IBM Programmer's Guide to the Server-Requester Programming Interface for the IBM Personal Computer and the IBM 3270 PC

First Edition (September 1986)

This edition applies to Release 1.0 of IBM System/370 to IBM Personal Computer Enhanced Connectivity Facilities and to all subsequent releases and modifications until otherwise indicated in new editions or Technical Newsletters. Changes are made periodically to the information herein.

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

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

Comments may be addressed to IBM Corporation, Department 95H, 11400 Burnet Road, Austin, Texas 78758. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

About This Book

The purpose of this book is to explain the concepts and procedures for writing requesters. A requester is a program that requests a server to perform a task, using the Server-Requester Programming Interface (SRPI). See "Server-Requester Programming Interface" on page 1-5 for details of the SRPI.

This book shows how to write requesters in the following languages for the IBM Personal Computer:

- IBM Pascal
- IBM C
- IBM Macro Assembler.

This book also explains:

- Requesters
- Servers
- Routers
- Server-Requester Programming Interface (SRPI)
- The send_request function.

Abbreviations

This book uses the following abbreviations:

- **Pascal** refers to IBM Pascal Compiler, Version 2.00
- **C** refers to IBM C Compiler, Version 1.00
- **Macro Assembler** refers to IBM Macro Assembler Version 1.00 or 2.00
- **DOS** refers to Release 3.10 or 3.20 of IBM PC Disk Operating System (DOS)
- **MVS** refers to the IBM System/370 running Multiple Virtual Storage/Extended Architecture (MVS/XA) with Time Sharing Option (TSO)
- **Personal Computer** and **PC** refer to one of the following IBM Personal Computers:
  - PC
  - PC/XT
  - PC/AT

- Portable PC
- 3270 PC
- 3270 PC/AT.

- The term **IBM host computer** refers to the IBM mainframe computers (30xx series) and the IBM intermediate computers (43xx series) that support the MVS/System Product (MVS/XA) and the VM/System Product.

- **VM** refers to the IBM System/370 running Virtual Machine/System Product (VM/SP) Release 4, with Conversational Monitor System (CMS).

Audience

This book is intended primarily for:

- Application programmers

- Application/system designers.

It is intended secondarily for:

- System programmers

- IBM technical support personnel.

You should be familiar with one or more of the following programming languages:

- IBM Personal Computer Pascal Language

- IBM Personal Computer C Language

- IBM Personal Computer Macro Assembler Language.

How to Use This Book

Chapter 1 provides an overview of the Server-Requester Programming Interface and explains the send_request function and semantics. Chapters 2 through 4 describe the language interface and syntax for Pascal, C, and Macro Assembler.

Organization

This manual contains the following chapters:

Chapter 1, "The SRPI and The Send_Request Function," defines the Server-Requester Programming Interface (SRPI), routers, requesters, and servers. It explains how the send_request concept functions in the SRPI and lists the send_request parameters supplied by the requester. This chapter describes the parameters returned from the server in a send_reply

operation. This chapter also describes the format of the Connectivity Programming Request Block (CPRB).

Chapter 2, "Language Interface and Syntax for Pascal," discusses SRPI record definitions, request record initialization, the Pascal sendrequest function, and linking subroutines. This chapter is for programmers who are writing a requester program in the Pascal language. This chapter also provides a Pascal sample program.

Chapter 3, "Language Interface and Syntax for C," discusses the SRPI structure definition, request record initialization, the C send_request function, and linking subroutines. This chapter provides language-specific notes for C. This chapter is for programmers who are writing a requester program in the C language. This chapter also provides a C language sample program.

Chapter 4, "Language Interface and Syntax for Macro Assembler," discusses macro definitions, macro parameters, and CPRB mapping. This chapter is for programmers who are writing a requester program in the Macro Assembler language. This chapter also provides a Macro Assembler sample program.

Appendix A, "SRPI Return Codes," describes SRPI return codes for successful and unsuccessful tasks.

Appendix B, "ASCII to EBCDIC Translation Table," provides a table that the SRPI uses for translating the server name from ASCII to EBCDIC.

Appendix C, "Product Requirements," describes the product requirements for the IBM Personal Computer, IBM Requesters/Servers, MVS/XA environment, and VM environment.

The glossary defines key terms used in the book.

Prerequisite Publication

Introduction to IBM System/370 to IBM Personal Computer Enhanced Connectivity Facilities, GC23-0957

This book provides a high-level overview of the services available through IBM Enhanced Connectivity Facilities.

Related Publications

- *TSO Extensions Programmer's Guide to the Server-Requester Programming Interface for MVS/Extended Architecture*, SC28-1309

  This book explains how to write, install, test, and debug servers to use with MVSSERV. It is intended for application designers and programmers who design and write servers and server initialization/termination programs and system programmers who install MVS/XA servers.

- *IBM Programmer's Guide to the Server-Requester Programming Interface for VM/System Product*, SC24-5291

  This book explains how to write and install servers in a VM/SP system. Explanations also cover the use of the router and the messages/MNOTES which it issues.

- *IBM PC 3270 Emulation Program, Version 3.0, User's Guide*

  This book explains how to install, load, and use this emulation program to communicate with an IBM System/370.

- *IBM PC 3270 Emulation Program, Version 3.0, System Planner's and User's Reference*

  This book describes network planning, problem determination procedures, keyboard remapping and keyboard extensions.

- *IBM 3270 PC Control Program, Version 3.0, User's Guide*, 58X9968

  This book explains how to install, load, and use this program to communicate with an IBM System/370.

- *IBM TSO/E Servers and CMS Servers Installation and Programmer's Guide*, SH20-9677

  This book is a reference manual for the system programmer who installs software and for the application programmer who writes special user conversion routines (user exits) on VM or TSO/E.

- *IBM PC Requesters User's Guide*, 6316993

  This book is for the personal computer user, included with the program, and cannot be ordered separately. It describes how to install and use the IBM PC Requesters product.

Compatibility

The supported languages are Pascal, C, and Macro Assembler.

Product requirements are one or more of the following:

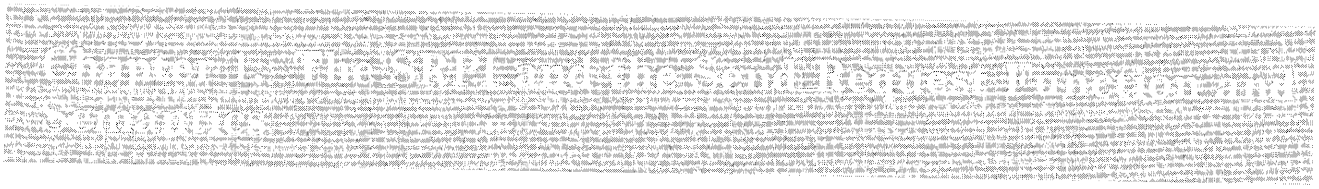
IBM Personal Computer

- Pascal Compiler, Version 2.00
- C Compiler, Version 1.00
- Macro Assembler, Version 1.00
- Macro Assembler, Version 2.00.

See Appendix C, "Product Requirements" on page C-1 for additional information about product requirements.

Contents

Figures



CONTENTS

About This Chapter

This chapter summarizes IBM System/370 to IBM Personal Computer Enhanced Connectivity Facilities:

- Server-Requester Programming Interface (SRPI)
- Routers
- Requesters and servers.

This chapter also explains the send_request concept and its function in the SRPI and:

- Shows how to use the Server-Requester Programming Interface (SRPI)
- Lists the supplied and returned parameters of the send_request function
- Describes the format of the Connectivity Programming Request Block (CPRB)
- Provides a sample SRPI program flow.

Summary of IBM System/370 to IBM Personal Computer Enhanced Connectivity Facilities

IBM System/370 to IBM Personal Computer Enhanced Connectivity Facilities provides a method for communicating and moving functions between unlike systems. IBM Enhanced Connectivity Facilities are a set of programs for interconnecting IBM Personal Computers and IBM System/370 host computers operating with the MVS/XA or VM/SP environment.

IBM Enhanced Connectivity Facilities is patterned after the *call/return* function available in many high-level programming languages. Customers can either write their own *Requesters/Servers* or use those available from IBM. See Appendix C, "Product Requirements" on page C-1 for information about the IBM Requesters/Servers.

IBM Enhanced Connectivity Facilities provide a common structure for sending and receiving functions on a connection between an IBM host computer[1] and IBM Personal Computers[2].

IBM Enhanced Connectivity Facilities is designed to shield end users and application programs from the differences between two connected systems, including details of the operating systems, the location of the systems, and the communication protocols.

IBM Enhanced Connectivity Facilities help simplify the way unlike systems use services over a connection. IBM Enhanced Connectivity Facilities provide a single interface that allows application programmers to write personal computer and host applications that run on a variety of communication connections. This interface is used for the exchange of such functions as reading, transferring, or printing.

IBM Enhanced Connectivity Facilities has the following characteristics:

- A consistent interface for application programs in a personal computer to request services, data, or both from a host. The requesting program is referred to as the *requester*.
- A consistent interface for programs in a host to reply to requests for services, data, or both from personal computers. The program that services the request is referred to as the *server*.
- A consistent interface for handling communications between requesters and servers. The program, provided in personal computers and host

---

1 The term **IBM host computer** refers to the IBM mainframe computers (30xx series) and the IBM intermediate computers (43xx series) that support the MVS/System Product (MVS/XA) and the VM/System Product.

2 In this publication, the term **Personal Computer** refers to the properly-configured members of the IBM Personal Computer family, including the PC, the PC/XT, the Personal Computer AT, the Portable Personal Computer, the IBM 3270 Personal Computer, and the 3270 Personal Computer AT.

computers, is referred to as the *router*. The router provides a new *Server-Requester Programming Interface (SRPI)*. The SRPI is a request interface for requesters, or a reply interface for servers. This interface isolates requesters and servers from the underlying communication environment. See "Server-Requester Programming Interface" for details.

The requester and server programs operate in pairs, with the requester on the personal computer and the server on the host computer.

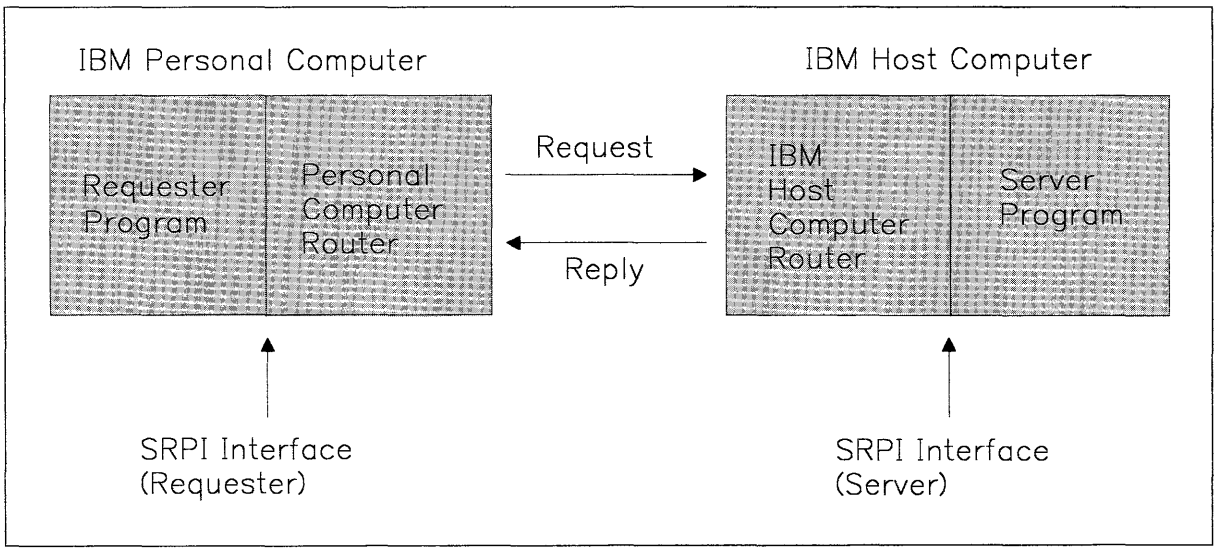


Figure 1-1. Example of a Requester and Server

Server-Requester Programming Interface

The application programming interface between requesters from the IBM Personal Computer and servers on the IBM host computer is the Server-Requester Programming Interface (SRPI).

Note: For information about a corresponding interface for servers on the IBM host computer, see one of the following:

- *TSO Extensions Programmer's Guide to the Server-Requester Programming Interface for MVS/Extended Architecture*
- *IBM Programmer's Guide to the Server-Requester Programming Interface for VM/System Product.*

The SRPI for the IBM Personal Computer is part of the IBM PC 3270 Emulation Program, Version 3.0, and part of the IBM 3270 PC Control Program, Release 3.0, for the 3270 PC. The SRPI on the IBM Personal Computer supports only requesters. It provides a call/return function for application-to-application communications. Using the *send_request* function, a program on an IBM Personal Computer calls (requests) for service from a partner program on an IBM host computer, which returns (services) the results.

