z/OS Communications Server 2.5

IP IMS Sockets Guide



SC27-3653-50

Note:

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 303.

This edition applies to Version 2 Release 5 of z/OS° (5650-ZOS), and to subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2021-09-11

[©] Copyright International Business Machines Corporation 2000, 2021.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Tables. xiii About this document. xvii Who should read this document. xvii How this document is organized. xviii How to use this document. xviii How to contact IBM service. xviii Conventions and terminology that are used in this information. xviii How to read a syntax diagram. xix Prerequisite and related information. xxiii Summary of changes for IP IMS Sockets Guide. xxviii Changes made in z/OS Communications Server Version 2 Release 5. xxviii Changes made in z/OS Communications Server Version 2 Release 4. xxviii Changes made in z/OS Communications Server Version 2 Release 3. xxviii Changes made in z/OS Communications Server Version 2 Release 3. xxviii Changes made in z/OS Communications Server Version 2 Release 3. xxviii Changes made in z/OS Communications Server Version 2 Release 3. xxviii Changes made in z/OS Communications Server Version 2 Release 3. xxviii Changes made in z/OS Communications Server Version 2 Release 3. xxviii Changes made in z/OS Communications Server Version 2 Release 3. xxviii Changes made in z/OS Communications Server Version 2 Release 3.
About this document. xvii Who should read this document. xvii How this document is organized. xviii How to use this document. xviii How to contact IBM service. xviii Conventions and terminology that are used in this information. xviii How to read a syntax diagram. xix Prerequisite and related information. xxii Summary of changes for IP IMS Sockets Guide. xxvii Changes made in z/OS Communications Server Version 2 Release 5. xxvii Changes made in z/OS Communications Server Version 2 Release 4. xxvii Changes made in z/OS Communications Server Version 2 Release 3. xxvii Changes made in z/OS Communications Server Version 2 Release 4. xxvii Changes made in z/OS Communications Server Version 2 Release 3. xxvii Changes made in z/OS Communications Server Version 2 Release 3. xxvii Changes made in z/OS Communications Server Version 2 Release 3. xxvii Changes made in z/OS Communications Server Version 2 Release 3. xxvii Changes made in z/OS Communications Server Version 2 Release 3. xxvii Changes made in z/OS Communications Server Version 2 Release 3. xxvii Changes made in z/OS
Who should read this document
Who should read this document. Xviii How this document is organized. Xviii How to use this document. Xviii How to contact IBM service. Xviii Conventions and terminology that are used in this information. Xviii How to read a syntax diagram. Xix Prerequisite and related information. Xviii Changes made in z/OS Communications Server Version 2 Release 5. Xxvii Changes made in z/OS Communications Server Version 2 Release 4. Xxvii Changes made in z/OS Communications Server Version 2 Release 4. Xxvii Changes made in z/OS Communications Server Version 2 Release 3. Xxvii Changes made in z/OS Communications Server Version 2 Release 4. Xxvii Changes made in z/OS Communications Server Version 2 Release 3. Xxvii Changes made in z/OS Communications Server Version 2 Release 3. Xxvii Changes made in z/OS Communications Server Version 2 Release 3. Xxvii Changes made in z/OS Communications Server Version 2 Release 3. Xxvii Changes made in z/OS Communications Server Version 2 Release 3. Xxvii Changes made in z/OS Communications Server Version 2 Release 3. Xxvii Changes made in z/OS Communications Server Version 2 Release 3.
How this document is organized
How to date this document
Conventions and terminology that are used in this information
How to read a syntax diagram
Prerequisite and related information
Summary of changes for IP IMS Sockets Guide .xxvii Changes made in z/OS Communications Server Version 2 Release 5
Changes made in z/OS Communications Server Version 2 Release 5
Changes made in z/OS Communications Server Version 2 Release 4
Changes made in z/OS Communications Server Version 2 Release 3
Chapter 1. Using TCP/IP in the IMS environment
The role of IMS TCP/IP
IMS TCP/IP feature components
The IMS Listener
The IMS Assist module2
The MVS TCP/IP socket application programming interface (Sockets Extended)
Chapter 2. IMS TCP/IP
Using IMS with SNA or TCP/IP3
TCP/IP internets
Mainframe interactive processing4
Client/server processing
TCP, UDP, and IP
The socket API5
Programming with sockets5
Socket types
Addressing TCP/IP hosts
A typical client/server program flow chart
Concurrent and iterative servers
The basic socket calls
Server TCP/IP calls
Server SUCKET call
Server BIND call
Server LISTEN Call
Server GIVESOCKET and TAKESOCKET calls
Server DEAD and WDITE calls
Client TCD/ID calls
Client SOCKET coll
Client CONNECT call 12
Client Read/Write calls — the conversation

Client CLOSE call	
Other socket calls	
The SELECT call	
IOCTL and FCNTL calls	
GIVESOCKET and TAKESOCKET calls	
What you need to run IMS TCP/IP	
A summary of what IMS TCP/IP provides	
Chapter 3. Principles of operation of the Listener and the Assist mod	19 Iule 19
The role of the IMS Listener	
The role of the IMS Assist module	19
Client/server logic flow	20
How the connection is established	20
How the server exchanges data with the client	
How the IMS Listener manages multiple connection requests	25
Use of the IMS message queue	
Call sequence for the IMS Listener	26
Application design considerations	
Restrictions for operation of the Listener and the Assist module	
Chapter 4. How to write an IMS TCP/IP client program	
Explicit-mode client program logic flow	
Explicit mode client program togic now	29
Explicit-mode application data	30
Implicit-mode client logic flow	30
Implicit-mode client call sequence	31
Implicit-mode application data stream	32
Implicit-mode application data	
IMS TCP/IP message segment formats.	
Transaction-request message segment (client to Listener)	
Request-status message segment	
Complete-status message segment	
End-of-message segment (EOM)	
PL/I coding	
Chapter F. How to write on IMC TCD/ID conver program	20
General server program logic flow	
Explicit-mode server program logic flow	
Explicit-mode call sequence	
Explicit-mode application data	
Transaction-initiation message segment	38
Program design considerations	
I/O PCB explicit-mode server	ر 3 ۱۸
Explicit mode server PL/I programming considerations	40- 40
Implicit-mode server program logic flow	40 40
Implicit-mode server call sequence	40 40
Implicit-mode application data	40 Д1
Programming to the Assist module interface	۲+ 42
Implicit-mode server PI /I programming considerations	Δ2
Implicit-mode server Clanguage programming considerations	42 ب 12
I/O PCB implicit-mode server	
Chapter 6. How to customize and operate the IMS Listener	45
How to start the IMS Listener	45 //۲
How to stop the IMS Listener	

The IMS Listener configuration file	
TCPIP statement	
LISTENER statement	
TRANSACTION statement	47
The IMS Listener security exit	
TCP/IP services definitions	48
The <i>bla</i> PROFILE TOPIP data set	
The hlg TCPIP DATA data set	
Chapter 7. CALL instruction application programming interface	
CALL instruction API environmental restrictions and programming requirements	51
CALL instruction API output register information	ـــــــــــــــــــــــــــــــــــــ
CALL instruction API compatibility considerations	525
CALL instruction application programming interface (API)	
Understanding COPOL Accomptor and DL/Leall formate	
COPOL landuade cell formet	ວວ ເລ
COBOL language call format	
Assembly language call format.	
PL/I language call format	
Converting parameter descriptions	
Diagnosing problems in applications using the CALL instruction AP1	
CALL instruction API error messages and return codes	54
Code CALL instructions	55
ACCEPT	
BIND	57
BIND2ADDRSEL	59
CLOSE	62
CONNECT	63
FCNTL	66
FREEADDRINFO	
GETADDRINFO	69
GETCLIENTID	
GETHOSTBYADDR	
GETHOSTBYNAME	80
GETHOSTID	
GETHOSTNAME	
GETIBMOPT	85
GETNAMEINEO	87
GETPEERNAME	
GETSOCKNAME	ده ۲۷
	112
INE16_IS_SRCADDR	
LISTEN	
NTOP	
PTON	
READ	
READV	
RECV	
RECVFROM	
RECVMSG	
SELECT	
SELECTEX	
SEND	153
SENDMSG	155 155
SENDTO	150
SEND TO.	

SETSOCKOPT	
SHUTDOWN	
SOCKET	
TAKESOCKET	
TERMAPI	
WRITE	
WRITEV	
Using data translation programs for socket call interface	
Assembly language utility programs call format	
Data translation	
Bit-string processing	
Call interface sample programs	
Sample code for IPv4 server program	
Sample program for IPv4 client program	
Sample code for IPv6 server program	205
Sample program for IPv6 client program	
Common variables used in PL/I sample programs	
Common variables used in COBOL sample programs	
COBOL call interface sample IPv6 server program	
COBOL call interface sample IPv6 client program	231
Chapter 8. IMS Listener samples	237
IMS TCP/IP control statements	237
JCL for starting a message processing region	
JCL for linking the IMS Listener	
Listener IMS definitions	
Sample program explicit-mode	240
Sample explicit-mode program flow	
Sample explicit-mode client program (C language)	240
Sample explicit-mode server program (Assembly language)	
Sample program implicit-mode	
Sample implicit-mode program flow	
Sample implicit-mode client program (C language)	
Sample implicit-mode server program (Assembly language)	
Sample program - IMS MPP client	
Sample IMS MPP client program flow	
Sample client program for non-IMS server	
Sample server program for IMS MPP client	
Appendix A. Return codes	
Sockets return codes (ERRNOs)	
Appendix B. Related protocol specifications	
Appendix C. Accessibility	
Notices	
Terms and conditions for product documentation	
IBM Online Privacy Statement	
Policy for unsupported hardware	
Minimum supported hardware	
Policy for unsupported hardware	
Trademarks	
Bibliography	

Index	
Communicating your comments to IBM	

Figures

1. The use of TCP/IP with IMS	3
2. TCP/IP protocols when compared to the OSI Model and SNA	4
3. A typical client/server session	8
4. An iterative server	9
5. A concurrent server	9
6. The SELECT call	13
7. How user applications access TCP/IP networks with IMS TCP/IP	
8. IMS TCP/IP message flow for transaction initiation	21
9. IMS TCP/IP message flow for explicit-mode input/output	23
10. IMS TCP/IP message flow for implicit mode input/output	24
11. JCL: Sample run Listener procedure	45
12. Definition of the TCP/IP profile	49
13. The TCPIPJOBNAME Parameter in the DATA data set	49
14. Storage definition statement examples	
15. ACCEPT call instructions example	56
16. BIND call instruction example	58
17. BIND2ADDRSEL call instruction example	61
18. CLOSE call instruction example	63
19. CONNECT call instruction example	64
20. FCNTL call instruction example	67
21. FREEADDRINFO call instruction example	68
22. GETADDRINFO call instruction example	70
23. GETCLIENTID call instruction example	77

24. GETHOSTBYADDR call instruction example	78
25. HOSTENT structure that is returned by the GETHOSTBYADDR call	79
26. GETHOSTBYNAME call instruction example	80
27. HOSTENT structure returned by the GETHOSTYBYNAME call	82
28. GETHOSTID call instruction example	83
29. GETHOSTNAME call instruction example	84
30. GETIBMOPT call instruction example	
31. Example of name field	87
32. GETNAMEINFO call instruction example	
33. GETPEERNAME call instruction example	92
34. GETSOCKNAME call instruction example	94
35. GETSOCKOPT call instruction example	96
36. GIVESOCKET call instruction example	
37. INET6_IS_SRCADDR call instruction example	
38. INITAPI call instruction example	
39. IOCTL call instruction example	
40. COBOL language example for SIOCGHOMEIF6	
41. COBOL language example for SIOCGIFNAMEINDEX	
42. COBOL II example for SIOCGIFCONF	
43. LISTEN call instruction example	
44. NTOP call instruction example	
45. PTON call instruction example	
46. READ call instruction example	
47. READV call instruction example	
48. RECV call instruction example	

49. RECVFROM call instruction example	
50. SELECT call instruction example	
51. SELECTEX call instruction example	150
52. SEND call instruction example	
53. SENDTO call instruction example	
54. SETSOCKOPT call instruction example	
55. SHUTDOWN call instruction example	
56. SOCKET call instruction example	
57. TAKESOCKET call instruction example	
58. TERMAPI call instruction example	
59. WRITE call instruction example	
60. WRITEV call instruction example	
61. EZACIC04 EBCDIC-to-ASCII table	
62. EZACIC04 call instruction example	
63. EZACIC05 ASCII-to-EBCDIC table	
64. EZACIC05 call instruction example	
65. EZACIC06 call instruction example	
66. EZAZIC08 call instruction example	
67. EZACIC09 call instruction example (Part 1 of 2)	
68. EZACIC09 call instruction example (Part 2 of 2)	
69. EZACIC14 EBCDIC-to-ASCII table	
70. EZACIC14 call instruction example	
71. EZACIC15 ASCII-to-EBCDIC table	
72. EZACIC15 call instruction example	
73. EZASOKPS PL/1 sample server program for IPv4	

74. EZASOKPC PL/1 sample client program for IPv4	
75. EZASO6PS PL/1 sample server program for IPv6	
76. EZASO6PC PL/1 sample client program for IPv6	
77. CBLOCK PL/1 common variables	218
78. EZACOBOL COBOL common variables	223
79. EZASO6CS COBOL call interface sample IPv6 server program	230
80. EZASO6CC COBOL call interface sample IPv6 client program	236
81. Cross zone Lnk IMS application interface	239
82. Sample C client to drive IMS Listener	
83. Sample assembler IMS server	247
84. Sample C client to drive IMS Listener	
85. Sample assembler IMS server	253
86. Sample of IMS program as a client	260
87. Sample of IMS program as a server	

Tables

1. First fullword passed in a bit string in select	14
2. Second fullword passed in a bit string in select	15
3. Socket calls	17
4. Format of data passed to the security exit	
5. ACCEPT call requirements	
6. BIND call requirements	57
7. BIND2ADDRSEL call requirements	60
8. CLOSE call requirements	62
9. CONNECT call requirements	64
10. FCNTL call requirements	66
11. FREEADDRINFO call requirements	
12. GETADDRINFO call requirements	69
13. GETCLIENTID call requirements	
14. GETHOSTBYADDR call requirements	78
15. GETHOSTBYNAME call requirements	80
16. GETHOSTID call requirements	
17. GETIBMOPT call requirements	85
18. GETNAMEINFO call requirements	87
19. GETPEERNAME call requirement	
20. GETSOCKNAME call requirements	
21. GETSOCKOPT call requirements	
22. OPTNAME options for GETSOCKOPT and SETSOCKOPT	
23. GIVESOCKET call requirements	

24. INET6_IS_SRCADDR call requirements	
25. INITAPI call requirements	116
26. IOCTL call requirements	118
27. IOCTL call arguments	125
28. LISTEN call requirements	128
29. NTOP call requirements	129
30. PTON call requirements	131
31. READ call requirements	133
32. READV call requirements	134
33. RECV call requirements	
34. RECVFROM call requirements	
35. RECVMSG call requirements	
36. SELECT call requirements	
37. SELECTEX call requirements	
38. SEND call requirements	153
39. SENDMSG call requirements	
40. SENDTO call requirements	
41. SETSOCKOPT call requirements	161
42. OPTNAME options for GETSOCKOPT and SETSOCKOPT	
43. SHUTDOWN call requirements	178
44. SOCKET call requirements	
45. TAKESOCKET call requirements	
46. TERMAPI call requirements	183
47. WRITE call requirements	
48. WRITEV call requirements	

49. Sockets ERRNOs

About this document

This document describes how to use IP Services with IMS Version 7 and later. It describes the IMS call interface and the supporting functions.

This information includes descriptions of support for both IPv4 and IPv6 networking protocols. Unless explicitly noted, descriptions of IP protocol support concern IPv4. IPv6 support is qualified within the text.

This information refers to Communications Server data sets by their default SMP/E distribution library name. Your installation might, however, have different names for these data sets where allowed by SMP/E, your installation personnel, or administration staff. For instance, this information refers to samples in SEZAINST library as simply in SEZAINST. Your installation might choose a data set name of SYS1.SEZAINST, CS390.SEZAINST or other high level qualifiers for the data set name.

This document addresses the following topics:

- IMS client/server application design
- The IMS Listener
- The IMS Assist function
- The IMS socket calls, including call syntax conventions

Who should read this document

This document is intended for programmers who have some familiarity with IMS Transaction Manager and IP Services, and who need to develop IMS client/server applications.

To ensure proper interprogram communication, the two halves of a client/server program must be developed together. At a minimum, they must agree on protocol and data formats. To complicate matters (particularly in the case of a UNIX processor talking to an IMS mainframe), the technology differences are so extensive that the two halves will often be coded by different individuals — one, an IP socket programmer; the other, an IMS programmer.

This document has been designed for users with a variety of backgrounds and needs:

- Application designers need to know how the various components of IMS TCP/IP interact to provide program-to-program communication. These readers should read <u>Chapter 3</u>, "Principles of operation of the Listener and the Assist module," on page 19.
- Experienced IP socket programmers need to know the protocol and message formats necessary to establish communication with the IMS Listener and with the server program. These readers should read <u>Chapter 4, "How to write an IMS TCP/IP client program," on page 29</u> and <u>Chapter 7, "CALL instruction</u> application programming interface," on page 51.
- Experienced IMS application programmers will be familiar with IMS input/output calls (GU, GN, ISRT). These programmers have two choices:
 - Programmers with IMS experience and little or no TCP/IP programming experience will probably want to use the IMS Assist module, which accepts standard IMS I/O calls, and converts them to equivalent socket calls. They should read the sections on implicit-mode programming.
 - IMS programmers with socket experience can choose to code native C language or use the Sockets Extended API. These programmers should read the sections on explicit-mode programming and Chapter 7, "CALL instruction application programming interface," on page 51.
- IMS system programmers and communication programmers are responsible for the IMS system itself. These readers should read Chapter 6, "How to customize and operate the IMS Listener," on page 45.

How this document is organized

z/OS Communications Server: IP IMS Sockets Guide contains the following information:

- An overview of TCP/IP as it is used with IMS and the types of applications for which it is intended to be used.
- Information about the IMS Listener, including principles of operation, writing and customizing client and server programs, use of the CALL Instruction API, and samples.
- Appendix A, "Return codes," on page 267, Appendix B, "Related protocol specifications," on page 279, and Appendix C, "Accessibility," on page 299 provide additional information for this document.
- "Notices" on page 303 contains notices and trademarks used in this information.
- <u>"Bibliography" on page 307</u> contains descriptions of the documents in the z/OS Communications Server library.

How to use this document

To use this information, you should be familiar with z/OS TCP/IP services and the TCP/IP suite of protocols.

How to contact IBM service

For immediate assistance, visit this website: https://www.ibm.com/mysupport

Most problems can be resolved at this website, where you can submit questions and problem reports electronically, and access a variety of diagnosis information.

For telephone assistance in problem diagnosis and resolution (in the United States or Puerto Rico), call the IBM Software Support Center anytime (1-800-IBM[®]-SERV). You will receive a return call within 8 business hours (Monday – Friday, 8:00 a.m. – 5:00 p.m., local customer time).

Outside the United States or Puerto Rico, contact your local IBM representative or your authorized IBM supplier.

If you would like to provide feedback on this publication, see <u>"Communicating your comments to IBM" on</u> page 317.

Conventions and terminology that are used in this information

Commands in this information that can be used in both TSO and z/OS UNIX environments use the following conventions:

- When describing how to use the command in a TSO environment, the command is presented in uppercase (for example, NETSTAT).
- When describing how to use the command in a z/OS UNIX environment, the command is presented in bold lowercase (for example, **netstat**).
- When referring to the command in a general way in text, the command is presented with an initial capital letter (for example, Netstat).

All the exit routines described in this information are *installation-wide exit routines*. The installation-wide exit routines also called installation-wide exits, exit routines, and exits throughout this information.

The TPF logon manager, although included with VTAM[®], is an application program; therefore, the logon manager is documented separately from VTAM.

Samples used in this information might not be updated for each release. Evaluate a sample carefully before applying it to your system.

z/OS no longer supports mounting HFS data sets (The POSIX style file system). Instead, a z/OS File System (ZFS) can be implemented. The term hierarchical file system, abbreviated as HFS, is defined as

a data structure that has a hierarchical nature with directories and files. References to hierarchical file systems or HFS might still be in use in z/OS Communications Server publications.

