASCII
  - Send the record to the requester for processing
- Function 2 sends a request to server IBMABAS2 in "Sample Server IBMABAS2" on page 21 to:
  - Receive a record with a positive balance from the requester
  - Translate the record back into EBCDIC
  - Put the record into an accounts receivable data set on MVS

The initialization/termination program for these servers is shown in "Sample Initialization/Termination Program" on page 32.

The recovery routine in IBMABASE covers errors in the server itself. Errors in the called servers (IBMABAS1 and IBMABAS2) are handled by MVSSERV's recovery routine, which informs IBMABASE if they fail.

Sample Server IBMABASE

```
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.
         L     2,0(,1)               Obtain the CPRB address.
         USING CHSDCPRB,2            Establish addressability to it.
         L     3,4(,1)               Obtain the CED address.
         USING CHSCED,3              Establish addressability to it.
         L     4,8(,1)               Obtain server parameter address.
         USING PARAMETERS,4          Establish addressability to them.
         L     11,DYNAMIC_ADDR       Obtain the address for the dynamic
*                                    storage.
         USING DYNAREA,11            Establish addressability to the
*                                    dynamic area.
         ST    13,BASESAVE+4         Save the callers savearea address.
         LA    15,BASESAVE           Obtain our savearea address.
         ST    15,8(,13)             Chain it in the caller's savearea.
         LR    13,15                 Point register 13 to our savearea.
         ST    3,CED_ADDR            Save the address of the CED.
         EJECT
***********************************************************************
* TITLE: IBMABASE MAINLINE
*
* LOGIC: Determine the function requested, and invoke the appropriate
*        server.
*
* OPERATION:
* 1. Establish a recovery environment.
* 2. If the data sets are not open:
*      - Open them.
* 3. Determine the function requested.
* 4. If function 1 is requested:
*      - Issue the CHSTRACE macro to output a message to the TRACE
*        data set.
*      - Issue the SENDREQ macro to invoke the appropriate server.
*      - Copy IBMABAS1's reply into the requester CPRB.
* 5. If function 2 is requested:
*      - Issue the CHSTRACE macro to output a message to the TRACE
*        data set.
*      - Issue the SENDREQ macro to invoke the appropriate server.
* 6. Else an invalid function is requested:
*      - Issue the CHSTRACE macro to output a message to the TRACE
*        data set.
*      - Set an error return code.
* 7. Check the SENDREQ return code:
*      - If the SENDREQ failed, then set an error return code.
* 8. Cancel the recovery environment.
* 9. Return to the caller with return code.
***********************************************************************
         SPACE 2
***********************************************************************
* Establish a recovery environment
***********************************************************************
         SPACE
         ESTAE RECOVERY,PARAM=(11),MF=(E,ESTLIST)
         EJECT
***********************************************************************
* OPEN the data sets.
***********************************************************************
         SPACE
         CLI   STATUS,OPENED         Are the data sets opened?
         BE    OPEN                  Yes, then don't try to open them.
         L     6,DCBIN_ADDR          Load the INPUT DCB address.
         L     7,DCBOUT_ADDR         Load the OUTPUT DCB address.
         L     8,DCBLOG_ADDR         Load the LOG DCB address.
         L     9,OPEN_ADDR           Load the list form address.
         OPEN  ((6),,(7),,(8)),MF=(E,(9)) Open the data sets.
         MVI   STATUS,OPENED         Indicate that they are open.
         EJECT
***********************************************************************
* Determine the FUNCTION requested.
```

```
************************************************************************
         SPACE
OPEN     DS    0H
         LA    5,1                   Load the first function ID.
         CH    5,CRBFID              Is function one requested?
         BE    FUNCTION_1            Yes, branch to the function.
         LA    5,2                   Load the second function ID.
         CH    5,CRBFID              Is function two requested?
         BE    FUNCTION_2            Yes, branch to the function.
         SPACE 3
***********************************************************************
* Issue the CHSTRACE macro to output the INVALID FUNCTION message.
***********************************************************************
         SPACE
         CHSTRACE DEST=TRACE,CED=CHSCED,BUFFER=INV_MSG,                *
               BUFLEN=MSG_LEN,MF=(E,CHSLIST,COMPLETE)
         B     ERROR                 Exit the server.
         EJECT
FUNCTION_1 DS  0H
         SPACE
***********************************************************************
* Issue the CHSTRACE macro to output the FUNCTION 1 message.
***********************************************************************
         SPACE
         CHSTRACE DEST=TRACE,CED=CHSCED,BUFFER=FUN1_MSG,               *
               BUFLEN=MSG_LEN,MF=(E,CHSLIST,COMPLETE)
***********************************************************************
* Issue the SENDREQ macro to invoke IBMABAS1.
***********************************************************************
         SPACE
         DROP  2
         LA    5,CPRBSTOR            Obtain the address for the new
*                                    CPRB.
         USING CHSDCPRB,5            Establish addressability to it.
         SENDREQ CPRB=CHSDCPRB,CED=CHSCED,SERVER=SERVER1_NAME,         *
               REQPARM=(CRBRQPRM-CHSDCPRB(,2),CRBRQPLN-CHSDCPRB(,2)),  *
               REPDATA=(CRBRPDAT-CHSDCPRB(,2),CRBRPDLN-CHSDCPRB(,2)),  *
               MF=(E,SENDLIST,COMPLETE)
***********************************************************************
* Copy IBMABAS1's reply into the REQUESTER CPRB.
***********************************************************************
         SPACE
         L     8,CRBRPDAT            Obtain the address of the reply
*                                    data.
         L     6,CRBRPDLN            Obtain the length of the reply
*                                    data.
         DROP  5
         USING CHSDCPRB,2            Restore addressability to the
*                                    requester CPRB.
         ST    6,CRBRPDLN            Store the reply data length in the
*                                    CPRB for the requester.
         L     7,CRBRPDAT            Obtain the address to place the
*                                    reply data.
         BCTR  6,0
         EX    6,MOVDATA1            Copy the reply data into the CPRB
*                                    for the requester.
         B     EXIT
MOVDATA1 MVC   0(0,7),0(8)
         EJECT
FUNCTION_2 DS  0H
         SPACE
***********************************************************************
* Issue the CHSTRACE macro to output the FUNCTION 2 message.
***********************************************************************
         SPACE
         CHSTRACE DEST=TRACE,CED=CHSCED,BUFFER=FUN2_MSG,               *
               BUFLEN=MSG_LEN,MF=(E,CHSLIST,COMPLETE)
***********************************************************************
* Issue the SENDREQ macro to invoke IBMABAS2.
***********************************************************************
         SPACE
         DROP  2
         LA    5,CPRBSTOR            Obtain the address for the new
*                                    CPRB.
         USING CHSDCPRB,5            Establish addressability to it.
         SENDREQ CPRB=CHSDCPRB,CED=CHSCED,SERVER=SERVER2_NAME,         *
               REQPARM=(CRBRQPRM-CHSDCPRB(,2),CRBRQPLN-CHSDCPRB(,2)),  *
               REQDATA=(CRBRQDAT-CHSDCPRB(,2),CRBRQDLN-CHSDCPRB(,2)),  *
               MF=(E,SENDLIST,COMPLETE)
         EJECT
***********************************************************************
* Leave the server.
```

```
***********************************************************************
         SPACE
EXIT     DS    0H
         LTR   15,15                 Check SENDREQ return code.
         BNZ   ERROR                 Error? - Then set bad return code.
         L     15,CRBSRTNC           Otherwise obtain the SERVER return
*                                    code.
         B     LEAVE                 Exit the SERVER.
ERROR    DS    0H
         LA    15,8                  Set bad return code.
LEAVE    DS    0H
         LR    2,15                  Save the return code.
         ESTAE 0                     Remove the recovery environment.
         LR    15,2                  Restore the return code.
         L     13,BASESAVE+4         Restore caller's savearea 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.
         EJECT
************************************************************************
* TITLE: IBMABASE RECOVERY
*
* LOGIC: Issue a message to the terminal and trace data set indicating
*        that the server ABENDed and is no longer available.
*
* OPERATION:
* 1. If an SDWA is available then:
*      - Establish addressability to the recovery routine parameters
*        (IBMABASE dynamic storage address).
*      - Obtain the address of the CED.
*      - Issue the CHSTRACE macro to output a message to the TERMINAL
*        and the TRACE data set.
*      - Issue the SETRP macro to issue a DUMP and CONTINUE WITH
*        TERMINATION.
* 2. Else an SDWA is not available so:
*      - Set the return code to indicate to CONTINUE WITH TERMINATION.
* 3. Return to the caller (with return code in no SDWA case).
***********************************************************************
         SPACE 2
RECOVERY DS    0H
         USING RECOVERY,15
         C     0,=F'12'              SDWA supplied?
         BE    NO_SDWA               No, then leave recovery.
         STM   14,12,12(13)
         LR    12,15
         USING RECOVERY,12
         DROP  15
         L     11,0(,1)              Obtain the recovery parameters
*                                    (Dynamic storage address).
***********************************************************************
* Use IBMABAS1's savearea for the recovery savearea.
***********************************************************************
         SPACE
         ST    13,BAS1SAVE+4         Save the callers savearea address.
         LA    15,BAS1SAVE           Obtain our savearea address.
         ST    15,8(,13)             Chain it in the caller's savearea.
         LR    13,15                 Point register 13 to our savearea.
         EJECT
         LR    2,1                   Save the address of the SDWA.
         L     3,CED_ADDR            Obtain the address of the CED.
         USING CHSCED,3              Establish addressability to it.
         SPACE
***********************************************************************
*
* Here is where diagnostic information that would useful in debugging
* any problems would be placed in the SDWA and the VRA.
*
***********************************************************************
* Issue the CHSTRACE macro to output the ABEND message.
***********************************************************************
         SPACE
         CHSTRACE DEST=BOTH,CED=CHSCED,BUFFER=REC_MSG,                 *
               BUFLEN=MSG_LEN,MF=(E,CHSLIST,COMPLETE)
***********************************************************************
* Issue the SETRP macro to issue a DUMP and CONTINUE WITH TERMINATION.
***********************************************************************
         SETRP WKAREA=(2),DUMP=YES,RC=0
         EJECT
***********************************************************************
* Leave the recovery routine.
***********************************************************************
         SPACE
         L     13,BAS1SAVE+4         Restore caller's savearea address.
         LM    14,12,12(13)          Restore the caller's registers.
NO_SDWA  DS    0H
         SLR   15,15                 Indicate CONTINUE WITH TERMINATION
*                                    for the no SWDA case.
         BR    14
         EJECT
**********************************************************************
* Constants.
**********************************************************************
         SPACE
**********************************************************************
* SERVER names.
**********************************************************************
         SPACE
SERVER_NAME  DC CL8'IBMABASE'        Server name.
SERVER1_NAME DC CL8'IBMABAS1'        Server name.
SERVER2_NAME DC CL8'IBMABAS2'        Server name.
         SPACE
```

```
**********************************************************************
* TRACE data set messages.
**********************************************************************
         SPACE
FUN1_MSG DC    CL80' Server IBMABASE entered.  SENDREQ issued for IBMABA*
               S1.'
FUN2_MSG DC    CL80' Server IBMABASE entered.  SENDREQ issued for IBMABA*
               S2.'
INV_MSG  DC    CL80' Server IBMABASE entered.  An invalid function was r*
               equested.'
REC_MSG  DC    CL80' Server IBMABASE ABENDed.  The server is no longer a*
               vailable.'
MSG_LEN  DC    A(*-REC_MSG)          Length of message
         EJECT
**********************************************************************
* Dynamic Area.
*
* NOTE:  This mapping is shared between IBMABASE, IBMABAS1 and
*        IBMABAS2.  Any change must be incorporated into all modules.
**********************************************************************
         SPACE
DYNAREA  DSECT                       DYNAMIC area common mapping
         SPACE
BASESAVE DS    18F                   Save area.
BASESUBS DS    15F                   Subroutine save area.
         SPACE
BAS1SAVE DS    18F                   Save area.
         SPACE
BAS2SAVE DS    18F                   Save area.
         SPACE
CED_ADDR DS    F                     Address of the CED.
         SPACE
WORKAREA DS    D                     Work area for CVB and CVD.
         SPACE
BINARY_BAL DS  F                     Holds binary form of the balance.
         SPACE
ED_AREA  DS    0CL8                  EDIT instruction work area.
         DS    CL1                   Fill character position.
         DS    CL3                   Digit positions.
ED_RESULT DS   CL4                   EDIT result digits.
         SPACE
STATUS   DS    X                     Status word.
OPENED   EQU   X'01'                 Data sets are opened.
CLOSED   EQU   X'00'                 Data sets are closed.
         SPACE
CPRBSTOR DS    0D                    Storage for the CPRB to be used
*                                    for IBMABAS1 and IBMABAS2.
         ORG   *+CRBSIZE
CPRBEND  DS    0D
**********************************************************************
* Issue the CHSTRACE macro list form to supply a parameter list.
**********************************************************************
         SPACE
         CHSTRACE MF=(L,CHSLIST)
         SPACE
**********************************************************************
* Issue the SENDREQ macro list form to supply a parameter list.
**********************************************************************
         SPACE
         SENDREQ MF=(L,SENDLIST)
         SPACE
```

```
**********************************************************************
* Issue the ESTAE macro list form to supply a parameter list.
**********************************************************************
         SPACE
ESTLIST  ESTAE MF=L
         EJECT
***********************************************************************
* Server parameter list mapping.
***********************************************************************
         SPACE
PARAMETERS DSECT
DYNAMIC_ADDR DS A                    Dynamic Storage address.
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.
         SPACE
***********************************************************************
* CPRB reply buffer mapping.
***********************************************************************
         SPACE
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
         EJECT
***********************************************************************
* CPRB mapping
***********************************************************************
         SPACE
         CHSDCPRB DSECT=YES
         EJECT
**********************************************************************
* CED mapping.
**********************************************************************
         SPACE
         CHSCED   DSECT=YES
         EJECT
**********************************************************************
* SDWA mapping.
**********************************************************************
         SPACE
         IHASDWA
         END   IBMABASE
```