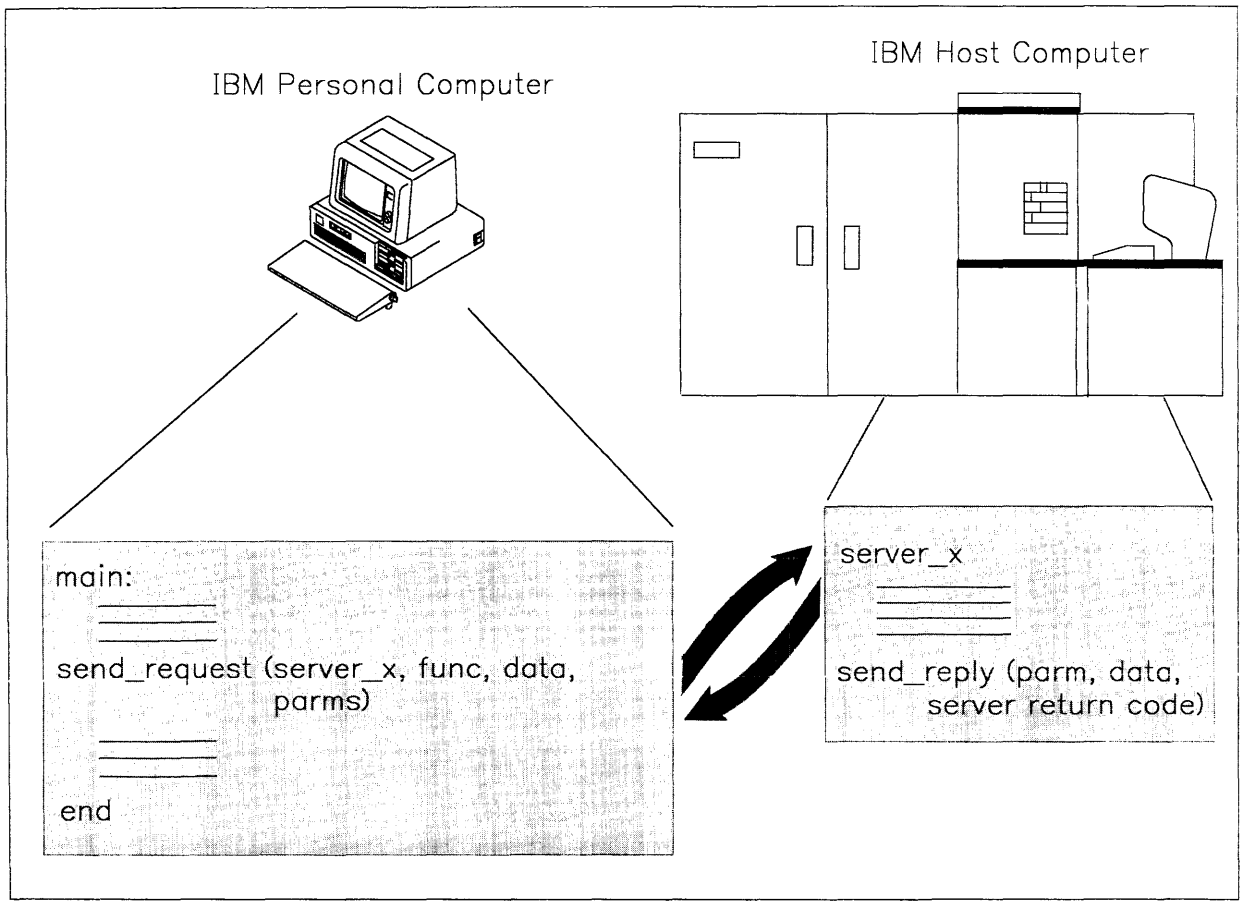


Figure 1-2. IBM Personal Computer Requester and IBM Host Computer Server Relationship

Applications use the SRPI by issuing the send_request function. See "The Send_Request Function" on page 1-8 for further information on this operation.

When an IBM Personal Computer requester issues the send_request function using the SRPI:

1. The PC router converts the request into a structure that the IBM host computer router recognizes.
2. The PC router passes the request to the IBM host computer router.
3. The IBM host computer router passes the request to the appropriate IBM host computer server.
4. The IBM host computer server processes the request and passes a reply to the IBM host computer router.
5. The IBM host computer router then passes the reply to the PC router.
6. The PC router returns the reply to the originating requester application.

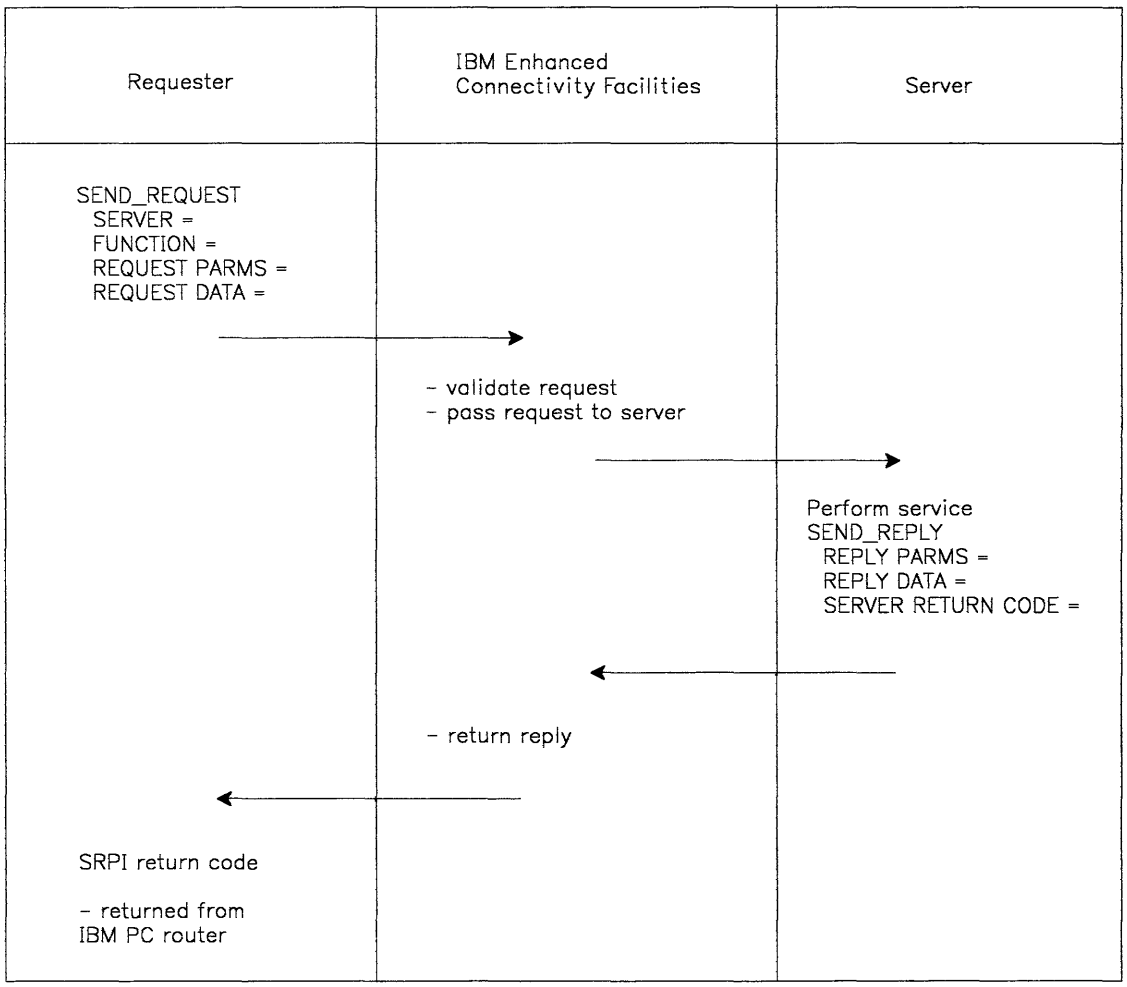


Figure 1-3. Example of a Requester and Server Flow

The Send_Request Function

You may issue the send_request function to the SRPI interface in the following ways:

- The C language support programs provided by IBM.
- The Pascal language support programs provided by IBM.
- The Macro Assembler support programs provided by IBM.
- Directly accessing the SRPI interface without the use of any of the support programs listed above.

The language support programs provided by IBM generate the send_request function.

You must perform the following steps, if you are not using the language support programs provided by IBM, to generate the send_request function:

1. Initialize the appropriate Connectivity Programming Request Block (CPRB) fields. This includes the CPRB length field, the CPRB version ID and the CPRB verb type. Any unused fields should be set to the appropriate default value. The following five CPRB fields should be initialized to zero if unused:

   - Function ID
   - Request Parameters Length
   - Request Data Length
   - Reply Parameters Buffer Length
   - Reply Data Buffer Length.

   See "Connectivity Programming Request Block" on page 1-12 for details of the CPRB.

2. Load register pair ES:DI with the address of the CPRB.

On Entry	Register Contents
ES:DI	Address of the Connectivity Programming Request Block (CPRB)

Figure 1-4. CPRB Register Address

3. Set register AX to X'0103'.
4. Invoke software interrupt X'7F'.

5. Examine register AX after the interrupt. If register AX is set to zero, the SRPI has processed the request and the SRPI return code may be examined. If register AX is non-zero, SRPI is not loaded and has not processed the request.

   Note: The SRPI return code X'01000404' (PC router is not loaded) is returned in the CPRB only when the language support programs provided by IBM are utilized.

Send_Request Parameters

The PC router sends the request to the IBM host computer router using the necessary communication facility. The SRPI returns control to the requester with an appropriate return code, optional parameters, and data.

The parameters and data associated with the send_request function are described on the following pages.

Supplied Parameters

Name of Parameter	Required/ Optional	Default Value	Description
Server Name	Required	Blanks	The name of the IBM host computer server must be 8 bytes long (PC/ASCII), left justified, and padded with blanks (X'20'); leading spaces, embedded spaces, and names consisting of all spaces are invalid. The name is converted to EBCDIC before the request is sent to the IBM host computer. See Appendix B, "ASCII to EBCDIC Translation Table" on page B-1.
Function ID	Optional	0	A 2-byte binary number that specifies the server function being requested. Values of 0 to 65,535 are valid for specification by a requester.
Request Parameters Length	Optional	0	A 2-byte unsigned binary number that specifies the byte length of the request parameters to be passed to the server. Values of 0 to 32,763 are valid. A value of 0 indicates that there are no request parameters to be passed.
Request Parameters	Optional	0	The 4-byte address of the parameters, if any, to be passed to the server. A non-zero value in the request parameters length indicates that there are parameters to be passed. See Note 3 on page 1-11.
Request Data Length	Optional	0	A 2-byte unsigned binary number that specifies the byte length of the request data to be passed to the server. Values of 0 to 65,535 are valid. A value of 0 indicates that there is no request data to be passed.

Figure 1-5 (Part 1 of 2). Parameters Supplied by the Requester

Name of Parameter	Required/ Optional	Default Value	Description
Request Data	Optional	0	The 4-byte address of the data, if any to be passed to the server. A non-zero value in the request data length indicates that there is data to be passed. See Note 3 on page 1-11.
Reply Parameters Buffer Length	Optional	0	A 2-byte unsigned binary number that specifies the length in bytes of the reply parameter buffer supplied by the requester. Values of 0 to 32,763 are valid. A value of 0 indicates that no reply parameters are expected.
Reply Parameters Buffer	Optional	0	The 4-byte address of the reply parameter buffer. Its presence is indicated by a non-zero reply parameters buffer length. See Note 3 on page 1-11.
Reply Data Buffer Length	Optional	0	A 2-byte unsigned binary number that specifies the length in bytes of the reply data buffer supplied by the requester. Values of 0 to 65,535 are valid. A value of 0 indicates that no reply data will be received.
Reply Data Buffer	Optional	0	The 4-byte address of the reply data buffer. A non-zero value in the reply data buffer length indicates that there is reply data to be received. See Note 3 on page 1-11.

Figure 1-5 (Part 2 of 2). Parameters Supplied by the Requester

Notes:

1. *The default values for each language interface are set during the request record initialization function.*

2. *In the C language interface, the INIT_SEND_REQ_PARMS function initializes the server name pointer to zero. The SEND_REQUEST function checks the server name pointer for the zero value. If the server name pointer is set to zero, then the CPRB server name is set to blanks (X'20'). The server name pointer remains set to zero.*

Returned Parameters

Name of Parameter	Description
SRPI Return Code	A 4-byte value that specifies the results of the send_request execution. See Appendix A, "SRPI Return Codes" on page A-1 for a complete description of SRPI return codes.
Server Return Code	A 4-byte value returned by the server. The content and meaning of the return status are defined by the Requester/Server, but the length of the field is always 32 bits.
Replied Parameter Length	A 2-byte unsigned binary length that specifies the number in bytes of the parameters returned by the server. Values of 0 to 32,763 are valid. A value of 0 indicates that no reply parameters were received from the server.
Replied Data Length	A 2-byte unsigned binary length that specifies the number of bytes of the data returned by the server. Values of 0 to 65,535 are valid. A value of 0 indicates that no reply data was received from the server.

Figure 1-6. Parameters Returned to the Requester

Notes:

1. *The PC router is not re-entrant. If the PC router is re-entered with a request while it is processing a request, the second request is rejected with a return code of X'0100 0408' (PC router busy).*

2. *The server name is used by the remote IBM host computer to route the request to the server.*

3. *The address supplied is made up of a segment address and an offset into the segment. The PC router does not validate this field. The segment address and offset must give full addressability of the buffer; that is, the sum of the offset and the buffer length does not exceed 64K - 1 (65,535).*

4. *The Requesters/Servers determine the contents and meaning of the buffers defined by the CPRB.*

Connectivity Programming Request Block

The Connectivity Programming Request Block (CPRB) is used to pass a request to a server through the PC router. Requester applications written in C and Pascal do not require knowledge of the CPRB. The format of the CPRB is shown on the following pages.