Note: In this information, you might see the following Shared Memory Communications over Remote Direct Memory Access (SMC-R) terminology:

- RoCE Express[®], which is a generic term representing IBM 10 GbE RoCE Express, IBM 10 GbE RoCE Express2, and IBM 25 GbE RoCE Express2 feature capabilities. When this term is used in this information, the processing being described applies to all of these features. If processing is applicable to only one feature, the full terminology, for instance, IBM 10 GbE RoCE Express will be used.
- RoCE Express2, which is a generic term representing an IBM RoCE Express2[®] feature that might operate in either 10 GbE or 25 GbE link speed. When this term is used in this information, the processing being described applies to either link speed. If processing is applicable to only one link speed, the full terminology, for instance, IBM 25 GbE RoCE Express2 will be used.
- RDMA network interface card (RNIC), which is used to refer to the IBM 10 GbE RoCE Express, IBM[®] 10 GbE RoCE Express2, or IBM 25 GbE RoCE Express2 feature.
- Shared RoCE environment, which means that the "RoCE Express" feature can be used concurrently, or shared, by multiple operating system instances. The feature is considered to operate in a shared RoCE environment even if you use it with a single operating system instance.

Clarification of notes

Information traditionally qualified as Notes is further qualified as follows:

Attention

Indicate the possibility of damage

Guideline

Customary way to perform a procedure

Note

Supplemental detail

Rule

Something you must do; limitations on your actions

Restriction

Indicates certain conditions are not supported; limitations on a product or facility

Requirement

Dependencies, prerequisites

Result

Indicates the outcome

Тір

Offers shortcuts or alternative ways of performing an action; a hint

How to read a syntax diagram

This syntax information applies to all commands and statements that do not have their own syntax described elsewhere.

The syntax diagram shows you how to specify a command so that the operating system can correctly interpret what you type. Read the syntax diagram from left to right and from top to bottom, following the horizontal line (the main path).

Symbols and punctuation

The following symbols are used in syntax diagrams:

Symbol

Description

Marks the beginning of the command syntax.

Indicates that the command syntax is continued.

I.

Marks the beginning and end of a fragment or part of the command syntax.

►◀

Marks the end of the command syntax.

You must include all punctuation such as colons, semicolons, commas, quotation marks, and minus signs that are shown in the syntax diagram.

Commands

Commands that can be used in both TSO and z/OS UNIX environments use the following conventions in syntax diagrams:

- When describing how to use the command in a TSO environment, the command is presented in uppercase (for example, NETSTAT).
- When describing how to use the command in a z/OS UNIX environment, the command is presented in bold lowercase (for example, netstat).

Parameters

The following types of parameters are used in syntax diagrams.

Required

Required parameters are displayed on the main path.

Optional

Optional parameters are displayed below the main path.

Default

Default parameters are displayed above the main path.

Parameters are classified as keywords or variables. For the TSO and MVS[™] console commands, the keywords are not case sensitive. You can code them in uppercase or lowercase. If the keyword appears in the syntax diagram in both uppercase and lowercase, the uppercase portion is the abbreviation for the keyword (for example, OPERand).

For the z/OS UNIX commands, the keywords must be entered in the case indicated in the syntax diagram.

Variables are italicized, appear in lowercase letters, and represent names or values you supply. For example, a data set is a variable.

Syntax examples

In the following example, the PUt subcommand is a keyword. The required variable parameter is *local_file*, and the optional variable parameter is *foreign_file*. Replace the variable parameters with your own values.



Longer than one line

If a diagram is longer than one line, the first line ends with a single arrowhead and the second line begins with a single arrowhead.

```
The first line of a syntax diagram that is longer than one line
The continuation of the subcommands, parameters, or both
```

Required operands

Required operands and values appear on the main path line. You must code required operands and values.

► REQUIRED_OPERAND →

Optional values

Optional operands and values appear below the main path line. You do not have to code optional operands and values.

Selecting more than one operand

An arrow returning to the left above a group of operands or values means more than one can be selected, or a single one can be repeated.



Nonalphanumeric characters

If a diagram shows a character that is not alphanumeric (such as parentheses, periods, commas, and equal signs), you must code the character as part of the syntax. In this example, you must code OPERAND=(001,0.001).

► OPERAND — = — (— 001 — , — 0.001 —) →

Blank spaces in syntax diagrams

If a diagram shows a blank space, you must code the blank space as part of the syntax. In this example, you must code OPERAND=(001 FIXED).

► OPERAND — = — (— 001 — FIXED —) →

Default operands

Default operands and values appear above the main path line. TCP/IP uses the default if you omit the operand entirely.



Variables

A word in all lowercase italics is a *variable*. Where you see a variable in the syntax, you must replace it with one of its allowable names or values, as defined in the text.

🍽 variable 🛏

Syntax fragments

Some diagrams contain syntax fragments, which serve to break up diagrams that are too long, too complex, or too repetitious. Syntax fragment names are in mixed case and are shown in the diagram and in the heading of the fragment. The fragment is placed below the main diagram.

🗭 Syntax fragment 🛛 🛏

Syntax fragment

▶ 1ST_OPERAND — , — 2ND_OPERAND — , — 3RD_OPERAND →

Prerequisite and related information

z/OS Communications Server function is described in the z/OS Communications Server library. Descriptions of those documents are listed in "Bibliography" on page 307, in the back of this document.

Required information

Before using this product, you should be familiar with TCP/IP, VTAM, MVS, and UNIX System Services.

Softcopy information

Softcopy publications are available in the following collection.

Titles	Description
IBM Z Redbooks	The IBM Z ^{®®} subject areas range from e-business application development and enablement to hardware, networking, Linux [®] , solutions, security, parallel sysplex, and many others. For more information about the Redbooks [®] publications, see <u>http://www.redbooks.ibm.com/</u> and <u>http://www.ibm.com/</u> systems/z/os/zos/zfavorites/.

Other documents

This information explains how z/OS references information in other documents.

When possible, this information uses cross-document links that go directly to the topic in reference using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see z/OS Information Roadmap (SA23-2299). The Roadmap describes what level of documents are supplied with each release of z/OS Communications Server, and also describes each z/OS publication.

To find the complete z/OS library, visit the <u>z/OS library</u> in <u>IBM Documentation</u> (https://www.ibm.com/ docs/en/zos). Relevant RFCs are listed in an appendix of the IP documents. Architectural specifications for the SNA protocol are listed in an appendix of the SNA documents.

The following table lists documents that might be helpful to readers.

Title	Number
DNS and BIND, Fifth Edition, O'Reilly Media, 2006	ISBN 13: 978-0596100575
Routing in the Internet, Second Edition, Christian Huitema (Prentice Hall 1999)	ISBN 13: 978-0130226471
<i>sendmail,</i> Fourth Edition, Bryan Costales, Claus Assmann, George Jansen, and Gregory Shapiro, O'Reilly Media, 2007	ISBN 13: 978-0596510299
SNA Formats	GA27-3136
<i>TCP/IP Illustrated, Volume 1: The Protocols,</i> W. Richard Stevens, Addison-Wesley Professional, 1994	ISBN 13: 978-0201633467
<i>TCP/IP Illustrated, Volume 2: The Implementation,</i> Gary R. Wright and W. Richard Stevens, Addison-Wesley Professional, 1995	ISBN 13: 978-0201633542
TCP/IP Illustrated, Volume 3: TCP for Transactions, HTTP, NNTP, and the UNIX Domain Protocols, W. Richard Stevens, Addison-Wesley Professional, 1996	ISBN 13: 978-0201634952
TCP/IP Tutorial and Technical Overview	GG24-3376
Understanding LDAP	SG24-4986
z/OS Cryptographic Services System SSL Programming	SC14-7495
z/OS IBM Tivoli Directory Server Administration and Use for z/OS	SC23-6788
z/OS JES2 Initialization and Tuning Guide	SA32-0991
z/OS Problem Management	SC23-6844
z/OS MVS Diagnosis: Reference	GA32-0904
z/OS MVS Diagnosis: Tools and Service Aids	GA32-0905
z/OS MVS Using the Subsystem Interface	SA38-0679
z/OS Program Directory	GI11-9848
z/OS UNIX System Services Command Reference	SA23-2280
z/OS UNIX System Services Planning	GA32-0884
z/OS UNIX System Services Programming: Assembler Callable Services Reference	SA23-2281
z/OS UNIX System Services User's Guide	SA23-2279
z/OS XL C/C++ Runtime Library Reference	SC14-7314
Open Systems Adapter-Express Customer's Guide and Reference	SA22-7935

Redbooks publications

The following Redbooks publications might help you as you implement z/OS Communications Server.

Title	Number
IBM z/OS Communications Server TCP/IP Implementation, Volume 1: Base Functions, Connectivity, and Routing	SG24-8096

Title	Number
IBM z/OS Communications Server TCP/IP Implementation, Volume 2: Standard Applications	SG24-8097
IBM z/OS Communications Server TCP/IP Implementation, Volume 3: High Availability, Scalability, and Performance	SG24-8098
IBM z/OS Communications Server TCP/IP Implementation, Volume 4: Security and Policy-Based Networking	SG24-8099
IBM Communication Controller Migration Guide	SG24-6298
IP Network Design Guide	SG24-2580
Managing OS/390 TCP/IP with SNMP	SG24-5866
Migrating Subarea Networks to an IP Infrastructure Using Enterprise Extender	SG24-5957
SecureWay Communications Server for OS/390 V2R8 TCP/IP: Guide to Enhancements	SG24-5631
SNA and TCP/IP Integration	SG24-5291
TCP/IP in a Sysplex	SG24-5235
TCP/IP Tutorial and Technical Overview	GG24-3376
Threadsafe Considerations for CICS	SG24-6351

Where to find related information on the Internet

z/OS

This site provides information about z/OS Communications Server release availability, migration information, downloads, and links to information about z/OS technology

http://www.ibm.com/systems/z/os/zos/

z/OS Internet Library

Use this site to view and download z/OS Communications Server documentation

http://www.ibm.com/systems/z/os/zos/library/bkserv/

z/OS Communications Server product

The page contains z/OS Communications Server product introduction

https://www.ibm.com/products/zos-communications-server

IBM Communications Server product support

Use this site to submit and track problems and search the z/OS Communications Server knowledge base for Technotes, FAQs, white papers, and other z/OS Communications Server information

https://www.ibm.com/mysupport

IBM Communications Server performance information

This site contains links to the most recent Communications Server performance reports

http://www.ibm.com/support/docview.wss?uid=swg27005524

IBM Systems Center publications

Use this site to view and order Redbooks publications, Redpapers, and Technotes

http://www.redbooks.ibm.com/

z/OS Support Community

Search the z/OS Support Community Library for Techdocs (including Flashes, presentations, Technotes, FAQs, white papers, Customer Support Plans, and Skills Transfer information)

z/OS Support Community

Tivoli[®] NetView[®] for z/OS

Use this site to view and download product documentation about Tivoli NetView for z/OS

http://www.ibm.com/support/knowledgecenter/SSZJDU/welcome

RFCs

Search for and view Request for Comments documents in this section of the Internet Engineering Task Force website, with links to the RFC repository and the IETF Working Groups web page

http://www.ietf.org/rfc.html

Internet drafts

View Internet-Drafts, which are working documents of the Internet Engineering Task Force (IETF) and other groups, in this section of the Internet Engineering Task Force website

http://www.ietf.org/ID.html

Information about web addresses can also be found in information APAR II11334.

Note: Any pointers in this publication to websites are provided for convenience only and do not serve as an endorsement of these websites.

DNS websites

For more information about DNS, see the following USENET news groups and mailing addresses:

USENET news groups

comp.protocols.dns.bind

BIND mailing lists

https://lists.isc.org/mailman/listinfo

BIND Users

- Subscribe by sending mail to bind-users-request@isc.org.
- Submit questions or answers to this forum by sending mail to bind-users@isc.org.

BIND 9 Users (This list might not be maintained indefinitely.)

- Subscribe by sending mail to bind9-users-request@isc.org.
- Submit questions or answers to this forum by sending mail to bind9-users@isc.org.

The z/OS Basic Skills Information Center

The z/OS Basic Skills Information Center is a web-based information resource intended to help users learn the basic concepts of z/OS, the operating system that runs most of the IBM mainframe computers in use today. The Information Center is designed to introduce a new generation of Information Technology professionals to basic concepts and help them prepare for a career as a z/OS professional, such as a z/OS systems programmer.

Specifically, the z/OS Basic Skills Information Center is intended to achieve the following objectives:

- Provide basic education and information about z/OS without charge
- · Shorten the time it takes for people to become productive on the mainframe
- Make it easier for new people to learn z/OS

To access the z/OS Basic Skills Information Center, open your web browser to the following website, which is available to all users (no login required): <u>https://www.ibm.com/support/knowledgecenter/</u>zosbasics/com.ibm.zos.zbasics/homepage.html?cp=zosbasics

Summary of changes for IP IMS Sockets Guide

This document contains terminology, maintenance, and editorial changes, including changes to improve consistency and retrievability. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Changes made in z/OS Communications Server Version 2 Release 5

This information contains no technical change for this release.

Changes made in z/OS Communications Server Version 2 Release 4

This document contains information previously presented in z/OS Communications Server: IP IMS Sockets Guide, which supported z/OS Version 2 Release 3.

Changed information

AT-TLS support for TLS v1.3, see Sockets return codes (ERRNOs)

Changes made in z/OS Communications Server Version 2 Release 3

This document contains information previously presented in z/OS Communications Server: IP IMS Sockets Guide, which supported z/OS Version 2 Release 1.

Changed information

• IPv6 getaddrinfo() API standards compliance, see <u>Parameter values set by the application</u> (GETADDRINFO code call).

xxviii z/OS Communications Server: z/OS V2R5.0 Communications Server: IP IMS Sockets Guide

Chapter 1. Using TCP/IP in the IMS environment

For peer-to-peer applications that use SNA communication facilities, remote programmable devices communicate with IMS through the advanced program-to-program communication (APPC) API. For peer-to-peer applications that use TCP/IP communication facilities, remote programmable devices communicate with IMS through facilities provided by IMS TCP/IP.

The IMS TCP/IP feature provides the services necessary to establish and maintain connection between a TCP/IP-connected host and an IMS MPP. In addition, you can develop client/server applications by using the TCP/IP socket application programming interface.

In operation, when a TCP/IP client requires program-to-program communication with an IMS server message processing program (MPP), the client sends its request to TCP/IP Services. TCP/IP passes the request to the IMS Listener, which schedules the requested MPP and transfers control of the connection to it. After control of the connection is passed, data transfer between the server and the remote client is done by using socket calls.

The role of IMS TCP/IP

The IMS/ESA® database and transaction management facility is used throughout the world. For many enterprises, IMS is the data processing backbone, supporting large personnel and financial databases, manufacturing control files, and inventory management facilities. IMS backup and recovery features protect valuable data assets, and the IMS Transaction Manager provides high-speed access for thousands of concurrent users.

Traditionally, many IMS users have used 3270-type protocol to communicate with the IMS Transaction Manager. In that environment, all the processing, including display screen formatting, is done by the IMS mainframe. During the decade of the 1980s, users began to move some of the processing outboard into personal computers. However, these PCs were typically connected to IMS through SNA 3270 protocol.

During that period, although most IMS users were focused on 3270 PC emulation, many non-IMS users were busy building a network based on a different protocol, called TCP/IP. As this trend developed, the need for an access path between TCP/IP-communicating devices and the still-indispensable processing power of IMS became clear. IMS TCP/IP provides that access path. Its role can be more easily understood when one distinguishes between traditional 3270 applications (in which the IMS processor does all the work), and the more complex client/server applications (in which the application logic is divided between the IMS processor and another programmable device such as a TCP/IP host).

MVS TCP/IP supports both application types:

- When a TCP/IP host needs access to a traditional 3270 Message Format Service (MFS) application, it does not have touse the IMS TCP/IP feature; it can connect to IMS directly through Telnet which provides 3270 emulation services for TCP/IP-connected clients. Telnet is a part of the base TCP/IP Services product. (See z/OS Communications Server: IP User's Guide and Commands for more information).
- When a TCP/IP host has to support a client/server application, it should use the IMS TCP/IP feature of TCP/IP Services. This feature supports two-way client/server communication between an IMS message processing program (MPP) and a TCP/IP host.

As used in this information, the term *client* means a program that requests services of another program, which is known as the *server*. The client is often a UNIX-based program; however, DOS, Windows, Linux, CMS, and MVS-based programs can also act as clients. Similarly, the term *server* means a program that is often an IMS message processing program (MPP); however, the server can be a TCP/IP host, responding to an IMS MPP client.

IMS TCP/IP feature components

The IMS TCP/IP feature consists of the following components:

- The IMS Listener, which provides connectivity
- The IMS Assist module, which simplifies TCP/IP communications programming
- The Sockets Extended application programming interface (API)

The IMS Listener

The purpose of the Listener is to provide clients with a single point of contact to IMS. The IMS Listener is a batch program (BMP) that waits for connection requests from remote TCP/IP-connected hosts. When a request arrives, the Listener schedules the appropriate transaction (the server) and passes a TCP/IP socket (representing the connection) to that server.

The IMS Listener maintains connection requests until the requested MPP takes control of the socket. The Listener can maintain a variable number of concurrent connection requests.

Tip: The backlog value specified on the listen call cannot be larger than the value configured by the SOMAXCONN statement in the stack's TCPIP PROFILE (the default value is 10). No error is returned if a larger backlog is requested. If you want a larger backlog, update the SOMAXCONN statement. See <u>z/OS</u> Communications Server: IP Configuration Reference for details.

The IMS Assist module

The Assist module is a subroutine that is a part of the server program. Its use is optional. With the Assist module, you can use conventional IMS calls for TCP/IP communication between client and server. The Assist module intercepts the IMS calls and issues the corresponding socket commands. Consequently, IMS MPP programmers who use the IMS Assist module require no TCP/IP skills.

Programs that do use the Assist module are known as *implicit-mode* programs because the socket calls are issued implicitly by the Assist module.

Programs that do not use the Assist module issue socket calls directly. Such programs are known as *explicit-mode* programs because of their explicit use of the calls.

The MVS TCP/IP socket application programming interface (Sockets Extended)

The socket call interface provides a set of programming calls that can be used in an IMS message processing program to conduct a conversation with a peer program in another TCP/IP processor. The interface is derived from BSD 4.3 socket, a commonly used communications programming interface in the TCP/IP environment. Socket calls include connection, initiation, and termination functions, and basic read/write communication. The MVS TCP/IP socket call interface makes it possible to issue socket calls from programs written in COBOL, PL/I, and assembly language.

The IMS socket calls are a subset of the TCP/IP socket calls. They are designed to be used in programs written in other than C language; hence the term Sockets Extended.

Chapter 2. IMS TCP/IP

With the IMS TCP/IP feature, remote users can access IMS client/server applications over TCP/IP internets. It is a feature of TCP/IP Services. Figure 1 on page 3 shows how IMS TCP/IP gives a variety of remote users peer-to-peer communication with IMS applications.

It is important to understand that IMS TCP/IP is primarily intended to support *peer-to-peer* applications, as opposed to the traditional IMS mainframe interactive applications in which the IMS system contained all programmable logic, and the remote terminal was often referred to as a "dumb" terminal. To connect a TCP/IP host to one of those traditional applications, you should first consider the use of Telnet, a function of TCP/IP Services which provides 3270 emulation. With Telnet, you can access existing 3270-style Message Format Services applications without modification. You should consider IMS TCP/IP only when developing new peer-to-peer applications in which both ends of the connection are programmable.



Figure 1. The use of TCP/IP with IMS

IMS TCP/IP provides a variant of the BSD 4.3 Socket interface, which is widely used in TCP/IP networks and is based on the UNIX system and other operating systems. The socket interface consists of a set of calls that IMS application programs can use to set up connections, send and receive data, and perform general communication control functions. The programs can be written in COBOL, PL/I, assembly language, or C.

Using IMS with SNA or TCP/IP

IMS is an online transaction processing system. This means that application programs that use IMS can handle large numbers of data transactions from large networks of computers and terminals.

Communication throughout these networks has often been based on the Systems Network Architecture (SNA) family of protocols. IMS TCP/IP offers IMS users an alternative to SNA — the TCP/IP family of protocols for those users whose native communications protocol is TCP/IP.