Sample Server IBMABAS1

```
IBMABAS1 CSECT
IBMABAS1 AMODE 24
IBMABAS1 RMODE 24
         STM   14,12,12(13)          Save the caller's registers.
         LR    12,15                 Establish addressability within
         USING IBMABAS1,12           this CSECT.
         L     2,0(,1)               Obtain the CPRB address.
         USING CHSDCPRB,2            Establish addressability to it.
         L     3,4(,1)               Obtain the CED address.
         USING CHSCED,3              Establish addressability to it.
         L     4,8(,1)               Obtain server parameter address.
         USING PARAMETERS,4          Establish addressability to them.
         L     11,DYNAMIC_ADDR       Obtain the address for the dynamic
*                                    storage.
         USING DYNAREA,11            Establish addressability to the
*                                    dynamic area.
         ST    13,BAS1SAVE+4         Save the callers savearea address.
         LA    15,BAS1SAVE           Obtain our savearea address.
         ST    15,8(,13)             Chain it in the caller's savearea.
         LR    13,15                 Point register 13 to our savearea.
         EJECT
***********************************************************************
* TITLE: IBMABAS1 MAINLINE
*
```

```
* LOGIC: Read a record from the input file.
*
* OPERATION:
* 1. Issue the CHSTRACE macro to output a message to the TRACE data
*    set.
* 2. Issue the GET macro to read an input file record.
* 3. If the end of file was encountered:
*      - Issue the CHSTRACE macro to output a message to the TRACE
*        data set.
*      - Close the data sets.
*      - Set end of file return code
* 4. Else, no end of file encountered:
*      - If the transaction should be logged:
*          a. Issue the PUT macro to output the log message to the
*             log file.
*      - Translate the reply data into ASCII.
* 5. Return to the caller with return code.
***********************************************************************
         SPACE
**********************************************************************
* Issue the CHSTRACE macro to output the IBMABAS1 message.
**********************************************************************
         SPACE
         CHSTRACE DEST=TRACE,CED=CHSCED,BUFFER=BAS1_MSG,               *
               BUFLEN=MSG_LEN,MF=(E,CHSLIST,COMPLETE)
         L     5,CRBRPDAT            Obtain the address of the reply
*                                    buffer.
         USING REPLY_BUFFER,5        Establish addressability to it.
         SPACE
         L     6,DCBIN_ADDR          Obtain INPUT DCB address.
         USING IHADCB,6
         MVC   DCBEODA,=AL3(END_OF_FILE) Set end of file exit.
         SPACE
**********************************************************************
* Issue the GET macro to read an input record.
**********************************************************************
         SPACE
         GET   (6),REPLY             Get the record.
         DROP  6
         SPACE
         L     6,CRBRQPRM            Load request parameter address.
         CLI   0(6),X'01'            Should we log the transaction?
         BNE   NO_LOG                No, branch around logging.
         EJECT
**********************************************************************
* LOG the transaction. Issue the PUT macro to output records to the
* log data set.
**********************************************************************
         SPACE
         L     6,DCBLOG_ADDR         Obtain LOG DCB address.
         PUT   (6),INPUT_LOG         Output the log message and
         PUT   (6),REPLY             the record.
         PUT   (6),BLANK             Insert a blank line.
         EJECT
**********************************************************************
* Convert the EBCDIC message to ASCII.
**********************************************************************
         SPACE
NO_LOG   DS    0H
         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.
         SPACE
**********************************************************************
* Convert the balance to binary.
**********************************************************************
         SPACE
DO_PACK  DS    0H
         PACK  WORKAREA(8),CUST_BAL(4) Convert balance to decimal.
         CVB   7,WORKAREA            Convert balance to binary.
         ST    7,BINARY_BAL          Save the balance.
         SPACE
***********************************************************************
* Move the balance into the reply area, taking into account the PC's
* method of reverse byte retrieval.
***********************************************************************
         SPACE
         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.
         SPACE
***********************************************************************
* Store the reply statistics in the CPRB.
***********************************************************************
         SPACE
         LA    6,REPLY_LEN           Get the length of the reply,
         ST    6,CRBRPDLN            and store it into the CPRB.
         LA    6,0                   Set the reply parameter length,
         ST    6,CRBRPPLN            and store it into the CPRB.
         SLR   15,15                 Set a good return code.
         B     EXIT
         EJECT
**********************************************************************
* END OF FILE routine.
**********************************************************************
         SPACE
END_OF_FILE DS 0H
         SPACE
**********************************************************************
* Issue the CHSTRACE macro to output the END OF FILE message.
**********************************************************************
         SPACE
         CHSTRACE DEST=TRACE,CED=CHSCED,BUFFER=EOF_MSG,                *
               BUFLEN=MSG_LEN,MF=(E,CHSLIST,COMPLETE)
**********************************************************************
* Close the data sets.
**********************************************************************
         SPACE
         L     6,DCBIN_ADDR          Load the INPUT DCB address.
         L     7,DCBOUT_ADDR         Load the OUTPUT DCB address.
         L     8,DCBLOG_ADDR         Load the LOG DCB address.
         L     9,CLOSE_ADDR          Load the list form address.
         CLOSE ((6),,(7),,(8)),MF=(E,(9)) Close the data sets.
         MVI   STATUS,CLOSED         Indicate that they are closed.
         SPACE
         LA    15,4                  Set end of file return code.
         EJECT
**********************************************************************
* Leave the server.
**********************************************************************
         SPACE
EXIT     DS    0H
         L     13,BAS1SAVE+4         Restore caller's savearea 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.
         EJECT
**********************************************************************
* Constants.
**********************************************************************
         SPACE
***********************************************************************
* EBCDIC to ASCII translate table.
***********************************************************************
         SPACE
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'
         SPACE
**********************************************************************
* TRACE data set messages.
**********************************************************************
         SPACE
BAS1_MSG DC    CL80' Server IBMABAS1 entered.'
EOF_MSG  DC    CL80' End of file encountered on customer records.'
MSG_LEN  DC    A(*-EOF_MSG)          Length of message
         SPACE
**********************************************************************
```

```
* LOG data set messages.
**********************************************************************
         SPACE
INPUT_LOG DS   0CL109                Input log message.
         DC    CL109'The following customer record was read from the cu*
               stomer files.'
         SPACE
BLANK    DS    0CL109                Blank line
         DC    CL109' '
         EJECT
**********************************************************************
* Dynamic Area.
*
* NOTE:  This mapping is shared between IBMABASE, IBMABAS1 and
*        IBMABAS2.  Any change must be incorporated into all modules.
**********************************************************************
         SPACE
DYNAREA  DSECT                       DYNAMIC area common mapping
         SPACE
BASESAVE DS    18F                   Save area.
BASESUBS DS    15F                   Subroutine save area.
         SPACE
BAS1SAVE DS    18F                   Save area.
         SPACE
BAS2SAVE DS    18F                   Save area.
         SPACE
CED_ADDR DS    F                     Address of the CED.
         SPACE
WORKAREA DS    D                     Work area for CVB and CVD.
         SPACE
BINARY_BAL DS  F                     Holds binary form of the balance.
         SPACE
ED_AREA  DS    0CL8                  EDIT instruction work area.
         DS    CL1                   Fill character position.
         DS    CL3                   Digit positions.
ED_RESULT DS   CL4                   EDIT result digits.
         SPACE
STATUS   DS    X                     Status word.
OPENED   EQU   X'01'                 Data sets are opened.
CLOSED   EQU   X'00'                 Data sets are closed.
         SPACE
CPRBSTOR DS    0D                    Storage for the CPRB to be used
*                                    for IBMABAS1 and IBMABAS2.
         ORG   *+CRBSIZE
CPRBEND  DS    0D
**********************************************************************
* Issue the CHSTRACE macro list form to supply a parameter list.
**********************************************************************
         SPACE
         CHSTRACE MF=(L,CHSLIST)
         SPACE
**********************************************************************
* Issue the SENDREQ macro list form to supply a parameter list.
**********************************************************************
         SPACE
         SENDREQ MF=(L,SENDLIST)
         SPACE
**********************************************************************
* Issue the ESTAE macro list form to supply a parameter list.
**********************************************************************
         SPACE
ESTLIST  ESTAE MF=L
         EJECT
***********************************************************************
* Server parameter list mapping.
***********************************************************************
         SPACE
PARAMETERS DSECT
DYNAMIC_ADDR DS A                    Dynamic Storage address.
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.
         SPACE
***********************************************************************
* CPRB reply buffer mapping.
***********************************************************************
         SPACE
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
         EJECT
***********************************************************************
* CPRB mapping
***********************************************************************
         SPACE
         CHSDCPRB  DSECT=YES
         EJECT
***********************************************************************
* CED mapping
***********************************************************************
         SPACE
         CHSCED   DSECT=YES
         EJECT
***********************************************************************
* DCB mapping
***********************************************************************
         SPACE
         DCBD  DSORG=PS
         END   IBMABAS1
```

Sample Server IBMABAS2

```
IBMABAS2 CSECT
IBMABAS2 AMODE 24
IBMABAS2 RMODE 24
         STM   14,12,12(13)          Save the caller's registers.
         LR    12,15                 Establish addressability within
         USING IBMABAS2,12           this CSECT.
         L     2,0(,1)               Obtain the CPRB address.
         USING CHSDCPRB,2            Establish addressability to it.
         L     3,4(,1)               Obtain the CED address.
         USING CHSCED,3              Establish addressability to it.
         L     4,8(,1)               Obtain server parameter address.
         USING PARAMETERS,4          Establish addressability to them.
         L     11,DYNAMIC_ADDR       Obtain the address for the dynamic
*                                    storage.
         USING DYNAREA,11            Establish addressability to the
*                                    dynamic area.
         ST    13,BAS2SAVE+4         Save the callers savearea address.
         LA    15,BAS2SAVE           Obtain our savearea address.
         ST    15,8(,13)             Chain it in the caller's savearea.
         LR    13,15                 Point register 13 to our savearea.
         EJECT
***********************************************************************
* TITLE: IBMABAS2 MAINLINE
*
* LOGIC: Determine the function requested, and perform that function.
*
* OPERATION:
* 1. Issue the CHSTRACE macro to output a message to the TRACE data
*    set.
* 2. Translate the request data into EBCDIC.
* 3. Issue the PUT macro to output the record to the output file.
*      - If the transaction should be logged:
*          a. Issue the PUT macro to output the log message to the
*             log file.
* 4. Return to the caller with return code.
***********************************************************************
         SPACE
**********************************************************************
* Issue the CHSTRACE macro to output the IBMABAS2 message.
**********************************************************************
         SPACE
         CHSTRACE DEST=TRACE,CED=CHSCED,BUFFER=BAS2_MSG,               *
               BUFLEN=MSG_LEN,MF=(E,CHSLIST,COMPLETE)
         L     5,CRBRQDAT            Obtain the address of the request
*                                    buffer.
         USING REPLY_BUFFER,5        Establish addressability to it.
         SPACE
***********************************************************************
* Convert the ASCII message to EBCDIC.
```

```
***********************************************************************
         SPACE
         TR    TRANS_PART,TRANS_EBCDIC Translate the record to EBCDIC.
         SPACE
***********************************************************************
* Move the reply balance into the work area, taking into account the
* PC's method of reverse byte retrieval.
***********************************************************************
         SPACE
         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.
         SPACE
***********************************************************************
* Convert the balance to EBCDIC
***********************************************************************
         SPACE
         L     7,BINARY_BAL          Prepare for CVD.
         CVD   7,WORKAREA            Convert the balance to decimal.
         MVC   ED_AREA,ED_PATTERN    Copy in the EDIT pattern.
         ED    ED_AREA,WORKAREA+4    EDIT the balance.
         MVC   CUST_BAL,ED_RESULT    Place the results in the record.
         SPACE
***********************************************************************
* Issue the PUT macro to write the record.
***********************************************************************
         SPACE
         L     6,DCBOUT_ADDR         Obtain OUTPUT DCB address.
         PUT   (6),REPLY             Output the record.
         L     6,CRBRQPRM            Load request parameter address.
         CLI   0(6),X'01'            Should we log the transaction?
         BNE   NO_LOG                No, branch around logging.
         EJECT
***********************************************************************
* LOG the transaction. Issue the PUT macro to output records to the
* log data set.
***********************************************************************
         SPACE
         L     6,DCBLOG_ADDR         Obtain LOG DCB address.
         PUT   (6),OUTPUT_LOG        Output the log message and
         PUT   (6),REPLY             the record.
         PUT   (6),BLANK             Insert a blank line.
         EJECT
***********************************************************************
* Leave the server.
***********************************************************************
         SPACE
NO_LOG   DS    0H
         LA    15,0                  Set the return code.
EXIT     DS    0H
         L     13,BAS2SAVE+4         Restore caller's savearea 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.
         EJECT
**********************************************************************
* Constants.
**********************************************************************
         SPACE
***********************************************************************
* ASCII to EBCDIC translate table.
***********************************************************************
         SPACE
TRANS_EBCDIC DS 0CL256
         DC    X'00010203372D2E2F1605250B0C0D0E0F'
         DC    X'101112133C3D322618193F27221D351F'
         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'30424757EE33B6E1CDED3644CMVSSERV31AA'
         DC    X'FC9EAE8CDDDC39FB80AFFD7876B29FFF'
         SPACE
```

```
ED_PATTERN DC X'4020202020202020'    Edit pattern for balances.
         SPACE
**********************************************************************
* TRACE data set messages.
**********************************************************************
         SPACE
BAS2_MSG DC    CL80' Server IBMABAS2 entered.'
MSG_LEN  DC    A(*-BAS2_MSG)         Length of message
         SPACE
**********************************************************************
* LOG data set messages.
**********************************************************************
         SPACE
OUTPUT_LOG DS  0CL109                Output log message.
         DC    CL109'The following customer record was written to the b*
               illing file.'
         SPACE
BLANK    DS    0CL109                Blank line
         DC    CL109' '
         EJECT

**********************************************************************
* Dynamic Area.
*
* NOTE:  This mapping is shared between IBMABASE, IBMABAS1 and
*        IBMABAS2.  Any change must be incorporated into all modules.
**********************************************************************
         SPACE
DYNAREA  DSECT                       DYNAMIC area common mapping
         SPACE
BASESAVE DS    18F                   Save area.
BASESUBS DS    15F                   Subroutine save area.
         SPACE
BAS1SAVE DS    18F                   Save area.
         SPACE
BAS2SAVE DS    18F                   Save area.
         SPACE
CED_ADDR DS    F                     Address of the CED.
         SPACE
WORKAREA DS    D                     Work area for CVB and CVD.
         SPACE
BINARY_BAL DS  F                     Holds binary form of the balance.
         SPACE
ED_AREA  DS    0CL8                  EDIT instruction work area.
         DS    CL1                   Fill character position.
         DS    CL3                   Digit positions.
ED_RESULT DS   CL4                   EDIT result digits.
         SPACE
STATUS   DS    X                     Status word.
OPENED   EQU   X'01'                 Data sets are opened.
CLOSED   EQU   X'00'                 Data sets are closed.
         SPACE
CPRBSTOR DS    0D                    Storage for the CPRB to be used
*                                    for IBMABAS1 and IBMABAS2.
         ORG   *+CRBSIZE
CPRBEND  DS    0D
         SPACE
**********************************************************************
* Issue the CHSTRACE macro list form to supply a parameter list.
**********************************************************************
         SPACE
         CHSTRACE MF=(L,CHSLIST)
         SPACE
**********************************************************************
* Issue the SENDREQ macro list form to supply a parameter list.
**********************************************************************
         SPACE
         SENDREQ MF=(L,SENDLIST)
         SPACE
**********************************************************************
* Issue the ESTAE macro list form to supply a parameter list.
**********************************************************************
         SPACE
ESTLIST  ESTAE MF=L
         EJECT
***********************************************************************
* Server parameter list mapping.
***********************************************************************
         SPACE
PARAMETERS DSECT
DYNAMIC_ADDR DS A                    Dynamic Storage address.
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.
         SPACE
***********************************************************************
* CPRB reply buffer mapping.
***********************************************************************
         SPACE
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
         EJECT
***********************************************************************
* CPRB mapping
***********************************************************************
         SPACE
         CHSDCPRB DSECT=YES
         EJECT
***********************************************************************
* CED mapping
***********************************************************************
         SPACE
         CHSCED   DSECT=YES
         END   IBMABAS2
```

Chapter 3. Designing and Writing a Server Initialization/Termination Program

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

Program Design

The initialization/termination programs are logically grouped in separate subtasks. They 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 6 on page 25 shows the position of initialization/termination programs in the logical MVSSERV task structure.