Field	Byte Offset	Byte Length	Contents
CPRB length	0	2	The length in bytes of the CPRB.
PC router version ID	2	2	PC router version number.
SRPI return code	4	4	SRPI return code.
SRPI verb type	8	1	Type of SRPI request.
Reserved	9	1	Reserved
Function ID	10	2	Function ID; defined default of 0.
Reserved	12	2	Reserved
Request parameter length	14	2	Request parameter length; defined default of 0.
Request parameter	16	4	Request parameter pointer.
Request data length	20	2	Request data length; defined default of 0.
Request data	22	4	Request data pointer.
Reply parameter buffer length	26	2	Reply parameter buffer length; defined default value of 0.
Reply parameter buffer	28	4	Reply parameter buffer pointer.
Reply data buffer length	32	2	Reply data buffer length; defined default value of 0.
Reply data buffer	34	4	Reply data buffer pointer.
Reserved	38	2	Reserved
Server return code	40	4	Server return code.
Replied parameter length	44	2	Replied parameter length.
Replied data length	46	2	Replied data length.
Work area	48	46	The SRPI reserves this area; requesters should not use it.
Server name length	94	2	Number of bytes reserved for the server name.
Server name	96	8	Server name value supplied by the requester; the name is assumed to be left justified and padded with blanks.

Figure 1-7. CPRB Format

Notes:

1. *The PC router version ID is used to verify that the provided CPRB format can be processed. If the version ID is not valid, an error code is returned in the CPRB.*

2. *The following fields should be initialized to the values indicated:*

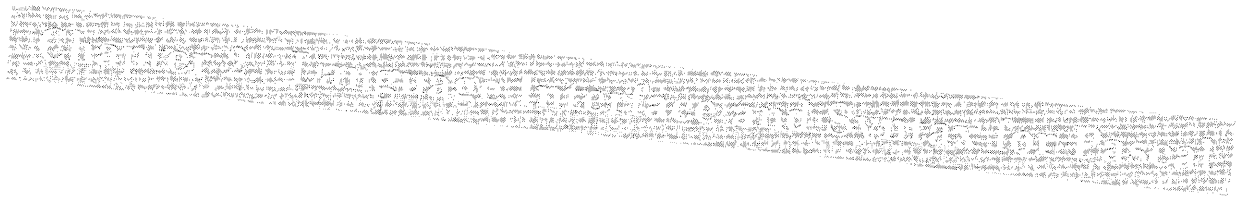
   - *CPRB length = Length of the CPRB (X'68')*
   - *PC router version id = Version number of the router (X'0100')*
   - *SRPI verb type = X'01'*
   - *Server name length = X'0008'*

   The IBM language support programs provided for Pascal, C, and Macro Assembler automatically initialize these fields.

3. *The IBM Personal Computer stores word (2-byte) values in memory in a byte-reversed format. For example, X'0102' is stored in memory as X'0201'. Doubleword (4-byte) values are stored in memory in a word-reversed and byte-reversed format. For example, X'0102 0304' is stored in memory as X'0403 0201'. The PC router does not alter this format when these values are sent to the IBM host computer as request data or request parameters. When a word value is sent to the IBM host computer, the low order byte is sent first, followed by the high order byte. The IBM host computer does not use the byte-reversed format. You must ensure that data and parameters passed between the requester and the server are in the proper format for the Requester|Server.*

4. *PC router pointers are stored using the doubleword format. See Note 3 on page 1-13. The first word in memory contains the offset value for the field. The second word in memory contains the segment value for the field. For example, a pointer with a segment value of X'1E00' and an offset value of X'0100' is stored in memory as X'0001 001E'.*

5. *The return code values are defined as doublewords by the provided IBM language interface. For example, the SRPI return code X'0100 0402 is stored in the CPRB memory as X'0204 0001'. See Note 3 on page 1-13.*



CONTENTS

About This Chapter

This chapter is for programmers who want to become familiar with writing a requester in Pascal.

This chapter describes:

- Pascal sendrequest function
- SRPI record definitions
- SRPI return codes
- Request record initialization
- Linking subroutines
- A Pascal sample program.

Note: The function called *send_request* in other chapters is spelled as one word (*sendrequest*) in this chapter.

Pascal SendRequest Function

The sendrequest parameters are grouped in a single Pascal record structure of type UERCPRB. The INIT_SEND_REQ_PARMS procedure initializes all the default sendrequest parameters. This allows the default values to be set once for parameters not used by a requester. The sendrequest function has a single parameter which is the 32-bit address (ADS) of a UERCPRB record.

The mapping is not the same for the UERCPRB record and the CPRB. Application programs should make no assumptions about the mapping of the UERCPRB record to the CPRB.

The SRPI provides for sending a buffer of parameters and/or a buffer of data to the server and receiving a buffer of parameters and/or a buffer of data from the server. A generic type is used for these parameters of the sendrequest function because any type of data can be sent using this interface. For Pascal, the type ADSMEM is used for these buffer pointers. This is the type predeclared to be ADS OF ARRAY [0..32765] OF BYTE, so it can point to data of whatever type is convenient. It uses the ADS operator to get the segment and offset address of the data object. Array indexing accesses specific offsets from the pointer. If the request parameters and/or data consist of more than a single structure, such as several records, the application must convert the data and/or parameters into a single *flat* structure that can be used as a buffer. A single flat structure is a contiguous sequence of bytes. You can use explicit field offset extension that allows you to assign an exact byte offset to fields within a record. This ensures that the fields within a record have consistent offsets.

The requesting program is responsible for packaging the request parameters and data in a format that can be recognized by the server.

The same memory area can be used for both request and reply parameters. In addition, the same memory area can be used for both request and reply data. The application program must ensure that reply data and parameters are written into the request data and parameters buffers only when the over-written data is no longer needed.

The object code for the Pascal procedures, the declaration files for the procedures, the record type, and the SRPI return codes are provided on diskette.

The Pascal object code linked with the requester program can push up to 12 words onto the application program stack. The PC router uses an additional 5 words of application program stack. Seventeen words of application program stack are required. Ensure that your application program stack is large enough to meet this requirement.

SRPI Record Definitions

The UERCPRB record type defines a record being passed to the PC router using the sendrequest function. The UERCPRB record type is defined in an application program by using the $INCLUDE metacommand to include the UUPCPRB.INC file. See "Supplied Parameters" on page 1-9 and "Returned Parameters" on page 1-11 for the definition of the supplied and returned parameters. The following is the SRPI record definition:

```
Type UERCPRBPTR = ADS of uercprb;

   uercprb = RECORD

  {Supplied Parameters                Parameter Description}

   uerserver      : string (8);    {ASCII name of server}
   uerfunct       : word;          {Function ID}
   uerqparml      : word;          {Request Parameters Length}
   uerqparmad     : adsmem;        {Request Parameters Address}
   uerqdatal      : word;          {Request Data Length}
   uerqdataad     : adsmem;        {Request Data Address}
   uerrparml      : word;          {Reply Parameters Buffer Length}
   uerrparmad     : adsmem;        {Reply Parameters Buffer Address}
   uerrdatal      : word;          {Reply Data Buffer Length}
   uerrdataad     : adsmem;        {Reply Data Buffer Address}

  {Returned Parameters}

   uerretcode     : integer4;      {SRPI Return Code}
   uerservrc      : integer4;      {Server Return Code}
   uerrepldplen   : word;          {Replied Parameters Length }
   uerreplddlen   : word;          {Replied Data Length }

END;
```

Notes:

1. *The name in the server name field must be left justified and padded with blanks (X'20') to a length of 8 bytes.*

2. *The supplied parameters are not changed by the sendrequest function.*

3. *The following output fields are undefined unless the SRPI return code value returned in uerretcode by the PC router is successful:*

   - *Server Return Code (uerservrc)*
   - *Replied Parameter Length (uerrepldplen)*
   - *Replied Data Length (uerreplddlen).*

   These fields may or may not have been altered by the PC router, and they may or may not have been initialized to zero by the PC router. The calling application should not expect these fields to be either maintained or altered across any unsuccessful call to the PC router.

4. *The value returned from the sendrequest function is identical to the value in the field uerretcode in the UERCPRB record.*

SendRequest Function Definition

The sendrequest function is defined in an application program by using the $INCLUDE metacommand to include the UUPPROCS.INC file. The sendrequest function declaration heading follows:

```
FUNCTION SendRequest
      (vars cprbptr  :  UERCPRBPTR)  : integer4; extern;
```

where *integer4* is a 4-byte field.

SRPI Return Codes

To incorporate SRPI return code definitions in an application program, use the $INCLUDE metacommand to include the UUPCPRB.INC file. The return code constants, their hexadecimal values, and their meanings are as follows:

```
CONST
  UERERROK       = #00000000;   { Successful                              }

{ Type 1 Errors }

  UERERRT1START = #01000402;   { Not started                             }
  UERRT1LOAD    = #01000404;   { Not loaded                              }
  UERERRT1BUSY  = #01000408;   { Busy                                    }
  UERERRT1VER   = #0100040A;   { Unsupported version ID                  }
  UERERRT1EMU   = #0100040C;   { PC 3270 Emulation, 3.0 not loaded       }
  UERERRT1QPLEN = #01000602;   { Request parameters length too large     }
  UERERRT1RPLEN = #01000604;   { Reply parameters length too large       }
  UERERRT1VERB  = #01000606;   { Invalid verb type                       }
  UERERRT1SERV  = #01000608;   { Invalid server name                     }
  UERERRT1QPAD  = #0100060C;   { Invalid request parameters address      }
  UERERRT1QDAD  = #0100060E;   { Invalid request data address            }
  UERERRT1RPAD  = #01000610;   { Invalid reply parameters address       }
  UERERRT1RDAD  = #01000612;   { Invalid reply data address             }
  UERERRT1TOPV  = #01000616;   { TOPVIEW not supported                   }
  UERERRT1CNCL  = #01000802;   { Cancelled by the IBM host computer      }
  UERERRT1CONV  = #01000C00;   { Unable to maintain conversation         }
  UERERRT1ISE   = #01000C02;   { Internal software error                 }
  UERERRT1PROT  = #01000C04;   { Protocol violation                      }
  UERERRT1SYTN  = #01000C06;   { System inconsistency                    }

{ Type 2 and Type 3 Errors }

  UERERRT2      = #02;         { Acknowledge sent (Type 2 Error)         }
  UERERRT3      = #03;         { Acknowledge received (Type 3 Error)     }
```

See Appendix A, "SRPI Return Codes" on page A-1 for a complete description of SRPI return codes.

Request Record Initialization

The INIT_SEND_REQ_PARMS function sets all sendrequest parameters in the UERCPRB record that have a default value. An application program that does not use all the sendrequest parameters may initialize them once.

The INIT_SEND_REQ_PARMS function sets default values in the UERCPRB record for the following sendrequest parameters:

- Request Parameters (pointer and length)
- Request Data (pointer and length)
- Reply Parameters Buffer (pointer and length)
- Reply Data Buffer (pointer and length)
- Function ID
- Server Name (set to blanks).

The request record initialization function is defined in an application program by using the $INCLUDE metacommand to include the UUPPROCS.INC file. The procedure declaration heading for the INIT_SEND_REQ_ PARMS procedure is:

```
PROCEDURE init_send_req_parms(
                vars cprbptr  :  UERCPRBPTR); extern;
```

Linking Subroutines

The INIT_SEND_REQ_PARMS routine initializes the Pascal UERCPRB record. The object module for the INIT_SEND_REQ_PARMS routine is UUPINIT.OBJ. The sendrequest routine calls the PC router. The object module for the sendrequest routine is UUPSENDR.OBJ.

Each object module should be included in the list of object modules passed to the LINK program.

Writing a Requester

The following Pascal sample program invokes a server using the Pascal interface routines. The program requests records from a customer records data set on the IBM host computer. The IBM host computer sends the customer records to the requester program for processing.

The requester examines the customer's balance returned from the server. If the customer's balance is positive, the customer's balance is sent to the server. The server puts the positive balance into an accounts receivable data set on the IBM host computer.

Warning: This program is provided solely as an example of how the Pascal interface routines can be used to invoke a server. It is not intended to produce meaningful output for your use or to provide a complete programming example that accounts for all possible error situations. It is not a programming tutorial.