TCP/IP internets

This topic describes some of the basic ideas behind the TCP/IP family of protocols.

Like SNA, TCP/IP is a set of communication protocols used between physically separated computer systems. Unlike SNA and most other protocols, TCP/IP is not designed for a particular hardware technology. TCP/IP can be implemented on a wide variety of physical networks, and is specially designed

for communicating between systems on different physical networks (local and wide area). This is called *internetworking*.

Mainframe interactive processing

TCP/IP Services supports traditional 3270 mainframe interactive (MFI) applications with an emulator function called Telnet (TN3270). For these applications, all program logic runs in the mainframe, and the remote host uses only that amount of logic necessary to provide basic communications services. Thus, if your requirement is simply to provide access from a remote TCP/IP host to existing IMS MFI applications, you should consider Telnet rather than IMS TCP/IP as the communications vehicle. Telnet 3270-emulation functions allow your TCP/IP host to communicate with traditional applications without modification.

Client/server processing

TCP/IP also supports *client/server* processing, where processes are either:

- Servers that provide a particular service and respond to requests for that service
- · Clients that initiate the requests to the servers

With IMS TCP/IP, remote client systems can initiate communications with IMS and cause an IMS transaction to start. It is anticipated that this will be the most common mode of operation. (Alternatively, the remote system can act as a server with IMS initiating the conversation.)

TCP, UDP, and IP

TCP/IP is a family of protocols that is named after its two most important members. Figure 2 on page 4 shows the TCP/IP protocols used by IMS TCP/IP, in terms of the layered Open Systems Interconnection (OSI) model, which is widely used to describe data communication systems. For IMS users who might be more accustomed to SNA, the left side of Figure 2 on page 4 shows the SNA layers, which correspond very closely to the OSI layers.



Figure 2. TCP/IP protocols when compared to the OSI Model and SNA

The protocols implemented by TCP/IP Services and used by IMS TCP/IP, are highlighted in Figure 2 on page 4:

Transmission Control Protocol (TCP)

In terms of the OSI model, TCP is a transport-layer protocol. It provides a reliable virtual-circuit connection between applications; that is, a connection is established before data transmission begins. Data is sent without errors or duplication and is received in the same order as it is sent. No boundaries are imposed on the data; TCP treats the data as a stream of bytes.

User Datagram Protocol (UDP)

UDP is also a transport-layer protocol and is an alternative to TCP. It provides an unreliable datagram connection between applications (that is, data is transmitted link by link; there is no end-to-end

connection). The service provides no guarantees: data can be lost or duplicated, and datagrams can arrive out of order.

Internet Protocol (IP)

In terms of the OSI model, IP is a network-layer protocol. It provides a datagram service between applications, supporting both TCP and UDP.

The socket **API**

The socket API is a collection of socket calls that enable you to perform the following primary communication functions between application programs:

- Set up and establish connections to other users on the network
- · Send and receive data to and from other users
- Close down connections

In addition to these basic functions, the API enables you to:

- Interrogate the network system to get names and status of relevant resources
- Perform system and control functions as required

IMS TCP/IP provides two TCP/IP socket application program interfaces (APIs), similar to those used on UNIX systems. One interfaces to C language programs, the other to COBOL, PL/I, and System/370* assembly language programs.

- **C language**. Historically, TCP/IP has been associated with the C language and the UNIX operating system. Textbook descriptions of socket calls are usually given in C, and most socket programmers are familiar with the C interface to TCP/IP. For these reasons, TCP/IP Services includes a C language API. If you are writing new TCP/IP applications and are familiar with C language programming, you might prefer to use this interface. Refer to the <u>z/OS Communications Server</u>: IP Sockets Application Programming Interface Guide and Reference for the C language socket calls supported by MVS TCP/IP.
- Sockets Extended API (COBOL, PL/I, Assembly Language). The Sockets Extended API (Sockets Extended) is for those who want to write in COBOL, PL/I, or assembly language, or who have COBOL, PL/I, or assembly language programs that need to be modified to run with TCP/IP. The Sockets Extended API enables you to do this by using CALL statements. If you are writing new TCP/IP applications in COBOL, PL/I, or assembly language, you might prefer to use the Sockets Extended API. With this interface, C language is not required. See Chapter 7, "CALL instruction application programming interface," on page 51 for details of this interface.

Programming with sockets

The original UNIX socket interface was designed to hide the physical details of the network. It included the concept of a *socket*, which represents the connection to the programmer, yet shields the program (as much as possible) from the details of communication programming. **A socket is an endpoint for communication that can be named and addressed in a network.** From an application program perspective, a socket is a resource that is allocated by the TCP/IP address space. A socket is represented to the program by an integer called a *socket descriptor*.

Socket types

The MVS socket APIs provide a standard interface to the transport and internetwork layer interfaces of TCP/IP. They support three socket types: *stream, datagram, and raw*. Stream and datagram sockets interface to the transport layer protocols, and raw sockets interface to the network layer protocols. All three socket types are described here for background purposes. While CICS[®] supports stream and datagram sockets, stream sockets provide the most reliable form of data transfer offered by TCP/IP.

Stream sockets transmit data between TCP/IP hosts that are already connected to one another. Data is transmitted in a continuous stream; in other words, there are no record length or newline character boundaries between data. Communicating processes ¹ must agree on a scheme to ensure that both client and server have received all data. One way of doing this is for the sending process to send the *length* of

the data, followed by the data itself. The receiving process reads the length and then loops, accepting data until all of it has been transferred.

In TCP/IP terminology, the stream socket interface defines a reliable connection-oriented service. In this context, the word *reliable* means that data is sent without error or duplication and is received in the same order as it is sent. Flow control is built in to avoid data overruns.

The **datagram** socket interface defines a connectionless service. Datagrams are sent as independent packets. The service provides no guarantees; data can be lost or duplicated, and datagrams can arrive out of order. The size of a datagram is limited to the size that can be sent in a single transaction (currently the default is 8192 and the maximum is 65507). No disassembly and reassembly of packets is performed by TCP/IP.

The **raw** socket interface allows direct access to lower layer protocols, such as IP and Internet Control Message Protocol (ICMP). This interface is often used for testing new protocol implementations.

Addressing TCP/IP hosts

This information describes how one TCP/IP host addresses another TCP/IP host.²

Address families

An address family defines a specific addressing format. Applications that use the same addressing family have a common scheme for addressing socket end-points. TCP/IP for CICS IMS supports the AF_INET address family.

Socket addresses

A socket address in the AF_INET family comprises 4 fields: the name of the address family itself (AF_INET), a port, an IP address, and an 8-byte reserved field. In COBOL, a socket address looks like this:

01 NAME

03 FAMILYPIC 9(4) BINARY.03 PORTPIC 9(4) BINARY.03 IP_ADDRESSPIC 9(8) BINARY.03 RESERVEDPIC X(8).

You will find this structure in every call that addresses another TCP/IP host.

In this structure, FAMILY is a half-word that defines which addressing family is being used. In CICS, IMS, FAMILY is always set to a value of 2, which specifies the AF_INET IP address family. ³ The PORT field identifies the application port number; it must be specified in network byte order. The IP_ADDRESS field is the IP address of the network interface used by the application. It also must be specified in network byte order. The RESERVED field should be set to all zeros.

IP addresses

An IP address is a 32-bit field that represents a network interface. An IP address is commonly represented in *dotted decimal* notation such as *129.5.25.1*. Every IP address within an administered AF_INET domain must be unique. A common misunderstanding is that a host must have only one IP address. In fact, a single host can have several IP addresses — one for each network interface.

¹ In TCP/IP terminology, a *process* is essentially the same as an application program.

² In TCP/IP terminology, a host is simply a computer that is running TCP/IP. There is no connotation of "mainframe" or large processor within the TCP/IP definition of the word *host*.

³ Note that sockets support many address families, but TCP/IP for CICS, IMS supports only the IP address family.

Ports

A port is a 16-bit integer that defines a specific application, within an IP address, in which several applications use the same network interface. The port number is a qualifier that TCP/IP uses to route incoming data to a specific application within an IP address. Some port numbers are reserved for particular applications and are called *well-known ports*, such as Port 23, which is the well-known port for Telnet.

As an example, an MVS system with an IP address of 129.9.12.7 might have CICS, IMS as port 2000, and Telnet as port 23. In this example, a client desiring connection to CICS, IMS would issue a CONNECT call, requesting port 2000 at IP address 129.9.12.7.

Note: It is important to understand the difference between a socket and a port. TCP/IP defines a port to represent a certain process on a certain machine (network interface). A port represents the location of one process in a host that can have many processes. A bound socket represents a specific port and the IP address of its host. In the case of CICS, the Listener has a listening socket which has a port to receive incoming connection requests. When a connection request is received, the Listener creates a new socket representing the endpoint of this connection and passes it to the applications by way of the givesocket/takesocket calls.

Domain names

Because dotted decimal IP addresses are difficult to remember, TCP/IP also allows you to represent host interfaces on the network as alphabetic names, such as Alana.E04.IBM.COM, or CrFre@AOL.COM. Every Domain Name has an equivalent IP address or set of addresses. TCP/IP includes service functions (GETHOSTBYNAME and GETHOSTBYADDR) that will help you convert from one notation to another.

Network byte order

In the open environment of TCP/IP, IP addresses must be defined in terms of the architecture of the machines. Some machine architectures, such as IBM mainframes, define the lowest memory address to be the high-order bit, which is called *big endian*. However, other architectures, such as IBM PCs, define the lowest memory address to be the low-order bit, which is called *little endian*.

Network addresses in a given network must all follow a consistent addressing convention. This convention, known as network byte order, defines the bit-order of network addresses as they pass through the network. The TCP/IP standard network byte order is big-endian. In order to participate in a TCP/IP network, little-endian systems usually bear the burden of conversion to network byte order.

Note: The socket interface does not handle application data bit-order differences. Application writers must handle these bit order differences themselves.

A typical client/server program flow chart

Stream-oriented socket programs generally follow a prescribed sequence. See Figure 3 on page 8 for a diagram of the logic flow for a typical client and server. As you study this diagram, keep in mind the fact that a concurrent server typically starts before the client does, and waits for the client to request connection at step **3**. It then continues to wait for additional client requests after the client connection is closed.



Figure 3. A typical client/server session

Concurrent and iterative servers

An *iterative server* handles both the connection request and the transaction involved in the call itself. Iterative servers are fairly simple and are suitable for transactions that do not last long.

However, if the transaction takes more time, queues can build up quickly. In Figure 4 on page 9, once Client A starts a transaction with the server, Client B cannot make a call until A has finished.
TCP/IP



Figure 4. An iterative server

So, for lengthy transactions, a different sort of server is needed — the *concurrent server*, as shown in <u>Figure 5 on page 9</u>. Here, Client A has already established a connection with the server, which has then created a *child server process* to handle the transaction. This allows the server to process Client B's request without waiting for A's transaction to complete. More than one child server can be started in this way.

TCP/IP provides a concurrent server program called the **IMS Listener**. It is described in <u>Chapter 6</u>, "How to customize and operate the IMS Listener," on page 45.





Figure 3 on page 8 illustrates a concurrent server at work.

The basic socket calls

This topic provides an overview of the basic socket calls.

The following calls are used by the server:

SOCKET

Obtains a socket to read from or write to.

BIND

Associates a socket with a port number.

LISTEN

Tells TCP/IP that this process is listening for connections on this socket.

SELECT

Waits for activity on a socket.

ACCEPT

Accepts a connection from a client.

The following calls are used by a concurrent server to pass the socket from the parent server task (Listener) to the child server task (user-written application).

GIVESOCKET

Gives a socket to a child server task.

TAKESOCKET

Accepts a socket from a parent server task.

GETCLIENTID

Optionally used by the parent server task to determine its own address space name (if unknown) prior to issuing the GIVESOCKET.

The following calls are used by the client:

SOCKET

Allocates a socket to read from or write to.

CONNECT

Allows a client to open a connection to a server's port.

The following calls are used by both the client and the server:

WRITE

Sends data to the process on the other host.

READ

Receives data from the other host.

CLOSE

Terminates a connection, deallocating the socket.

For full discussion and examples of these calls, see <u>Chapter 7</u>, "CALL instruction application programming interface," on page 51.

Server TCP/IP calls

To understand Socket programming, the client program and the server program must be considered separately. In this topic the call sequence for the *server* is described. <u>"Client TCP/IP calls" on page 12</u> contains the typical call sequence for a *client*. Server TCP/IP calls are presented first because the server is usually already in execution before the client is started. The step numbers (such as **5**) in this topic refer to the steps in Figure 3 on page 8.

Server SOCKET call

The server must first obtain a socket **1**. This socket provides an end-point to which clients can connect.

A socket is actually an index into a table of connections in the TCP/IP address space, so TCP/IP usually assigns socket numbers in ascending order. In COBOL, the programmer uses the SOCKET call to obtain a new socket.

The socket function specifies the address family (AF_INET), the type of socket (STREAM), and the particular networking protocol (PROTO) to use. (When PROTO is set to zero, the TCP/IP address space automatically uses the appropriate protocol for the specified socket type). Upon return, the newly allocated socket's descriptor is returned in RETCODE.

Server BIND call

At this point **2**, an entry in the table of communications has been reserved for the application. However, the socket has no port or IP address associated with it until the BIND call is issued. The BIND function requires three parameters:

- The socket descriptor that was just returned by the SOCKET call.
- The number of the port on which the server wants to provide its service
- The IP address of the network connection on which the server is listening. If the application wants to receive connection requests from any network interface, the IP address should be set to zeros.

Server LISTEN call

After the bind, the server has established a specific IP address and port upon which other TCP/IP hosts can request connection. Now it must notify the TCP/IP address space that it intends to listen for connections on this socket. The server does this with the LISTEN**3** call, which puts the socket into passive open mode. *Passive open mode* describes a socket that can accept connection requests, but cannot be used for communication. A passive open socket is used by a listener program like the CICS IMS Listener to await connection requests. Sockets that are directly used for communication between client and server are known as *active open* sockets. In passive open mode, the socket is open for client contacts; it also establishes a backlog queue of pending connections.

This LISTEN call tells the TCP/IP address space that the server is ready to begin accepting connections. Normally, only the number of requests specified by the BACKLOG parameter will be queued.

Tip: The backlog value specified on the listen call cannot be larger than the value configured by the SOMAXCONN statement in the stack's TCPIP PROFILE (the default value is 10). No error is returned if a larger backlog is requested. If you want a larger backlog, update the SOMAXCONN statement. See the *z*/OS Communications Server: IP Configuration Reference for details.

Server ACCEPT call

At this time **5**, the server has obtained a socket, bound the socket to an IP address and port, and issued a LISTEN to open the socket. The server main task is now ready for a client to request connection **4**. The ACCEPT call temporarily blocks further progress. ⁴

The default mode for Accept is blocking. Accept behavior changes when the socket is non-blocking. The FCNTL() or IOCTL() calls can be used to disable blocking for a given socket. When this is done, calls that would normally block continue regardless of whether the I/O call has completed. If a socket is set to non-blocking and an I/O call issued to that socket would otherwise block (because the I/O call has not completed) the call returns with ERRNO 35 (EWOULDBLOCK).

When the ACCEPT call is issued, the server passes its socket descriptor, S, to TCP/IP. When the connection is established, the ACCEPT call returns a new socket descriptor (in RETCODE) that represents the connection with the client. **This is the socket upon which the server subtask communicates with the client**. Meanwhile, the original socket (S) is still allocated, bound and ready for use by the main task to accept subsequent connection requests from other clients.

To accept another connection, the server calls ACCEPT again. By repeatedly calling ACCEPT, a concurrent server can establish simultaneous sessions with multiple clients.

Server GIVESOCKET and TAKESOCKET calls

The GIVESOCKET and TAKESOCKET functions are not supported with the IMS TCP/IP OTMA Connection server. A server handling more than one client simultaneously acts like a dispatcher at a messenger service. A messenger dispatcher gets telephone calls from people who want items delivered and the

⁴ Blocking is a UNIX concept in which the requesting process is suspended until the request is satisfied. It is roughly analogous to the MVS wait. A socket is blocked while an I/O call waits for an event to complete. If a socket is set to block, the calling program is suspended until the expected event completes.

dispatcher sends out messengers to do the work. In a similar manner, the server receives client requests, and then spawns tasks to handle each client.

In UNIX-based servers, the *fork()* system call is used to dispatch a new subtask after the initial connection has been established. When the *fork()* command is used, the new process automatically inherits the socket that is connected to the client.

Because of architectural differences, CICS sockets does not implement the *fork()* system call. Tasks use the GIVESOCKET and TAKESOCKET functions to pass sockets from parent to child. The task passing the socket uses GIVESOCKET, and the task receiving the socket uses TAKESOCKET. See <u>"GIVESOCKET and</u> TAKESOCKET calls" on page 15 for more information about these calls.

Server READ and WRITE calls

Once a client has been connected with the server, and the socket has been transferred from the main task (parent) to the subtask (child), the client and server exchange application data, using various forms of READ/WRITE calls. See <u>"Client Read/Write calls – the conversation" on page 12</u> for details about these calls.

Client TCP/IP calls

The TCP/IP call sequence for a client is simpler than the one for a concurrent server. A client has to support only one connection and one conversation. A concurrent server obtains a socket upon which it can listen for connection requests, and then creates a new socket for each new connection.

Client SOCKET call

In the same manner as the server, the first call **1** issued by the client is the SOCKET call. This call causes allocation of the socket on which the client will communicate.

CALL 'EZASOKET' USING SOCKET-FUNCTION SOCTYPE PROTO ERRNO RETCODE.

Client CONNECT call

After the SOCKET call has allocated a socket to the client, the client can then request connection on that socket with the server through use of the CONNECT call **4**.

The CONNECT call attempts to connect socket descriptor (S) to the server with an IP address of NAME. The CONNECT call blocks until the connection is accepted by the server. On successful return, the socket descriptor (S) can be used for communication with the server.

This is essentially the same sequence as that of the server; however, the client need not issue a BIND command because the port of a client has little significance. The client need only issue the CONNECT call, which issues an implicit BIND. When the CONNECT call is used to bind the socket to a port, the port number is assigned by the system and discarded when the connection is closed. Such a port is known as an *ephemeral* port because its life is very short as compared with that of a concurrent server, whose port remains available for a prolonged time.

Client Read/Write calls — the conversation

A variety of I/O calls is available to the programmer. The READ and WRITE, READV and WRITEV, and SEND **6** and RECV **6** calls can be used only on sockets that are in the connected state. The SENDTO and RECVFROM, and SENDMSG and RECVMSG calls can be used regardless of whether a connection exists.

The WRITEV, READV, SENDMSG, and RECVMSG calls provide the additional features of scatter and gather data. Scattered data can be located in multiple data buffers. The WRITEV and SENDMSG calls gather the scattered data and send it. The READV and RECVMSG calls receive data and scatter it into multiple buffers.

The WRITE and READ calls specify the socket S on which to communicate, the address in storage of the buffer that contains, or will contain, the data (BUF), and the amount of data transferred (NBYTE). The server uses the socket that is returned from the ACCEPT call.

These functions return the amount of data that was either sent or received. Because stream sockets send and receive information in streams of data, it can take more than one call to WRITE or READ to transfer all of the data. It is up to the client and server to agree on some mechanism of signaling that all of the data has been transferred.

Client CLOSE call

When the conversation is over, both the client and server call CLOSE to end the connection. The CLOSE call also deallocates the socket, freeing its space in the table of connections.

Other socket calls

Several other calls that are often used — particularly in servers — are the SELECT call, the GIVESOCKET/ TAKESOCKET calls, and the IOCTL and FCTL calls. These calls are discussed next.

The SELECT call

Applications such as concurrent servers often handle multiple sockets at once. In such situations, the SELECT call can be used to simplify the determination of which sockets have data to be read, which are ready for data to be written, and which have pending exceptional conditions. An example of how the SELECT call is used can be found in Figure 6 on page 13.