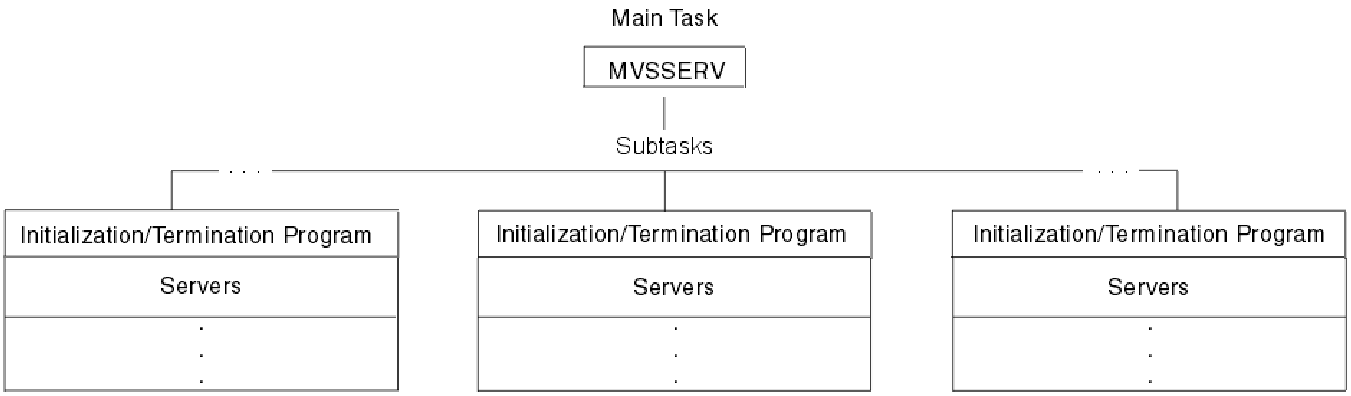

Main Task
MVSSERV
Subtasks
Initialization/Termination Program
Servers
Initialization/Termination Program
Servers
Initialization/Termination Program
Servers

Figure 6: 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 and data sets 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 related 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 do not want the initialization/termination program to load the server, place the initialization/termination program and server in the *same* load module. The initialization/termination program can use a constant server address to define the server to MVSSERV.
- If you want the initialization/termination program to load the server, place the initialization/termination program and server in *different* load modules. The initialization/termination program can get the server address from the LOAD macro to define the server to MVSSERV.

3. Decide whether the initialization/termination program server should use 24- or 31-bit addressing. Initialization/termination programs can execute in AMODE 24 or 31, and RMODE 24 or ANY.
4. Select a name for the initialization/termination program. Names 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 5, “Installing Programs and Data Sets for Use with MVSSERV,” on page 43).

Writing an Initialization/Termination Program

Figure 7 on page 26 gives an overview of an initialization/termination program's processing.

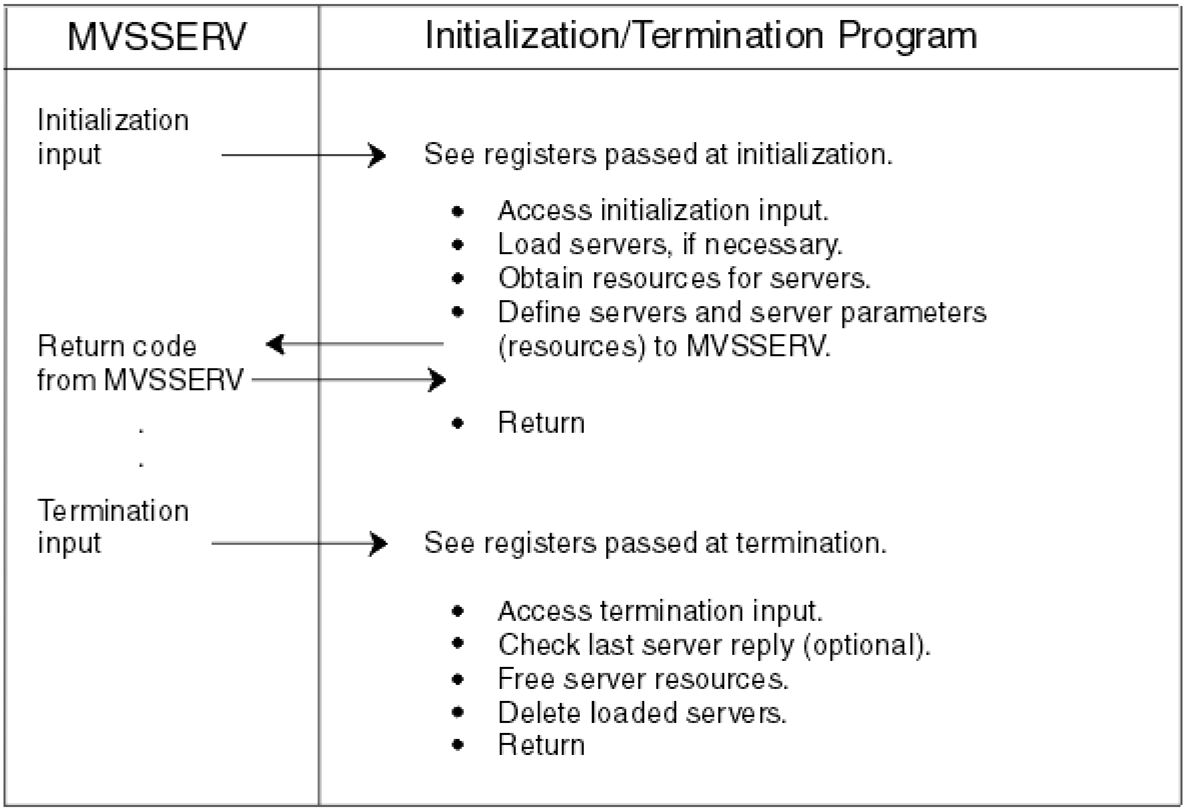

MVSSERV
Initialization/Termination Program
Initialization input
See registers passed at initialization.
• Access initialization input.
• Load servers, if necessary.
• Obtain resources for servers.
• Define servers and server parameters (resources) to MVSSERV.
Return code from MVSSERV
• Return
.
.
Termination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

Figure 7: 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 5, “Installing Programs and Data Sets for Use with MVSSERV,” on page 43.

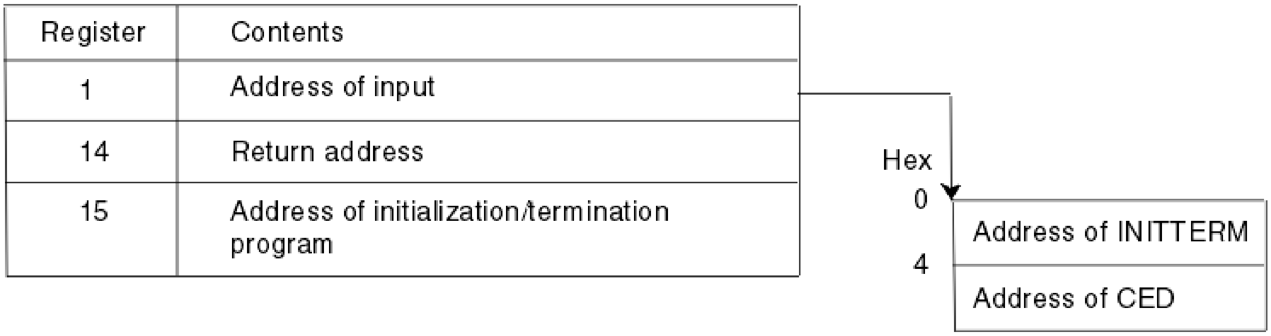
Input to the Initialization/Termination Program

Figure 8 on page 27 and Table 4 on page 27 show the input that MVSSERV makes available to the initialization/termination programs.

When MVSSERV gets control, it invokes your server initialization/termination programs in problem program state, key 8.

As shown in Figure 8 on page 27, 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 the CED (connectivity environment descriptor). Of the two:

- INITTERM indicates whether the call is for initialization or termination.
- The CED is for MVSSERV use only. (If the program issues the DEFSERV, SENDREQ, or CHSTRACE macros, it must pass the CED address.)



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

Figure 8: Registers Passed at Initialization

You can use the INITTERM mapping macro to obtain input from the INITTERM control block. For details, see “INITTERM Macro” on page 54. Table 4 on page 27 shows the INITTERM control block with the initialization input.

Table 4: INITTERM Control Block with Initialization Input

Offset Dec(Hex)	Number of Bytes	Field Name	Contents
0(0)	4	INTINIT[1]	Initialization or termination indicator. X'00000000' indicates the call is for initialization. X'00000001' indicates termination.
4(4)	4	INTWALEN[2]	Work area length. Specify the length of a work area that the program can use at termination time.
8(8)	4	INTWAPTR[2]	Work area address. Specify the address of a work area that the program can use at termination time.
12(C)	16		Reserved
28(1C)	4	INTENVRN	Address of the TSO/E 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/E command processor or uses TSO/E services such as SCAN or PARSE. For more information about the CPPL, see *z/OS TSO/E Programming Guide*.
32(20)	4		Reserved

Notes:

[1] Check for initialization or termination indicator.

2

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 must 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

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

- Open data sets that the servers need.
- Obtain storage, such as a work area to be shared among the servers, by issuing the GETMAIN macro.

The initialization/termination program makes resources available to the server by pointing to them in a server parameter list (parmlist) as part of the server definition process. When MVSSERV passes a service request to the server, it passes the server parmlist 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 servers using the DEFSERV macro. The DEFSERV macro fills in fields of a connectivity programming request block (CPRB) that does the following:

- Defines the server to MVSSERV.
- Specifies a parmlist for the server.

For details about the DEFSERV macro, see “DEFSERV Macro” on page 55.

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 parmlist. The CPRB and its significant fields are shown in Table 5 on page 28.

The DEFSERV Request CPRB

Table 5: CPRB Control Block Used to Define a Server

Offset Dec(Hex)	Number of Bytes	Field Name	Contents
0(0)	1	CRBF1	The control block's version number (first four bits) and modification level number (last four bits).
1(1)	2		Reserved
3(3)	1	CRBF4	The type of request (X'03' indicates a Define Server request).
4(4)	4	CRBCPRB	The value of C‘CPRB’.
8(8)	8		Reserved

Table 5: CPRB Control Block Used to Define a Server (continued)

Offset Dec(Hex)	Number of Bytes	Field Name	Contents
16(10)	8	CRBSNAME	The server name specified in the DEFSERV parameter SERVNAME.
24(18)	32		Reserved
56(38)	4	CRBRQPLN	The value X'0003', indicating the length of the define server parameter area.
60(3C)	4	CRBRQPRM	The address of the define server parameter area.
64(40)	48		Reserved

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 parmlist - resources passed to the server when it is called.

Figure 9 on page 29 shows the format of the define server parameter area.

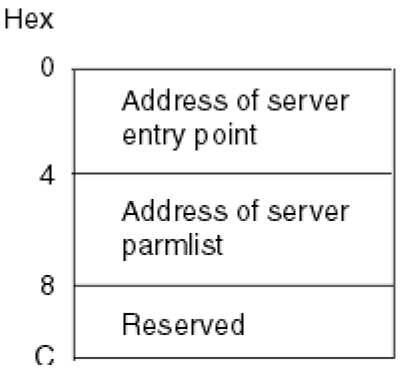


Figure 9: The Define Server Parameter Area

Sending a Service Request

An initialization/termination program can send service requests to servers that it defines. For example, at termination an initialization/termination program can check the status of the last reply (see Table 6 on page 30) sent to the PC. If the last reply had an unsuccessful return code caused by a communication failure, the initialization/termination program could send a request to the server that issued the reply, directing the server to cancel its last service.

To send a service request to a server, use the CHSDCPRB macro to create a CPRB and the SENDREQ macro to initialize and send the CPRB. For details, see “SENDREQ Macro” on page 57. The SENDREQ macro sends a service request to another server in a CPRB identical to the one shown in Table 1 on page 9.

Receiving a Service Reply

On return from the SENDREQ macro, an updated CPRB and reply buffers are returned, indicating the status of the requested service. Table 3 on page 11 shows the CPRB on return from issuing a service request.

Issuing Messages

Initialization/termination programs can issue messages to the terminal, to the MVSSERV trace data set, or to both. To issue a message and specify its destination, use the CHSTRACE macro. For details, see "CHSTRACE Macro" on page 59. For information about the MVSSERV trace data set, see "Trace Data Set" on page 44.

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 prevents all the servers in the subtask from processing any more requests.

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.

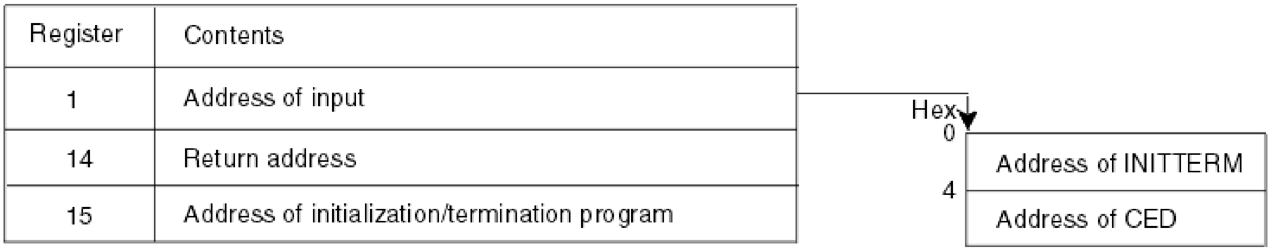
For more information about using the ESTAE macro and recovery routines, refer to *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG*.