The following books contain sample server programs:

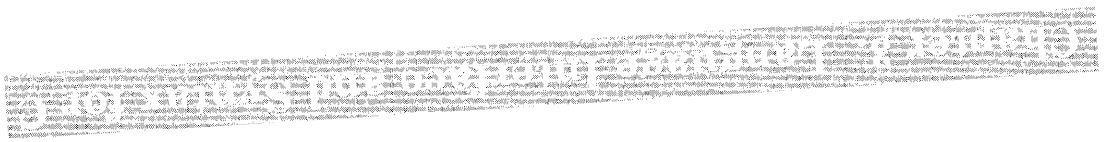
- *TSO Extensions Programmer's Guide to the Server-Requester Programming Interface for MVS/Extended Architecture*
- *IBM Programmer's Guide to the Server-Requester Programming Interface for VM/System Product.*

Pascal Sample Program

```
(******************** PROLOGUE ***************************************
*                                                                     *
*  MODULE NAME = PSAMPL.PAS                                           *
*                                                                     *
* DESCRIPTIVE NAME = Pascal Sample Program                            *
*                                                                     *
* COPYRIGHT = (C) COPYRIGHT IBM CORP. 1984, 1987                      *
*             LICENSED MATERIAL - PROGRAM PROPERTY OF IBM             *
*             ALL RIGHTS RESERVED                                     *
*                                                                     *
*                                                                     *
* FUNCTION = Invoke a hypothetical server via the Pascal              *
*            interface routines.                                      *
*                                                                     *
*            This sample program reads a customer record              *
*            from a host computer, examines the customer's            *
*            balance, and writes the customer record to               *
*            a file containing customer records if the                *
*            balance is greater than zero.                            *
*                                                                     *
* NOTES =                                                             *
*                                                                     *
*   RESTRICTIONS = This sample program is provided solely as          *
*                  an example of how the Pascal interface             *
*                  routines can be used to invoke a server.           *
*                                                                     *
* MODULE TYPE = IBM Personal Computer Pascal Compiler                 *
*               Version 2.00                                          *
*                                                                     *
* CHANGE ACTIVITY =                                                   *
*                                                                     *
******************** END PROLOGUE ***********************************)
(******************** DEFINITIONS ************************************)
program psampl;
        (*$SUBTITLE : 'CPRB Record Definition'*)
        (*$PAGE+*)
        (*$INCLUDE: 'UUPCPRB.INC'*)
        (*$SUBTITLE : 'Definitions Section'*)
        (*$INCLUDE: 'UUPPROCS.INC'*)

const                                       (* Miscellaneous consts      *)
        pfunc1   = 1;                       (* Get Record                *)
        pfunc2   = 2;                       (* Update AR file            *)

        pcrecsiz = 109;                     (* Customer Record size      *)

        prcok    = #00000000;               (* Server Return Code OK     *)
        plstr    = #00000004;               (* Last Record               *)

        poper    = 'ADMIN   ';              (* Default operator          *)
        pserver  = 'IBMabase';              (* Server Name               *)

type    custrec = record                    (* Customer Record           *)
           cusname [00]:  string(25);       (* Customer Name             *)
           cusaddr [25]:  string(25);       (* Street Address            *)
           cuscity [50]:  string(15);       (* City                      *)
           cusstat [65]:  string(15);       (* State                     *)
           cuszip  [80]:  string(9);        (* Zip Code                  *)
           cusacct [89]:  string(16);       (* Account Number            *)
           cusbal [105]:  integer4;         (* Balance                   *)
        end;
```

```
type    qparms = record                  (* Request Parameters        *)
           qpaflags [00]: byte;          (* Processing Flags          *)
           qpaoper  [01]: string(8);     (* Requesting Operator       *)
        end;
const                                    (* Values for qpaflags       *)
        qpalog  = #01;                   (* Log the transaction       *)
        qpacom  = #02;                   (* Commit transaction        *)

var     pcprb     :   uercprb;           (* CPRB record               *)
        pcustrec  :   custrec;           (* Customer Record           *)
        pqparms   :   qparms;            (* Request Parameters        *)
        pretcod   :   integer4;          (* SRPI Return Code          *)
        pcprbads  :   UERCPRBPTR;        (* CPRB address              *)

(****************** END DEFINITIONS *********************************)
      (*$SUBTITLE:  'Main procedure'*)
      (*$PAGE+*)
(********************* PSEUDOCODE ***********************************)
(*                     PROC (MAIN)                                   *)
(*                    1. SET PROCESSING OPTION = COMMIT              *)
(*                         TRANSACTION                               *)
(*                    1. SET REQUESTING OPERATOR ID                  *)
(*                    1. INITIALIZE SRPI RETURN CODE                 *)
(*                    1. INITIALIZE SERVER RETURN CODE               *)
(*                    1. DO WHILE SERVER RETURN CODE IS NOT          *)
(*                         LAST RECORD AND SRPI RETURN CODE          *)
(*                         IS GOOD                                   *)
(*                    2. . INITIALIZE THE CPRB RECORD                *)
(*                           <INIT_SEND_REQ_PARMS>                   *)
(*                    2. . MOVE SERVER NAME AND FUNCTION (GET        *)
(*                           RECORD) INTO CPRB RECORD                *)
(*                    2. . SET CPRB REQUEST PARAMETERS BUFFER        *)
(*                           INFORMATION                             *)
(*                    2. . SET CPRB REPLY DATA BUFFER INFORMATION    *)
(*                    2. . SEND THE REQUEST TO THE SERVER            *)
(*                           <SEND REQUEST>                          *)
(*                    2. . IF THE SRPI RETURN CODE IS GOOD           *)
(*                    3. . . IF THE SERVER RETURN CODE IS GOOD       *)
(*                    4. . . . IF THE ACCOUNT BALANCE IS POSITIVE    *)
(*                    5. . . . . SET CPRB FUNCTION = UPDATE          *)
(*                                 ACCOUNTS RECEIVABLE               *)
(*                    5. . . . . SET CPRB REQUEST DATA = CUSTOMER    *)
(*                                 RECORD                            *)
(*                    5. . . . . UPDATE THE ACCOUNTS RECEIVABLE      *)
(*                                 FILE <SEND REQUEST>               *)
(*                    4. . . . ENDIF                                 *)
(*                    3. . . ENDIF                                   *)
(*                    2. . ENDIF                                     *)
(*                    1. ENDWHILE                                    *)
(*                     END PROC (MAIN)                               *)
(******************** END PSEUDOCODE ********************************)
(************************ PROCEDURE *********************************)
begin                                    (* PROC  (MAIN)              *)

pqparms.qpaflags  := qpacom;             (* SET PROCESSING OPTION=    *)
                                         (* COMMIT TRANSACTION        *)

pqparms.qpaoper   := poper;              (* SET REQUEST OPERATOR ID   *)

pcprb.uerservrc   := UERERROK;           (* INITIALIZE SERVER         *)
                                         (* RETURN CODE               *)

pretcod           := prcok;              (* INITIALIZE SRPI RETURN CODE*)

while  (pcprb.uerservrc  <>  plstr) and (pretcod = prcok) do
begin                                    (* DO WHILE SERVER RETURN    *)
```

```
                                        (* CODE IS NOT LAST RECORD    *)

    pcprbads  := ADS pcprb;             (* INITIALIZE THE CPRB RECORD *)
    init_send_req_parms(pcprbads);      (* <INIT_SEND_REQ_PARMS       *)

    pcprb.uerserver  := pserver;        (* MOVE SERVER NAME AND       *)
    pcprb.uerfunct   := pfunc1;         (* FUNCTION INTO CPRB         *)

    pcprb.uerqparml  := sizeof(pqparms); (* SET CPRB REQUEST PARAMETERS*)
    pcprb.uerqparmad := ADS pqparms;    (* BUFFER INFORMATION         *)

    pcprb.uerrdatal  := pcrecsiz;       (* SET CPRB REPLY DATA        *)
    pcprb.uerrdataad := ADS pcustrec;   (* BUFFER INFORMATION         *)


    pretcod := sendrequest(pcprbads);   (* SEND THE REQUEST TO SERVER *)
                                        (* <SEND REQUEST>             *)

    if pretcod = UERERROK then          (* IF THE SRPI RETURN         *)
                                        (* CODE IS GOOD               *)
    begin

      if pcprb.uerservrc = prcok then   (* IF THE SERVER RETURN CODE  *)
                                        (* IS GOOD                    *)
      begin

        if pcustrec.cusbal > 0 then     (* IF THE ACCOUNT BALANCE     *)
                                        (* IS POSITIVE                *)
        begin

          pcprb.uerfunct := pfunc2;     (* SET CPRB FUNCTION = UPDATE *)
                                        (* ACCOUNTS RECEIVABLE        *)

          pcprb.uerqdatal := pcrecsiz;       (* SET CPRB REQUEST DATA *)
          pcprb.uerqdataad:= ADS pcustrec;   (* = CUSTOMER RECORD     *)

          pretcod := sendrequest(pcprbads);  (* UPDATE THE ACCOUNTS   *)
                                             (* RECEIVABLE FILE       *)
                                             (* <SEND REQUEST>        *)

    end;                                (* ENDIF                      *)
  end;                                  (* ENDIF                      *)
 end;                                   (* ENDIF                      *)
end;                                    (* ENDWHILE                   *)
end.                                    (* ENDPROC (MAIN)             *)
(********************** END PROCEDURE ********************************)
```



CONTENTS

About This Chapter

This chapter is for programmers who want to become familiar with writing a requester in the C language.

This chapter describes:

- C send_request function
- SRPI structure definition
- SRPI return codes
- Request record initialization
- Linking subroutines
- Language-specific notes
- A C sample program.

C Send_Request Function

The parameters of the SEND_REQUEST function are grouped in a single C structure of type UERCPRB. The INIT_SEND_REQ_PARMS function initializes the SEND_REQUEST parameters that have default values. This allows default values to be set only once for parameters not used by a requester. The SEND_REQUEST function has a single parameter that is a pointer to a structure of type UERCPRB.

The parameters in the C UERCPRB structure are the same as the parameters in the CPRB. The mapping is not necessarily the same. Application programs should make no assumptions about the mapping of the UERCPRB record to the CPRB.

The SRPI provides for sending a buffer of parameters and/or a buffer of data to the server and receiving a buffer of parameters and/or a buffer of data from the server. Any data can be sent using this interface. A generic type is used for these parameters of the SEND_REQUEST function. C uses the type pointer-to-character for these buffer pointers: for example, `char far *my_buffer_pointers;`

If the request parameters or data consist of several structures, the application must convert the data or parameters into a single flat structure that consists of a contiguous sequence of bytes which are used as a buffer. The requesting program must package the request parameters and data in a format recognizable by the server.

Structure members are stored sequentially in the same order in which they are declared. The first member has the lowest memory address. The last member has the highest memory address. The storage for each member begins on a memory boundary appropriate to its type. Unnamed blanks can occur between the members of a structure in memory.

You should compile your C application programs with the /ZP option. When you use the /ZP option, each structure member after the first member is stored beginning at the first available byte. This ensures a contiguous sequence of bytes within a structure. See "Language-Specific Notes" on page 3-8 for additional information about compiler options.

The same memory area can be used for both request and reply parameters. In addition, the same memory area can also be used for both request and reply data. The application program must ensure that the reply data and parameters are written into the request data and parameters buffer when the request data and parameters are no longer needed.

The object code for the C functions are on diskette. The declaration files for the functions, the structure type, and the return codes are also on the diskette.

The C object code linked with the requester program can push up to 16 words onto an application program stack. The PC router uses an additional 5 words of application program stack. Twenty-one words of application program stack are required. Ensure that your application program stack is large enough to meet this requirement.

SRPI Structure Definition

The UERCPRB structure type defines a structure passed to the PC router using the send_request function. The structure is defined in an application program by using the **#include** preprocessor directive to include the UUCCPRB.H file. See "Supplied Parameters" on page 1-9 and "Returned Parameters" on page 1-11 for the definitions and value ranges of the supplied and returned parameters.

The following is the SRPI structure definition:

```
typedef struct {

/* Supplied Parameters            Parameter Description              */

   char far *uerserver;       /* Address of ASCII name of server    */
   unsigned int uerfunct;     /* Function ID                        */

   /* Request Parameters and Data                                    */

   int      uerqparml;        /* Request Parameters Length          */
   char far *uerqparmad;      /* Request Parameters Address         */
   unsigned int uerqdatal;    /* Request Data Length                */
   char far *uerqdataad;      /* Request Data Address               */

   /* Reply Parameters and Data                                      */

   int      uerrparml;        /* Reply Parameters Buffer Length     */
   char far *uerrparmad;      /* Reply Parameters Buffer Address    */
   unsigned int uerrdatal;    /* Reply Data Buffer Length           */
   char far *uerrdataad;      /* Reply Data Buffer Address          */


/* Returned Parameters                                              */

   long int uerretcode;       /*  SRPI Return Code                  */
   long int uerservrc;        /*  Server Return Code                */
   int      uerrepldplen;     /*  Replied Parameters Length         */
   unsigned int uerreplddlen; /*  Replied Data Length               */
  } UERCPRB;
```

Notes:

1. *The pointer in uerserver must point to an 8-byte, left-justified, blank-padded (X'20') server name.*

2. *The supplied parameters are not changed by the SEND_REQUEST function.*

3. *All pointers in the UERCPRB structure are 32 bits.*

4. *When the return code value returned in uerretcode by the PC router is not successful the following output fields are undefined:*

   - *Server Return Code (uerservrc)*
   - *Replied Parameter Length (uerrepldplen)*
   - *Replied Data Length (uerreplddlen)*

The PC router may or may not have altered these fields. The PC router may or may not have initialized these fields to zero. The calling application should not expect these fields to be maintained or altered across any unsuccessful call to the PC router.