WORKING	STORAGE	
01	SOC-FUNCTION	PIC X(16) VALUE IS 'SELECT'.
01	MAXSOC	PIC 9(8) BINARY VALUE 50.
01	TIMEOUT.	
	03 TIMEOUT-S	SECONDS PIC 9(8) BINARY.
	03 TIMEOUT-N	1ILLISEC PIC 9(8) BINARY.
01	RSNDMASK	PIC X(50).
01	WSNDMASK	PIC X(50).
01	ESNDMASK	PIC X(50).
01	RRETMASK	PIC X(50).
01	WRETMASK	PIC X(50).
01	ERETMASK	PIC X(50).
01	ERRNO	PIC 9(8) BINARY.
01	RETCODE	PIC S9(8) BINARY.
PROCEDU	RE	
CA	LL 'EZASOKET'	USING SOC-FUNCTION MAXSOC TIMEOUT
	F	RSNDMASK WSNDMASK ESNDMASK
	F	RETMASK WRETMASK ERETMASK
	E	ERRNO RETCODE.

Figure 6. The SELECT call

In this example, the application *sends* bit sets (the xSNDMASK sets) to indicate which sockets are to be tested for certain conditions, and *receives* another set of bits (the xRETMASK sets) from TCP/IP to indicate which sockets meet the specified conditions.

The example also indicates a time-out. If the time-out parameter is NULL, this is the C language API equivalent of a wait forever. (In Sockets Extended, a negative timeout value is a wait forever.) If the time-out parameter is nonzero, SELECT waits only the timeout amount of time for at least one socket to become ready on the indicated conditions. This is useful for applications servicing multiple connections that cannot afford to wait for data on a single connection. If the xSNDMASK bits are all zero, SELECT acts as a timer.

With the Socket SELECT call, you can define which sockets you want to test (the xSNDMASKs) and then wait (block) until one of the specified sockets is ready to be processed. When the SELECT call returns, the program knows only that some event has occurred, and it must test a set of bit masks (xRETMASKs) to determine which of the sockets had the event, and what the event was.

To maximize performance, a server should test only those sockets that are active. The SELECT call allows an application to select which sockets will be tested, and for what. When the Select call is issued, it blocks until the specified sockets are ready to be serviced (or, optionally) until a timer expires. When the select call returns, the program must check to see which sockets require service, and then process them.

To allow you to test any number of sockets with just one call to SELECT, place the sockets to test into a bit set, passing the bit set to the select call. A bit set is a string of bits where each possible member of the set is represented by a 0 or a 1. If the member's bit is 0, the member is not to be tested. If the member's bit is 1, the member is to be tested. Socket descriptors are actually small integers. If socket 3 is a member of a bit set, then bit 3 is set; otherwise, bit 3 is zero.

Therefore, the server specifies 3 bit sets of sockets in its call to the SELECT function: one bit set for sockets on which to receive data; another for sockets on which to write data; and any sockets with exception conditions. The SELECT call tests each selected socket for activity and returns only those sockets that have completed. On return, if a socket's bit is raised, the socket is ready for reading data or for writing data, or an exceptional condition has occurred.

The format of the bit strings is a bit awkward for an assembler programmer who is accustomed to bit strings that are counted from left to right. Instead, these bit strings are counted from right to left.

The first rule is that the length of a bit string is always expressed as a number of fullwords. If the highest socket descriptor you want to test is socket descriptor number three, you have to pass a 4-byte bit string, because this is the minimum length. If the highest number is 32, you must pass 8 bytes (2 fullwords).

The number of fullwords in each select mask can be calculated as:

INT(highest socket descriptor / 32) + 1

Table 1. First fullword passed in a bit string in select								
Socket descriptor numbers represented by byte	Bit O	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
Byte 0	31	30	29	28	27	26	25	24
Byte 1	23	22	21	20	19	18	17	16
Byte 2	15	14	13	12	11	10	9	8
Byte 3	7	6	5	4	3	2	1	0

Look at the first fullword you pass in a bit string in Table 1 on page 14.

In these examples, we use standard assembler numbering notation; the leftmost bit or byte is relative zero.

If you want to test socket descriptor number 5 for pending read activity, you raise bit 2 in byte 3 of the first fullword (X'0000020'). If you want to test both socket descriptor 4 and 5, you raise both bit 2 and bit 3 in byte 3 of the first fullword (X'0000030').

If you want to test socket descriptor number 32, you must pass two fullwords, where the numbering scheme for the second fullword resembles that of the first. Socket descriptor number 32 is bit 7 in byte 3 of the second fullword. If you want to test socket descriptors 5 and 32, you pass two fullwords with the following content: X'00000200000001'.

The bits in the second fullword represents the socket descriptor numbers shown in Table 2 on page 15. Subsequent mask words continue this pattern; word 3 for sockets 64 - 95, word 4 for sockets 96 - 127, and so on.

Table 2. Second fullword passed in a bit string in select								
Socket descriptor numbers represented by byte	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
Byte 4	63	62	61	60	59	58	57	56
Byte 5	55	54	53	52	51	50	49	48
Byte 6	47	46	45	44	43	42	41	40
Byte 7	39	38	37	36	35	34	33	32

If you develop your program in COBOL or PL/I, you might find that the EZACICO6 routine, which is provided as part of TCP/IP for MVS, will make it easier for you to build and test these bit strings. This routine translates between a character string mask (one byte per socket) and a bit string mask (one bit per socket).

In addition to its function of reporting completion on Read/Write events, the SELECT call can also be used to determine completion of events associated with the LISTEN and GIVESOCKET calls.

- When a connection request is pending on the socket for which the main process issued the LISTEN call, it will be reported as a pending read.
- When the parent process has issued a GIVESOCKET, and the child process has taken the socket, the parent's socket descriptor is selected with an exception condition. The parent process is expected to close the socket descriptor when this happens.

IOCTL and FCNTL calls

In addition to SELECT, applications can use the IOCTL or FCNTL calls to help perform asynchronous (nonblocking) socket operations.

The IOCTL call has many functions; establishing blocking mode is only one of its functions. The value in COMMAND determines which function IOCTL will perform. The REQARG of 0 specifies non-blocking (a REQARG of 1 would request that socket S be set to blocking mode). When this socket is passed as a parameter to a call that would block (such as RECV when data is not present), the call returns with an error code in RETCODE, and ERRNO set to EWOULDBLOCK. Setting the mode of the socket to nonblocking allows an application to continue processing without becoming blocked.

GIVESOCKET and **TAKESOCKET** calls

The GIVESOCKET and TAKESOCKET functions are not supported with the IMS TCP/IP OTMA Connection server. Tasks use the GIVESOCKET and TAKESOCKET functions to pass sockets from parent to child.

For programs using TCP/IP for MVS, each task has its own unique 8-byte name. The main server task passes three arguments to the GIVESOCKET call:

- The socket number it wants to give
- Its own name ⁵
- The name of the task to which it wants to give the socket

If the server does not know the name of the subtask that will receive the socket, it blanks out the name of the subtask. ⁶ The first subtask calling TAKESOCKET with the server's unique name receives the socket.

⁵ If a task does not know its address space name, it can use the GETCLIENTID function call to determine its unique name.

⁶ This is the case in IMS because the Listener has no way of knowing which Message Processing Region will inherit the socket.

The subtask that receives the socket must know the main task's unique name and the number of the socket that it is to receive. This information must be passed from main task to subtask in a work area that is common to both tasks.

- In IMS, the parent task name and the number of the socket descriptor are passed from parent (Listener) to child (MPP) through the message queue.
- In CICS, the parent task name and the socket descriptor number are passed from the parent (Listener) to the transaction program by means of ` the EXEC CICS START and EXEC CICS RETREIVE function.

Because each task has its own socket table, the socket descriptor obtained by the main task is not the socket descriptor that the subtask will use. When TAKESOCKET accepts the socket that has been given, the TAKESOCKET call assigns a new socket number for the subtask to use. This new socket number represents the same connection as the parent's socket. (The transferred socket might be referred to as socket number 54 by the parent task and as socket number 3 by the subtask; however, both socket descriptors represent the same connection.)

Once the socket has successfully been transferred, the TCP/IP address space posts an exceptional condition on the parent's socket. The parent uses the SELECT call to test for this condition. When the parent task SELECT call returns with the exception condition on that socket (indicating that the socket has been successfully passed) the parent issues CLOSE to complete the transfer and deallocate the socket from the main task.

To continue the sequence, when another client request comes in, the concurrent server (Listener) gets another new socket, passes the new socket to the new subtask, and dissociates itself from that connection. And so on.

Summary of passing the socket process

The process of passing the socket is accomplished in the following way:

- After creating a subtask, the server main task issues the GIVESOCKET call to pass the socket to the subtask. If the subtask's address space name and subtask ID are specified in the GIVESOCKET call, (as with CICS) only a subtask with a matching address space and subtask ID can take the socket. If this field is set to blanks, (as with IMS) any MVS address space requesting a socket can take this socket.
- The server main task then passes the socket descriptor and concurrent server's ID to the subtask using some form of commonly addressable technique such as the IMS Message Queue.
- The concurrent server issues the SELECT call to determine when the GIVESOCKET has successfully completed.
- The subtask calls TAKESOCKET with the concurrent server's ID and socket descriptor and uses the resulting socket descriptor for communication with the client.
- When the GIVESOCKET has successfully completed, the concurrent server issues the CLOSE call to complete the handoff.

An example of a concurrent server is the IMS Listener. It is described in <u>Chapter 6</u>, "How to customize and operate the IMS Listener," on page 45. Figure 5 on page 9 shows a concurrent server.

What you need to run IMS TCP/IP

IMS TCP/IP using the IMS Listener and IMS Assist Module is designed for use on an MVS/SP host system running IMS/ESA Version 4 or later.

A TCP/IP host can communicate with any remote IMS or non-IMS system that runs TCP/IP. The remote system can, for example, run a UNIX or OS/2 operating system.

TCP/IP services is not described in this information because it is a prerequisite for IMS TCP/IP. However, much material from the TCP/IP library has been repeated in this information in an attempt to make it independent of that library.

A summary of what IMS TCP/IP provides

Figure 7 on page 17 shows how IMS TCP/IP allows IMS applications to access the TCP/IP network. It shows that IMS TCP/IP makes the following facilities available to your application programs:

The sockets calls (1 and 2 in Figure 7 on page 17)

The socket API is available both in the C language and in COBOL, PL/I, or assembly language. <u>Table 3 on</u> page 17 shows the socket calls included in the socket API.

Table 3. Socket calls				
Call type	Calls			
Basic	socket, bind, connect, listen, accept, shutdown, close			
Read/write	send, sendto, recvfrom, read, write			
Advanced	gethostname, gethostbyaddr, gethostbyname, getpeername, getsockname, getsockopt, setsockopt, fcntl, ioctl, select			
IBM-specific	initapi, getclientid, givesocket, takesocket			



Figure 7. How user applications access TCP/IP networks with IMS TCP/IP

IMS TCP/IP provides for both connection-oriented and connectionless (datagram) services, using the TCP and UDP protocols respectively. TCP/IP does not support the IP (raw socket) protocol.

The Listener (4) in Figure 7 on page 17.

IMS TCP/IP includes a concurrent server application, called the Listener, to which the client makes initial connection requests. The Listener passes the connection request on to the user-written server, which is typically an IMS Message Processing Program.

Conversion routines (5) in Figure 7 on page 17.

IMS TCP/IP provides the following conversion routines, which are part of the base TCP/IP Services product:

- An EBCDIC-to-ASCII conversion routine, used to convert EBCDIC data to the ASCII format that is used in TCP/IP networks and workstations. The conversion routine is run by calling the EBCDIC-to-ASCII translation table EZACIC04, shown in the z/OS Communications Server: IP Configuration Reference.
- A corresponding ASCII-to-EBCDIC conversion routine (EZACIC05), shown in the z/OS Communications Server: IP Configuration Reference.
- An alternative EBCDIC-to-ASCII conversion routine (EZACIC14).
- Corresponding ASCII-to-EBCDIC conversion routine (EZACIC15).
- A module that converts COBOL character arrays into bit-mask arrays used in TCP/IP. This module, which is run by calling EZACIC06, is used with the socket SELECTSELECT call.
- A module that interprets a C language structure known as Hostent (EZACIC08).

Chapter 3. Principles of operation of the Listener and the Assist module

This information describes the operation of the Listener and the Assist module. Its purpose is to explain how a TCP/IP-to-IMS connection is established, and how the client and server exchange application data. For specific data formats and the socket protocols used when coding a TCP/IP client or server, see Chapter 4, "How to write an IMS TCP/IP client program," on page 29 and Chapter 5, "How to write an IMS TCP/IP server program," on page 37.

Overview of the Listener and the Assist module

The IMS TCP/IP feature consists of 3 components: the IMS Listener, the IMS Assist module, and the Sockets Extended API. ⁷ The Sockets Extended API can either be used independently, or with the other 2 components. When the Sockets Extended interface is used independently, an IMS MPP can either serve as a client or as a server.

When the IMS Listener is used, the IMS MPP acts as a server, and the TCP/IP remote acts as the client. The Assist module is dependent upon the IMS Listener; therefore, when the Assist module is used, IMS is the server.

Because the Listener and the Assist module are designed to support IMS as a server, this information is based on that assumption. For a discussion of IMS as **client**, see <u>"When the client is an IMS MPP" on page</u> 27, and the sample program on <u>"Sample program - IMS MPP client" on page 253</u>.

The role of the IMS Listener

Because the IMS Transaction Manager does not support direct connection with TCP/IP, some other program must establish that connection. When IMS is acting as a *server* to a TCP/IP-connected *client*, that program is the IMS Listener — an IMS batch message program (BMP) whose main function it is to establish connection between the client and the requested IMS transaction.

When the client requests the services of an IMS message processing program (MPP), it sends a message to the IMS host containing the transaction code of that MPP. The IMS Listener receives that request and schedules the requested MPP; it then holds the connection until the MPP starts and accepts the connection. Once the MPP owns the connection, the Listener is no longer involved with it.

The role of the IMS Assist module

The IMS Assist module is a subroutine, called from an IMS MPP (server) that translates conventional IMS communication calls into the corresponding socket calls. Its use is optional. Its purpose is to shield the programmer from having to understand TCP/IP programming. To exchange data with the client, the server program issues traditional IMS message queue calls (GU, GN, ISRT). These calls are intercepted by the Assist module, which issues the appropriate socket calls.

Pros and cons for the use of the IMS Assist module

The Assist module makes message processing program (MPP) coding easier, but is accompanied by a series of trade-offs. This information discusses the trade-offs between implicit mode and explicit mode.

• Implicit-mode application programmers use conventional IMS Transaction Manager (TM) calls and require no special training; explicit-mode application programmers must understand TCP/IP socket calls and protocols.

© Copyright IBM Corp. 2000, 2021

⁷ Shipped with the TCP/IP Services base product.

- Implicit-mode transactions must adhere to constraints imposed by the IMS Assist module. By contrast, explicit-mode transactions use the TCP/IP socket call interface and have no specific protocol requirements other than the orderly initiation and termination of the transaction.
- Implicit-mode transactions obtain their message input from the IMS message queue. Because the Listener must put the input message segments on the queue before the server begins execution, the client sends all application data with the transaction request. Explicit-mode transactions bypass the message queue for all application data both input, and output.
- Implicit-mode transactions are limited to a single GU-GN/ISRT iteration (one input of one or more segments, followed by one output of one or more segments) for each message retrieved from the IMS message queue. By contrast, explicit-mode transactions have no such limit. Unlimited read/write sequences make it possible to design conversations in which the two programs talk back and forth without limit. ⁸

Client/server logic flow

This information describes the flow of a client/server application through the system — starting with the client and continuing on through the Listener to the server. The complete transaction, including initiation, execution, and termination is traced.

How the connection is established

The following paragraphs describe the functions the Listener performs in coordinating between the client and the server. With the exception of paragraph 6, the Listener performs the same steps for both explicitand implicit-mode servers. Paragraph numbers correspond to the step numbers in Figure 8.

⁸ Because of the potential for long running conversations, MPPs with multiple conversational iterations should be carefully designed to avoid the possibility of extended message processing region occupancy.



Figure 8. IMS TCP/IP message flow for transaction initiation

1. Connection request

The IMS Listener is an IMS batch message processing program (BMP). When the Listener starts, it establishes a socket on which it can "listen" for connection requests. It binds itself to the specified port, and then listens for requests from TCP/IP clients. When a client sends a connection request, MVS TCP/IP notifies the Listener of the request.

2. Connection processing

When the Listener receives a connection request, it issues a socket ACCEPT call, which creates a new socket specifically for that connection.

3. Transaction-Request Message

The client then sends a transaction-request message (TRM) segment, which includes the 8-byte name of the requested IMS server transaction (otherwise known as the TRANCODE).

4. Transaction verification

The Listener performs several tests to ensure that the requested transaction should be accepted:

- The TRANCODE is tested against IMS Listener configuration file TRANSACTION statements to ensure that the requested transaction is eligible to be run from a TCP/IP client.
- If security data is included in the transaction-request message (TRM), that data is passed to a user-written security exit. The purpose of this exit is to validate the credentials of the client before allowing the transaction to be scheduled.
- The Listener issues an IMS CHNG call to a modifiable alternate PCB, specifying the TRANCODE of the required transaction. It then issues an IMS INQY call to ensure that the transaction is not stopped (due to previous abend or Master Terminal Operator action).

The following actions depend on the results of the verification:

- If the transaction request is *rejected*, the IMS Listener returns a request-status message (RSM) segment to the client with an indication of the reason for rejecting the request; it then closes the connection.
- If the transaction request is *accepted* the requested transaction is scheduled (the Listener *does not* return a status message to the client).
- 5. Transaction Initiation Message (TIM)

The Listener then inserts (ISRT) a transaction initiation message (TIM) segment to the IMS message queue. This message contains information needed by the server program when it takes responsibility for the connection.

Note: The client sends the transaction *request* message (TRM) to the Listener. The Listener sends the transaction *initiation* message (TIM) to the server.

6. Client-to-server input data transfer (implicit mode only)

If the transaction is in implicit mode, the Listener reads the client-to-server input data and places it on the message queue.

7. Pass the socket to the server

Next, the Listener issues a GIVESOCKET call, which makes the socket available to the server program.

8. Schedule the transaction

Finally, the Listener issues an IMS SYNC call to schedule the requested IMS transaction and waits for the server program to take responsibility for the connection.

When the server issues a TAKESOCKET call, the Listener has completed its responsibility for the socket and dissociates itself from the connection.

Note: The Listener is a never-ending IMS Batch Message Program, which processes multiple concurrent transactions.

How the server exchanges data with the client

Once the server begins execution, the protocol to pass input data to the server is a function of whether the transaction mode is explicit or implicit.

Explicit-mode transactions

The following information describes an explicit-mode server program which exchanges application data with a client.

Step numbers in Figure 9 correspond to the paragraph numbers below.



Figure 9. IMS TCP/IP message flow for explicit-mode input/output

- 1. Once an explicit-mode server begins execution, it issues an IMS GU call to obtain the transaction initiation message (TIM) segment, an INITAPI to establish connection with MVS TCP/IP, and a TAKESOCKET call to establish direct connection between client and server.
- 2. Subsequently, socket READ and WRITE commands are used to exchange data between client and server. The conversation can consist of any number of database calls and socket READ/WRITE exchanges. ⁹ Client data is not passed through the IMS message queue and is not subject to any predefined protocols.
- 3. The transaction indicates completion by issuing another GU to the I/O PCB. This notifies the Transaction Manager that the database changes should be committed. At this point, the server

⁹ Because of the potential for long running conversations, MPPs with multiple conversational iterations should be carefully designed to avoid the possibility of extended message processing region occupancy.

program might send a message to the client indicating that the database changes have been successfully completed.

If another message awaits this transaction, the GU will cause the first segment of that message to be retrieved and the program should issue a new TAKESOCKET call to start the process again.

4. When the GU call returns with a QC status code, the server ends the conversation by closing the socket.

Implicit-mode transactions

The following information describes how the Assist module and the server program interact to exchange application data with the client. The paragraph numbers correspond to the step numbers in Figure 3.



Figure 10. IMS TCP/IP message flow for implicit mode input/output

1. Server GU

GU must be the first IMS call issued by the server to the I/O PCB. The Assist module retrieves the first segment from the message queue and examines it (for *LISTNR* in the first field) to determine whether it is a transaction initiation message. (If the message was not sent by the Listener, the Assist module assumes the transaction was started by an SNA terminal and immediately passes the input segment to the server. In this case, subsequent I/O PCB calls (as well as database calls) are passed directly through to IMS without further consideration.)