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, preventing them from processing requests, and immediately invokes the initialization/termination program for termination.

Termination

Before MVSSERV ends, it calls the initialization/termination program again to delete the servers (if loaded) and free any resources obtained for them. The termination input to the initialization/termination program is shown in Figure 10 on page 30 and Table 6 on page 30, with the significant fields indicated.



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

Figure 10: Registers Passed at Termination

Table 6: INITTERM Control Block with Termination Input

Offset Dec(Hex)	Number of Bytes	Field Name	Contents
0(0)	4	INTINIT 1	Initialization or termination indicator. X'00000001' indicates that the call is for termination.
4(4)	4	INTWALEN	Work area length. The length of a work area, if any, specified at initialization.
8(8)	4	INTWAPTR	Work area address. The address of a work area, if any, specified at initialization.
12(C)	8	INTSNAME 2	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.

Table 6: INITTERM Control Block with Termination Input (continued)

Offset Dec(Hex)	Number of Bytes	Field Name	Contents
20(14)	4	INTRSN[3]	Return code for last reply. Contains one of the following return codes: **0(0)** Processing was successful. **4(4)** The last reply *may not* have been successfully received by the requester. **8(8)** The last reply was not successfully received by the requester. **10(A)** The last reply CPRB from the server was not valid.
24(18)	4		Reserved
28(1C)	4	INTENVRN	CPPL address (see Table 4 on page 27)
32(20)	4		Reserved

Notes:

[1] Check for initialization or termination.

[2] Check the name of the last server to send a reply (optional).

[3] If the last server was defined by the initialization/termination program, check the status of the last reply (optional). If the last reply had an unsuccessful return code, the initialization/termination program could send a request to the server that issued the reply, directing the server to cancel its last service.

Freeing Resources

The initialization/termination program must release any resources that it obtained. For example, the program must:

- Use the FREEMAIN macro to free any storage that it obtained by GETMAIN during initialization.
- Close any data sets that it opened.

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 an initialization/termination program, you must compile or assemble it and link-edit it. For information about preparing and running a program in TSO/E, see *z/OS TSO/E Programming Guide*.

Sample Initialization/Termination Program

The following initialization/termination program corresponds to the sample server in "Sample Server IBMABASE" on page 13. The initialization/termination program does the following:

- Loads the server.
- Issues the DEFSERV macro.
- Cleans up at termination.

```
**********************************************************************
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.
         LA    0,DYNSIZE             Obtain the dynamic storage size.
         GETMAIN RU,LV=(0)           Obtain the dynamic storage.
         LR    11,1                  Place the storage address in the
*                                    dynamic area register.
         USING DYNAREA,11            Establish addressability to the
*                                    dynamic area.
         ST    13,SAVEAREA+4         Save the callers savearea address.
         ST    11,8(,13)             Chain our savearea to the callers.
         LM    15,1,16(13)           Restore registers 15,0, and 1.
         LA    13,SAVEAREA           Point register 13 to our savearea.
         EJECT
**********************************************************************
* TITLE: IBMINTRM MAINLINE
*
* LOGIC: Perform server initialization/termination.
*
* OPERATION:
* 1. Determine if we are in initialization or termination.
* 2. If initialization:
*   - Call INIT_SERVER to load and define the servers to MVSSERV
*   - If the servers are defined to MVSSERV:
*     A. Exit the init/term program.
*   - Else:
*     A. Call CLEAN_UP to delete the servers.
*     B. Exit the init/term program.
* 3. Else, termination:
*   - Call CLEAN_UP to delete the servers.
* 4. Return to caller with return code.
**********************************************************************
         SPACE
         L     2,0(,1)               Load the init/term area address.
         USING INITTERM,2            Establish addressability to it.
         SPACE
         L     3,4(,1)               Load the CED address.
         USING CHSCED,3              Establish addressability to it.
         SPACE
**********************************************************************
* Determine if we are in INITIALIZATION or TERMINATION
**********************************************************************
         SPACE
         LA    4,INITIAL             Obtain the initialization equate.
         C     4,INTINIT             Are we in initialization?
         BNE   TERMINATE             No, then we must terminate.
         SPACE
**********************************************************************
* Perform INITIALIZATION processing
**********************************************************************
         SPACE
         BAL   14,INIT_SERVERS       Yes, Call INIT_SERVERS.
         LTR   15,15                 Are the servers defined to MVSSERV?
         BZ    EXIT                  Leave the init/term program.
         SPACE
**********************************************************************
* Perform TERMINATION processing
**********************************************************************
         SPACE
TERMINATE DS   0H
         BAL   14,CLEAN_UP           Call CLEAN_UP.
         EJECT
**********************************************************************
```

```
* Leave the INIT/TERM program
***********************************************************************
         SPACE
EXIT     DS    0H
         L     13,SAVEAREA+4         Restore the callers savearea
*                                    address.
         LR    2,15                  Save the return code.
         LR    1,11                  Obtain dynamic area address.
         LA    0,DYNSIZE             Obtain the dynamic storage size.
         FREEMAIN RU,LV=(0),A=(1)    Release the dynamic area.
         LR    15,2                  Restore the return code.
         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.
         EJECT
***********************************************************************
* TITLE: INIT_SERVERS
*
* LOGIC: Define the servers to MVSSERV.
*
* OPERATION:
* 1. Issue the CHSTRACE macro to output a message to the TRACE data
*    set.
* 2. Issue the GETMAIN macro to obtain the SERVER parameter storage.
* 3. Clear the SERVER parameter storage and initialize the macro
*    list forms.
* 4. Load the servers.
* 5. Issue the DEFSERV macro for each server to attempt to define
*    the server to MVSSERV.
* 6. Save the return codes.
* 7. Return to the mainline.
***********************************************************************
         SPACE
INIT_SERVERS DS 0H
         STM   14,12,SUBSAVE         Save the caller's registers.
         SPACE 2
***********************************************************************
* Issue the CHSTRACE macro to output the initialization message.
***********************************************************************
         SPACE
         CHSTRACE DEST=TRACE,CED=CHSCED,BUFFER=INIT_MSG,               *
               BUFLEN=MSG_LEN,MF=(E,CHSLIST,COMPLETE)
         SPACE
***********************************************************************
* Obtain the Server Parameter Area.
***********************************************************************
         SPACE
         LA    0,SERVER_PARMS_SIZE   Obtain the length of the server
*                                    parameter area.
         GETMAIN RU,LV=(0)
         LR    4,1                   Obtain the address of the storage.
         USING SERVPARM,4            Establish addressability to the
*                                    server parameters.
         ST    0,INTWALEN            Save the server parameter area
*                                    length.
         ST    4,INTWAPTR            Save the server parameter area
*                                    address.
         SPACE
***********************************************************************
* Initialize the macro list forms.
***********************************************************************
         SPACE
         LA    5,L'SERVER_STORAGE
         SLR   6,6
         SLR   7,7
         MVCL  4,6
         L     4,INTWAPTR            Restore server parameter area
*                                    address.
         MVC   DCBIN(SDCBIN_LEN),SDCBIN
         MVC   DCBOUT(SDCBOUT_LEN),SDCBOUT
         MVC   DCBLOG(SDCBLOG_LEN),SDCBLOG
         MVC   OPEN_LIST(SOPEN_LEN),SOPEN_LIST
         MVC   CLOSE_LIST(SCLOSE_LEN),SCLOSE_LIST
         SPACE
***********************************************************************
* Issue the LOAD macro to load the servers into storage.
***********************************************************************
         SPACE
         LOAD  EP=IBMABASE
         ST    0,SERVER_ADDR         Save IBMABASE's address.
         SPACE
         LOAD  EP=IBMABAS1
         ST    0,SERVER1_ADDR        Save IBMABAS1's address.
         SPACE
         LOAD  EP=IBMABAS2
         ST    0,SERVER2_ADDR        Save IBMABAS2's address.
         SPACE
**********************************************************************
* Initialize the SERVER parameter list.
**********************************************************************
         SPACE
         LA    5,CHSDCPRB            Get the address of the CPRB.
         SPACE
         LA    6,SERVER_STORAGE      Get the Server dynamic storage
*                                    address.
         ST    6,PARM_LIST           Place it in the server parameter.
         LA    6,DCBIN               Get the INPUT DCB address.
         ST    6,PARM_LIST+4         Place it in the server parameter.
         LA    6,DCBOUT              Get the OUTPUT DCB address.
         ST    6,PARM_LIST+8         Place it in the server parameter.
         LA    6,DCBLOG              Get the LOG DCB address.
         ST    6,PARM_LIST+12        Place it in the server parameter.
         LA    6,OPEN_LIST           Get the OPEN macro list form.
         ST    6,PARM_LIST+16        Place it in the server parameter.
         LA    6,CLOSE_LIST          Get the CLOSE macro list form.
         ST    6,PARM_LIST+20        Place it in the server parameter.
         SPACE
         LA    6,PARM_LIST           Get the address of the server
*                                    parameter list.
         SPACE
**********************************************************************
* Issue the DEFSERV macro to define the servers to MVSSERV.
**********************************************************************
         SPACE
         DEFSERV CPRB=(5),CED=(3),SERVNAME=SERVER_NAME,                *
               SERVEPA=SERVER_ADDR,SERVPARM=(6),MF=(E,DEFLIST)
         LTR   15,15                 Check the return code.
         BNZ   DEFSERV_ERROR         If it is non-zero, then leave.
         L     15,CRBCRTNC           Obtain the return code.
         LTR   15,15                 Check the return code.
         BNZ   DEFSERV_ERROR         If it is non-zero, then leave.
         SPACE
         DEFSERV CPRB=(5),CED=(3),SERVNAME=SERVER1_NAME,               *
               SERVEPA=SERVER1_ADDR,SERVPARM=(6),MF=(E,DEFLIST)
         LTR   15,15                 Check the return code.
         BNZ   DEFSERV_ERROR         If it is non-zero, then leave.
         L     15,CRBCRTNC           Obtain the return code.
         LTR   15,15                 Check the return code.
         BNZ   DEFSERV_ERROR         If it is non-zero, then leave.
         SPACE
         DEFSERV CPRB=(5),CED=(3),SERVNAME=SERVER2_NAME,               *
               SERVEPA=SERVER2_ADDR,SERVPARM=(6),MF=(E,DEFLIST)
         LTR   15,15                 Check the return code.
         BNZ   DEFSERV_ERROR         If it is non-zero, then leave.
         L     15,CRBCRTNC           Obtain the return code.
         LTR   15,15                 Check the return code.
         BNZ   DEFSERV_ERROR         If it is non-zero, then leave.
         B     LEAVE                 Everything is O.K., so leave.
         SPACE
DEFSERV_ERROR DS 0H
         LA    15,4                  Set a bad return code.
         SPACE
LEAVE    DS    0H
         L     14,SUBSAVE            Restore the caller's registers
         LM    0,12,SUBSAVE+8        except for 15 (return code).
         BR    14                    Return to caller with return code.
         EJECT
**********************************************************************
* TITLE: CLEAN_UP
*
* LOGIC: Remove the servers.
*
* OPERATION:
* 1. Issue the CHSTRACE macro to output a message to the TRACE data
*    set.
* 2. Issue the FREEMAIN macro to release the SERVER parameter storage.
* 3. Delete the servers.
* 4. Return to the mainline.
**********************************************************************
         SPACE
CLEAN_UP DS 0H
         STM   14,12,SUBSAVE         Save the caller's registers.
         SPACE 2
**********************************************************************
```

```
* Issue the CHSTRACE macro to output the termination message.
***********************************************************************
         SPACE
         CHSTRACE DEST=TRACE,CED=CHSCED,BUFFER=TERM_MSG,               *
               BUFLEN=MSG_LEN,MF=(E,CHSLIST,COMPLETE)
         SPACE
***********************************************************************
* Release the Server Parameter Area.
***********************************************************************
         SPACE
         L     1,INTWAPTR            Obtain the address of the server
*                                    parameter area.
         L     0,INTWALEN            Obtain the length of the server
*                                    parameter area.
         FREEMAIN RU,LV=(0),A=(1)
         SPACE
***********************************************************************
* Issue the DELETE macro to delete the servers from storage.
***********************************************************************
         SPACE
         DELETE EP=IBMABASE
         SPACE
         DELETE EP=IBMABAS1
         SPACE
         DELETE EP=IBMABAS2
         SPACE
         LA    15,0
         L     14,SUBSAVE            Restore the caller's registers
         LM    0,12,SUBSAVE+8        except for 15 (return code).
         BR    14                    Return to caller with return code.
         EJECT
***********************************************************************
* Constants.
***********************************************************************
         SPACE
***********************************************************************
* SERVER names.
***********************************************************************
         SPACE
SERVER_NAME DC  CL8'IBMABASE'        Server name.
SERVER1_NAME DC CL8'IBMABAS1'        Server name.
SERVER2_NAME DC CL8'IBMABAS2'        Server name.
         SPACE
***********************************************************************
* TRACE data set messages.
***********************************************************************
         SPACE
INIT_MSG DC    CL80' Initialization/termination program IBMINTRM entered*
                for INITIALIZATION.'
TERM_MSG DC    CL80' Initialization/termination program IBMINTRM entered*
                for TERMINATION.'
MSG_LEN  DC    A(*-TERM_MSG)         Length of message
         SPACE
***********************************************************************
* OPEN macro (static list form).
***********************************************************************
         SPACE
SOPEN_LIST OPEN (,(INPUT),,(OUTPUT),,(EXTEND)),MF=L
SOPEN_LEN EQU  *-SOPEN_LIST
         SPACE
***********************************************************************
* CLOSE macro (static list form).
***********************************************************************
         SPACE
SCLOSE_LIST CLOSE (,,,,,),MF=L
SCLOSE_LEN EQU *-SCLOSE_LIST
         EJECT
***********************************************************************
* DCB macro (static input).
***********************************************************************
         SPACE
SDCBIN   DCB   DDNAME=CUSTRECS,DSORG=PS,MACRF=GM
SDCBIN_LEN EQU *-SDCBIN
         EJECT
***********************************************************************
* DCB macro (static output).
***********************************************************************
         SPACE
SDCBOUT  DCB   DDNAME=ACCTRECS,DSORG=PS,MACRF=PM
SDCBOUT_LEN EQU *-SDCBOUT
         EJECT
***********************************************************************
```

```
* DCB macro (static log).
**********************************************************************
         SPACE
SDCBLOG  DCB   DDNAME=LOGTRANS,DSORG=PS,MACRF=PM
SDCBLOG_LEN EQU *-SDCBLOG
         EJECT
**********************************************************************
* Dynamic Area.
**********************************************************************
         SPACE
DYNAREA  DSECT
         SPACE
SAVEAREA DS    18F                   IBMINTRM's save area.
SUBSAVE  DS    15F                   IBMINTRM subroutine's save area.
SERVER_ADDR DS F                     Used to hold the servers entry
*                                    point.
SERVER1_ADDR DS F                    Used to hold the servers entry
*                                    point.
SERVER2_ADDR DS F                    Used to hold the servers entry
*                                    point.
         SPACE
**********************************************************************
* Issue the DEFSERV macro list form to supply a parameter list.
**********************************************************************
         SPACE
DEFLIST  DEFSERV  MF=L
         SPACE
**********************************************************************
* Issue the CHSTRACE macro list form to supply a parameter list.
**********************************************************************
         SPACE
         CHSTRACE MF=(L,CHSLIST)
         EJECT
**********************************************************************
* CPRB
**********************************************************************
         SPACE
         CHSDCPRB DSECT=NO
         SPACE
DYNSIZE  EQU   *-DYNAREA             Size of the dynamic area.
         EJECT
**********************************************************************
* Server parameters.
**********************************************************************
         SPACE
SERVPARM DSECT
         SPACE
**********************************************************************
* Dynamic storage for server (saves GETMAIN and FREEMAIN in server)
*
* NOTE:  SERVER_STORAGE must be changed if the DYNAMIC area for
*        IBMABASE, IBMABAS1 and IBMABAS2 exceeds the current size.
**********************************************************************
         SPACE
SERVER_STORAGE DS CL500
         SPACE
**********************************************************************
* OPEN macro (dynamic list form).
**********************************************************************
         SPACE
OPEN_LIST OPEN (,(INPUT),,(OUTPUT),,(OUTPUT)),MF=L
         SPACE
**********************************************************************
* CLOSE macro (dynamic list form).
**********************************************************************
         SPACE
CLOSE_LIST CLOSE (,,,,,),MF=L
         EJECT
**********************************************************************
* DCB macro (dynamic input).
**********************************************************************
         SPACE
DCBIN    DCB   DDNAME=CUSTRECS,DSORG=PS,MACRF=GM
         EJECT
**********************************************************************
* DCB macro (dynamic output).
**********************************************************************
         SPACE
DCBOUT   DCB   DDNAME=ACCTRECS,DSORG=PS,MACRF=PM
         EJECT
**********************************************************************
* DCB macro (dynamic log).
```

```
***********************************************************************
         SPACE
DCBLOG   DCB   DDNAME=LOGTRANS,DSORG=PS,MACRF=PM
         EJECT
         SPACE
***********************************************************************
* Server parameter list, contains the addresses of:
*        The Server Dynamic Storage
*        The INPUT DCB
*        The OUTPUT DCB
*        The LOG DCB
*        The OPEN LIST FORM
*        The CLOSE LIST FORM
***********************************************************************
         SPACE
PARM_LIST DS 6A
         SPACE
SERVER_PARMS_SIZE EQU *-SERVPARM     Size of server parameter area.
         EJECT
***********************************************************************
* CED mapping.
***********************************************************************
         SPACE
         CHSCED   DSECT=YES
         EJECT
***********************************************************************
* INIT/TERM mapping.
***********************************************************************
         SPACE
         INITTERM DSECT=YES
         END   IBMINTRM
```