5. *The value returned from the SEND_REQUEST function is identical to the value in the UERRETCODE field in the UERCPRB structure.*

SEND_REQUEST Function Definition

The SEND_REQUEST function is defined in an application program by using the **#include** pre-processor directive to include the UUCCPRB.H file. Following is the function declaration:

```
extern long int send_request (UERCPRB far *);
```

SRPI Return Codes

To incorporate SRPI return code definitions in an application program, use the **#include** preprocessor directive to include the UUCCPRB.H file. The return code constants, their hexadecimal values, and their meanings are as follows:

```
#define UERERROK        0x00000000   /* Successful                          */

/* Type 1 Errors                                                          */

#define UERERRT1START   0x01000402   /* Not started                         */
#define UERERRT1LOAD    0x01000404   /* Not loaded                          */
#define UERERRT1BUSY    0x01000408   /* Busy                                */
#define UERERRT1VER     0x0100040A   /* Unsupported version ID              */
#define UERERRT1EMU     0x0100040C   /* PC 3270 Emulation, 3.0 not loaded   */
#define UERERRT1QPLEN   0x01000602   /* Request parameters length too large */
#define UERERRT1RPLEN   0x01000604   /* Reply parameters length too large   */
#define UERERRT1VERB    0x01000606   /* Invalid verb type                   */
#define UERERRT1SERV    0x01000608   /* Invalid server name                 */
#define UERERRT1QPAD    0x0100060C   /* Invalid request parameters address  */
#define UERERRT1QDAD    0x0100060E   /* Invalid request data address        */
#define UERERRT1RPAD    0x01000610   /* Invalid reply parameters address    */
#define UERERRT1RDAD    0x01000612   /* Invalid reply data address          */
#define UERERRT1TOPV    0x01000616   /* TOPVIEW not supported               */
#define UERERRT1CNCL    0x01000802   /* Cancelled by the IBM host computer  */
#define UERERRT1CONV    0x01000C00   /* Unable to maintain conversation     */
#define UERERRT1ISE     0x01000C02   /* Internal software error             */
#define UERERRT1PROT    0x01000C04   /* Protocol violation                  */
#define UERERRT1SYIN    0x01000C06   /* System inconsistency                */

/* Type 2 and Type 3 Errors                                               */

#define UERERRT2        0x02        /*  Acknowledge sent (Type 2 Error)    */
#define UERERRT3        0x03        /*  Acknowledge received (Type 3 Error) */
```

See Appendix A, "SRPI Return Codes" on page A-1 for a complete description of SRPI return codes.

Request Record Initialization

The initialization routine is defined in an application program by using the **#include** preprocessor directive to include the UUCCPRB.H file. The initialization routine sets all parameters that have default values to their corresponding default values.

The INIT_SEND_REQ_PARMS function sets all SEND_REQUEST parameters in the UERCPRB structure that have a default value. An application program that does not use all of the SEND_REQUEST parameters can initialize them once.

The INIT_SEND_REQ_PARMS function sets default values in the UERCPRB structure for the following send_request parameters:

- Request Parameters (pointer and length)
- Request Data (pointer and length)
- Reply Parameters Buffer (pointer and length)
- Reply Data Buffer (pointer and length)
- Function ID
- Server Name (pointer).

The INIT_SEND_REQ_PARMS function initializes the server name pointer to zero. The SEND_REQUEST function checks the server name pointer for the value zero. If the server name pointer is set to zero, then the CPRB server name is set to blanks (X'20'). The server name pointer remains set to zero.

The INIT_SEND_REQ_PARMS function declaration follows:

```
extern void INIT_SEND_REQ_PARMS(UERCPRB far *);
```

Linking Subroutines

The INIT_SEND_REQ_PARMS function initializes the C UERCPRB structure. The object module for the INIT_SEND_REQ_PARMS function is UUCINIT.OBJ. The SEND_REQUEST function calls the PC router. The object module for the SEND_REQUEST function is UUCSENDR.OBJ.

Each object module should be included in the list of object modules passed to the linking program.

Language-Specific Notes

Compiler options and program statements must be chosen so that only long pointers are passed to the PC router. The IBM C Compiler provides support by way of certain memory models and the *far* keyword used in declarations.

Compile C application programs with the */AL*, */ZE*, and */ZP* options.

Writing a Requester

The following C sample program invokes a server using the C interface functions. The program requests records from a customer records data set on the IBM host computer. The IBM host computer sends the customer records to the requester program for processing.

The requester examines the customer's balance returned from the server. If the customer's balance is positive, it is sent to the server. The server puts the positive balance into an accounts receivable data set on the IBM host computer.

Warning: This program is provided solely as an example of how the C interface functions can be used to invoke a server. It is not intended to produce meaningful output for your use or to provide a complete programming example that accounts for all possible error situations. It is not a programming tutorial.

The following books contain sample server programs:

- *TSO Extensions Programmer's Guide to the Server-Requester Programming Interface for MVS/Extended Architecture*
- *IBM Programmer's Guide to the Server-Requester Programming Interface for VM/System Product.*

C Sample Program

```
/******************** PROLOGUE ***************************************
*                                                                     *
*  MODULE NAME = CSAMPL.C                                             *
*                                                                     *
*  DESCRIPTIVE NAME = C Sample Program                                *
*                                                                     *
*  COPYRIGHT = (C) COPYRIGHT IBM CORP. 1984, 1987                     *
*              LICENSED MATERIAL - PROGRAM PROPERTY OF IBM            *
*              ALL RIGHTS RESERVED                                    *
*                                                                     *
*                                                                     *
*  FUNCTION = Invoke a hypothetical server via the C interface        *
*             routines.                                               *
*                                                                     *
*             This sample program reads a customer record             *
*             from a host computer, examines the customer's           *
*             balance and writes the customer record to               *
*             a file containing customer records if the balance       *
*             is greater than zero.                                   *
*                                                                     *
*  NOTES =                                                            *
*                                                                     *
*    RESTRICTIONS = This sample program is provided solely as         *
*                   an example of how the C interface routines        *
*                   can be used to invoke a server.                   *
*                                                                     *
*  MODULE TYPE = IBM Personal Computer C Compiler Version 1.00        *
*                                                                     *
*  CHANGE ACTIVITY =                                                  *
*                                                                     *
********************* END PROLOGUE ************************************
/******************** DEFINITIONS ************************************/
#include <uuccprb.h>
char    cserver[9] = "IBMabase";   /* Server Name                    */

char    coper[9] = "ADMIN   ";     /* Default operator name          */

main()                             /* PROC (MAIN)                    */
{
UERCPRB ccprb;                     /* CPRB structure                 */
struct  {                          /* Customer Record Structure      */

        char     cusname[25];      /* Customer Name                  */
        char     cusaddr[25];      /* Street Address                 */
        char     cuscity[15];      /* City                           */
        char     cusstat[15];      /* State                          */
        char     cuszip[9];        /* Zip Code                       */
        char     cusacct[16];      /* Account Number                 */
        long int cusbal;           /* Balance                        */

        }        ccustrec;

struct  {                          /* Request Parameters Structure   */

        char     qpaflags;         /* Processing Flags               */
#define QPALOG 0x01                /* Log the transaction            */
#define QPACOM 0x02                /* Commit the transaction         */
        char     qpaoper[8];       /* Requesting operator's          */
                                   /* sign-on ID                     */
```

```
        }       cqparms;
#define CFUNC1 1                /* Func Code: Get Record              */
#define CFUNC2 2                /* Func Code: Update accounts         */
                                /* receivable file                    */
#define CRCOK 0x00000000        /* Server Return Code OK              */
#define CLSTR 0x00000004        /* Last Record                        */

        int      cctr;          /* general purpose counter            */
        long int cretcod;       /* SRPI return code                   */

/****************** END DEFINITIONS ***********************************/
/********************** PSEUDOCODE ************************************/
/*                      PROC (MAIN)                                   */
/*                     1. SET PROCESSING OPTION = COMMIT              */
/*                          TRANSACTION                               */
/*                     1. SET REQUESTING OPERATOR ID                  */
/*                     1. INITIALIZE SERVER RETURN CODE               */
/*                     1. INITIALIZE SRPI RETURN CODE                 */
/*                     1. DO WHILE SERVER RETURN CODE IS NOT LAST     */
/*                          RECORD AND SRPI RETURN CODE IS GOOD       */
/*                     2. . INITIALIZE THE CPRB STRUCTURE             */
/*                          <INIT_SEND_REQ_PARMS>                     */
/*                     2. . MOVE SERVER NAME ADDRESS AND FUNCTION     */
/*                          (GET RECORD) INTO CPRB STRUCTURE          */
/*                     2. . SET CPRB REQUEST PARAMETERS BUFFER        */
/*                          INFORMATION                               */
/*                     2. . SET CPRB REPLY DATA BUFFER INFORMATION    */
/*                     2. . SEND THE REQUEST TO THE SERVER            */
/*                     2. . IF THE SRPI RETURN CODE IS GOOD           */
/*                     3. . . IF THE SERVER RETURN CODE IS GOOD       */
/*                     4. . . . IF THE ACCOUNT BALANCE IS POSITIVE    */
/*                     5. . . . . SET CPRB FUNCTION = UPDATE          */
/*                                  ACCOUNTS RECEIVABLE               */
/*                     5. . . . . SET CPRB REQUEST DATA = CUSTOMER    */
/*                                  RECORD                            */
/*                     5. . . . . UPDATE THE ACCOUNTS RECEIVABLE      */
/*                                  FILE <SEND_REQUEST>               */
/*                     4. . . . ENDIF                                 */
/*                     3. . . ENDIF                                   */
/*                     2. . ENDIF                                     */
/*                     1. ENDWHILE                                    */
/*                      ENDPROC (MAIN)                                */
/******************** END PSEUDOCODE ***********************************/
/************************ PROCEDURE*************************************/

ccprb.uerservrc = CRCOK;                /* INITIALIZE SERVER          */
                                        /* RETURN CODE                */

cretcod = UERERROK;                     /* INITIALIZE SRPI RETURN CODE  */

cqparms.qpaflags = QPACOM;              /* SET PROCESSING OPTION =    */
                                        /* COMMIT TRANSACTION         */

for (cctr = 0; cctr <= (sizeof cqparms.qpaoper) - 1; cctr++)
  cqparms.qpaoper[cctr] = coper[cctr];
                                        /* SET REQUESTING OPERATOR ID   */

while  (ccprb.uerservrc != CLSTR && cretcod == UERERROK)
{                                       /* DO WHILE SERVER RETURN CODE  */
                                        /* IS NOT LAST RECORD         */

  init_send_req_parms(&ccprb);          /* INITIALIZE CPRB STRUCTURE    */

  ccprb.uerserver = cserver;            /* MOVE SERVER NAME AND       */
  ccprb.uerfunct = CFUNC1;              /* FUNCTION INTO CPRB STRUCTURE */
```

```
  ccprb.uerqparml = sizeof cqparms;   /* SET CPRB REQUEST PARAMETER   */
  ccprb.uerqparmad = &cqparms;        /* BUFFER INFORMATION           */

  ccprb.uerrdatal = sizeof ccustrec;  /* SET CPRB REPLY DATA BUFFER   */
  ccprb.uerrdataad = &ccustrec;       /* INFORMATION                  */

  cretcod = send_request(&ccprb);     /* SEND REQUEST TO SERVER       */

  if (cretcod == UERERROK)            /* IF THE SRPI RETURN           */
                                      /* CODE IS GOOD                 */
  {

    if (ccprb.uerservrc == CRCOK)     /* IF THE SERVER RETURN CODE    */
                                      /* IS GOOD                      */
    {

      if (ccustrec.cusbal > 0)        /* IF THE ACCOUNT BALANCE IS    */
                                      /* POSITIVE                     */
      {

        ccprb.uerfunct = CFUNC2;      /* SET CPRB FUNCTION = UPDATE   */
                                      /* ACCOUNTS RECEIVABLE          */

        ccprb.uerqdatal = sizeof ccustrec; /* SET CPRB REQUEST        */
        ccprb.uerqdataad = &ccustrec;      /* DATA = CUSTOMER RECORD  */

        cretcod = send_request(&ccprb);    /* UPDATE ACCOUNTS         */
                                           /* RECEIVABLE FILE         */
                                           /* <SEND_REQUEST>          */

      }                               /* ENDIF                        */

    }                                 /* ENDIF                        */

  }                                   /* ENDIF                        */

 }                                    /* ENDWHILE                     */

}                                     /* ENDPROC (MAIN)               */
/************************ END PROCEDURE ********************************/
```

Chapter 4. Language Interface and Syntax for Macro Assembler

CONTENTS

About This Chapter

This chapter is for programmers who want to become familiar with writing a requester in the Macro Assembler language.

This chapter describes:

- Macro definitions
- SRPI return codes
- Macro parameters
- CPRB mapping
- A Macro Assembler sample program.

Macro Definitions

Macro definitions:

- Provide CPRB mapping
- Initialize the CPRB with default values
- Set the required parameters in the CPRB
- Set the request buffers parameters which are optional in the CPRB
- Set the reply buffers parameters which are optional in the CPRB
- Execute the send_request interrupt
- Move the returned fields from the CPRB to user-defined data fields.

The Macro Assembler INCLUDE pseudo-op includes the PC router macros during assembly. The files to be included are:

- The UUMCPRB.INC file, which is the UERCPRB structure definition.
- The UUMINFAC.MAC file, which includes the interface macros used during assembly.

The application program provides storage for the CPRB. The UUMCPRB file defines the required size of the CPRB.

SEND_REQUEST Macro Definitions

Invoking the macros does not cause changes to registers, except for the SEND_REQUEST macro which modifies the AX and BX registers. To maintain register contents, the application program must have a valid stack pointer in the SS:SP registers. The stack pointer is required because the instructions, which the macros generate, push register values onto the stack prior to altering a register's contents. Up to 6 words may be pushed on the stack. The PC router uses an additional 5 words of application program stack. Eleven words of application program stack are required.

See "Supplied Parameters" on page 1-9 and "Returned Parameters" on page 1-11 for the semantics and value ranges of the supplied and returned parameters.