2. Transaction Initiation Message (TIM)

If the message was sent by the Listener, the initial message segment is the transaction initiation message (TIM); the Assist module *does not* return it to the server. Instead, the Assist module uses the TIM contents to issue the TAKESOCKET to establish connection between the client and the server program.

3. Server input data

After the server owns the socket, the Assist module issues a GN to retrieve the first segment of the client input message and returns it to the server program. Thus, the server program never sees the TIM; it receives the first data segment in response to its GU. Subsequent GN calls from the server cause the Assist module to retrieve the remaining segments of the message. When the Assist module reads the last input segment for that transaction from the message queue, it receives a QD status code from IMS, which it returns to the server program.

After the initial GU to the I/O PCB, server GN calls, ISRT calls, and database calls can be intermixed.

4. Server output data

When the server program issues ISRT calls to send output message segments to the client, the IMS Assist module accumulates the output segments, up to maximum of 32KB, into a buffer.

5. Commit

The server signals completion by issuing a GU to the I/O PCB.

6. TCP/IP writes application data to the client.

When the server issues the GU, the Assist module issues WRITE calls to send the data to the client and passes the GU to the IMS Transaction Manager to commit the database changes.

7. Confirmation

If the GU is successful, (that is, QC status or spaces) the Assist module sends a complete-status message segment (CSM) to the client to confirm the successful commit and passes the status code back to the server.

8. Close the socket

After the complete-status message has been sent to the client, the Assist module closes the socket, ending the connection.

If the GU in the previous step resulted in a 'bb' status code (indicating successful return of another message) the program logic returns to step 2 to process the new message.

How the IMS Listener manages multiple connection requests

The IMS Listener uses two queues for the management of connection requests:

- 1. The *backlog* queue (managed by MVS TCP/IP) contains client connection requests that have not yet been accepted by the Listener. If a client requests a connection while the backlog queue is full, TCP/IP rejects the connection request. The number of requests allowed in the backlog queue is specified in the LISTENER startup configuration statement (BACKLOG parameter), see <u>"LISTENER statement" on page 46</u>.
- 2. The *active sockets* queue contains the sockets that are held by the Listener while they wait for assignment to a server program. After the Listener has accepted the connection, the connection belongs to the Listener until it is accepted by the server. If the Listener uses all of its sockets and cannot accept any more connections, subsequent requests go into the backlog queue. The

maximum number of sockets available is specified in the LISTENER startup configuration statement, (MAXACTSKT parameter), see "LISTENER statement" on page 46.

Tip: The backlog value specified on the listen call cannot be larger than the value configured by the SOMAXCONN statement in the stack's TCPIP PROFILE (the default value is 10). No error is returned if a larger backlog is requested. If you want a larger backlog, update the SOMAXCONN statement. See the z/OS Communications Server: IP Configuration Reference for details.

Use of the IMS message queue

In conventional 3270 applications, the IMS message queue is a mechanism for passing communications between an MPP and another entity, such as a 3270-type terminal, or another message processing program (MPP). The IMS TCP/IP feature uses the message queue for communication between the Listener and the MPP. Messages from and to TCP/IP hosts bypass IMS message format services (MFS). The following information describes how IMS TCP/IP uses the IMS message queue:

Input messages

(Messages that are *input* to the MPP)

- Explicit-mode transactions use only the message queue to pass the transaction initiation message (TIM) from the Listener to the server. All application data sent by the client is received by the server using sockets READ calls, thus bypassing the IMS message queue.
- Implicit-mode transactions use the message queue both for the TIM (which is trapped by the Assist module and not passed on to the server) and for all client-to-server application data (which is passed to the server in response to IMS GU, GN calls).

Output messages

All messages that are *output* from the server go directly via TCP/IP to the client; they do not pass through the message queue.

- Explicit-mode servers use socket WRITE calls to send application data directly to the client.
- Implicit-mode servers use the IMS ISRT call for output, but the inserted data is trapped by the Assist module which, in turn, issues socket WRITE calls to send the data to the client.

Call sequence for the IMS Listener

Although you will probably not be writing a Listener program, it is important that you match the sequence of calls issued by the Listener when you write your client program. The Listener call sequence is:

INITIALIZE LISTENER

INITAPI

Connect the Listener to MVS TCP/IP at Listener startup. (This call is used only in programs written to the Sockets Extended interface.

SOCKET

Create a socket descriptor.

BIND

Allocate the local port for the socket. This port is used by clients when requesting connection to IMS.

LISTEN

Create a queue for incoming connections.

WAIT FOR CONNECTION REQUEST

SELECT

Wait for an incoming connection request.

ACCEPT

Accept the incoming connection request; create a new socket descriptor to be used by the server for this specific connection.

READ

Read TRM; determine the IMS TRANCODE.

CHNG

Change the modifiable alternate PCB to reflect the desired IMS TRANCODE.

INQY

Ensure the desired IMS TRANCODE is available for scheduling.

ISRT

Use the alternate PCB to insert the transaction initiation message (TIM) and pass control information and user input data to the server.

GIVESOCKET

Pass the newly created socket to the server.

SYNC

Schedule the requested transaction.

SELECT

Wait for the server to take the socket.

CLOSE

Release the socket.

END OF CONNECTION REQUEST

Return to "WAIT FOR CONNECTION REQUEST"

SHUTDOWN LISTENER

CLOSE

Close the socket through which the Listener receives connection requests from MVS TCP/IP.

TERMAPI

Disconnect the Listener from MVS TCP/IP before shutting down.

Application design considerations

The following information is a set of guidelines and limitations that should be considered when you are designing IMS TCP/IP applications.

Programs that are not started by the IMS Listener

In most cases, IMS server applications are started by the IMS Listener. Such programs are known as *dependent* programs because the Listener establishes the TCP/IP connection.

Under some circumstances, application design considerations require an application to establish its own connection between TCP/IP and IMS. For example, an IMS message processing program (MPP) might require the services of a UNIX processor that is connected through TCP/IP. An IMS application of this type is known as an *independent* program because it is not started by the Listener. Because independent programs do not use Listener services, they must define their own protocol.

When the client is an IMS MPP

For this example, the underlying assumption is that the TCP/IP host acts as client and the IMS MPP acts as server; however, this is not always the case.

Consider an IMS MPP that requires the services of an AIX[®] host that is connected through TCP/IP. Such an MPP (acting as a client) initiates a TCP/IP conversation by issuing the *client* TCP/IP call sequence. The TCP/IP host would respond with the *server* TCP/IP call sequence. This application design is supported because the MPP communicates directly with MVS TCP/IP. The IMS TCP/IP feature does not impose any unique restrictions on the type and usage of socket calls issued by such a program; however, because of the unique and unstructured communication requirements of this application design, you must use explicit mode for this type of program.

Abend processing

When a task that owns a socket fails, MVS TCP/IP closes the socket. Therefore, when an IMS MPP abnormally ends as a the result of an error condition, regardless of the reason, the socket is no longer available and communication between the server and the client is no longer possible.

True abends

If an IMS TCP/IP server program abnormally ends (for example, because of an SOCx condition), database changes in progress are backed out and the transaction task is terminated, which breaks the TCP/IP connection. When the connection is broken, the client receives a negative status code and an error number that indicates that the connection has been broken. Upon receipt of this indication, the client should assume that the transaction did not complete and that the database changes have not been made. The client could reschedule the transaction, but the IMS TM will have probably stopped it from further running as a result of the abnormal end.

The solution is to correct the reason for the abnormal end and restart the transaction.

Pseudo abends

Under certain situations IMS applications cannot complete. When such a condition occurs, IMS abnormally ends the MPR with a status code (such as U0777) and reschedules it. This action is not apparent to the conventional 3270-type user.

However, when an IMS TCP/IP transaction is abnormally ended, the action is apparent to the client because the connection between client and server is lost when the server MPR is abnormally ended. In this case, IMS TM reschedules the transaction and places the input message (including the TIM) back on the message queue. When the transaction is rescheduled and issues a GU for the TIM, the socket described in the TIM no longer represents a valid connection. and the associated TAKESOCKET call fails. At this time, the Assist module detects the failure of the socket call and returns a ZZ status code to the server program. Upon receipt of this status code, the server program should end normally.

Note: At the time of the abnormal end, the IMS TM backs out database changes, so the client should restart the transaction.

Guideline: For deadlock situations you should define the transaction as INIT STATUS GROUP B, which allows the application program to regain control after a deadlock with a BC status code (instead of terminating with a U0777 abend). The server program can regain control after the deadlock and notify the client while the connection is still available.

Implicit-mode support for ROLB processing

If a server program issues the IMS ROLB call, all database changes are reversed, and all output messages are erased from the IMS message queue. However, the client is not automatically notified of this action and will (when the transaction completes normally) receive a CSMOKY message, indicating normal completion.

As a result, for transactions that conditionally issue the ROLB call, the server should send a message to the client indicating whether the ROLB command was issued. Otherwise, the client might incorrectly interpret the CSMOKY message to mean that database changes have been made (when in fact, the message simply denotes successful termination of the transaction).

Restrictions for operation of the Listener and the Assist module

- Transactions must be defined as MODE=SNGL in the IMS TRANSACT macro; this ensures that the database buffers are emptied (flushed) to direct access storage when the second and subsequent GU calls are issued.
- Transactions must not reference other systems (MSC is not supported).
- Transactions must not be conversational [that is, they must not use the IMS Scratch Pad Area (SPA)].

Chapter 4. How to write an IMS TCP/IP client program

When writing an IMS TCP/IP client program, the programmer must follow conventions established by the IMS Listener and by the IMS Assist module (if used). This information describes the call sequences and input/output data formats to be used by the client program. For server programming, see <u>Chapter 5</u>, "How to write an IMS TCP/IP server program," on page 37.

In this information, a "client" is typically a TCP/IP host that is requesting the services of an IMS message processing program (MPP). This is considered to be the normal case. However, in some situations, an MPP can start as a server and then (because it needs the services of another program) switch roles from server to client.

In this information, the client will be assumed to be the TCP/IP host and the server, the IMS MPP.

General client program logic flow

For both explicit- and implicit-mode clients the logic flow is essentially the same:

The client initiates the request for a specific IMS MPP server by communicating with MVS TCP/IP, which passes the request on to the IMS Listener. The Listener schedules the transaction and the client then exchanges application data with the server. When the transaction is complete, the connection is closed; each client request for an IMS transaction requires a new TCP/IP connection.

The following topics provide more details about the programming requirements for explicit-mode and implicit-mode clients, respectively.

Explicit-mode client program logic flow

When the client requests the services of an explicit-mode server, the only protocol imposed by IMS TCP/IP is that the client must begin by establishing TCP/IP connectivity and sending a transaction-request message (TRM).

The Listener uses contents of the transaction-request message (TRM) to determine which transaction to schedule. If the request is not accepted (for example, because of failure to pass the security exit, or because the transaction was stopped by the IMS master terminal operator), the Listener returns a request-status message (RSM) to the client with an indication of the cause of failure. (See <u>"Request-status message segment" on page 34</u> for the format of the request-status message).

Once an explicit-mode client and server are in communication, there is no predefined input/output protocol. Rules of the conversation are established by agreement between the two programs. Any number of READ/WRITE calls can be issued. Upon termination, the server program should commit any database changes, notify the server of successful completion, and close the socket.

It is suggested that, when all database updates have been committed, the server notify the client by sending a "success" message to the client. This notifies the client that the transaction has completed properly and that all database updates have been committed. Unless such a message is sent, the client has no way of knowing that the transaction completed properly.

Explicit-mode client call sequence

The call sequence to be used by an explicit-mode client program is:

Call

Explanation of Function

INITAPI

Open the interface. (Required only for client programs that use MVS TCP/IP socket calls).

SOCKET

Obtain a socket descriptor.

CONNECT

Request connection to the IMS Listener port.

WRITE

Send a transaction-request message (TRM).

READ

Test for successful transaction initiation. ¹⁰

WRITE/READ

Explicit-mode transactions can issue any number of READ or WRITE socket call sequences.

READ

Ensure that the server ended normally and that the database changes are committed.

CLOSE

Terminate the connection and release socket resources.

Explicit-mode application data

The following information describes explicit-mode application data.

Format

Explicit-mode clients must initiate the connection with the server by sending the transaction-request message (TRM) to the IMS host. The format of this message is defined later in this topic. Explicit-mode application data is formatted according to agreement between client and server. Explicit-mode imposes no application data format requirements.

Data translation

In explicit-mode, application data translation from ASCII to EBCDIC (if necessary) is the responsibility of the client and server programs. Data is not translated by the IMS TCP/IP feature.

Network byte order

Fixed-point binary integers (used for segment lengths in TRM and RSM) are specified using the TCP/IP network byte ordering convention (big-endian notation). This means that if the high-order byte is stored at address n, the low-order byte is stored at address n+1. (Little-endian notation stores the other way around).

MVS also uses the big-endian convention. Because this is the same as the network convention, IMS TCP/IP MPP's should not need to convert data from little-endian to big-endian notation. If the client uses little-endian notation, it is responsible for the conversion.

End-of-message indicator

IMS TCP/IP does not define an End-of-message indicator for explicit-mode messages.

Implicit-mode client logic flow

When the client requests the services of an implicit-mode client, the protocol is predefined by IMS TCP/IP.

¹⁰ If the Listener is unable to initiate the transaction, it sends a request-status message (RSM) to the client indicating the reason for failure. Therefore, the client must be prepared to receive that message. It is suggested that a convention be established that the server initiate the conversation by sending an opening message. By following this convention, the client will receive either positive or negative notification of transaction status before initiating application data exchange.

The client requests an IMS MPP by sending the transaction-request message (TRM). (See <u>"Transaction-request message segment (client to Listener)</u>" on page 33 for the format of the TRM.) The TRM includes the name of the transaction the Listener is to schedule.

If the transaction cannot be scheduled (for example, because of failure to pass the security exit, or because the transaction was stopped by the IMS master terminal operator), the Listener returns the request-status message with an indication of the cause of failure. (See <u>"Request-status message</u> segment" on page 34 for the format of the request-status message).

For implicit-mode applications, the input data stream consists of the TRM, immediately followed by all segments of application data and an end-of message-segment. The Listener uses the TRM contents to schedule the server and then places the TIM and all of the application data on the IMS message queue for retrieval by the Assist module.

Implicit-mode transactions are limited to one multisegment input message and one multisegment output message. In other words, implicit-mode applications cannot enter into conversations.

When the transaction is complete, the IMS Assist module sends a complete-status message (CSMOKY) segment to the client. If the client receives this message, the client can safely assume that the database changes have been committed. If the client doesn't receive this message, the client cannot determine what has happened. The transaction might have completed normally and database changes committed, or the transaction might have failed with database changes backed out. For this reason, clients that work with implicit mode servers should include application logic that, upon failure to receive the CSMOKY message segment, reestablishes contact with IMS and confirms the success of the previously submitted update.

Implicit-mode client call sequence

The call sequence to be used by an implicit-mode client program is:

Call

Explanation of Function

INITAPI

Open the interface. (Required only for client programs that use MVS TCP/IP Sockets calls).

SOCKET

Obtain a socket descriptor.

CONNECT

Request connection to the IMS Listener port.

WRITE

Send a transaction-request message (TRM).

WRITE

Send server input data formatted as IMS segments.

READ

Receive response.

- If the request was rejected, a request-status message (RSM) will be received.
- If the transaction was scheduled and executed properly, application data will be received.

Thus, logic in the client must test the output message for the characters *REQSTS* to distinguish between application data and a request-status message (RSM).

READ

Upon successful completion of the database updates, the Assist module sends a complete-status message (*CSMOKY*) to the client, indicating that the transaction has completed successfully.

If this message is not received, the client must assume that the application failed to complete properly; in this case, a return code of -1 and ERRNO (typically set to 54) will indicate that application failed. The client must take whatever action is appropriate (for example, reschedule the transaction, resynchronize data).

CLOSE

Terminate the connection and release the socket resources.

Implicit-mode application data stream

The following information describes the types of implicit-mode application data streams.

Client-to-server data stream

In implicit mode, the client sends the following data stream:

llzz transaction-request message (TRM) *llzz* application data segment 1 *llzz* application data segment 2 (optional) *llzz* ... *llzz* application data segment n (optional) *04zz* end-of-message segment

WHERE:

ll

The length in bytes of this data segment in binary.

zz

Reserved; must be set to binary.

transaction-request message (TRM)

The initial client request.

application data segment 1 – n

Data to be passed to the server application.

end of message segment

A segment with no data. Therefore, its segment length is 04 (2 for the length field plus 2 for the reserved field.)

Server-to-client data stream

Data received by the client is formatted (by the Assist module) as above. It consists of n segments of application data including the CSM segment, followed by an end-of-message segment.

Implicit-mode application data

The following information describes implicit-mode application data.

Format

Data exchanged between implicit-mode client and server is transmitted in a format that resembles an IMS message segment. These segments have the following format: ¹¹

Field	Format	Description
Length	Н	Length of the data segment (including this field)
Reserved (zz)	CL2	Reserved field
Data	CLn	Client-supplied data

The length field contains the total length of the message in binary. The length (ll) includes the length of the ll and zz fields.

Data translation

The IMS Listener tests the initial input data string (the TRM) to determine whether the terminal is transmitting in ASCII. If the terminal is transmitting in ASCII, and the transaction is defined as *implicit*-

¹¹ This example uses Assembly language notation. See Chapter 7, "CALL instruction application programming interface," on page 51 for COBOL and PL/I equivalents.

mode in the TRANSACTION configuration statement, the Listener translates the ASCII application data into EBCDIC.

Note: When data translation takes place, the entire application data portion of the segment is translated from ASCII to EBCDIC, and vice versa; therefore, the segment should contain only printable characters that are common to both character sets. (For example, the EBCDIC cent sign and the ASCII left square bracket are both printable in their respective native environments, but they are not translated because they do not have an equivalent in the other character set.)

End-of-message segment

The last segment in a message (either sent by the client, or received from the server) is indicated by an end-of-message (EOM) segment. (See "End-of-message segment (EOM)" on page 35).

- Implicit-mode messages sent by the client are received by the Listener. When the client program sends an EOM segment, the Listener interprets the EOM as an indication that no more message segments are to be received and inserts the segments onto the IMS message queue.
- Implicit-mode messages received by the client are actually written by the Assist module on behalf of the server program. When the server program sends application data to the client (using the ISRT call), the Assist module intercepts the output data and accumulates it in an output buffer. When the server program issues a subsequent GU to the I/O PCB, the Assist module interprets the GU as an indication that the server has inserted the last segment for that message. The Assist module then adds an end-of-message segment to the output data and issues WRITE commands, which transmit the data to the client. (The client program should test for the EOM segment to determine when the last segment of the message has been sent by the server program.)

IMS TCP/IP message segment formats

The client sends or receives several types of message segments whose formats are defined by the Listener and the Assist module.

- Transaction-request message segment (TRM)
- Request-status message segment (RSM)
- Complete-status message segment (CSMOKY)
- End-of-message segment (EOM)

The following paragraphs describe the formats for each of these segments:

Transaction-request message segment (client to Listener)

To initiate a connection with an IMS server, the client first issues a transaction-request message segment (TRM), which tells the Listener which transaction to schedule.

The format of the transaction-request message segment (TRM) is:

Field	Format	Meaning
TRMLen	Н	Length of the segment (in binary) including this field. This field is sent in network byte order.
TRMRsv	CL2	Reserved
TRMId	CL8	Identifying string. Always *TRNREQ*. If the client data stream will be sent in ASCII, the TRMId field should also be transmitted in ASCII because the Listener uses this field to determine whether ASCII to EBCDIC translation is required.

Field	Format	Meaning
TRMTrnCod	CL8	The transaction code (TRANCODE) of the IMS transaction to be started. It must not begin with a / character; it must follow the naming rules for IMS transactions. If the Listener has determined that data will be transmitted in ASCII, it translates the transaction code to EBCDIC before any further processing is done.
TRMUsrDat	XLn	This variable-length field contains client data that is passed directly to the security exit without translation.