Chapter 4. Writing an Access Method Driver

This chapter describes the role of MVSSERV access method drivers and MVSSERV's access method driver interface.

What is an Access Method Driver?

Access method drivers (AMDs) are programs that provide the communications link between MVSSERV and a PC. Access method drivers on the host and PC work in pairs, passing data between them in a format appropriate to the mode of PC-to-Host attachment. At the host, MVSSERV's access method driver converts requests into CPRBs, sends them to servers, and converts reply CPRBs back into the proper communications format for transmission to the PC.

MVSSERV provides access method drivers that manage communications with PCs attached to the host through an IBM 3174 or 3274 control unit in:

- Distributed Function Terminal (DFT) mode
- Control Unit Terminal (CUT) mode

MVSSERV also provides an AMD interface that lets you write and install other access method drivers to support other modes of attachment. Figure 11 on page 39 shows the position of the AMD interface in the MVSSERV environment.

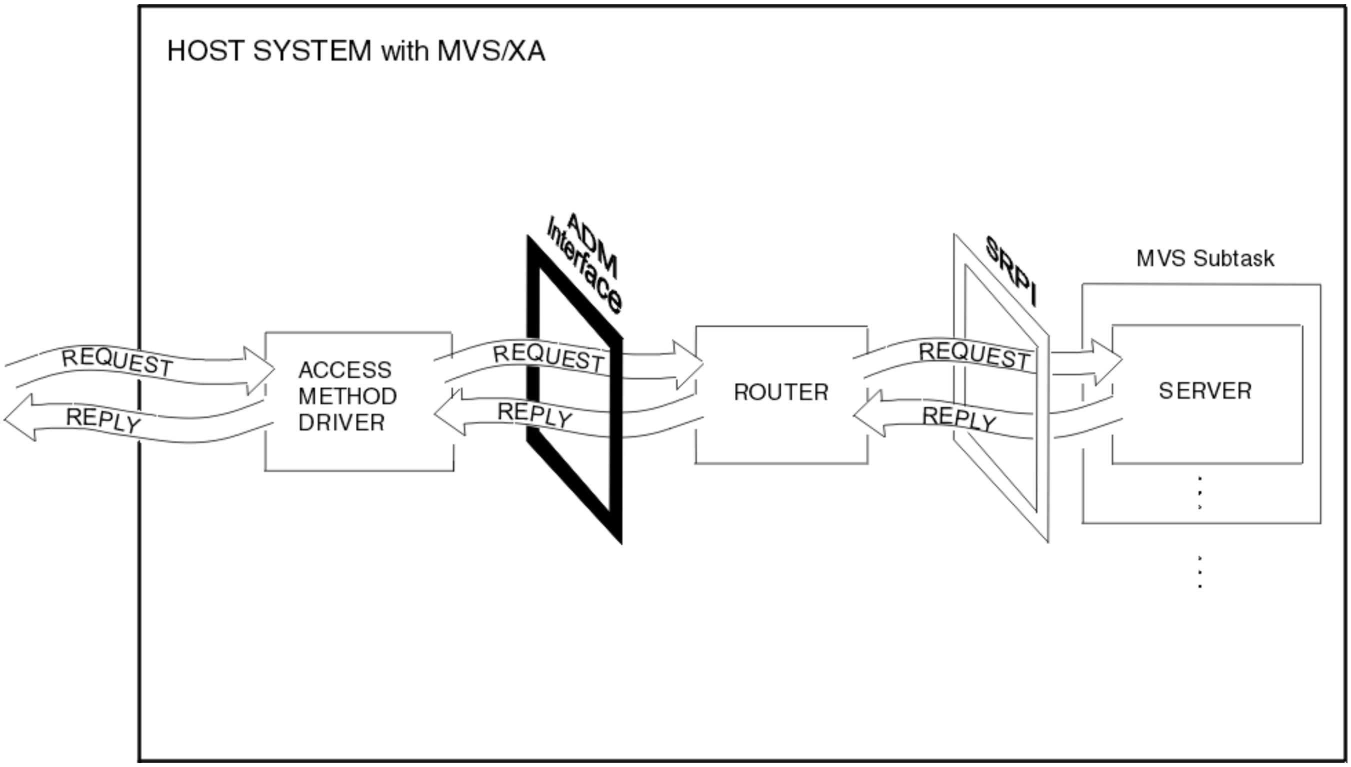


Figure 11: The MVSSERV Enhanced Connectivity Environment

Using the AMD Interface

The purpose of the AMD interface is to let installations write and install access method drivers to support different modes of PC-to-host attachment. If an access method driver is properly installed on MVS and defined to MVSSERV, MVSSERV invokes the access method driver. Then MVSSERV routes service requests from the access method driver to the servers, and routes service replies back to the access method driver. An access method driver on the host must have a counterpart on the PC; the access method drivers are responsible for ensuring that requests from the PC reach MVSSERV in the proper format, and that replies from MVSSERV reach the PC properly.

Writing an Access Method Driver

You can write access method drivers to support different modes of PC-to-host attachment. An access method driver must do the following:

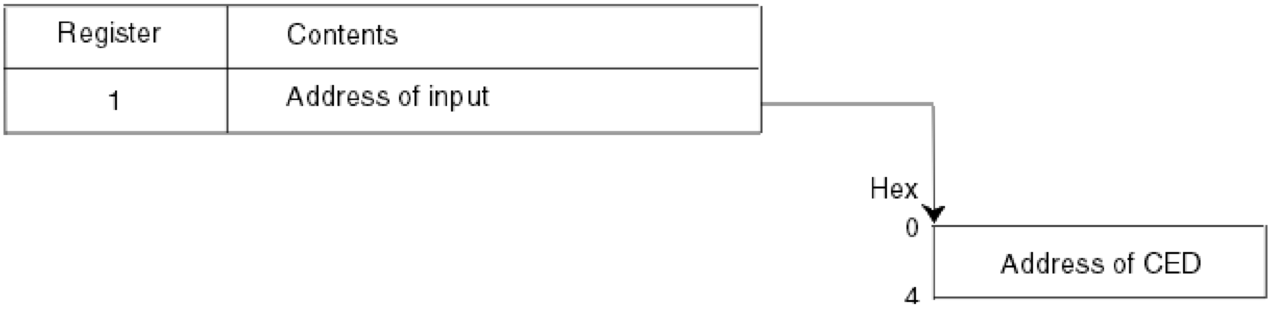
- Receive requests from the PC
- Use the SENDREQ macro to send the requests to servers
- Receive the server replies
- Send the replies to the requester in the appropriate form.
- At termination, free any resources and notify the PC counterpart.

Installing and Defining an Access Method Driver

To make an access method driver available to MVSSERV, you must install the access method driver on MVS and define it in MVSSERV's input parameter data set. See Chapter 5, "Installing Programs and Data Sets for Use with MVSSERV," on page 43for information about installing access method drivers and defining them in the MVSSERV input parameter data set.

AMD Invocation

When MVSSERV finds an access method driver defined in the input parameter data set, MVSSERV loads and invokes it. MVSSERV passes a single parameter to the access method driver: the address of a Connectivity Environment Descriptor (CED), as shown in Figure 12 on page 40.



Register	Contents
1	Address of input

Figure 12: MVSSERV Input to an Access Method Driver

The CED address is for MVSSERV use only; to issue the SENDREQ or CHSTRACE macros, the access method driver must pass the CED address in the macro.

Considerations for Writing Access Method Drivers

Installation-written access method drivers must:

- Run AMODE 31 and RMODE ANY.
- Provide their own recovery routines.
- Display their own screens or logos. MVSSERV does not display its logo when a user-defined access method driver is running.

In addition, access method drivers may not issue the DEFSERV macro to define servers. In order for MVSSERV to route requests to servers, the servers must be defined to MVSSERV by initialization/termination programs or by other servers.

Sending a Service Request

The primary function of an access method driver is to receive service requests, send the service requests to servers, receive service replies from the servers, and send the service replies back to the requester. Service requests must be sent to servers in a CPRB control block.

To send a service request, you can use the CHSDCPRB macro to create a CPRB and the SENDREQ macro to initialize the CPRB with the request and send it to the server. For details, see "SENDREQ Macro" on page 57. The SENDREQ macro sends the service request in a CPRB as shown in Table 1 on page 9.

Receiving a Service Reply

On return from a service request, an updated CPRB and reply buffers are returned, indicating the results of the requested service. (Table 3 on page 11 shows the CPRB on return from a service request.) The access method driver on the host must convert the reply information into the appropriate format and send the information to its PC counterpart. The PC counterpart must convert the information back into a CPRB and send it to the requester. For complete information about the format of the CPRB that the requester expects, see the *IBM Programmer's Guide to the Server-Requester Programming Interface for the IBM Personal Computer and the IBM 3270 PC*.

Issuing Messages

An access method driver can issue messages to the terminal, to the MVSSERV trace data set, or to both. To issue a message and specify its destination, use the CHSTRACE macro. For details, see "CHSTRACE Macro" on page 59. For information about the MVSSERV trace data set, see "Trace Data Set" on page 44.

Sample Access Method Driver

The following sample is provided to illustrate use of the AMD interface. The sample does not represent a functional access method driver.

```
IBMAMD   CSECT
IBMAMD   AMODE 31
IBMAMD   RMODE ANY
         STM   14,12,12(13)          Save the caller's registers.
         LR    12,15                 Establish addressability within
         USING IBMAMD,12             this CSECT.
         LA    0,DYNSIZE             Obtain the dynamic storage size.
         GETMAIN RU,LV=(0)           Obtain the dynamic storage.
         LR    11,1                  Place the storage address in the
*                                    dynamic area register.
         USING DYNAREA,11            Establish addressability to the
*                                    dynamic area.
         ST    13,SAVEAREA+4         Save the caller's savearea address.
         ST    11,8(,13)             Chain our savearea to the callers.
         LM    15,1,16(13)           Restore registers 15,0, and 1.
         LA    13,SAVEAREA           Point register 13 to our savearea.
         L     2,0(,1)               Obtain the CED address.
         USING CHSCED,2              Establish addressability to it.
         EJECT
***********************************************************************
* TITLE: IBMAMD MAINLINE
*
* LOGIC: Receive the PC request and route it to the appropriate
*        server.
***********************************************************************
         SPACE 2
*
* An Access Method Driver receives and sends communications and is
* responsible for initiating service requests on the host.
*
* The format of the communication depends on the protocol
* that is used to communicate between the requester and the host
* AMD.
*
* The AMD can use the CHSTRACE macro to issue messages to the
* terminal and/or the trace data set.  Messages can indicate
* that a communication was received and the type of communication
* (such as a valid server request, invalid server request, termination
* request, and so on).
*
* If a valid request for a server was received and all of the
* parameters were received, the AMD can issue the SENDREQ macro to
* invoke the server.
*
* Upon return from the SENDREQ macro, the AMD should send the reply
* to the requester.
*
* The AMD should then await another request or reply communication
* until a predetermined termination indicator is received.  When the
* AMD terminates, it returns control to MVSSERV.

         EJECT
***********************************************************************
* Leave the AMD.
***********************************************************************
         SPACE
EXIT     DS    0H
         L     13,SAVEAREA+4         Restore the caller's savearea
*                                    address.
         LR    2,15                  Save the return code.
         LR    1,11                  Obtain dynamic area address.
         LA    0,DYNSIZE             Obtain the dynamic storage size.
         FREEMAIN RU,LV=(0),A=(1)    Release the dynamic area.
         LR    15,2                  Restore the return code.
         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.
         EJECT
***********************************************************************
* Dynamic Area.
***********************************************************************
         SPACE
DYNAREA  DSECT                       DYNAMIC area common mapping
         SPACE
SAVEAREA DS    18F                   Save area.
         SPACE
***********************************************************************
* Issue the CHSTRACE macro list form to supply a parameter list.
***********************************************************************
         SPACE
         CHSTRACE MF=(L,CHSLIST)
         SPACE
***********************************************************************
* Issue the SENDREQ macro list form to supply a parameter list.
***********************************************************************
         SPACE
         SENDREQ MF=(L,SENDLIST)
         EJECT
************************************************************************
* CPRB
************************************************************************
         SPACE
         CHSDCPRB DSECT=NO
DYNSIZE  EQU   *-DYNAREA
         EJECT
***********************************************************************
* CED mapping.
***********************************************************************
         SPACE
         CHSCED  DSECT=YES
         END   IBMAMD
```

Chapter 5. Installing Programs and Data Sets for Use with MVSSERV

This chapter describes how to install servers, initialization/termination programs, and access method drivers for use with MVSSERV. This chapter uses the term *program* to collectively refer to the above programs.