SRPI Return Codes

You can include SRPI return code definitions in an application program by using the INCLUDE pseudo-op to include the UUMCPRB.INC file. The return code constants, their hexadecimal values, and their meanings are as follows:

```
; Error Types   (High word of Type 0 and Type 1 Errors)
uererrokeq      EQU    0000H   ;Successful
uererrt1eq      EQU    0100H   ;Request failed       (Type 1 Error)

; Error Types   (High byte of Type 2 and Type 3 Errors)
uererrt2eq      EQU    02H     ;Acknowledge sent     (Type 2 Error)
uererrt3eq      EQU    03H     ;Acknowledge received (Type 3 Error)

;Type 1 Errors (Low word of return code)

uererrt1start   EQU    0402H   ;Not started
uererrt1loadt   EQU    0404H   ;Not loaded
uererrt1busy    EQU    0408H   ;Busy
uererrt1ver     EQU    040AH   ;Unsupported version ID
uererrt1emu     EQU    040CH   ;PC 3270 Emulation, 3.0 not loaded
uererrt1qplen   EQU    0602H   ;Request parameters length too large
uererrt1rplen   EQU    0604H   ;Reply parameters length too large
uererrt1verb    EQU    0606H   ;Invalid verb type
uererrt1serv    EQU    0608H   ;Invalid server name
uererrt1qpad    EQU    060CH   ;Invalid request parameters address
uererrt1qdad    EQU    060EH   ;Invalid request data address
uererrt1rpad    EQU    0610H   ;Invalid reply parameters address
uererrt1rdad    EQU    0612H   ;Invalid reply data address
uererrt1topv    EQU    0616H   ;TOPVIEW not supported
uererrt1cncl    EQU    0802H   ;Cancelled by the host computer
uererrt1conv    EQU    0C00H   ;Unable to maintain conversation
uererrt1ise     EQU    0C02H   ;Internal software error
uererrt1prot    EQU    0C04H   ;Protocol violation
uererrt1syin    EQU    0C06H   ;System inconsistency
```

See Appendix A, "SRPI Return Codes" on page A-1 for a complete description of SRPI return codes.

Macro Parameters

The ES:DI registers must point to the CPRB whenever invoking any of the macros. Several of the parameters are specified as *locations*. A location indicates that the actual parameter should be a variable name or register designation giving a memory location, using Macro Assembler syntax. The offset register designation may be [BX] or one of the index registers [SI] or [DI]. It is assumed that DS is the data segment register. To override this assumption, use the ES: segment override prefix.

The following examples are valid location parameters:

- my_variable_name
- my_variable_name [BX]
- my_variable_name [SI]
- my_variable_name [BX] [DI]
- ES:my_variable_name
- ES:my_variable_name [BX].

Vectors are doubleword address fields containing an offset followed by a segment value, with the bytes within a word reversed. Vectors are used where the parameter is a pointer (for example, to a buffer or to the CPRB).

The macros can be invoked with null parameters. When a parameter is null, the corresponding field in the CPRB is not accessed. All parameters are optional in terms of invoking macros. The requester application should not issue the send_request verb until all fields in the CPRB have been set to their intended values. For an example of using positional parameters, see <SET_REPLY_PARMS> on page 4-17.

SEND_REQ_INIT Macro

The SEND_REQ_INIT macro sets default values in the CPRB for the following send_request parameters:

- Request Parameters (pointer and length)
- Request Data (pointer and length)
- Reply Parameters Buffer (pointer and length)
- Reply Data Buffer (pointer and length)
- Function ID
- Server Name (set to blanks).

The SEND_REQ_INIT macro syntax is as follows:

```
SEND_REQ_INIT
```

SET_REQ_PARMS Macro

The SET_REQ_PARMS macro sets all the send_request parameters except the request and reply buffer information. The SET_REQ_PARMS macro syntax is as follows:

```
SET_REQ_PARMS  SERV_NAM,FUNCT
```

SERV_NAM: The location of the server name which is assumed to be 8 bytes in length, left-justified and padded with blanks. The SERV_NAM value must be reachable from the DS register and indicated in one of the following ways:

1. The character string: DSSI.

   The DS:SI register pair points to the left character of the server name to be moved into the CPRB.

2. Any valid **source** operand for the **LEA SI,source** instruction.

FUNCT: The location of a word containing the Function ID value, a literal value, or a label equated to the Function ID. The location is indicated in one of the following ways:

1. The character string: AX.

   The AX register contains the function ID to be moved into the CPRB.

2. Any valid **source** operand for the **MOV AX,source** instruction.

SET_REQ_BUFFERS Macro

The SET_REQ_BUFFERS macro sets the values of the request data and request parameters buffers and the corresponding lengths. The SET_REQ_BUFFERS macro syntax is as follows:

```
SET_REQ_BUFFERS QPARM_BUF,QPARM_LEN,QDATA_BUF,QDATA_LEN
```

QPARM_BUF: The location of a vector that points to the request parameter buffer; must be specified so as to be a valid substitution for the `source` operand in the `MOV AX,source` and `MOV AX,source + 2` instructions.

QPARM_LEN: The location of a word that contains the length of the request parameters buffer or a label equated to the length. The length is indicated in one of the following ways:

1. The character string: CX.

   The CX register contains the length to be moved into the CPRB.

2. Any valid `source` operand for the `MOV CX,source` instruction.

QDATA_BUF: The location of a vector that points to the request data buffer; must be specified so as to be a valid substitution for the `source` operand in the `MOV AX,source` and `MOV AX,source + 2` instructions.

QDATA_LEN: The location of a word that contains the length of the request data buffer or a label equated to the length. This is indicated in one of the following ways:

1. The character string: DX.

   The DX register contains the length to be moved into the CPRB.

2. Any valid `source` operand for the `MOV DX,source` instructions.

SET_REPLY_BUFFERS Macro

The SET_REPLY_BUFFERS macro sets the value of the reply data and reply parameters buffers and the corresponding pointers. The SET_REPLY_BUFFERS macro syntax is as follows:

```
SET_REPLY_BUFFERS PARM_BUF,PARM_LEN,DATA_BUF,DATA_LEN
```

PARM_BUF: The location of the vector that points to the reply parameters buffer; must be specified so as to be a valid substitution for the **source** operand in the **MOV AX,source** and **MOV AX,source + 2** instructions.

PARM_LEN: The location of a word that contains the length of the reply parameters buffer or a label equated to the length. The location is indicated in one of the following ways:

1. The character string: CX.

   The CX register contains the length to be moved into the CPRB.

2. Any valid **source** operand for the **MOV CX,source** instruction.

DATA_BUF: The location of the vector that points to the reply data buffer; must be specified so as to be a valid substitution for the **source** operand in the **MOV AX,source** and **MOV AX,source + 2** instructions.

DATA_LEN: The location of a word that contains the length of the reply data buffer or a label equated to the length. This is indicated in the following ways:

1. The character string: DX.

   The DX register contains the length to be moved into the CPRB.

2. Any valid **source** operand for the **MOV DX,source** instruction.

SEND_REQUEST Macro

The SEND_REQUEST macro executes the send_request verb by issuing an interrupt. The SEND_REQUEST macro syntax is as follows:

```
SEND_REQUEST
```

ES:DI must contain the segment and offset of the CPRB when this macro is invoked.

Calling the PC router modifies the AX and BX registers. When the PC router processes a request successfully, the AX register is set to zero upon return to the calling application and the BX register is undefined. If the AX register is not zero, the PC router is not loaded and the request is not processed. The CPRB fields are not updated.

Note: Application programs which use the SEND_REQUEST macro to invoke the PC router do not need to examine the contents of the AX register to determine whether or not the PC router is loaded. The instructions expanded by the SEND_REQUEST macro move the appropriate value into the return code field in the CPRB when the PC router is not loaded.

GET_REPLY Macro

The GET_REPLY macro retrieves the parameters returned when a send_request has been processed. The GET_REPLY syntax is as follows:

```
GET_REPLY RET_CODE,SERV_RC,REP_PARM_LEN,REP_DATA_LEN
```

RET_CODE: Location of a doubleword to which the return code should be moved; must be specified so as to be a valid substitution for the **target** operand in the **MOV target,CX** and **MOV target + 2,CX** instructions.

SERV_RC: Location of a doubleword to which the server return code should be moved; must be specified so as to be a valid substitution for the **target** operand in the **MOV target,CX** and **MOV target + 2,CX** instructions.

REP_PARM_LEN: The CPRB Replied Parameters Length are moved to this location of a word. This is indicated in one of the following ways:

1. The character string: BX.

   The field is moved into register BX.

2. Any valid **target** operand for the **MOV target,CX** instruction.

REP_DATA_LEN: The CPRB Replied Data Length is moved to this location of a word. This is indicated in one of the following ways:

1. The character string: CX.

   The field is moved into the CX register.

2. Any valid **target** operand for the **MOV target,CX** instruction.

CPRB Mapping

A pseudo-op called the Macro Assembler STRUC is used to define the CPRB. To define the CPRB in an application program, use the INCLUDE pseudo-op to include the UUMCPRB.INC file. The following is the CPRB structure definition:

```
uercprb       STRUC
uerrbsiz       dw            ?          ;Size of CPRB in bytes
uerversion     dw            ?          ;Version Number
uerretcode     dd            ?          ;Return Code
uerverbtyp     db            ?          ;Verb Type
               db            ?          ;Reserved

uerfunct       dw            ?          ;Function ID
               dw            ?          ;Reserved
uerqparml      dw            ?          ;Request Parameters Length
uerqparmad     dd            ?          ;Request Parameters Address
uerqdatal      dw            ?          ;Request Data Length
uerqdataad     dd            ?          ;Request Data Address
uerrparml      dw            ?          ;Reply Parameters Length
uerrparmad     dd            ?          ;Reply Parameters Address
uerrdatal      dw            ?          ;Reply Data Length
uerrdataad     dd            ?          ;Reply Data Address

               dw            ?          ;Reserved
uerservrc      dd            ?          ;Server Return Code
uerrepldplen   dw            ?          ;Replied Parameters Length
uerrepldd1en   dw            ?          ;Replied Data Length
uerwkarea      db          46 dup(?)    ;Work Area
uersrvnml      dw           ?           ;Server Name field length
uerserver      db          8 dup(?)     ;Server Name
uercprb        ENDS
```

The following two values are also defined in the UUMCPRB file:

```
uerversnum     equ           0100H;     ;Version number
uersendreq     equ           1          ;Send_Request
```

The UUMCPRB file does not allocate memory for the CPRB. The requester program allocates memory for the CPRB by using the define byte. An example of the define byte follows:

```
uumcprbseg     SEGMENT  'data'
uumcprb        db       SIZE uercprb dup  (OFFH)   ;Allocate space
;                                                    for CPRB
uumcprbseg     ENDS
```

Note: The following CPRB output fields are undefined when the return code value in uerretcode returned by the PC router is any value other than successful:

- Server Return Code (uerservrc)
- Replied Parameter Length (uerrepldplen)
- Replied Data Length (uerrepldd1en).

The PC router may or may not have altered or initialized these fields to zero. The calling application should not expect these fields to be maintained or altered across any unsuccessful call to the PC router.

Writing a Requester

The following Macro Assembler sample program invokes a server using the Macro Assembler interface macros. The application program requests records from a customer records data set on the IBM host computer. The IBM host computer sends the customer records to the requester program for processing.

The requester examines the customer's balance returned from the server. If the customer's balance is positive, it is sent to the server. The server puts the positive balance into an accounts receivable data set on the IBM host computer.

Warning: This program is provided solely as an example of how the Macro Assembler macros can be used to invoke a server. It is not intended to produce meaningful output for your use or to provide a complete programming example that accounts for all possible error situations. It is not a programming tutorial.