Request-status message segment

If a transaction request is accepted, the IMS Listener does not send the request-status message segment; if the transaction request is rejected, the IMS Listener sends a request-status message segment (RSM) to the client. This segment has the following format:

Field	Format	Description
RSMLen	Н	Length of message (in binary), including this field.
RSMRsv	CL2	Reserved
RSMId	CL8	Identifying string. Always *REQSTS*. This field is translated to ASCII if the Listener has determined that the client is transmitting in ASCII.
F	Return code, sent in network byte order. Set to nonzero (for example, 4, 8, 12) to indicate an error. The nonzero value is further explained by the reason code (RSMRsnCod).	
RSMRsnCod	F	Reason Code, sent in network byte order. Reason codes 0 — 100 are reserved for use by the IMS Listener. Codes greater than 100 can be assigned by the user-written security exit.

Request-status message reason codes

If the IMS Listener sends a request-status message (RSM) segment to the client (indicating that it is unable to complete the processing of the client's transaction-request message (TRM), it sets the return and reason code in the RSM.

- If the security exit rejects a transaction request, it sets the return code and reason code, and returns control to the Listener, which sends the request-status message segment to the client.
- If the Listener detects other errors that cause a request to be rejected, it sets a return code of 8 and a reason code from the following list.

1

The transaction was not defined to the IMS Listener.

2

An IMS error occurred and the transaction was unable to be started.

3

The transaction failed to perform the TAKESOCKET call within the 3 minute time frame.

4

The input buffer is full as the client has sent more than 32KB of data for an implicit transaction.

5

An AIB error occurred when the IMS Listener tried to confirm if the transaction was available to be started.

6

The transaction is not defined to IMS or is unavailable to be started.

7

The transaction-request message (TRM) segment was not in the correct format.

9

The application data buffer for the Client-to-Server Data Stream contains an invalid value for the data segment length.

100 up

Reason codes of 100 or higher are defined by the user-supplied security exit.

Complete-status message segment

The complete-status message segment is sent by the Assist module to indicate the successful completion of an implicit-mode transaction, including the fact that database updates have been committed. The format of the complete-status message segment is:

Field	Format	Description
Length	Н	Length of the data segment (in binary) including this field
CSMRsv	Н	Reserved field; must be set to zero
CSMId	CL8	*CSMOKY* This field is translated to ASCII if the client is transmitting in ASCII.

End-of-message segment (EOM)

The end-of-message segment is defined as an IMS-type segment (with *llzz* fields) but no application data. Thus, the EOM segment has an *llzz* field of '0400'; 04 is the length of the *llzz* field.

PL/I coding

PL/I programmers should note that (although the segments exchanged between the Listener and implicitmode servers resemble IMS segments) the segments are actually sent by TCP/IP socket calls and do not necessarily follow the standard IMS convention for the PL/I language interface. Specifically, the length field in a segment (TRM or RSM), which is passed via a TCP/IP socket call, *must* be a halfword (FIXED BIN(15)) and not a fullword.

36 z/OS Communications Server: z/OS V2R5.0 Communications Server: IP IMS Sockets Guide

Chapter 5. How to write an IMS TCP/IP server program

When writing an IMS TCP/IP server program, the programmer must follow conventions established by the IMS Listener; by the IMS Assist module (if the server program uses it); and by the TCP/IP client. This topic describes the call sequences and input/output formats necessary for communication between a TCP/IP client program and an IMS server program. (See <u>Chapter 4</u>, "How to write an IMS TCP/IP client program," on page 29 for a discussion of client programming).

General server program logic flow

An IMS TCP/IP server program is executed in response to a transaction request from a TCP/IP host. The server program can either explicitly issue TCP/IP socket calls, or implicitly issue them through the IMS Assist module. However, the same TCP/IP functions are completed in either case.

The following topics describe the server logic flow for each mode.

Explicit-mode server program logic flow

When an explicit-mode server begins execution, the Listener has received the transaction-request message (TRM) from the client and has inserted the transaction-initiation message (TIM) to the IMS message queue. The Listener has also issued a GIVESOCKET call to pass the connection to the server.

The server's first action is to obtain the TIM from the IMS message queue. This message contains the information needed to issue the INITAPI and TAKESOCKET calls.

Once the server has issued the TAKESOCKET call, the connection is between client and server; the two can now communicate directly using socket READ/WRITE calls. The number of reads/writes, and the format of the data exchanged, is determined by agreement between the two programs.

At the end of processing a client's request, the application program should follow the IMS DC programming standard of issuing another GU to the IO/PCB. This informs IMS that the database changes should be committed, and that the database buffers should be emptied (flushed).

Note: For this reason, a transaction invoked by a TCP/IP client should be defined (by the IMS-gen TRANSACT macro) as MODE=SNGL.

Explicit-mode call sequence

The suggested call sequence for an explicit-mode server follows. See <u>Chapter 7</u>, "CALL instruction application programming interface," on page 51 for the call syntax of the socket calls.

Server call

Explanation of Function

CALL CBLTDLI (GU) I/O PCB

Obtain transaction-initiation message (TIM) from IMS message queue.

INITAPI

Initialize the connection with TCP/IP.

Parameter

Meaning

ADSNAME

Server address space (TIMSrvAddrSpc from the TIM)

SUBTASK

Server task ID (TIMSrvTaskID from the TIM)

TCPNAME

TCP address space (TIMTCPAddrSpc from the TIM)

TAKESOCKET

Accept the socket from the Listener.

Parameter

Meaning

CLIENT.name

Listener address space (TIMLstAddrSpc from the TIM)

CLIENT.task

Listener task ID (TIMLstTaskID from the TIM)

SOCRECV

Socket descriptor (TIMSktDesc from the TIM)

Note that the TAKESOCKET call returns a new socket descriptor which must be used for the rest of the process. (Do not continue to use the descriptor passed by the Listener in TIMSktDesc.)

READ/WRITE

Exchange application data with the client.

Database calls

Read/write database records.

Note: TCP/IP and database calls can be intermixed.

GU

Force IMS synchronization point; update the database from the buffers.

WRITE

Send complete-status message to the client.

CLOSE

Shut down the socket and release resources associated with it.

TERMAPI

End processing on the call interface.

Explicit-mode application data

The following information describes explicit-mode application data.

Format

Other than the initial transaction-initiation message, explicit-mode imposes no restrictions on the format of application data exchanged between client and server.

EBCDIC and ASCII data translation

If the TCP/IP host is transmitting ASCII data, explicit-mode servers are responsible for data translation from EBCDIC to ASCII and from ASCII to EBCDIC. Data translation is not performed by IMS TCP/IP. You can use the data translation subroutines (EZACIC04 and EZACIC05 or EZACIC14 and EZACIC15) described in Chapter 7, "CALL instruction application programming interface," on page 51 for this purpose.

When the conversation is complete, the server should force an IMS commit and close the connection. This causes IMS to complete the database updates. Explicit-mode server logic is responsible for notifying the client of the success or failure of the commit process.

Transaction-initiation message segment

Once the server has been started, the first segment it receives from the message queue is the transaction-initiation message (TIM) segment, which was created by the IMS Listener.

Field	Format	Explanation
TIMLen ¹²	Н	The length of the transaction- initiation message segment (in binary) , including the length of this field. (X'0038')
TIMRsv	н	Reserved field set to zero. (X'0000').
TIMId	CL8	Identifies the message as having been created by the IMS Listener. Always contains the characters *LISTNR*.
TIMLstAddrSpc	CL8	Listener address space name. Used in server TAKESOCKET.
TIMLstTaskId	CL8	Listener task ID. Used in server TAKESOCKET.
TIMSrvAddrSpc	CL8	Server address space name. Used in server INITAPI. Server address space IDs are generated by the Listener and consist of the 2-character prefix specified in the Listener configuration file (Listener statement) followed by a unique 6-character hexadecimal number.
TIMSrvTaskID	CL8	Server task ID. Used in server INITAPI.
TIMSktDesc	Н	Contains the descriptor of the socket given by Listener. Used in server TAKESOCKET.
TIMTCPAddrSpc	CL8	The TCP/IP address space name of TCP/IP. Used in INITAPI.
TIMDataType	Н	Indicates the data type of the client messages: ASCII(0) or EBCDIC(1).

Program design considerations

- Because MVS TCP/IP ends the connection when a server MPP completes, the client has no way of knowing that the database changes have been committed. Therefore, it is suggested that explicit-mode servers send a message to the client confirming the COMMIT before terminating. (Implicit-mode servers send the CSMOKY segment when the database changes have been committed.)
- When an explicit-mode server issues a ROLB command, the client has no automatic way of knowing that the database updates have been rolled back. It is suggested, therefore, that the server send a message to the client when a rollback call completes.

¹² If you use PL/I, you must define the LLLL field as a binary fullword.

I/O PCB explicit-mode server

When an IMS MPP issues a call for IMS TM services (like a GU or an ISRT), IMS returns information about the results of the call in a control block called the I/O program control block (I/O PCB). The contents of the I/O PCB are:

LTERM NAME

Blanks (8 bytes)

RESERVED

X'00' (2 bytes)

STATUS CODE

See "Status codes" on page 40 (2 bytes)

DATE/TIME

Undefined (8 bytes)

INPUT MSG. SEQ. #

Undefined (4 bytes)

MESSAGE OUTPUT DESC. NAME

Blanks (8 bytes)

USERID

PSBname of Listener (8 bytes)

Status codes

The I/O PCB status code is set by IMS in response to the server GU for the TIM. A status code of bb indicates successful completion of the GU call. Because the only data explicit-mode servers receive from the message queue is the TIM, the only call issued by the server is a GU, requesting a new TIM. Thus, the only status codes an explicit-mode server should receive are bb, which indicates successful completion of the GU; and QC, which indicates that there are no more messages on the message queue for that transaction. In response to the QC status code, the server program should end normally.

Explicit-mode server PL/I programming considerations

PL/I programmers should note that I/O areas used to retrieve IMS segments must follow standard IMS conventions. That is, the length field for the TIM segment must be defined as a fullword (FIXED BIN(31)).

Implicit-mode server program logic flow

An implicit-mode server must perform all of the functions previously described for an explicit-mode server (see <u>"Explicit-mode server program logic flow" on page 37</u>). However, the IMS Assist module issues the TCP/IP calls on behalf of the server program; consequently, the implicit-mode application programmer need issue only standard IMS Input/Output calls.

Implicit-mode server call sequence

When writing an implicit-mode program, you must call the IMS Assist module (CBLADLI, PLIADLI, ASMADLI, CADLI, as appropriate for the language you are using) instead of the conventional IMS equivalent (CBLTDLI, PLITDLI, ASMTDLI, CTDLI). This will cause the I/O PCB calls to be intercepted and processed (if necessary) by the Assist module. The Assist module will pass database calls directly to IMS for processing; it will intercept I/O PCB calls and issue the appropriate sockets calls. A sample call sequence (using COBOL syntax) for an implicit-mode server follows:

IMS Server Call

Resulting Assist Module Function

CALL CBLADLI (GU) I/O PCB

Issue CALL CBLTDLI (GU) to obtain the (TIM).

CALL CBLADLI (GN) I/O PCB

(optional) Issue CALL CBLTDLI (GN), which returns a subsequent segment of client input data for each call.

CALL CBLADLI

Read/write database records. ¹³

CALL CBLADLI (ISRT) I/O PCB

Store segments in the sockets output buffer.

CALL CBLADLI (GU) I/O PCB

Issue WRITE to empty output buffers.

Implicit-mode application data

The following information describes implicit-mode application data.

Format

All data exchanged between the client and an implicit-mode server is formatted into IMS segments. Each data segment has the following format:

Field	Format	Description
Length	Н	Length of the data segment (in binary) including this field.
Reserved	Н	Reserved field; must be set to zero.
Data	CLn	Application data.

Data translation

Translation of input data (when necessary) is done by the Listener. As a result, all data on the IMS message queue is in EBCDIC; output data is translated (when necessary) by the Assist module.

Note that when data translation takes place, the entire application data portion of the segment is translated from ASCII to EBCDIC, and vice versa; therefore, the segment should contain only printable characters common to both character sets. (For example, the EBCDIC cent sign and the ASCII left bracket are both printable in their respective environments but are not translated because they do not have an equivalent in the other character set.)

End-of-message segment

The last segment in a message (either sent by the client, or received from the server) is indicated by an end-of-message (EOM) segment. (See <u>"End-of-message segment (EOM)"</u> on page 35).

• Implicit-mode messages sent by the client are received by the Listener and inserted onto the IMS message queue. The end-of-message segment indicates to the Listener that there are no more segments to be inserted for this message.

Note: The server program will *not* receive the EOM segment; it will receive a QD status code, indicating that there are no more segments for this message.

• Implicit-mode messages to be sent by the server are actually written by the Assist module on behalf of the server program. When the server program sends application data to the client (using the ISRT call), the Assist module intercepts the output data and accumulates it in an output buffer. When the server program issues a subsequent GU to the I/O PCB, the Assist module interprets the GU as an indication that the server has inserted the last segment for that message. The Assist module then adds an end-of-message segment to the output data and issues WRITE commands, which transmit the data to the client.

¹³ Database PCB and I/O PCB calls can be intermixed.

Note: The server program should *not* attempt to insert an EOM segment to the I/O PCB.

Programming to the Assist module interface

Programs written to the Assist module interface are very similar (in terms of I/O calls) to conventional IMS Transaction Manager (TM) MPPs.

- To communicate with IMS TM, use the following calls (depending upon programming language) — CBLADLI, PLIADLI, ASMADLI, or CADLI — instead of CBLTDLI, PLITDLI, ASMTDLI, and CADLI, respectively.
- Use the same parameters as with the IMS TM counterparts.
- The first IMS call to the I/O PCB must be GU. Subsequent IMS calls to the I/O PCB can be GN and/or ISRT (with intervening database calls, as appropriate).
- When the transaction is complete, the server program should issue another GU to the I/O PCB to finalize processing of the present message. If the server program receives a bb status code, (indicating another message has been received for that program), it should loop back and process that message. Note that the Assist module will have closed the previous connection and opened a new connection associated with the new message. When the GU returns a QC status code, no more messages have been received for that program should end.

A set of one GU, one or more GN calls, and one or more ISRT calls to the I/O PCB (with intervening database calls, as required) constitute a transaction. The Assist module interprets each GU as the start of a new transaction.

- The PURG call cannot be used to indicate end-of-message; the server should not issue PURG calls to the I/O PCB.
- The Assist module GU reads the TIM into the I/O area defined in the server program; consequently, the I/O area you define in the server must be at least 56 bytes in length (the length of the TIM).
- If the server program attempts to insert more than 32KB, the Assist module flags this as an error by terminating processing and returning a status code of ZZ.

Implicit-mode server PL/I programming considerations

PL/I programmers should note that I/O areas passed to the Assist module must follow standard IMS conventions. That is, the length field for a segment must be defined as a fullword (FIXED BIN(31)). This applies to both input and output data segments; however, the actual segment that is received from and sent to the client uses a halfword (FIXED BIN(15)) length field. Thus, the messages exchanged between the client and server are programming-language independent.

Implicit-mode server C language programming considerations

The following statements are required in IMS implicit-mode servers written in C language:

```
#pragma runopts(env(IMS),plist(IMS))
#pragma linkage(cadli, OS)
```

This is in addition to the standard requirements for using C language programs in IMS.

I/O PCB implicit-mode server

When an IMS MPP issues a call for IMS TM services (like a GU or an ISRT), IMS returns information about the results of the call in a control block called the I/O program control block (I/O PCB). When using the Assist module, the contents of the I/O PCB are:

LTERM NAME

Blanks (8 bytes)

RESERVED

See "Status codes" on page 43 (2 bytes)

STATUS CODE

See "Status codes" on page 43 (2 bytes)

DATE/TIME

Undefined (8 bytes)

INPUT MSG. SEQ.

Undefined (4 bytes)

MESSAGE OUTPUT DESC. NAME

Blanks (8 bytes)

USERID

PSBname of Listener (8 bytes)

Status codes

The I/O PCB status code is set by IMS in response to the IMS calls that the Assist module makes on behalf of the server. For example, GU and GN calls usually result in bb, QC, or QD status codes. However, when the Assist module detects a TCP/IP error, it sets the status code field of the I/O PCB to ZZ with further information about the error in the **reserved** field of the I/O PCB. This field should be initially tested as a signed, fixed binary halfword:

- If the halfword is positive, then a socket error has occurred, and the field should continue to be treated as a signed fixed binary halfword. The field contains the 2 low-order bytes from the ERRNO resulting from the socket call. (See Appendix A, "Return codes," on page 267).
- If the halfword is negative, then an IMS or other type of error has occurred, and the field should be treated as a fixed-length, 2-byte character string containing one of the following information:

Code

Meaning

EA

A call that used the AIB interface to determine the I/O PCB address failed.

EB

The output buffer is full. An attempt was made to insert (ISRT) more than 32KB (including the segment length and reserved bytes) to be sent to the client.

EC

A QD status code was received in response to a GU or ROLB call when attempting to retrieve the first segment of data after the transaction-initiation message (TIM) segment. This implies that the client sent only the TIM segment followed by an end-of-message segment with no actual data segments.

44 z/OS Communications Server: z/OS V2R5.0 Communications Server: IP IMS Sockets Guide
Chapter 6. How to customize and operate the IMS Listener

The IMS Listener is an IMS batch message program (BMP) whose main purpose is to validate connection requests from TCP/IP clients and to schedule IMS message processing programs (MPP) servers.

This topic describes the IMS Listener and the user-written security exit that can be used to validate incoming transaction requests.

How to start the IMS Listener

The IMS Listener is executed as an MVS 'started task' using job control language (JCL) statements. Copy the sample job in the *hlq*.SEZAINST(EZAIMSJL) to your system or recognized PROCLIB and modify it to suit your conditions. The following information shows a sample of the JCL needed for the Listener BMP. Note the STEPLIB statements pointing to MVS TCP/IP. Also note the EZAIMSJL G.LSTNCFG DD statement points to the Listener configuration file. For more information on configuring the IMS Listener, see "The IMS Listener configuration file" on page 46.

((PROG NER FRATNOLN DOR FRATNOLN THOTE THE OFE TOPTHO COUT A
//EZAIMSJL	PROC MBR=EZAIMSLN, PSB=EZAIMSLN, IMSID=IMS, CFG=ICPIMS, SOUI=A
//*	
//LISTENER	EXEC PROC=IMSBATCH.MBR=&MBRSOUT=&SOUTIMSID=&IMSID
//	PSB=&PSB., CPUTIME=1440
//G.STEPLIB	DD DSN=IMSVS31.&SYS2.RESLIB,DISP=SHR
11	DD DSN=IMSVS31.&SYS2.PGMLIB,DISP=SHR
11	DD DSN=TCPIP.SEZALOAD,DISP=SHR
11	DD DSN=TCPIP.SEZATCP,DISP=SHR
//G.LSTNCFG	DD DSN=TCPIP.LSTNCFG(&CFG.),DISP=SHR
//G.SYSPRINT	DD SYSOUT=&SOUT,DCB=(LRECL=137,RECFM=VBA,BLKSIZE=1374),
//	SPACE=(141,(2500,100),RLSE,,ROUND)

Figure 11. JCL: Sample run Listener procedure

Once you have configured your JCL, you can start the Listener using the MVS START command. The basic syntax and parameters of this command are:



procname

The name of the cataloged procedure that defines the IMS Listener job to be started.

identifier

A user-determined name which, with the procedure name, (*procname*) uniquely identifies the started job. This name can be up to 8 characters long with the first character being alphabetic. If the identifier is omitted, MVS automatically uses the procedure name as the identifier.

How to stop the IMS Listener

The Listener is normally ended by issuing an MVS MODIFY command. The syntax and parameters of this command are:

► MODIFY _______ identifier ____, ___ STOP ►

procname

The name of the cataloged procedure that was used to start the Listener. This is required only if an identifier that was different from *procname* was specified with the START command when the Listener was started.

identifier

The user-determined identifier used on the START command when the Listener was started. If an explicit identifier was not specified (on the START command), MVS automatically uses the procedure name (*procname*) on the START command as the default identifier.

stop

Stops the Listener.

On receipt of a MODIFY command, the Listener closes the socket bound to the listening port so that no new requests can be accepted. It ends once all other sockets have been closed following acceptance of each socket by the corresponding server.

As a BMP, the Listener can be forcibly ended by issuing the IMS STOP REGION command with the ABDUMP option.

The IMS Listener configuration file

The IMS Listener obtains startup parameters from a configuration file. In the EZAIMSJL G.LSTNCFG DD statement points to the Listener configuration file. This statement will be in the JCL sample you customize.