After a program has been written, compiled or assembled, and link-edited, you must install the program to make it available to users and to MVSSERV.

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

1. Install the program in a library.
2. Use the input parameter data set to identify initialization/termination programs and access method drivers.

Installing a Program

You can install a program in one of two ways:

- In a STEPLIB
- In a system library.

In a STEPLIB

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

To allocate a STEPLIB to a user, add the following JCL in the user's logon procedure:

```
//STEPLIB DD DSN=data_set_name,DISP=SHR
```

In a System Library

To make a program available to all system users, copy it to a member of a system library. The system library can be one that is defined in the linklist concatenation, such as SYS1.LINKLIB, or it can be SYS1.LPALIB, which is allocated at IPL and therefore always available.

Programs in system libraries should be (and programs in SYS1.LPALIB *must* be) reentrant--that is, they must use dynamic storage to allow multiple and concurrent executions of the program. The sample programs in this document are reentrant, and macro descriptions in Chapter 7, “Macro Syntax and Parameters,” on page 53 indicate steps to take to make a program reentrant.

Using the Input Parameter Data Set

Before issuing the MVSSERV command, you must name, in the input parameter data set, your initialization/termination programs and optionally, any access method driver. From this input, MVSSERV invokes the access method driver, if any, to manage communications, and 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 FILE(CHSPARM) DA('data_set_name') NEW LRECL(80) RECFM(F)
```

To make the input parameter data set available to an MVSSERV user, allocate the existing data set in the user's logon procedure, or in a CLIST, REXX exec, 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, REXX exec, or ISPF dialog, you can use the following command:

  ```
  ALLOCATE FILE(CHSPARM) DA('data_set_name') SHR
  ```

Be sure that the user has security authorization 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 or an access method driver, starting in column 1. The name can have up to eight characters, including the characters A-Z, 0-9, @, #, and $. The first character cannot be 0-9.

To distinguish access method drivers from initialization/termination programs, include the TYPE keyword in the input record anywhere between columns 9 and 72. An access method driver must be followed by the keyword TYPE(A); initialization/termination programs can be followed by the keyword TYPE(I) or by no keyword.

For example, in the following lines from an input parameter data set, the first two programs are initialization/termination programs and the third (AMDPROG) is an access method driver.

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

INTPROG1
INTPROG2  TYPE(I)
AMDPROG   TYPE(A)
```

MVSSERV invokes all initialization/termination programs it finds in the input parameter data set, but invokes only the first access method driver it finds, ignoring any other TYPE(A) programs.

Additional MVSSERV Data Sets

In addition to 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, REXX exec, or ISPF dialog that invokes MVSSERV, or in line mode TSO/E. 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, as well as messages issued by the CHSTRACE macro. For messages and data to be received in the trace data set, MVSSERV must be invoked with the TRACE or IOTRACE option. The level of trace data from MVSSERV varies with the option used:

- TRACE -- records events in the MVSSERV session, such as requests for servers, and MVSSERV errors.
- IOTRACE -- 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 FILE(CHSTRACE) DA('data_set_name') NEW LRECL(80) RECFM(F)
```

To make the trace data set available to an MVSSERV user, allocate the existing data set in the user's logon procedure, or in a CLIST, REXX exec, 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, REXX exec, or ISPF dialog, you can use the following command:

  ```
  ALLOCATE FILE(CHSTRACE) DA('data_set_name') OLD
  ```

For more information about the MVSSERV trace parameters and syntax, refer to Chapter 6, “Testing and Diagnosis,” on page 49.

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 dump data sets and how to read them, refer to *z/OS MVS Diagnosis: Tools and Service Aids*.

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, REXX exec, or ISPF dialog that issues MVSSERV for the user. Each user must have their own dump data set.

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

```
//SYSUDUMP DD DSN=data_set_name,DISP=OLD
```

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

```
ALLOCATE FILE(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 FILE(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, REXX exec, 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, REXX exec, or ISPF dialog, you can use the following command:

```
ALLOCATE FILE(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 OPEN 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 the following:

- *z/OS MVS System Codes*
- *z/OS MVS System Messages, Vol 1 (ABA-AOM)*
- *z/OS MVS System Messages, Vol 2 (ARC-ASA)*
- *z/OS MVS System Messages, Vol 3 (ASB-BPX)*
- *z/OS MVS System Messages, Vol 4 (CBD-DMO)*
- *z/OS MVS System Messages, Vol 5 (EDG-GFS)*
- *z/OS MVS System Messages, Vol 6 (GOS-IEA)*
- *z/OS MVS System Messages, Vol 7 (IEB-IEE)*
- *z/OS MVS System Messages, Vol 8 (IEF-IGD)*
- *z/OS MVS System Messages, Vol 9 (IGF-IWM)*
- *z/OS MVS System Messages, Vol 10 (IXC-IZP)*

Chapter 6. Testing and Diagnosis

This chapter describes the steps to follow to test servers and diagnose any server problems.

Testing Servers

After you have written and installed a server, you must 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 must start an MVSSERV session on TSO/E. On the PC, you must 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 following data sets available for your MVSSERV session:

   - 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.

   For information about allocating the data sets, refer to Chapter 5, "Installing Programs and Data Sets for Use with MVSSERV," on page 43. You may also want to have the dump data set and the dump suppression data set described in Chapter 5, "Installing Programs and Data Sets for Use with MVSSERV," on page 43.
2. To start the MVSSERV session, log on to TSO/E and issue the MVSSERV command.

   MVSSERV has the following syntax, with the default underlined:

   ```
   MVSSERV       [ NOTRACE ]
                 [ TRACE   ]
                 [ IOTRACE ]
   ```

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

   Note: The method used to refresh the MVSSERV logo depends on your type of terminal support.
3. Switch to the PC session. (If you are using a PC other than the 3270 PC, issue the appropriate Enhanced Connectivity Facility command for the PC.)
4. Invoke the requester that corresponds to the server you want to test.
5. Respond to 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 PF3 key to end MVSSERV.
8. Note any messages that appear on your screen. Each message has a message ID, beginning with CHS. In TSO/E, you can obtain online help for MVSSERV messages by typing the message ID in the following command:

   ```
   HELP MVSSERV MSG(CHSxxxxxxxx)
   ```

9. Read the trace data set. Because you used the TRACE option 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. The trace data set should also contain any messages that the server issued with the default, TRACE, or BOTH options of the CHSTRACE macro.

   For information about reading the trace data set messages, see "Diagnosing Servers" on page 50.
10. When the server works properly, you may want to copy it to a system library such as SYS1.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, REXX exec, or ISPF dialog, or in line mode TSO/E. After you copy the server to a system library, be sure to retest it.

Diagnosing Servers

This topic describes how to use information in the MVSSERV trace data set to diagnose and correct server problems.

Reading the Trace Data Set

When MVSSERV is issued with the TRACE or IOTRACE option, the trace data set contains messages from MVSSERV and any messages issued from servers, initialization/termination programs, or access method drivers using the CHSTRACE macro with options TRACE or BOTH.

To see messages about your most recent MVSSERV session, you can edit, browse, or print the MVSSERV trace data set. For explanations of the messages from MVSSERV, see *z/OS TSO/E Messages*.

The message explanations include information about what action, if any, you must take when you see a message.

The MVSSERV 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 CPRB address that was not valid.

Internal errors 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.

Figure 13 on page 51 shows a sample of a trace data set obtained using the TRACE option of MVSSERV. 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 13: Sample Trace Data Set

For explanations of messages that appear in your MVSSERV trace data set, look up the message ID (CHSxxxxxxx) in *z/OS TSO/E Messages*.

The message explanations tell what happened and why, and tell what action you should take (if any) to correct a problem.

Chapter 7. Macro Syntax and Parameters

This chapter describes the syntax and parameters of the following MVSSERV macros:

Table 7: MVSSERV Macros

Macro	Function	On page:
CHSDCPRB	CPRB mapping macro	“CHSDCPRB Macro” on page 53
CHSCED	CED mapping macro	“CHSCED Macro” on page 54
INITTERM	INITTERM mapping macro	“INITTERM Macro” on page 54
DEFSERV	Server definition macro	“DEFSERV Macro” on page 55
SENDREQ	Send request macro	“SENDREQ Macro” on page 57
CHSTRACE	Message issuing macro	“CHSTRACE Macro” on page 59

CHSDCPRB Macro

The CHSDCPRB macro provides a CPRB mapping DSECT or builds code to acquire storage for and partially initialize a CPRB control block. You can use the CHSDCPRB macro to:

- Access the fields of a CPRB to obtain service request input.
- Create a CPRB to use with the DEFSERV or SENDREQ macros.

Accessing the CPRB

Servers receive service request input in the CPRB. A server can use the CHSDCPRB macro to access the fields of a CPRB to obtain the input. To access a CPRB, use the CHSDCPRB macro with the following 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 to access a CPRB, see “Sample Servers” on page 12. Table 1 on page 9 shows the service request CPRB that servers access using macro CHSDCPRB.

Creating a CPRB for the DEFSERV or SENDREQ Macro

Before issuing the DEFSERV or SENDREQ macro, a program must create a CPRB. To create a CPRB, you can use the CHSDCPRB macro with the following syntax:

```
[label] CHSDCPRB DSECT=NO
```

Note: If the program is reentrant, use the GETMAIN macro to obtain storage for the CPRB.

For an example of using the CHSDCPRB macro with DEFSERV, see “Sample Initialization/Termination Program” on page 32.

For an example of using the CHSDCPRB macro with SENDREQ, see “Sample Servers” on page 12.

A program can use the same CPRB repeatedly to define multiple servers or send multiple requests. Therefore, a program only needs to issue the CHSDCPRB macro once to create one CPRB.

If you use the CHSDCPRB macro to obtain storage for the CPRB dynamically, the storage is freed when the program ends. If you use the GETMAIN macro to obtain storage, you must use the FREEMAIN macro to release it.

CHSCED Macro

MVSSERV's connectivity environment descriptor (CED) contains addresses that must be included in the DEFSERV, SENDREQ, and CHSTRACE macros. Programs can use the CHSCED mapping macro to obtain these addresses from the CED. The CHSCED macro has the following syntax:

```
[label] CHSCED [DSECT=YES|NO]
```

Code the macro with DSECT=YES (or omit the DSECT parameter) to build a DSECT that maps the CED fields. You can use the label CHSCED to address the control block with an assembler USING statement. For an example of using the CHSCED macro see “Sample Servers” on page 12.

Table 8 on page 54 shows the fields of the CED.

Table 8: Connectivity Environment Descriptor (CED)

Offset Dec(Hex)	Number of Bytes	Field Name	Contents
0(0)	4	CEDROUT	Address of router for DEFSERV and SENDREQ requests.
4(4)	8		Reserved
12(C)	4	CEDTRCE	Address of trace facility for CHSTRACE requests.
16(10)	80		Reserved

INITTERM Macro

The INITTERM control block provides input to server initialization/termination programs when MVSSERV begins and ends. Use the INITTERM mapping macro in an initialization/termination program to access fields of the INITTERM control block. The INITTERM macro has the following syntax:

```
[label] INITTERM [DSECT=YES|NO]
```

Code the macro with DSECT=YES (or omit the DSECT parameter) to build a DSECT that maps 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 INITTERM macro, see “Sample Initialization/Termination Program” on page 32.

Table 9 on page 55 shows the INITTERM control block. Note that some fields contain input for termination only.

Table 9: INITTERM Control Block

Offset Dec(Hex)	Number of Bytes	Field Name	Contents
0(0)	4	INTINIT	Initialization or termination indicator. X'0000' indicates initialization; X'0001' indicates termination.
4(4)	4	INTWALEN	Work area length. The length of a work area, if any, specified at initialization.
8(8)	4	INTWAPTR	Work area address. The address of a work area, if any, specified at initialization.
12(C)	8	INTSNAME[1]	Name of the last server that sent 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.
20(14)	4	INTRSN[1]	Return code for last reply. Contains one of the following return codes: **0(0)** Processing was successful. **4(4)** The last reply *may not* have been successfully received by the requester. **8(8)** The last reply was not successfully received by the requester. **10(A)** The last reply CPRB from the server was not valid.
24(18)	4		Reserved
28(1C)	4	INTENVRN	CPPL address. The CPPL must be in register 1 if a program invokes a TSO/E command processor or uses TSO/E services such as SCAN or PARSE.
32(20)	4		Reserved

Note:

[1] Input for termination only.

DEFSERV Macro

To define servers to MVSSERV, use the DEFSERV macro. Initialization/termination programs can issue the DEFSERV macro to define servers, and servers can also issue the DEFSERV macro to define other servers.

Register Contents for DEFSERV

Before issuing the DEFSERV macro, you must set register 13 to point to a 72-byte save area:

Register 13
Address of a standard 72-byte 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.

DEFSERV Syntax and Parameters

DEFSERV Macro Syntax shows the syntax of the DEFSERV macro. For an example of the DEFSERV macro, see “Sample Initialization/Termination Program” on page 32.

DEFSERV Macro Syntax

```
EXECUTE FORM

  label          DEFSERV    CPRB=address,
                            CED=address,
                            SERVNAME=server_name,
                            SERVEPA=server_address,
                            SERVPARM=parmlist_address
                            MF=(E,plist_name)
LIST FORM

  plist_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. The address must be in the same addressing mode (AMODE) as the issuing program.