The following books contain sample server programs:

- *TSO Extensions Programmer's Guide to the Server-Requester Programming Interface for MVS/Extended Architecture*
- *IBM Programmer's Guide to the Server-Requester Programming Interface for VM/System Product.*

Macro Assembler Sample Program

```
;******************** PROLOGUE ********************************
;                                                               *
; MODULE NAME = MSAMPL.ASM                                      *
;                                                               *
; DESCRIPTIVE NAME = Macro Assembler Sample Program             *
;                                                               *
; COPYRIGHT = (C) COPYRIGHT IBM CORP. 1984, 1987                *
;             LICENSED MATERIAL - PROGRAM PROPERTY OF IBM       *
;             ALL RIGHTS RESERVED                               *
;                                                               *
;                                                               *
; FUNCTION = Invoke a hypothetical server via the Macro         *
;            Assembler interface macros.                        *
;                                                               *
;            This sample program reads a customer record        *
;            from a host computer, examines the customer's      *
;            balance, and writes the customer record to         *
;            a file containing customer records if the          *
;            balance is greater than zero.                      *
;                                                               *
; NOTES =                                                       *
;                                                               *
;   RESTRICTIONS = This sample program is provided solely       *
;                  as an example of how the Macro Assembler     *
;                  macros can be used to invoke a server.       *
;                                                               *
; MODULE TYPE = Macro Assembler                                 *
;                                                               *
; CHANGE ACTIVITY =                                             *
;                                                               *
;******************** END PROLOGUE ****************************
;******************** DEFINITIONS *****************************
        INCLUDE uuminfac.mac
        SUBTTL  'CPRB Mapping'
        PAGE
        INCLUDE uumcprb.inc
;---------------------------------------------------------------
        SUBTTL  'Customer Record Mapping'
        PAGE
mcustrec      STRUC
mcusname      db      25 dup (?)      ;name
mcusaddr      db      25 dup (?)      ;street address
mcuscity      db      15 dup (?)      ;city
mcusstat      db      15 dup (?)      ;state
mcuszip       db      9  dup (?)      ;zip
mcusacct      db      16 dup (?)      ;account number
mcusbal       dd      ?               ;balance
mcustrec      ENDS
;---------------------------------------------------------------
        SUBTTL  'Request Parameters Mapping'
        PAGE
mqparms       STRUC
mqpaflags     db      ?               ;Processing flags
mqpaoper      db      8 dup (?)       ;Requesting operator

mqparms       ENDS
;
;Equates for processing flags defined in STRUC mqparms
mqpalog       equ     01H             ;Log the transaction
mqpacom       equ     02H             ;Commit the transaction
;
;---------------------------------------------------------------
```

```
        SUBTTL  'MWORK - Work Area Segment'
        PAGE
mwork           SEGMENT 'data'
mdabuf          db      SIZE mcustrec dup (?)  ;Allocate
;                                               buffer for customer
;                                               records
mdabuf@         dd      mdabuf                 ;Vector to customer
;                                               record buffer
mdabufl         equ     SIZE mcustrec          ;Length of a customer
;                                               record

mqprmbuf        db      SIZE mqparms dup (?) ;Allocate a buffer
;                                               for request parms
mqprmbuf@       dd      mqprmbuf               ;Vector to request
;                                               parameters buffer
mqprmbufl       equ     SIZE mqparms           ;Length of a request
;                                               parameters

mserver_l$      equ     $                      ;First character of
;                                               server name
mserver         db      'IBMabase'             ;Server name
mserver_len$    equ     $-mserver_l$           ;Length of server name

mfunc1          equ     1                      ;Func code: Get Record
mfunc2          equ     2                      ;Func code: Update AR file

mrcok           equ     0000H                  ;Server Return Code: OK
mlstrh          equ     00H                    ;Last Record high byte
mlstrl          equ     04H                    ;Last Record low byte

moper_l$        equ     $                      ;First byte - operator
;                                               name
moper           db      'ADMIN   '             ;Default operator name
moper_len$      equ     $-moper_l$             ;Length - operator name

mretcode        dd      ?                      ;SRPI Return Code
                org     mretcode-mwork
mrclow          dw      0                      ;Low word of return code
mrchigh         dw      0                      ;High word of return code

mservrc         dd      ?                      ;Server Return Code
                org     mservrc-mwork
msrvrclow       dw      0                      ;Low word of return code
msrvrchigh      dw      0                      ;High word of return code

mwork           ENDS
;---------------------------------------------------------------
mcprbseg        SEGMENT 'data'
mcprb           db      SIZE uercprb dup (0FFH) ;Allocate space
;                                                 for CPRB
mcprbseg        ENDS
;---------------------------------------------------------------
mstack          SEGMENT stack 'stack'
                dw      255 dup (0FFFFH) ;Allocate a stack
mstaktop        dw      0FFFFH           ;First stack entry
mstack          ENDS
;****************** END DEFINITIONS ****************************
        SUBTTL  'Main procedure'
        PAGE
;********************** PSEUDOCODE ******************************
;                       PROC (MAIN)
;                      1. ESTABLISH A STACK
;                      1. SET DS TO POINT TO WORK AREA
;                      1. GET ADDRESS OF REQUEST PARAMETERS
;                      1. SET PROCESSING OPTION = COMMIT
;                           TRANSACTION
```

```
;                       1. SET REQUESTING OPERATOR ID
;                       1. GET ADDRESS OF CPRB INTO ES:DI
;                       1. DO WHILE SERVER RETURN CODE IS NOT LAST
;                            RECORD AND SRPI RETURN CODE IS GOOD
;                       2. . INITIALIZE THE CPRB <SEND_REQ_INIT>
;                       2. . MOVE SERVER NAME AND FUNCTION (GET
;                            RECORD) INTO CPRB <SET_REQ_PARMS>
;                       2. . SET CPRB REQUEST PARAMETERS BUFFER
;                            INFORMATION <SET_REQ_BUFFERS>
;                       2. . SET CPRB REPLY DATA BUFFER INFORMATION
;                            <SET_REPLY_PARMS>
;                       2. . SEND THE REQUEST TO THE SERVER
;                            <SEND_REQUEST>
;                       2. . GET THE SRPI RETURN CODE AND SERVER RETURN
;                            CODE <GET_REPLY>
;                       2. . IF THE SRPI RETURN CODE IS GOOD
;                       3. . . IF THE SERVER RETURN CODE IS GOOD
;                       4. . . . IF THE ACCOUNT BALANCE IS POSITIVE
;                       5. . . . . SET CPRB FUNCTION = UPDATE
;                                    ACCOUNTS RECEIVABLE
;                                    <SET_REQ_PARMS>
;                       5. . . . . SET CPRB REQUEST DATA = CUSTOMER
;                                    RECORD <SET_REQ_BUFFERS>
;                       5. . . . . UPDATE THE ACCOUNTS RECEIVABLE
;                                    FILE <SEND_REQUEST>
;                       4. . . . ENDIF
;                       3. . . ENDIF
;                       2. . ENDIF
;                       1. ENDWHILE
;                       1. RETURN TO DOS
;                        ENDPROC (MAIN)
;******************** END PSEUDOCODE ****************************
msampl        segment 'code'

        assume  cs:msampl
;************************ PROCEDURE ****************************
;***************************************************************
;                        PROC (MAIN)
mentry:
;                       1. ESTABLISH A STACK
        assume  ss:mstack
        mov     ax,seg mstack
        mov     ss,ax
        mov     sp,offset mstaktop

;                       1. SET DS TO POINT TO WORK AREA
        assume  ds:mwork
        mov     ax,seg mwork
        mov     ds,ax
;                       1. GET ADDRESS OF REQUEST PARAMETERS
        assume  es:mwork
        les     di,mqprmbuf@                ;ES:DI  ->  request
;                                            parameters buffer

;                       1. SET PROCESSING OPTION = COMMIT
;                            TRANSACTION
        mov     BYTE PTR es:[di+mqpaflags],mqpacom


;                       1. SET REQUESTING OPERATOR ID
        mov     cx,moper_len$               ;length of operator name
        add     di,OFFSET mqpaoper          ;ES:DI  ->  operator name
;                                            field in req parms buf
        mov     si,OFFSET moper_1$          ;DS:SI  -> operator name
rep     movsb                               ;Move operator name to
;                                            request parms buffer
```

```
;                       1. GET ADDRESS OF CPRB INTO ES:DI
        assume  es:mcprbseg
        mov     ax,SEG mcprbseg
        mov     es,ax
        mov     di,OFFSET mcprb           ;ES:DI  ->  CPRB

;                       1. DO WHILE SERVER RETURN CODE IS NOT LAST
;                           RECORD AND SRPI RETURN CODE IS GOOD
loop:
        cmp     msrvrchigh,mlstrh
        jne     nowhile
        cmp     msrvrclow,mlstrl
        je      nowhile
        cmp     mrclow,uererrokeq
        jne     nowhile
        jmp     while
nowhile:
        jmp     exit
while:

;                       2. . INITIALIZE THE CPRB <SEND_REQ_INIT>
        SEND_REQ_INIT

;                       2. . MOVE SERVER NAME AND FUNCTION (GET
;                             RECORD) INTO CPRB <SET_REQ_PARMS>
        SET_REQ_PARMS   mserver,mfunc1

;                       2. . SET CPRB REQUEST PARAMETERS BUFFER
;                             INFORMATION <SET_REQ_BUFFERS>
        SET_REQ_BUFFERS mqprmbuf@,mqprmbufl

;                       2. . SET CPRB REPLY DATA BUFFER INFORMATION
;                             <SET_REPLY_PARMS>
        SET_REPLY_BUFFERS ,,mdabuf@,mdabufl

;                       2. . SEND THE REQUEST TO THE SERVER
;                             <SEND_REQUEST>
        SEND_REQUEST

;                       2. . GET THE SRPI RETURN CODE AND SERVER RETURN
;                             CODE <GET_REPLY>
        GET_REPLY  mretcode,mservrc

;                       2. . IF THE SRPI RETURN CODE IS GOOD
        cmp     mrchigh,uererrokeq
        je      goodrc1                   ;exit label is >127
        jmp     end                       ;bytes away
goodrc1:

;                       3. . . IF THE SERVER RETURN CODE IS GOOD
        cmp     msrvrchigh,mrcok          ;Compare high word of
;                                          server return code
        je      goodrc2                   ;exit label is >127
        jmp     end                       ;bytes away
goodrc2:
        cmp     msrvrclow,mrcok           ;Compare low word of
;                                          server return code
        jne     end
;                       4 . . . . IF THE ACCOUNT BALANCE IS POSITIVE
        mov     si,WORD PTR mdabuf@       ;get offset of data buf,
;                                          DS:SI -> data buffer
        mov     ax,WORD PTR [si+mcusbal];Get low word of
;                                          balance
        mov     dx,WORD PTR [si+mcusbal+2];Get high word
;                                          of balance
```

```
        sub     dx,0                    ;Subtract zero from the
;                                        high word
        jl      end                     ;Negative balance, quit
        jg      update                  ;Positive balance, update
;                                        the AR file
        cmp     ax,0                    ;Is low word zero?
        je      end                     ;Yes-zero balance, quit

;                       5. . . . . SET CPRB FUNCTION = UPDATE
;                                    ACCOUNTS RECEIVABLE
;                                    <SET_REQ_PARMS>
update:
        SET_REQ_PARMS ,mfunc2

;                       5. . . . . SET CPRB REQUEST DATA = CUSTOMER
;                                    RECORD <SET_REQ_BUFFERS>
        SET_REQ_BUFFERS ,,mdabuf@,mdabuf1

;                       5. . . . . UPDATE THE ACCOUNTS RECEIVABLE
;                                    FILE <SEND_REQUEST>
        SEND_REQUEST

;                       4. . . . ENDIF
;                       3. . . ENDIF
;                       2. . ENDIF
end:    jmp     loop
;                       1. ENDWHILE
exit:
;                       1. RETURN TO DOS
        mov     ax,4C00H                ;Return to DOS with
        int     21H                     ;return code zero
;                        ENDPROC (MAIN)
;********************** END PROCEDURE **************************
msampl        ENDS

        END     mentry
```

Appendix A. SRPI Return Codes

Error Handling

Unsuccessful execution of a service request in the SRPI environment can result from problems at any of the different layers. In keeping with its function, the SRPI shields applications from transport layer errors as much as possible. Errors within server processing are handled by the applications. Other errors arise directly from the use of the SRPI and are treated accordingly.

Transport Layer Errors

The SRPI tries to recover from transport layer errors if possible. When recovery is not possible, the SRPI returns to the requester with a return code indicating transport layer failure. Such failures should be handled using the problem determination procedures of the transport mechanism.

Application Errors

The SRPI is responsible for routing requests to servers and returning replies to requesters. Requesters and servers are responsible for handling errors, except for abend, that servers encounter. When a server ends abnormally, the SRPI returns to the requester with an abend notice in the SRPI return code.

The server return code is set by the server on the IBM host computer running under VM or MVS. The value and meaning of the server return code is dependent on the Requester/Server.

Send_Request Processing Errors

SRPI return codes distinguish among a number of errors in processing the Send_Request function. Such errors include:

- Invalid function parameters
- Unidentified server
- Inability to contact the server.

There are also system error codes for internal SRPI errors.

Types of SRPI Return Codes

SRPI return codes include types 0, 1, 2, and 3.

Type 0 return code indicates successful completion of the send_request function.

Type 1 return codes are errors detected by the PC router that prevent a request from being processed.

Type 2 return codes are errors detected by the PC router and reported to the remote computer by an acknowledge interchange unit.

Type 3 return codes are errors detected by the remote computer and reported to the PC router by an acknowledge interchange unit.

The return code values are word-reversed and byte-reversed within each word. For example, the SRPI return code X'0100 0402' is stored in the CPRB memory as X'0204 0001'.

Type 0 Return Code

The type 0 return code has the following format: `X'0000 0000'`

This value indicates that the SRPI function completed successfully.

Type 0 Return Code Definition

Return Codes	Return Code Definitions
X'0000 0000'	Successful completion.

Type 1 Return Codes

Type 1 return codes have the following format: `X'0100 nnnn'`

The nnnn bytes are the hexadecimal value that indicates the specific error detected.

Type 1 Return Code Definitions

Return Codes	Return Code Definitions	Definition Descriptions
X'0100 0402'	The SRPI is not started.	After loading the SRPI, type STARTSR. Press Enter before using the SRPI.
X'0100 0404'	The PC router is not loaded.	Pascal, C, or Macro Assembler language interface programs return this return code to applications when PSCAPI.COM is not loaded.
X'0100 0408'	The PC router is busy.	The PC router can only process one request at a time. If the PC router is processing a request and a subsequent request is made to the PC router, the latter request is rejected.
X'0100 040A'	Unsupported PC router version ID.	The version ID in the CPRB passed to the PC router is not supported by the resident portion of the PC router. The version ID is automatically put into the CPRB by the Macro, C, or Pascal interface facilities.
X'0100 040C'	The IBM PC 3270 Emulation Program, Version 3.0, is not loaded or the SRPI option was not chosen on the PC 3270 Emulation Program Communication Setup menu.	
X'0100 0602'	Request Parameters Length exceeds the maximum.	The maximum value allowed is 32763.
X'0100 0604'	Reply Parameters Buffer Length exceeds maximum.	The maximum value allowed is 32763.
X'0100 0606'	Invalid or unsupported verb type.	The verb type in the CPRB passed to the PC router is not recognized. The verb type is put into the CPRB automatically by the Pascal, C, or Macro Assembler interface facilities.
X'0100 0608'	Invalid server name.	One or more characters in the server name could not be converted to EBCDIC for sending to the host. See Appendix B, "ASCII to EBCDIC Translation Table" on page B-1.
X'0100 060C'	Invalid request parameter address.	The request parameter address is zero, and the request parameter length is non-zero.
X'0100 060E'	Invalid request data address.	The request data address is zero, and the request data length is non-zero.