The configuration file contains three types of statements which must appear in the following order:

1. TCPIP statement

2. LISTENER statement

3. TRANSACTION statements

The following information describes each of the configuration statements and their respective parameters.

TCPIP statement

Description: This statement is required and is used to specify the name of the TCP/IP address space.

```
► TCPIP — ADDRSPC=name →
```

- ADDRSPCPFX=prefix

ADDRSPC= *name*

Specifies the name of the TCP/IP address space. The name can be 1 to 8 characters long, consisting of the numbers 0–9, the letters A–Z, and the characters \$, @, and #.

LISTENER statement

Description: This statement is required. It is used to specify configuration information used by the IMS Listener.

```
LISTENER — PORT=port — MAXTRANS=maxtrans — MAXACTSKT=maxskt →

BACKLOG=10 —
```

BACKLOG=backlog

PORT= port

Port number that the Listener binds to for connection requests. Use an integer between 0 and 65535, inclusive.

MAXTRANS= maxtrans

The maximum number of TRANSACTION statements to be processed in the configuration file. Use an integer between 1 and 32767, inclusive.

MAXACTSKT= maxskt

The maximum number of sockets the Listener can have open awaiting an MPP TAKESOCKET at one time. This value is an integer from 1 to 2000, inclusive. The number includes the socket bound to the port through which it accepts incoming requests.

ADDRSPCPFX= prefix

One or two characters (consisting of the numbers 0–9, the letters A–Z, and the characters \$, @, and #) used in generating unique identifiers for started IMS transactions.

BACKLOG= backlog

This parameter is optional and is used to specify the length of the backlog queue maintained in TCP/IP for connection requests that have not yet been assigned sockets by the Listener. Use an unsigned number from 1 to 32767 inclusive. The default value is 10.

Tip: The backlog value specified on the listen call cannot be larger than the value configured by the SOMAXCONN statement in the stack's TCPIP PROFILE (the default value is 10), no error is returned if a larger backlog is requested. If you want a larger backlog, update the SOMAXCONN statement. See the z/OS Communications Server: IP Configuration Reference for details.

TRANSACTION statement

Description: This statement specifies which transactions can be started by the Listener. One statement is required for each transaction that can be initiated by a TCP/IP-connected client.

Note that the transactions named here are subject to limitations:

- They must be defined to IMS as MODE=SNGL in the IMS TRANSACT macro; this will ensure that the database buffers are emptied (flushed) to direct access storage when the second and subsequent GU calls are issued.
- They must not be IMS conversational transactions.
- They cannot name transactions that are executed in a remote Multiple Systems Coupling (MSC) environment.
- They must not use Message Format Services for messages to the client.



NAME= transid

The name of an IMS transaction that is designed to interact with a TCP/IP-connected program. This parameter must be 1 to 8 characters long, containing alphanumeric characters, or the characters @, \$, and #.

TYPE=

This parameter specifies whether the transaction uses the IMS Assist module. It must specify either EXPLICIT or IMPLICIT.

The IMS Listener security exit

The IMS Listener includes an exit (IMSLSECX), which can be programmed by the user to perform a security check on the incoming transaction-request. This Listener exit can be designed to validate the contents of the UserData field in the transaction request message.

To use the user-supplied security exit, you must define an entry point named IMSLSECX. If a module with this name is link-edited with the Listener (EZAIMSLN) load module, the security exit is called as part of transaction verification. The security exit is called using standard MVS linkage with register 1 (R1) pointing to the parameter list, shown in <u>Table 4 on page 48</u>. Note that the security exit must have the attribute AMODE(31).

The exit returns 2 indicators: a return code and a reason code. The Listener uses the return code to determine whether to honor the request. Both the return code and the reason code are passed back to the client. Data passed in the UserData field is not translated from ASCII to EBCDIC; this translation is the responsibility of the security exit. (EZACIC05 and EZACIC04 can be used to accomplish translation between ASCII and EBCDIC. See CALL instructions in z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference for a description of these utilities.)

Field	Format	Description
IpAddr	F	The address of a fullword containing the client's IP address.
Port	Н	The address of a halfword containing the client's port number.
TransNam	CL8	The address of an 8-character string defining the name of the requested transaction.
DataType	Н	The address of a halfword containing the data type (0 if ASCII or 1 if EBCDIC).
DataLen	F	The address of a fullword containing the length of the user data.
Userdata	XLn	The address of the user-supplied data.
RetnCode	F	The address of a fullword set by the security exit to indicate the return status. Set to nonzero (4, 8, 12,) to indicate an error.
ReasnCode	F	The address of a fullword set by the security exit as a reason code associated with the value of the return code. Reason codes 0–100 are reserved for use by the Listener. The security exit can use reason codes greater than 100.

TCP/IP services definitions

To run IMS, you need to modify the *tcpip*.PROFILE.TCPIP data set and the *hlq*.TCPIP.DATA data set that are part of the TCP/IP Services configuration file.

Guideline: In this information, the abbreviation *hlq* stands for an installation-dependent *high level qualifier* which you must supply.

The hlq.PROFILE.TCPIP data set

You define the hlq.PROFILE.TCPIP data set. In it, you must provide entries for the IMS socket Listener started task name in the PORT statement, as shown in Figure 12 on page 49.

The format for the PORT statement is:

► port_number — TCP — IMS_socket_Listener_jobname →

As an example, assume you want to define two different IMS control regions. Create a different line for each port that you want to reserve. Figure 12 on page 49 shows 2 entries, allocating port number 4000 for SERVA, and port number 4001 for SERVB. SERVA and SERVB are the names of the IMS socket Listener started task names.

These 2 entries reserve port 4000 for exclusive use by SERVA and port 4001 for exclusive use by SERVB. The Listener transactions for SERVA and SERVB should be bound to ports 4000 and 4001 respectively. Other applications that want to access TCP/IP on MVS are prevented from using these ports.

Ports that are not defined in the PORT statement can be used by any application, including SERVA and SERVB if they need other ports.

Figure 12. Definition of the TCP/IP profile

The hlq.TCPIP.DATA data set

For IMS, you do not have to make any extra entries in *hlq*.TCPIP.DATA. However, you need to check the TCPIPJOBNAME parameter that was entered during TCP/IP Services setup. This parameter is the name of the started procedure used to start the TCP/IP MVS address space. This must match the job name in the Listener configuration file TCPIP statement, as described in <u>"TCPIP statement" on page 46</u>. In the example shown in Figure 13 on page 49, TCPIPJOBNAME is set to TCPV3. The default name is TCPIP.



Figure 13. The TCPIPJOBNAME Parameter in the DATA data set

50 z/OS Communications Server: z/OS V2R5.0 Communications Server: IP IMS Sockets Guide

Chapter 7. CALL instruction application programming interface

This information describes the CALL instruction API for IPv4 or IPv6 socket applications. The following topics are included:

- "CALL instruction API environmental restrictions and programming requirements" on page 51
- "CALL instruction API output register information" on page 52
- "CALL instruction API compatibility considerations" on page 52
- "CALL instruction application programming interface (API)" on page 53
- <u>"Understanding COBOL</u>, Assembler, and PL/I call formats" on page 53
- "Converting parameter descriptions" on page 54
- <u>"Diagnosing problems in applications using the CALL instruction API" on page 54</u>
- <u>"CALL instruction API error messages and return codes" on page 54</u>
- <u>"Code CALL instructions" on page 55</u>
- "Using data translation programs for socket call interface" on page 187
- <u>"Call interface sample programs" on page 200</u>

CALL instruction API environmental restrictions and programming requirements

The following restrictions apply to both the Macro Socket API and the Callable Socket API:

Function	Restriction	
SRB mode	These APIs can be invoked only in TCB mode (task mode).	
Cross-memory mode	These APIs can be invoked only in a non-cross-memory environment (PASN=SASN=HASN).	
Functional Recovery Routine (FRR)	Do not invoke these APIs with an FRR set. This causes system recovery routines to be bypassed and severely damage the system.	
Locks	No locks should be held when issuing these calls.	
INITAPI and TERMAPI socket commands	The INITAPI and TERMAPI socket commands must be issued under the same task.	
Storage	Storage acquired for the purpose of containing data returned from a socket call must be obtained in the same key as the application program status word (PSW) at the time of the socket call.	
Nested socket API calls	You cannot issue nested API calls within the same task. That is, if a request block (RB) issues a socket API call and is interrupted by an interrupt request block (IRB) in an STIMER exit, any additional socket API calls that the IRB attempts to issue are detected and flagged as errors.	

Function	Restriction
Addressability mode (Amode) considerations	The EZASOKET API can be invoked while the caller is in either 31-bit or 24-bit Amode. However, if the application is running in 24-bit addressability mode at the time of the call, all addresses of parameters passed by the application must be addressable in 31-bit Amode. This implies that even if the addresses being passed reside in storage below the 16 MB line (and therefore addressable by 24-bit Amode programs) the high-order byte of these addresses needs to be 0.
Use of z/OS UNIX System Services	Address spaces using the EZASOKET API should not use any z/OS UNIX System Services socket API facilities such as z/OS UNIX Assembler Callable Services or Language Environment [®] for z/OS C/C++. Doing so can yield unpredictable results.

CALL instruction API output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register

Contents

0-1

Used as work registers by the system

2-13

Unchanged

14

Used as a work register by the system

15

Contains the entry point address EZASOKET

When control returns to the caller, the access registers (ARs) contain:

Register

Contents

0-1

Used as work registers by the system

2-14

Unchanged

15

Used as a work register by the system.

If a caller depends on register contents to remain the same before and after issuing a service, the caller must save the contents of a register before issuing the service and must restore them after the system returns control.

CALL instruction API compatibility considerations

Unless noted in <u>z/OS</u> Communications Server: New Function Summary, an application program compiled and link edited on a release of z/OS Communications Server IP can be used on higher level releases. That is, the API is upward compatible.

Application programs that are compiled and link edited on a release of z/OS Communications Server IP cannot be used on older releases. That is, the API is not downward compatible.

CALL instruction application programming interface (API)

This information describes the CALL instruction API for TCP/IP application programs written in the COBOL, PL/I, or System/370 Assembly language. The format and parameters are described for each socket call.

Note:

- Unless your program is running in a CICS environment, reentrant code and multithread applications are not supported by this interface.
- For a PL/I program, include the following statement before your first call instruction.

DCL EZASOKET ENTRY OPTIONS(ASM, INTER) EXT;

- If you use the CALL instruction from code that will run as a part of a CICS transaction, see the <u>z/OS</u> Communications Server: IP CICS Sockets Guide for additional considerations.
- The Sockets Extended module (EZASOKET) is located in the hlq.SEZATCP(EZASOKET) load module and should be resolved from there when it is processed by the binder. You can use the linkage editor MAP parameter to produce the module map report to verify where EZASOKET is resolved.

Understanding COBOL, Assembler, and PL/I call formats

This API is invoked by calling the EZASOKET program and performs the same functions as the C language calls. The parameters look different because of the differences in the programming languages.

COBOL language call format

The following syntax shows the 'EZASOKET' call format for COBOL language programs:

CALL 'EZASOKET' USING SOC-FUNCTION parm1, parm2, .. ERRNO, RETCODE.

SOC-FUNCTION

A 16-byte character field, left-aligned and padded on the right with blanks. Set to the name of the call. SOC-FUNCTION is case specific. It must be in uppercase.

parm*n*

A variable number of parameters depending on the type call.

ERRNO

If RETCODE is negative, there is an error number in ERRNO. This field is used in most, but not all, of the calls. It corresponds to the value returned by the tcperror() function in C.

RETCODE

A fullword binary variable containing a code returned by the EZASOKET call. This value corresponds to the normal return value of a C function.

Assembly language call format

The following syntax shows the EZASOKET call format for assembly language programs.

```
CALL EZASOKET, (SOC-FUNCTION, parm1, parm2, ... ERRNO, RETCODE), VL
```

PL/I language call format

The following syntax shows the EZASOKET call format for PL/I language programs:

CALL EZASOKET (SOC-FUNCTION parm1, parm2, ... ERRNO, RETCODE);

SOC-FUNCTION

A 16-byte character field, left-aligned and padded on the right with blanks. Set to the name of the call.

parm*n*

A variable number of parameters depending on the type call.

ERRNO

If RETCODE is negative, there is an error number in ERRNO. This field is used in most, but not all, of the calls. It corresponds to the value returned by the tcperror() function in C.

RETCODE

A fullword binary variable containing a code returned by the EZASOKET call. This value corresponds to the normal return value of a C function.

Converting parameter descriptions

The parameter descriptions in this information are written using the VS COBOL II PIC language syntax and conventions, but you should use the syntax and conventions that are appropriate for the language you want to use.

Figure 14 on page 54 shows examples of storage definition statements for COBOL, PL/I, and assembly language programs.

VS COBOL II PIC			
PIC S9(4) BINARY		HALFWORD BINARY	VALUE
PIC S9(8) BINARY		FULLWORD BINARY	VALUE
PIC X(n)		CHARACTER FIELD	OF N BYTES
COBOL PIC			
PIC S9(4) COMP		HALFWORD BINARY	VALUE
PIC S9(4) BINARY		HALFWORD BINARY	VALUE
PIC S9(8) COMP		FULLWORD BINARY	VALUE
PIC S9(8) BINARY		FULLWORD BINARY	VALUE
PIC X(n)		CHARACTER FIELD	OF N BYTES
PL/I DECLARE STATEMENT			
DCL HALF	FIXED BIN(15),	HALFWORD BINARY	VALUE
DCL FULL	FIXED BIN(31),	FULLWORD BINARY	VALUE
DCL CHARACTER	CHAR(n)	CHARACTER FIELD	OF n BYTES
ASSEMBLER DECLARATION			
DS H		HALFWORD BINARY	VALUE
DS F		FULLWORD BINARY	VALUE
DS CLn		CHARACTER FIELD	OF n BYTES

Figure 14. Storage definition statement examples

Diagnosing problems in applications using the CALL instruction API

TCP/IP provides a trace facility that can be helpful in diagnosing problems in applications using the CALL instruction API. The trace is implemented using the TCP/IP Component Trace (CTRACE) SOCKAPI trace option. The SOCKAPI trace option allows all Call instruction socket API calls issued by an application to be traced in the TCP/IP CTRACE. The SOCKAPI trace records include information such as the type of socket call, input, and output parameters and return codes. This trace can be helpful in isolating failing socket API calls and in determining the nature of the error or the history of socket API calls that might be the cause of an error. For more information about the SOCKAPI trace option, see <u>z/OS Communications</u> Server: IP Diagnosis Guide.

CALL instruction API error messages and return codes

For information about error messages, see z/OS Communications Server: IP Messages Volume 1 (EZA).

For information about error codes that are returned by TCP/IP, see <u>Appendix A</u>, "Return codes," on page 267.

Code CALL instructions

This information contains the description, syntax, parameters , and other related information for each call instruction included in this API.

ACCEPT

A server issues the ACCEPT call to accept a connection request from a client. The call points to a socket that was previously created with a SOCKET call and marked by a LISTEN call.

The ACCEPT call is a blocking call. When issued, the ACCEPT call:

- 1. Accepts the first connection on a queue of pending connections.
- 2. Creates a new socket with the same properties as s, and returns its descriptor in RETCODE. The original sockets remain available to the calling program to accept more connection requests.
- 3. The address of the client is returned in NAME for use by subsequent server calls.

Note:

- The blocking or nonblocking mode of a socket affects the operation of certain commands. The default is blocking; nonblocking mode can be established by use of the FCNTL and IOCTL calls. When a socket is in blocking mode, an I/O call waits for the completion of certain events. For example, a READ call will block until the buffer contains input data. When an I/O call is issued:
 - If the socket is blocking, program processing is suspended until the event completes.
 - If the socket is nonblocking, program processing continues.
- If the queue has no pending connection requests, ACCEPT blocks the socket unless the socket is in nonblocking mode. The socket can be set to nonblocking by calling FCNTL or IOCTL.
- When multiple socket calls are issued, a SELECT call can be issued prior to the ACCEPT to ensure that a connection request is pending. Using this technique ensures that subsequent ACCEPT calls will not block.
- TCP/IP does not provide a function for screening clients. As a result, it is up to the application program to control which connection requests it accepts, but it can close a connection immediately after discovering the identity of the client.

Condition	Requirement
Authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN.
Amode:	31-bit or 24-bit.
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.
ASC mode:	Primary address space control (ASC) mode.
Interrupt status:	Enabled for interrupts.
Locks:	Unlocked.
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Table 5. ACCEPT call requirements

Figure 15 on page 56 shows an example of ACCEPT call instructions.

```
WORKING-STORAGE SECTION.
   01 SOC-FUNCTION PIC X(16) VALUE IS 'ACCEPT'.
                        PIC 9(4) BINARY.
    01 S
* IPv4 socket address structure.
   01 NAME.
        03
            FAMILY
                        PIC 9(4) BINARY.
        03 PORT
                        PIC 9(4) BINARY.
            IP-ADDRESS PIC 9(8) BINARY.
        03
        03 RESERVED
                        PIC X(8).
* IPv6 socket address structure.
    01 NAME.
        03 FAMILY
                        PIC 9(4) BINARY.
                        PIC 9(4) BINARY.
PIC 9(8) BINARY.
        03
            PORT
        03 FLOWINFO
        03 IP-ADDRESS.
            10 FILLER
                        PIC 9(16) BINARY.
            10 FILLER PIC 9(16) BINARY.
        03
            SCOPE-ID PIC X(8) BINARY.
NO PIC 9(8) BINARY.
    01 ERRNO
    01 RETCODE
                        PIC S9(8) BINARY.
PROCEDURE DIVISION.
     CALL 'EZASOKET' USING SOC-FUNCTION S NAME ERRNO RETCODE.
```

Figure 15. ACCEPT call instructions example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing ACCEPT. Left-align the field and pad it on the right with blanks.

S

A halfword binary number specifying the descriptor of a socket that was previously created with a SOCKET call. In a concurrent server, this is the socket upon which the server listens.

Parameter values returned to the application

NAME

An IPv4 socket address structure that contains the client's socket address.

FAMILY

A halfword binary field specifying the IPv4 addressing family. The call returns the value decimal 2 for AF_INET.

PORT

A halfword binary field that is set to the client's port number.

IP-ADDRESS

A fullword binary field that is set to the 32-bit IPv4 IP address, in network byte order, of the client's host machine.

RESERVED

Specifies 8 bytes of binary zeros. This field is required, but not used.

An IPv6 socket address structure that contains the client's socket address.

FAMILY

A halfword binary field specifying the IPv6 addressing family. For TCP/IP the value is decimal 19, indicating AF_INET6.

PORT

A halfword binary field that is set to the client's port number.

FLOWINFO

A fullword binary field specifying the traffic class and flow label. This value of this field is undefined.

IP-ADDRESS

A 16-byte binary field that is set to the 128-bit IPv6 IP address, in network-byte-order, of the client's host machine.

SCOPE-ID

A fullword binary field which identifies a set of interfaces as appropriate for the scope of the address carried in the IPv6-ADDRESS field. For a link scope IPv6-ADDRESS, SCOPE-ID contains the link index for the IPv6-ADDRESS. For all other address scopes, SCOPE-ID is undefined.

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See <u>Appendix A</u>, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

If the RETCODE value is positive, the RETCODE value is the new socket number.

If the RETCODE value is negative, check the ERRNO field for an error number.

Value

Description

> 0

Successful call.

-1

Check **ERRNO** for an error code.

BIND

In a typical server program, the BIND call follows a SOCKET call and completes the process of creating a new socket.

The BIND socket command can specify the port or let the system choose the port. A listener program should always bind to the same well-known port so that clients know the socket address to use when issuing a CONNECT, SENDTO, or SENDMSG request.

In addition to the port, the application also specifies an IP address on the BIND socket command. Most applications typically specify a value of 0 for the IP address, which allows these applications to accept new TCP connections or receive UDP datagrams that arrive over any of the network interfaces of the local host. This enables client applications to contact the application using any of the IP addresses associated with the local host.

Alternatively, an application can indicate that it is interested in receiving only new TCP connections or UDP datagrams that are targeted towards a specific IP address associated with the local host. This can be accomplished by specifying the IP address in the appropriate field of the socket address structure passed on the NAME parameter.