Execute Form

CPRB=*address*
Specify the address of the DEFSERV CPRB. The CPRB must begin on a fullword boundary.

CED=*address*
Specify the address of the CED that was passed as input to the issuing program. (To map the CED, use the CHSCED macro.)

SERVNAME=*server_name*
Specify the name of the server being defined. You can also specify a general register (2–12) that points to an 8-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 this 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 (parmlist). If no parmlist is desired, code SERVPARM=0. The server parmlist should point to any resources that the issuing program obtained for the server, such as shared data sets and storage. MVSSERV passes this parmlist to the server when it calls the server to handle a service request.

MF=(E,*plist_name*)
Specify the name of a 20-byte area that will contain the DEFSERV parameter list (plist):

- The addresses of the CPRB and CED (8 bytes)
- The server entry point address and server parmlist address (the *define server parameter area* -- 12 bytes).

List Form

plist name MF=L
generates 20 bytes of storage to contain the addresses of the CPRB and CED (8 bytes) and the define server parameter area (12 bytes). The DEFSERV macro fills in this storage. The label on this statement must match the DEFSERV **plist 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 20 bytes of storage, and the FREEMAIN macro to release the storage when finished processing.

The DEFSERV CPRB

The DEFSERV macro fills in a CPRB as shown in Table 10 on page 57. The DEFSERV macro sends the CPRB to MVSSERV, which uses the CPRB to identify the server name with the server's address and parmlist.

Table 10: CPRB Control Block Used to Define a Server

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

For a list of return codes from the DEFSERV macro, see Chapter 8, “MVSSERV Return Codes,” on page 63.

SENDREQ Macro

To send service requests to servers that are defined in the current MVSSERV session, use the SENDREQ macro. Servers, initialization/termination programs, and access method drivers can issue the SENDREQ macro.

Register Contents for SENDREQ

Before issuing the SENDREQ macro, you must set register 13 to point to a 72-byte save area:

Register 13
Address of a standard 72-byte save area

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

SENDREQ Syntax and Parameters

SENDREQ Macro Syntax shows the syntax of the SENDREQ macro. Optional parameters are enclosed in brackets. For an example of the SENDREQ macro, see “Sample Server IBMABASE” on page 13.

```
EXECUTE FORM

  label   SENDREQ  CPRB=name or address,
                   CED=name or address,
                   SERVER=name or address,
                  [FUNCTION=name or address,]
                  [REQPARM=(address,length),]
                  [REQDATA=(address,length),]
                  [REPPARM=(address,length),]
                  [REPDATA=(address,length),]
                  [RETCODE=address,]
                   MF=(E,plist_address[,COMPLETE])
LIST FORM

          SENDREQ  MF=(L,plist_address[,attr])
```

Note: The addresses can be any address valid in an RX instruction, or the number of a general register (2)–(12) enclosed in parentheses. Addresses must be in the same addressing mode (AMODE) as the issuing program.

Execute Form

CPRB=*name* or *address*
Specify the name, or address in a register (2–12), of the CPRB control block. The CPRB must be obtained by the invoker of SENDREQ and must begin on a fullword boundary. (To map the CPRB, use the CHSDCPRB macro.)

CED=*name* or *address*
Specify the name, or address in a register (2–12), of the CED that was passed as input to the invoking program. (To map the CED, use the CHSCED macro.)

SERVER=*name* or *address*
Specify the name, or address in a register (2–12), of an field containing the name of the server to which the request is being sent. The maximum length of the field is eight characters.

[FUNCTION=*name* or *address*]
Specify the name, or address in a register (2–12), of a 2-byte field containing the function ID of the service function being requested. If FUNCTION is omitted, it defaults to 0.

[REQPARM=(*address*,*length*)]
Specify data describing the request parameter list to be passed to the server.

Provide the names, or addresses in registers (2–12), of a 4-byte field containing the address and a 4-byte field containing the length of the request parameter list. The maximum length is 32763 bytes. If address or length is omitted, it defaults to 0. If REQPARM is omitted, no request parameter list is passed.

[REQDATA=(*address*,*length*)]
Specify data describing the request data area to be passed to the server.

Provide the names, or addresses in registers (2–12), of a 4-byte field containing the address and a 4-byte field containing the length of the request data area. The maximum length is 65535 bytes. If address or length is omitted, it defaults to 0. If REQDATA is omitted, no request data area is passed.

[REPPARM=(*address*,*length*)]
Specify data describing the reply parameter area to be passed to the server.

Provide the names, or addresses in registers (2–12), of a 4-byte field containing the address and a 4-byte field containing the length of the reply parameter list. The maximum length is 32763 bytes. If address or length is omitted, it defaults to 0. If REPPARM is omitted, no reply parameter list is passed.

[REPDATA=(*address*,*length*)]
Specify data describing the reply data area to be passed to the server.

Provide the names, or addresses in registers (2–12), of a 4-byte field containing the address and a 4-byte field containing the length of the reply data area. The maximum length is 65535 bytes. If address or length is omitted, it defaults to 0. If REPDATA is omitted, no reply data area is passed.

[RETCODE=*variable*]
Specify the name or address of a 4-byte output variable to receive the SENDREQ return code from register 15. If you omit this parameter, you must obtain the return code from register 15.

MF=(E,*plist_address*[,COMPLETE])
specifies the execute form of the macro and the address of a storage area for the macro parameter list. The execute form generates code to put the parameters into a parameter list and invoke the desired server.

[,COMPLETE]
MVSSERV performs complete syntax checking, verifying that required SENDREQ parameters are specified and supplying default values for omitted optional parameters.

List Form

MF(L,*plist_address*[,*attr*])
Specify the list form of the macro and the address of a storage area for the macro parameter list. The list form defines an area to contain the parameter list.

[,*attr*]

Specify an optional input string that contains any special attributes for the parameter list, such as its word boundary. The maximum length of the string is 60 characters. If omitted, the default is BDY(DWORD).

Note: If the issuing program is reentrant, it must use the GETMAIN macro to allocate the storage area for the parameter list and the FREEMAIN macro to release the storage.

The SENDREQ CPRB

The SENDREQ macro fills in a CPRB as shown in Table 11 on page 59. The SENDREQ macro sends the CPRB to MVSSERV, which routes the CPRB to the requested server.

Table 11: CPRB Control Block for Sending a Request (SENDREQ)

Offset Dec(Hex)	Number of Bytes	Field Name	Contents
0(0)	1	CRBF1	The control block's version number (first four bits) and modification level number (last four bits).
1(1)	2		Reserved
3(3)	1	CRBF4	The type of request. X'01' indicates a service request. (X'03' indicates a define server (DEFSERV) request.)
4(4)	4	CRBCPRB	Control block identifier (‘CPRB’).
8(8)	8		Reserved
16(10)	8	CRBSNAME	The name of the requested server.
24(18)	2		Reserved
26(1A)	2	CRBFID	The ID number of the requested service function.
28(1C)	12		Reserved
40(28)	4	CRBRQDLN	The length of the request data.
44(2C)	4	CRBRQDAT	The address of the request data.
48(30)	4	CRBRPDLN	The length of the reply data (maximum length allowed by the invoker of SENDREQ).
52(34)	4	CRBRPDAT	The address of the buffer for reply data.
56(38)	4	CRBRQPLN	The length of the request parameters.
60(3C)	4	CRBRQPRM	The address of the request parameters.
64(40)	4	CRBRPPLN	The length of the reply parameters (maximum length allowed by the invoker of SENDREQ).
68(44)	4	CRBRPPRM	The address of the buffer for reply parameters.
72(48)	40		Reserved

For a list of return codes from the SENDREQ macro, see Chapter 8, “MVSSERV Return Codes,” on page 63.

CHSTRACE Macro

To issue messages to the terminal, the MVSSERV trace data set, or both, use the CHSTRACE macro. Servers, initialization/termination programs, and access method drivers can issue the CHSTRACE macro.

CHSTRACE Considerations

Messages from CHSTRACE must not exceed 80 characters in length. Any messages over 80 characters long are truncated after the 80th character.

Messages must begin with a message ID or a blank character. If an MVSSERV user has PROFILE NOMSGID specified, TSO/E removes the message ID or any other characters preceding the first blank character in the message.

CHSTRACE Syntax and Parameters

CHSTRACE Macro Syntax shows the syntax of the CHSTRACE macro. Optional parameters are shown in brackets. For an example of the CHSTRACE macro, see "Sample Server IBMABASE" on page 13.

```
EXECUTE FORM

 label CHSTRACE  [DEST=[TRACE|TERM|BOTH],]
                  CED=name or address,
                  BUFFER=name or address,
                  BUFLEN=name or address,
                 [RETCODE=variable,]
                  MF=(E,plist_address[,COMPLETE])
LIST FORM

       CHSTRACE   MF=(L,plist_address[attr])
```

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

Execute Form

[DEST=[TRACE | TERM | BOTH],]
Specify the destination of the message. TRACE sends the message to the MVSSERV trace data set. TERM sends the message to the terminal. BOTH sends the message to both the terminal and the trace data set. If you omit this parameter, messages go to the trace data set.

CED=*name* or *address*
Specify the name, or address in a register (2–12), of the CED that was passed as input to the invoking program. To map the CED, use the CHSCED macro.

BUFFER=*name* | *address*
Specify the name, or address in a register (2–12), of a message buffer that the macro is to issue.

BUFLEN=*address*
Specify the name, or address in a register (2–12), of a 4-byte field that contains the length in bytes of the message buffer to be issued. The maximum buffer length is 80 bytes. Messages that exceed 80 characters in length are truncated.

[RETCODE=*variable*]
Specify the name or address of a 4-byte output variable to receive the CHSTRACE return code from register 15. If you omit this parameter, you must obtain the return code from register 15.

MF=(E,*plist_address*[,COMPLETE])
specifies the execute form of the macro and the address of a storage area for the macro parameter list. The execute form generates code to put the parameters into a parameter list and invoke the desired server.

[,COMPLETE]
MVSSERV performs complete syntax checking, verifying that required CHSTRACE parameters are specified and supplying default values for omitted optional parameters.

List Form

MF(L,*plist_address*[,*attr*])
specifies the list form of the macro and the address of a storage area for the macro parameter list. The list form defines an area to contain the parameter list.

[,*attr*]
Specify an input string that contains any special attributes for the parameter list. The maximum length of the input string is 60 characters. If omitted, the default is BDY(DWORD).

Note: If the issuing program is reentrant, it must use the GETMAIN macro to allocate the storage area for the parameter list and the FREEMAIN macro to release the storage.

For a list of return codes from the CHSTRACE macro, see Chapter 8, “MVSSERV Return Codes,” on page 63.

Chapter 8. MVSSERV Return Codes

This chapter lists return codes from the DEFSERV, SENDREQ, and CHSTRACE macros.

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 Table 12 on page 63.

Table 12: Return Codes from the DEFSERV Macro

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" on page 63.
8(8)	The CPRB is not valid. Data fields in the CPRB, such as CPRBF4, contained information that was not valid.
12(C)	The CPRB is not valid. 24-bit addresses are 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 are not valid, causing MVSSERV to fail.

Return Codes from the DEFSERV CPRB

If the return code in register 15 is 4, the program must check for an additional return code in the DEFSERV CPRB, which MVSSERV returns after finishing with the DEFSERV macro. The additional return code, if any, is in field CRBCRTNC, as shown in Table 13 on page 63.

Table 13: Return Codes in the DEFSERV CPRB

Offset Dec(Hex)	Number of Bytes	Field Name	Contents or Meaning
0(0)	12		Reserved
12(C)	4	CRBCRTNC	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

Return Codes from the SENDREQ Macro

When a program resumes control after issuing the SENDREQ macro, the program must inspect register 15, or the variable defined in the RETCODE parameter, for a return code from MVSSERV. The possible return codes are shown in Table 14 on page 64.

Table 14: Return Codes from the SENDREQ Macro

Return Code Dec(Hex)	Meaning
0(0)	The request was successfully routed.
4(4)	The request was unsuccessfully routed. 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 SENDREQ CPRB" on page 64.
8(8)	The CPRB is not valid. Data fields in the CPRB, such as CPRBF4 (function ID), contained information that was not valid.
12(C)	The CPRB is not valid. 24-bit addresses are 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 are not valid, causing MVSSERV to fail.

Return Codes from the SENDREQ CPRB

If the return code in register 15 is 4, you must check for an additional return code in the CPRB, which MVSSERV returns after handling the SENDREQ macro. The additional return code, if any, is in field CRBCRTNC.

The return codes and their meanings are shown in Table 15 on page 64.

Table 15: Return Codes in the SENDREQ CPRB

Offset Dec(Hex)	Number of Bytes	Field Name	Contents
0(0)	12		Reserved
12(C)	4	CRBCRTNC	The return code from MVSSERV in the SENDREQ reply CPRB: **0000** Processing was successful. **0130** Request failed; the server was not found. **0131** Request failed; the server was unavailable. **0132** Request failed; the reply parameter length was not valid. **0133** Request failed; the reply data length was not valid. **0135** Request failed; the requested server failed. **0136** Request failed; MVSSERV error.
16(10)	96		Reserved

Return Codes from the CHSTRACE Macro

When a program resumes control after issuing the CHSTRACE macro, the program must inspect register 15, or the variable defined in the RETCODE parameter, for a return code from MVSSERV. The possible return codes are shown in Table 16 on page 65.

Table 16: Return Codes from the CHSTRACE Macro

Return Code Dec(Hex)	Meaning
0(0)	The message was successfully issued (to the terminal, the trace data set, or both).
4(4)	A failure occurred in message processing. Check the syntax of the CHSTRACE macro and the allocation of the trace data set.

Appendix A. Accessibility

Accessible publications for this product are offered through IBM Knowledge Center (www.ibm.com/support/knowledgecenter/SSLTBW/welcome).

If you experience difficulty with the accessibility of any z/OS information, send a detailed email message to mhvrcfs@us.ibm.com.

Accessibility features

Accessibility features help users who have physical disabilities such as restricted mobility or limited vision use software products successfully. The accessibility features in z/OS can help users do the following tasks:

- Run assistive technology such as screen readers and screen magnifier software.
- Operate specific or equivalent features by using the keyboard.
- Customize display attributes such as color, contrast, and font size.