Return Codes	Return Code Definitions	Definition Descriptions
X'0100 0610'	Invalid Reply Parameter Address.	The reply parameter buffer address is zero, and the reply parameter buffer length is non-zero.
X'0100 0612'	Invalid Reply Data Address.	The reply data buffer address is zero, and the reply data buffer length is non-zero.
X'0100 0616'	The TopView environment is not supported.	The PC router does not process service requests when TopView is running.
X'0100 0802'	The host cancelled the communications session.	The remote computer cancelled the communications session while the request was being processed. You can cause this to happen by stopping the remote program with the PF3 key. However, use of this value is not limited to user-initiated cancellation of the session. It is used any time SRPI receives notification from the host that the session is cancelled while processing a request.
X'0100 0C00'	A system error has occurred. Conversation with the host has ended.	Conversation with the host ended because of one of the following reasons: • The host communication session is not active. • A link-level communication error has occurred. • The system was unable to reliably transmit data to or from the host. For example, a sequence error has occurred.
X'0100 0C02'	A system error has occurred because of an internal software error.	This is a system software error in the PC router, the IBM PC 3270 Emulation Program, Version 3.0, or the IBM 3270 PC Control Program, Version 3.0.
X'0100 0C04'	A system error has occurred. This is a protocol violation error.	This is a system software error in the PC router or the host.
X'0100 0C06'	A system error has occurred. The error is caused by system inconsistency.	This is a system software error in the PC router.

Type 2 and Type 3 Return Codes

Type 2 return codes have the following format: `X'02xx yyzz'`

The three error-specific bytes consist of the following exception conditions from the acknowledge interchange unit:

- `xx` Exception Class
- `yy` Exception Code
- `zz` Exception Object

Type 3 return codes have the following format: `X'03xx yyzz'`

The three error specific bytes consist of the following exception conditions from the acknowledge interchange unit:

- xx Exception Class
- yy Exception Code
- zz Exception Object

Exception Class Definitions

The exception classes are syntax, semantic, and process.

The syntax exception class is used to report violations of the transmission unit syntax rules. For example, omitting the server return code parameter: `X'0202 yyzz'`

The semantic exception class is used to report conflicting parameters. For example, an invalid correlation value: `X'0203 1B00'`

The process exception class is used to report exception conditions during request processing. For example, server unknown: `X'0304 1E00'`

The exception class definitions are listed in the following table:

Value	Description
X'00' to X'01'	Reserved
X'02'	Syntax
X'03'	Semantic
X'04'	Process
X'05' to X'FF'	Reserved

Exception Code Values

The exception code defines the specific condition detected. An exception code is required with all errors. The exception code values are listed in the following table.

Value	Description
X'00'	Reserved
X'08'	Segmentation
X'0C'	Invalid operand ID
X'0F'	Invalid length
X'16'	Invalid subfield type
X'18'	Invalid subfield value
X'19'	Required operand missing

Value	Description
X'1A'	Required subfield missing
X'1B'	Correlation error
X'1C'	Data exceeds architected maximum
X'1D'	Resource not available
X'1E'	Server unknown
X'1F'	Server not available
X'20'	Parameter length
X'21'	Data length
X'22'	Normal termination
X'23'	Abnormal termination (server abend)
X'24'	Multiple occurrences of a subfield
X'25'	Multiple occurrences of operand

Note: All exception code values not specified in this table are reserved.

Exception Object Values

The exception object defines the transmission unit object that was incorrect. An exception object is required with syntax errors. The exception object values are listed in the following table.

Value	Description
X'00'	Not specified
X'01'	Prefix
X'07'	Command operand
X'08'	Command subfields
X'1C'	Parameters operand
X'1D'	Data operand
X'13'	Suffix

Note: All exception object values not specified in this table are reserved.

Server Return Codes

A server return code is a doubleword (4-byte) return code presented to the server's IBM Enhanced Connectivity Facilities, which is routed to the requester. The content and meaning of the return status are defined by the Requester/Server. For information about server return codes, contact your host personnel or see one of the following manuals:

- *TSO Extensions Programmer's Guide to the Server-Requester Programming Interface for MVS/Extended Architecture*
- *IBM Programmer's Guide to the Server-Requester Programming Interface for VM/System Product* .

Appendix B. ASCII to EBCDIC Translation Table

The SRPI translates the ASCII server name to EBCDIC. The following table is used to convert server names from ASCII to EBCDIC when using an English system:

ASCII HEX	ASCII CHAR	EBCDIC HEX	EBCDIC CHAR
20	' '	40	' '
23	#	7B	#
24	$	5B	$
30	0	F0	0
31	1	F1	1
32	2	F2	2
33	3	F3	3
34	4	F4	4
35	5	F5	5
36	6	F6	6
37	7	F7	7
38	8	F8	8
39	9	F9	9
40	@	7C	@
41	A	C1	A
42	B	C2	B
43	C	C3	C
44	D	C4	D
45	E	C5	E
46	F	C6	F
47	G	C7	G
48	H	C8	H
49	I	C9	I
4A	J	D1	J
4B	K	D2	K
4C	L	D3	L
4D	M	D4	M
4E	N	D5	N
4F	O	D6	O
50	P	D7	P
51	Q	D8	Q
52	R	D9	R
53	S	E2	S

ASCII HEX	ASCII CHAR	EBCDIC HEX	EBCDIC CHAR
54	T	E3	T
55	U	E4	U
56	V	E5	V
57	W	E6	W
58	X	E7	X
59	Y	E8	Y
5A	Z	E9	Z
61	a	C1	A
62	b	C2	B
63	c	C3	C
64	d	C4	D
65	e	C5	E
66	f	C6	F
67	q	C7	G
68	h	C8	H
69	i	C9	I
6A	j	D1	J
6B	k	D2	K
6C	l	D3	L
6D	m	D4	M
6E	n	D5	N
6F	o	D6	O
70	p	D7	P
71	q	D8	Q
72	r	D9	R
73	s	E2	S
74	t	E3	T
75	u	E4	U
76	v	E5	V
77	w	E6	W
78	x	E7	X
79	y	E8	Y
7A	z	E9	Z

Appendix C. Product Requirements

The following software programs must be at a specified maintenance level to provide IBM Enhanced Connectivity functions. Contact your IBM representative for the most recent maintenance release information.

IBM Personal Computer Environment Requirements

- PC DOS 3.1 or 3.2
- IBM PC 3270 Emulation Program, Version 3.0 (for the PC, PC/XT, PC/AT, Portable PC), includes IBM Enhanced Connectivity support
- IBM 3270 PC Control Program, Version 3.0 on the 3270 PC (except models 24 and 26) and on the 3270 Personal Computer AT, includes IBM Enhanced Connectivity support.

IBM Requesters/Servers Environment Requirements

- IBM PC Requesters (6316993)
- IBM TSO/E Servers (5665-396), or
- IBM CMS Servers (5664-327).

MVS/XA Environment Requirements

- MVS/System Product, Version 2, Release 1.2 (MVS/XA), JES2 or JES3 (5740-XYS or 5665-291)
- TSO/Extensions, Release 3 with MVS/XA feature (5665-285), includes IBM Enhanced Connectivity support
- ACF/VTAM Version 2, (5735-RC5) or higher
- When using IBM TSO/E Servers, the Interactive System Productivity Facility, (ISPF),Version 2, Release 2 (5665-319), is required
- When using DXT, or DB2, with IBM TSO/E Servers, one of the following must be coresident:
  - DXT, Version 2 (5668-788)
  - DB2, Release 1 (5740-XYR).

VM Environment Requirements

- VM/System Product, Release 4.0 (5665-167), with or without the High Performance Option (HPO) (5664-173), includes IBM Enhanced Connectivity support
- ACF/VTAM, Version 3 (for SNA/SDLC connection)
- When using IBM CMS Servers, the Interactive System Productivity Facility (ISPF), Version 2, Release 2 (5664-282), is required
- When using DXT or SQL/DS with IBM CMS Servers, one of the following must be coresident:
  - DXT, Version 2 (5668-973) (when using DXT)
  - SQL/DS, Release 3.5 (5748-XXJ) (when using SQL/DS).

Glossary

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

A

ABEND. Abnormal end of task.

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

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

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

application. See *application program.*

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

application program interface (API). The formally defined programming language interface between an IBM system control program or program product and its user.

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

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

attribute. A characteristic that you can redefine.

B

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

C

character string. A sequence of consecutive characters.

character variable. The name of a character data item whose value may be assigned or changed while the program is running.

CMS router. A program running under VM/SP that uses the Server-Requester Programming Interface (SRPI) to route requests from the PC to the corresponding server on the host. The CMS router is part of the CMSSERV command processor in VM/SP Release 4.

communication subsystem. A program, or a set of programs, specifically for managing the exchange of information between remotely connected computers and/or devices.

CMSSERV. (1) A program that provides the Server-Requester Programming Interface (SRPI) and a service request manager on an IBM System/370 using VM/CMS. (2) The implementation of Enhanced Connectivity Facilities on a VM/SP system with CMS installed.

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

computer. A complete electronic data processing system, with CPU, input and output devices, capable of executing an application program.

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

Connectivity Programming Request Block (CPRB). An interface control block used by requesters and servers to communicate information.

CPRB. See *Connectivity Programming Request Block.*

D

data communications. The transmission of data between computers, remote devices, or both.

data processing. The systematic performance of operations upon data, for example, merging, sorting, computing; synonymous with information processing.

data type. A category that identifies the internal representation of data.

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

diskette. A thin, flexible magnetic plate that is permanently sealed in a protective cover. It can be used to store information.

DOS. Disk Operating System, a group of programs that enables a personal computer to organize and use information on diskettes or fixed disks, including application programs.

E

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

embedded blanks. Blank characters that are surrounded by any other characters.

emulation. Imitation; for example, one computer imitating the characteristics of another type of computer.

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

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

Enhanced Connectivity Facilities verbs. The operations that define the protocol boundary between requesters and servers in an Enhanced Connectivity Facilities network.

enter. To send information to the computer by pressing the Enter key.

entry. A single input operation on a work station.

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

F

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

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

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

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

function keys. (1) Keys that request actions but may not display or print characters. Included are the keys that normally produce a printed character, but when used with another key produce a function instead. (2) On 3270 PC and System/370 keyboards, these are program function keys.

G

H

hex. See *hexadecimal.*

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

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

I

ID. Identification.

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

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

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

K

keyboard. An input device consisting of various keys that allow the user to input data, control cursor and pointer locations, and to control the dialog between the user and the work station.

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

L

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

M

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

memory. Main storage in a computer.

menu. A displayed list of items from which a user can make a selection.

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

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

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

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

N

O

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

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

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

P

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

PC router. A program that is part of the IBM PC 3270 Emulation Program, Version 3.0 or the IBM 3270 PC Control Program, Version 3.0 that uses the Server-Requester Programming Interface (SRPI) to route requests from the IBM PC Requesters to the corresponding router on the host.

personal computer. In this publication, the term personal computer refers to the properly-configured members of the IBM Personal Computer family, including the PC, the PC/XT, the Personal Computer AT, the Portable Personal Computer, the IBM 3270 Personal Computer, and the 3270 Personal Computer AT.

process. (1) A sequence of actions required to produce a desired result. (2) An entity receiving a portion of the processor's time for executing a program. (3) An activity within the system begun by entering a command, running a program, or being started by another process.

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

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

Q

R

record. A collection of fields treated as a unit.

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

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

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

request to send. A mode that causes the modem to activate the carrier signal.

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

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

reserved character. A character or symbol that has a special (non-literal) meaning unless quoted.

reserved word. A word that is defined in a programming language for a special purpose and that must not appear as a user-declared identifier.

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

router. The router provides a new Server-Requester Programming Interface (SRPI): a request interface for requesters, or a reply interface for servers. See also *CMS router, MVS router, PC router, SRPI.*

S

sequential access. An access method in which records are read from, written to, or removed from a file based on the order of the records in the file.

sequential processing. The processing of records in the order in which they exist in a file.

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

server return code. A doubleword (4-byte) return code presented to the server's Enhanced Connectivity Facilities, which is routed to the requester. The content and meaning of the return status are defined by the Requester/Server.

server system. A data processing system containing one or more servers providing services in response to a request from another computer.

Server-Requester Programming Interface (SRPI). (1) A protocol between requesters and servers in an Enhanced Connectivity Facilities network. (2) An application programming interface

used by requester and server programs to communicate with the PC or host routers.

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

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

SRPI. See *Server-Requester Programming Interface.*

SRPI return code. A doubleword (4-byte) return code from the SRPI interface that indicates the results of the send request execution. See also *Server-Requester Programming Interface.*

stack. An area in storage that stores temporary register information and returns addresses of subroutines.

stack buffer. A storage area that stores retrievable data in sequence. The last data stored is the first data removed.

stack pointer. A register providing the current location of the stack.

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

T

U

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

V

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

W

Index


9059X99700001