Consult assistive technologies

Assistive technology products such as screen readers function with the user interfaces found in z/OS. Consult the product information for the specific assistive technology product that is used to access z/OS interfaces.

Keyboard navigation of the user interface

You can access z/OS user interfaces with TSO/E or ISPF. The following information describes how to use TSO/E and ISPF, including the use of keyboard shortcuts and function keys (PF keys). Each guide includes the default settings for the PF keys.

- *z/OS TSO/E Primer*
- *z/OS TSO/E User's Guide*
- *z/OS ISPF User's Guide Vol I*

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users who access IBM Knowledge Center with a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line because they are considered a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or `3.1` or `3.1.1`. To hear these numbers correctly, make sure that the screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number `3.1`) are mutually exclusive alternatives. If you hear the lines `3.1 USERID` and `3.1 SYSTEMID`, your syntax can include either `USERID` or `SYSTEMID`, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The `*` symbol is placed next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element `*FILE` with dotted decimal number 3 is given the format `3 \* FILE`. Format `3* FILE` indicates that syntax element `FILE` repeats. Format `3* \* FILE` indicates that syntax element `* FILE` repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol to provide information about the syntax elements. For example, the lines `5.1*, 5.1 LASTRUN, and 5.1 DELETE` mean that if you use more than one of the `LASTRUN` and `DELETE` syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the `%` symbol, it indicates a reference that is defined elsewhere. The string that follows the `%` symbol is the name of a syntax fragment rather than a literal. For example, the line `2.1 %OP1` means that you must refer to separate syntax fragment OP1.

The following symbols are used next to the dotted decimal numbers.

? indicates an optional syntax element
The question mark (?) symbol indicates an optional syntax element. A dotted decimal number followed by the question mark symbol (?) indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example `5? NOTIFY`). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines `5 ?, 5 NOTIFY`, and `5 UPDATE`, you know that the syntax elements `NOTIFY` and `UPDATE` are optional. That is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

! indicates a default syntax element
The exclamation mark (`!`) symbol indicates a default syntax element. A dotted decimal number followed by the `!` symbol and a syntax element indicate that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the dotted decimal number can specify the ! symbol. For example, if you hear the lines `2? FILE, 2.1! (KEEP), and 2.1 (DELETE)`, you know that `(KEEP)` is the default option for the `FILE` keyword. In the example, if you include the `FILE` keyword, but do not specify an option, the default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the `FILE` keyword is omitted, the default `FILE(KEEP)` is used. However, if you hear the lines `2? FILE, 2.1, 2.1.1! (KEEP)`, and `2.1.1 (DELETE)`, the default option KEEP applies only to the next higher dotted decimal number, `2.1` (which does not have an associated keyword), and does not apply to `2? FILE`. Nothing is used if the keyword `FILE` is omitted.

*** indicates an optional syntax element that is repeatable**
The asterisk or glyph (*) symbol indicates a syntax element that can be repeated zero or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line `5.1* data area`, you know that you can include one data area, more than one data area, or no data area. If you hear the lines `3* , 3 HOST, 3 STATE`, you know that you can include `HOST, STATE`, both together, or nothing.

Notes:

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you can write `HOST STATE`, but you cannot write `HOST HOST`.

3. The * symbol is equivalent to a loopback line in a railroad syntax diagram.

+ indicates a syntax element that must be included

The plus (+) symbol indicates a syntax element that must be included at least once. A dotted decimal number followed by the + symbol indicates that the syntax element must be included one or more times. That is, it must be included at least once and can be repeated. For example, if you hear the line `6.1+ data area`, you must include at least one data area. If you hear the lines `2+, 2 HOST, and 2 STATE`, you know that you must include `HOST, STATE,` or both. Similar to the * symbol, the + symbol can repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loopback line in a railroad syntax diagram.

Notices

This information was developed for products and services that are offered in the USA or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for the Knowledge Centers. Use of hyperlinks in other output formats of this information is at your own risk.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Site Counsel
2455 South Road

Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user’s name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM’s Privacy Policy at ibm.com/privacy and IBM’s Online Privacy Statement at ibm.com/privacy/details in the section entitled “Cookies, Web Beacons and Other Technologies,” and the “IBM Software Products and Software-as-a-Service Privacy Statement” at ibm.com/software/info/product-privacy.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

Programming Interface Information

This document describes intended Programming Interfaces that allow the customer to write programs to obtain the services of z/OS TSO/E.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and Trademark information (www.ibm.com/legal/copytrade.shtml).

Index

Special Characters

<TSO/E>
  summary of changes xiv

A

abend
  obtaining a dump 45
  suppressing a dump 46
ABEND
  recovery from
    initialization/termination program 30
    server 12
access method driver (AMD)
  considerations for writing 39
  installation 43
  interface 39
  overview 5
  sample 41
accessibility
  contact IBM 67
  features 67
allocating
  dump data set 45
  dump suppression data set 46
  input parameter data set 43
  trace data set 45
AMD 5
AMODE
  access method driver 40
  initialization/termination program 26
  server 7
ASCII-to-EBCDIC data conversion
  performing in a server 10
assembling
  an initialization/termination program 31
  server 12
assistive technologies 67

B

buffer
  for message
    using with the CHSTRACE macro 60
  request and reply
    passing ECF data and parameters in 10

C

CED (connectivity environment descriptor)
  mapping to fields of 54
  on entry to the initialization/termination program 27
  pointer to
    on entry to the server 8
CED parameter
CED parameter *(continued)*
  CHSTRACE macro 60
  DEFSERV macro 56
  SENDREQ macro 58
CHSABEND data set 46
CHSCED macro 54
CHSDCPRB macro 9, 53
CHSPARM data set 43
CHSTRACE data set 44
CHSTRACE macro
  return code from 65
command processor parameter list(CPPL) 27
command syntax for MVSSERV 49
compiling
  an initialization/termination program 31
  server 12
concepts of the IBM Enhanced Connectivity Facility 1
connectivity environment descriptor (CED) 8
connectivity programming request block (CPRB) 3
contact
  z/OS 67
control block
  CED 54
  CPRB
    DEFSERV reply 63
    DEFSERV request 28, 56
    on entry to the server 9
    on exit from the server 11
    SENDREQ reply 64
    with reply from another server 11
  INITTERM
    at initialization 27
    at termination 30, 31
  sending a request (SENDREQ) 59
converting data
  from ASCII to EBCDIC, in a server 10
CPPL (command processor parameter list)
  address of
    as input at initialization 27
    as input at termination 31
CPRB (connectivity programming request block)
  DEFSERV reply 63
  DEFSERV request 28, 56
  on entry to the server
    with reply from another server 11
  on exit from the server 11
  overview 3
  sending a request (SENDREQ) 59
  SENDREQ reply 64
  using a CPRB
    to define a server to MVSSERV 28
    to receive a service reply 11
    to receive a service request 7
    to send a service reply 10
    to send a service request 11
  using one CPRB
    to define multiple servers 54

- CPRB (connectivity programming request block) *(continued)*
  - using one CPRB *(continued)*
    - to send multiple requests 54
- CPRB parameter
  - DEFSERV macro 56
  - SENDREQ macro 58

D

- data
  - for service requests and replies,
    - using buffers to pass 10
- data set for MVSSERV
  - dump data set 45
  - dump suppression data set 46
  - input parameter data set 43
  - trace data set
    - allocating 44
    - reading 50
    - sample 50
- define server parameter area
  - creating (in DEFSERV macro) 56
- defining a server to MVSSERV 28
- defining multiple servers with one CPRB 54
- defining server parameter area
  - content 29
- DEFSERV macro
  - return code from 63
- deleting a server 31
- designing a server 7
- designing an initialization/termination program 25
- DEST parameter
  - of the CHSTRACE macro 60
- diagnosis for MVS servers 50
- dump data set 45
- dump suppression data set 46
- dump, obtaining 45

E

- EBCDIC-to-ASCII data conversion
  - performing in a server 10
- ECF (Enhanced Connectivity Facility)
  - concepts 1
  - ECF environment using MVSSERV 3
  - ending a TSO/E Enhanced Connectivity session 49
  - starting a TSO/E Enhanced Connectivity session 49
- ending MVSSERV 49
- Enhanced Connectivity Facility (ECF) 1
- error message 50
- error recovery
  - initialization/termination program 30
  - server 12
- ESTAE macro
  - server recovery 12
- execution path trace table 50
- external trace data
  - creating a data set for 44
  - retrieving and reading 50

F

- feedback xiii
- freeing resource 31
- function ID
  - obtaining from the receive request CPRB 9

H

- handling a service request
  - overview 7
- help for MVSSERV message 49

I

- ID, service function
  - obtaining from the receive request CPRB 9
- informational message 50
- initialization/termination program
  - design 25
  - function of
    - defining server 28
    - deleting a server 31
    - freeing resources 31
    - loading a server 28
    - obtaining resources 28
  - in relation to a server 2, 25
  - input to
    - at initialization 27
    - at termination 30, 31
  - installation 43
  - naming 26
  - naming in the input parameter data set 43
  - overview 2
  - processing overview 26
  - recovery routine 30
- initializing
  - dump suppression data set 46
  - input parameter data set 44
- INITTERM control block
  - at initialization 27
  - at termination 30, 31
- INITTERM macro 54
- input
  - initialization/termination program
    - at initialization 27
    - at termination 30, 31
  - server 3
- input parameter data set 43
- installing
  - access method driver 43
  - initialization/termination program 43
  - server 43
- INTRSN field of INITTERM control block
  - at termination only 31
- INTSNAME field of INITTERM control block
  - at termination only 30
- IOTRACE option of MVSSERV
  - command syntax 49
- IOTRACE parameter of MVSSERV
  - trace data produced by 45
- issuing a message
  - from a server 12
  - from an access method driver 41
  - from an initialization/termination program 30
- issuing MVSSERV 49

K

keyboard
- navigation 67
- PF keys 67
- shortcut keys 67

L

link-editing
- an initialization/termination program 31
- server 12

load module
- linking a server to a 12
- linking an initialization/termination program to a 31

loading a server
- considerations 25
- example 28

M

macro
- CHSCED 54
- CHSDCPRB 53
- CHSTRACE 59
- DEFSERV 55
- INITTERM 54
- SENDREQ 57

macro syntax and parameter 53

mapping macro
- CHSCED
  - for mapping the CED 54
- CHSDCPRB
  - for creating a CPRB 53
  - for mapping a CPRB 53
  - for mapping the CPRB 9
- INITTERM
  - for mapping to initialization/termination input 54

message help (online) 49

message manual, using 51

message, issuing
- from a server 12
- from an access method driver 41
- from an initialization/termination program 30

MVSSERV command
- description 2
- diagnosis 50
- issuing 49
- message 50
- online message help 49
- sequence of events in an MVSSERV session 4
- syntax 49
- task structure 25
- termination 49

N

navigation
- keyboard 67

NOTRACE option of MVSSERV
- command syntax 49

O

obtaining resources for a server 28

online message help 49

operand of the MVSSERV command 49

overview
- of initialization/termination program processing 26
- of service request handling 7

P

parameter
- for service requests and replies,
  - using buffers to pass 10

parmlist, server
- content 28
- on entry to the server 8
- pointer from initialization/termination program 29, 56

preparing for execution
- initialization/termination program 31
- servers 12

procedure
- designing a server 7
- designing an initialization/termination program 25
- writing a server 4
- writing an initialization/termination program 5

R

receiving a service request 8

recovery routine
- for the initialization/termination program 30
- for the server 12

reentrant program
- installing in system library 43

register content
- at entry to server 8
- at exit from server 10
- at initialization 27
- at termination 30, 31
- required for DEFSERV macro 55
- required for SENDREQ macro 57

reply data 10

reply parameter 10

reply, service
- detecting reply failure 30
- overview 1
- sending from the server 10

request data 10

request parameter 10

requester
- books about 7
- overview of 1
- planning considerations 7

resource, for a server
- freeing 31
- obtaining 28

return address
- after initialization 27
- after termination 30, 31
- for a server 10

return code
- initialization/termination program 31

return code *(continued)*
- MVSSERV
  - CHSTRACE macro 65
  - DEFSERV CPRB 63
  - DEFSERV macro 63
  - last service reply 31
  - SENDREQ CPRB 64
  - SENDREQ macro 64
- server-requester pair 10

RMODE
- access method driver 40
- initialization/termination program 26
- server 7

S

sample
- access method driver 41
- initialization/termination program 32
- trace data set 50

sending a service reply 10
sending multiple requests with one CPRB 54
sending to IBM
- reader comments xiii

SENDREQ macro
- register content required for 57
- return code from 64

sequence of events in an MVSSERV session 4
server
- assembling 12, 31
- compiling 12, 31
- debugging 49
- deleting 31
- design 7
- diagnosis 50
- executing 49
- initialization 26
- input from MVSSERV 8
- input from requester 9
- installation 43
- loading 28
- naming 7
- overview 1
- recovery routine 12
- termination 30
- testing 49
- writing a 7

server parmlist
- content 28
- on entry to the server 8
- pointer from initialization/termination program 29, 56

server-requester programming interface (SRPI) 3
service function
- ID, obtaining from the receive request CPRB 9
- in relation to a server 2, 7
- overview 2
- packaging 7

service reply
- detecting reply failure 30
- overview 1
- sending from the server 10

service request
- overview 1
- server handling of 7

shortcut keys 67
SRPI (server-requester programming interface)
- overview 3
- summary of SRPI functions 3

starting MVSSERV 49
STEPLIB
- installing a program in a 43

steps
- designing a server 7
- designing an initialization/termination program 25
- writing a server 4
- writing an initialization/termination program 5

summary of changes
- <TSO/E> xiv

Summary of changes xiv
suppressing an MVSSERV dump 46
syntax of MVSSERV macro 53
SYSABEND dump, data set for 45
SYSMDUMP dump, data set for 45
system library
- installing a program in a 43

SYSUDUMP dump, data set for 45

T

task structure
- for MVSSERV 25

testing a server 49
trace data set
- allocating 44
- issuing a message to
  - from a server 12
  - from an access method driver 41
  - from an initialization/termination program 30
- reading 50

TRACE option of MVSSERV
- command syntax 49

TRACE parameter of MVSSERV
- trace data produced by 45

tracing
- MVSSERV message 50
- MVSSERV's execution path 50

trademarks 74
translating data
- from ASCII to EBCDIC, in a server 10

U

user interface
- ISPF 67
- TSO/E 67

W

writing
- a server 7
- an access method driver 39
- an initialization/termination program 25

SA32-0980-30