Tip: Even if an application specifies the value 0 for the IP address on the BIND, the system administrator can override that value by specifying the BIND parameter on the PORT reservation statement in the TCP/IP profile. The effect of this override is similar to the effect of the application specifying an explicit IP address on the BIND macro. For more information, see <u>z/OS Communications Server: IP Configuration</u> Reference.

Table 6. BIND call requirements

Condition	Requirement
Authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN.

Table 6. BIND call requirements (continued)	
Condition	Requirement
Amode:	31-bit or 24-bit.
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.
ASC mode:	Primary address space control (ASC) mode.
Interrupt status:	Enabled for interrupts.
Locks:	Unlocked.
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Figure 16 on page 58 shows an example of BIND call instructions.

```
WORKING-STORAGE SECTION.
    01 SOC-FUNCTION
                            PIC X(16) VALUE IS 'BIND'.
                            PIC 9(4) BINARY.
     01 S
 * IPv4 socket address structure.
    01 NAME.
         03 FAMILY
                            PIC 9(4) BINARY.
         03 PORT
                           PIC 9(4) BINARY.
         03 IP-ADDRESS PIC 9(8) BINARY.
03 RESERVED PIC X(8).
 * IPv6 socket address structure.
    01 NAME.
         03FAMILYPIC 9(4) BINARY.03PORTPIC 9(4) BINARY.03FLOWINFOPIC 9(8) BINARY.
         03 IP-ADDRESS.
         10 FILLER PIC 9(16) BINARY.
10 FILLER PIC 9(16) BINARY.
03 SCOPE-ID PIC 9(8) BINARY.
     01 ERRNO
                             PIC 9(8) BINARY.
    01 RETCODE
                           PIC S9(8) BINARY.
PROCEDURE DIVISION.
      CALL 'EZASOKET' USING SOC-FUNCTION S NAME ERRNO RETCODE.
```

Figure 16. BIND call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing BIND. The field is left-aligned and padded to the right with blanks.

S

A halfword binary number specifying the socket descriptor for the socket to be bound.

NAME

See <u>z/OS</u> Communications Server: IP Sockets Application Programming Interface Guide and Reference for more information.

Specifies the IPv4 socket address structure for the socket that is to be bound.

FAMILY

A halfword binary field specifying the IPv4 addressing family. The value is always set to decimal 2, indicating AF_INET.

PORT

A halfword binary field that is set to the port number to which you want the socket to be bound.

Note: To determine the assigned port number, call the GETSOCKNAME command after calling the BIND command.

IP-ADDRESS

A fullword binary field that is set to the 32-bit IPv4 IP address (network byte order) of the socket to be bound.

RESERVED

Specifies an 8-byte character field that is required but not used.

Specifies the IPv6 socket address structure for the socket that is to be bound.

FAMILY

A halfword binary field specifying the IPv6 addressing family. For TCP/IP the value is decimal 19, indicating AF_INET6.

PORT

A halfword binary field that is set to the port number to which you want the socket to be bound.

Note: To determine the assigned port number, call the GETSOCKNAME command after calling the BIND command.

FLOWINFO

A fullword binary field specifying the traffic class and flow label. This field must be set to 0.

IP-ADDRESS

A 16-byte binary field that is set to the 128-bit IPv6 IP address (network byte order) of the socket to be bound.

SCOPE-ID

A fullword binary field which identifies a set of interfaces as appropriate for the scope of the address carried in the IPv6-ADDRESS field. A value of 0 indicates the SCOPE-ID field does not identify the set of interfaces to be used, and can be specified for any address types and scopes. For a link scope IPv6-ADDRESS, SCOPE-ID can specify a link index which identifies a set of interfaces. For all other address scopes, SCOPE-ID must be set to 0.

Parameter values returned to the application

ERRNO

A fullword binary field. If RETCODE is negative, this field contains an error number. See <u>Appendix A</u>, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

0

Successful call.

-1

Check ERRNO for an error code.

BIND2ADDRSEL

The BIND2ADDRSEL call binds a socket to the local IP address that would be selected by the stack to communicate with the input destination IP address.

Use the BIND2ADDRSEL call when the application must verify that the local IP address assigned by the stack meets its address selection criteria as specified by the IPV6_ADDR_PREFERENCES socket option before the stack sends any packets to the remote host. In a TCP or UDP application, the BIND2ADDRSEL call usually follows the SETSOCKOPT call with option IPV6 ADDR PREFERENCES and precedes any communication with a remote host.

Result: The stack attempts to select a local IP address according to your application preferences. However, a successful BIND2ADDRSEL call does not guarantee that all of your source IP address selection preferences were met.

Guidelines

 Use the SETSOCKOPT call to set the IPV6_ADDR_PREFERENCES option to indicate your selection preferences of source IP address before binding the socket and before allowing an implicit bind of the socket to occur.

Result: If a socket has not been explicitly bound to a local IP address with a BIND or BIND2ADDRSEL call when a CONNECT, SENDTO, or SENDMSG call is issued, an implicit bind occurs. The stack chooses the local IP address used for outbound packets.

Requirement: When your application is using stream sockets, and must prevent the stack from sending any packets whatsoever (such as SYN) to the remote host before it can verify that the local IP address meets the values specified for the IPV6 ADDR PREFERENCES option, do not allow the CONNECT call to implicitly bind the socket to a local IP address. Instead, bind the socket with the BIND2ADDRSEL call and test the local IP address assigned with the INET6_IS_SRCADDR call. If the assigned local IP address is satisfactory, you can then use the CONNECT call to establish communication with the remote host.

• After you successfully issue the BIND2ADDRSEL call, use the GETSOCKNAME call to obtain the local IP address that is bound to the socket. When the local IP address is obtained, use the INET6_IS_SRCADDR

Table 7. BIND2ADDRSEL call requirements		
Condition	Requirement	
Authorization:	Supervisor state or problem state, any PSW key.	
Dispatchable unit mode:	Task.	
Cross memory mode:	PASN = HASN.	
Amode:	31-bit or 24-bit.	
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.	
ASC mode:	Primary address space control (ASC) mode.	
Interrupt status:	Enabled for interrupts.	
Locks:	Unlocked.	
Control parameters:	All parameters must be addressable by the caller and in the primary address space.	

Figure 17 on page 61 shows an example of BIND2ADDRSEL call instructions.

call to verify that the local IP address meets your address selection criteria.

```
WORKING-STORAGE SECTION.
         01 SOC-FUNCTION
                              PIC X(16) VALUE IS 'BIND2ADDRSEL'.
                             PIC 9(4) BINARY.
         01 S
      * IPv6 socket address structure.
         01 NAME.
             03
                 FAMILY
                              PIC 9(4) BINARY.
             03
                 PORT
                              PIC 9(4) BINARY.
                 FLOWINFO
                             PIC 9(8) BINARY.
             03
             03 IP-ADDRESS.
                 10 FILLER
                              PIC 9(16) BINARY.
                              PIC 9(16) BINARY.
PIC 9(8) BINARY.
                 10 FILLER
             03 SCOPE-ID
FRRNO
             ERRNO
         01
                              PIC 9(8) BINARY.
                              PIC S9(8) BINARY.
         01 RETCODE
     PROCEDURE DIVISION.
          CALL 'EZASOKET' USING SOC-FUNCTION S NAME ERRNO RETCODE.
```

Figure 17. BIND2ADDRSEL call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing BIND2ADDRSEL. The field is left-aligned and padded to the right with blanks.

S

A halfword binary number specifying the socket descriptor for the socket that is to be bound.

Requirement: The socket must be an AF_INET6 socket. The type can be SOCK_STREAM or SOCK_DGRAM.

NAME

Specifies the IPv6 socket address structure of the remote host that the socket will communicate with. The IPv6 socket structure must specify the following fields:

FAMILY

A halfword binary field specifying the IPv6 addressing family. This field must be set to the decimal value 19, indicating AF_INET6.

PORT

A halfword binary field. This field is ignored by BIND2ADDRSEL processing.

Tip: To determine the assigned port number, issue the GETSOCKNAME call after the BIND2ADDRSEL call completes.

FLOWINFO

A fullword binary field. This field is ignored by BIND2ADDRSEL processing.

IP-ADDRESS

A 16-byte binary field that is set to the 128-bit IPv6 IP address (network byte order) of the remote host that the socket will communicate with.

Rule: Specify an IPv4 address by using its IPv4-mapped IPv6 format.

SCOPE-ID

A fullword binary field that identifies a set of appropriate interfaces for the scope of the address that is specified in the IPv6-ADDRESS field. The value 0 indicates that the SCOPE-ID field does not identify the set of interfaces to be used.

Requirement: The SCOPE-ID value must be nonzero if the address is a link-local address. For all other address scopes, SCOPE-ID must be set to 0.

Parameter values returned to the application

ERRNO

A fullword binary field. If RETCODE is negative, this field contains an error number. See Appendix A, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

0

Successful call.

-1

Check ERRNO for an error code.

CLOSE

The CLOSE call performs the following functions:

- The CLOSE call shuts down a socket and frees all resources allocated to it. If the socket refers to an open TCP connection, the connection is closed.
- The CLOSE call is also issued by a concurrent server after it gives a socket to a child server program. After issuing the GIVESOCKET and receiving notification that the client child has successfully issued a TAKESOCKET, the concurrent server issues the close command to complete the passing of ownership. In high-performance, transaction-based systems the timeout associated with the CLOSE call can cause performance problems. In such systems, you should consider the use of a SHUTDOWN call before you issue the CLOSE call. See "SHUTDOWN" on page 178 for more information.

Note:

- If a stream socket is closed while input or output data is gueued, the TCP connection is reset and data transmission might be incomplete. The SETSOCKOPT call can be used to set a linger condition, in which TCP/IP will continue to attempt to complete data transmission for a specified time after the CLOSE call is issued. See SO-LINGER in the description of "SETSOCKOPT" on page 161.
- A concurrent server differs from an iterative server. An iterative server provides services for one client at a time; a concurrent server receives connection requests from multiple clients and creates child servers that actually serve the clients. When a child server is created, the concurrent server obtains a new socket, passes the new socket to the child server, and then dissociates itself from the connection. The CICS Listener is an example of a concurrent server.
- After an unsuccessful socket call, a close should be issued and a new socket should be opened. An attempt to use the same socket with another call results in a nonzero return code.

Table 8. CLOSE call requirements	
Condition	Requirement
Authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN.
Amode:	31-bit or 24-bit.
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.
ASC mode:	Primary address space control (ASC) mode.
Interrupt status:	Enabled for interrupts.

T / A A/AAF

Table 8. CLOSE call requirements (continued)

Condition	Requirement
Locks:	Unlocked.
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Figure 18 on page 63 shows an example of CLOSE call instructions.

```
WORKING-STORAGE SECTION.

01 SOC-FUNCTION PIC X(16) VALUE IS 'CLOSE'.

01 S PIC 9(4) BINARY.

01 ERRNO PIC 9(8) BINARY.

01 RETCODE PIC S9(8) BINARY.

PROCEDURE DIVISION.

CALL 'EZASOKET' USING SOC-FUNCTION S ERRNO RETCODE.
```

Figure 18. CLOSE call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte field containing CLOSE. Left-align the field and pad it on the right with blanks.

S

A halfword binary field containing the descriptor of the socket to be closed.

Parameter values returned to the application

ERRNO

A fullword binary field. If RETCODE is negative, this field contains an error number. See <u>Appendix A</u>, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

0

Successful call.

-1

Check ERRNO for an error code.

CONNECT

The CONNECT call is issued by a client to establish a connection between a local socket and a remote socket.

The call sequence issued by the client and server for stream sockets is:

- 1. The server issues BIND and LISTEN to create a passive open socket.
- 2. The *client* issues CONNECT to request the connection.
- 3. The server accepts the connection on the passive open socket, creating a new connected socket.

The blocking mode of the CONNECT call conditions its operation.

• If the socket is in blocking mode, the CONNECT call blocks the calling program until the connection is established, or until an error is received.

- If the socket is in nonblocking mode, the return code indicates whether the connection request was successful.
 - A 0 RETCODE indicates that the connection was completed.
 - A nonzero RETCODE with an ERRNO of 36 (EINPROGRESS) indicates that the connection is not completed. However, because the socket is nonblocking, the CONNECT call returns normally.

The caller must test the completion of the connection setup by calling SELECT and testing for the ability to write to the socket.

The completion cannot be checked by issuing a second CONNECT. For more information, see <u>"SELECT" on</u> page 145.

Table 9. CONNECT call requirements

Condition	Requirement
Authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN.
Amode:	31-bit or 24-bit.
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.
ASC mode:	Primary address space control (ASC) mode.
Interrupt status:	Enabled for interrupts.
Locks:	Unlocked.
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Figure 19 on page 64 shows an example of CONNECT call instructions.

WORKING-STORAGE SECTION 01 SOC-FUNCTION 01 S	PIC X(16) VALUE IS 'CONNECT'. PIC 9(4) BINARY.
* IPv4 socket address s 01 NAME.	tructure.
03 FAMILY 03 PORT 03 IP-ADDRESS	PIC 9(4) BINARY. PIC 9(4) BINARY. PIC 9(8) BINARY.
03 RESERVED	PIC X(8).
 * IPv6 socket address s 01 NAME. 	tructure.
03 FAMILY 03 PORT	PIC 9(4) BINARY. PIC 9(4) BINARY.
03 IP-ADDRESS 03 FLOWINFO 03 IP-ADDRESS	PIC 9(8) BINARY. PIC 9(8) BINARY.
10 FILLER	PIC 9(16) BINARY.
03 SCOPE-ID	PIC 9(8) BINARY.
01 RETCODE	PIC 9(8) BINARY. PIC S9(8) BINARY.
PROCEDURE DIVISION.	
CALL 'EZASOKET' USING S	OC-FUNCTION S NAME ERRNO RETCODE.

Figure 19. CONNECT call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Stream sockets

For stream sockets, the CONNECT call is issued by a client to establish connection with a server. The call performs two tasks:

- It completes the binding process for a stream socket if a BIND call has not been previously issued.
- It attempts to make a connection to a remote socket. This connection is necessary before data can be transferred.

UDP sockets

For UDP sockets, a CONNECT call need not precede an I/O call, but if issued, it allows you to send messages without specifying the destination.

Parameter values set by the application

SOC-FUNCTION

A 16-byte field containing CONNECT. Left-align the field and pad it on the right with blanks.

S

A halfword binary number specifying the socket descriptor of the socket that is to be used to establish a connection.

NAME

An IPv4 socket address structure that contains the IPv4 socket address of the target to which the local, client socket is to be connected.

FAMILY

A halfword binary field specifying the IPv4 addressing family. The value must be decimal 2 for AF_INET.

PORT

A halfword binary field that is set to the server's port number in network byte order. For example, if the port number is 5000 in decimal, it is stored as X'1388' in hex.

IP-ADDRESS

A fullword binary field that is set to the 32-bit IPv4 IP address of the server's host machine in network byte order. For example, if the IP address is 129.4.5.12 in dotted decimal notation, it would be represented as X'8104050C' in hex.

RESERVED

Specifies an 8-byte reserved field. This field is required, but is not used.

An IPv6 socket address structure that contains the IPv6 socket address of the target to which the local, client socket is to be connected.

FAMILY

A halfword binary field specifying the IPv6 addressing family. For TCP/IP the value is decimal 19 for AF_INET6.

PORT

A halfword binary field that is set to the server's port number in network byte order. For example, if the port number is 5000 in decimal, it is stored as X'1388' in hex.

FLOWINFO

A fullword binary field specifying the traffic class and flow label. This field must be set to 0.

IP-ADDRESS

A 16-byte binary field that is set to the 128-bit IPv6 IP address of the server's host machine in network byte order. For example, if the IPv6 IP address is 12ab:0:0:cd30:123:4567:89ab:cedf in colon hex notation, it is set to X'12AB0000000CD300123456789ABCDEF'.

SCOPE-ID

A fullword binary field which identifies a set of interfaces as appropriate for the scope of the address carried in the IPv6-ADDRESS field. A value of 0 indicates the SCOPE-ID field does not identify the set of interfaces to be used, and can be specified for any address types and scopes. For a link scope IPv6-ADDRESS, SCOPE-ID can specify a link index which identifies a set of interfaces. For all other address scopes, SCOPE-ID must be set to 0.

Parameter values returned to the application

ERRNO

A fullword binary field. If RETCODE is negative, this field contains an error number. See <u>Appendix A</u>, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

0

Successful call.

-1

Check ERRNO for an error code.

FCNTL

The blocking mode of a socket can either be queried or set to nonblocking using the FNDELAY flag described in the FCNTL call. You can query or set the FNDELAY flag even though it is not defined in your program.

See "IOCTL" on page 118 for another way to control a socket's blocking mode.

Values for commands that are supported by the z/OS UNIX Systems Services fcntl callable service will also be accepted. See <u>z/OS UNIX System Services Programming</u>: Assembler Callable Services Reference for more information.

Table 10. FCNTL call requireme	2/115
Condition	Requirement
Authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN.
Amode:	31-bit or 24-bit.
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.
ASC mode:	Primary address space control (ASC) mode.
Interrupt status:	Enabled for interrupts.
Locks:	Unlocked.
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Table 10. FCNTL call requirements

Figure 20 on page 67 shows an example of FCNTL call instructions.

WORKING-STORAGE SECTION 01 SOC-FUNCTION 01 S 01 COMMAND 01 REQARG 01 ERRNO 01 RETCODE	PIC X(16) VALUE IS 'FCNTL'. PIC 9(4) BINARY. PIC 9(8) BINARY. PIC 9(8) BINARY. PIC 9(8) BINARY. PIC 99(8) BINARY.
PROCEDURE DIVISION CALL 'EZASOKET' USING S ERRNO R	OC-FUNCTION S COMMAND REQARG ETCODE.

Figure 20. FCNTL call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing FCNTL. The field is left-aligned and padded on the right with blanks.

S

A halfword binary number specifying the socket descriptor for the socket that you want to unblock or query.

COMMAND

A fullword binary number with the following values:

Value

Description

3

Query the blocking mode of the socket.

4

Set the mode to blocking or nonblocking for the socket.

REQARG

A fullword binary field containing a mask that TCP/IP uses to set the FNDELAY flag.

- If COMMAND is set to 3 ('query') the REQARG field should be set to 0.
- If COMMAND is set to 4 ('set')
 - Set REQARG to 4 to turn the FNDELAY flag on. This places the socket in nonblocking mode.
 - Set REQARG to 0 to turn the FNDELAY flag off. This places the socket in blocking mode.

Parameter values returned to the application

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See <u>Appendix A</u>, <u>"Return codes," on page 267</u> for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values.

- If COMMAND was set to 3 (query), a bit string is returned.
 - If RETCODE contains X'00000004', the socket is nonblocking. (The FNDELAY flag is on.)
 - If RETCODE contains X'0000000', the socket is blocking. (The FNDELAY flag is off.)
- If COMMAND was set to 4 (set), a successful call is indicated by 0 in this field. In both cases, a RETCODE of -1 indicates an error (check the ERRNO field for the error number).

FREEADDRINFO

The FREEADDRINFO call frees all the address information structures returned by GETADDRINFO in the RES parameter.

Table 11. FREEADDRINFO call requirements		
Condition	Requirement	
Authorization:	Supervisor state or problem state, any PSW key.	
Dispatchable unit mode:	Task.	
Cross memory mode:	PASN = HASN.	
Amode:	31-bit or 24-bit.	
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.	
ASC mode:	Primary address space control (ASC) mode.	
Interrupt status:	Enabled for interrupts.	
Locks:	Unlocked.	
Control parameters:	All parameters must be addressable by the caller and in the primary address space.	

Figure 21 on page 68 shows an example of FREEADDRINFO call instructions.

WORKING	-STORAGE SECTI	DN.	'FREEADDRINFO'
01	SOC-FUNCTION	PIC X(16) VALUE IS	
01	ADDRINFO	PIC 9(8) BINARY.	
01	ERRNO	PIC 9(8) BINARY.	
01	RETCODE	PIC S9(8) BINARY.	
PROCEDU CALL 'E	RE DIVISION. ZASOKET' USING ERRNO	SOC-FUNCTION ADDRINFO RETCODE.	

Figure 21. FREEADDRINFO call instruction example

Parameter values set by the application

Keyword

Description

SOC-FUNCTION

A 16-byte character field containing FREEADDRINFO. The field is left-aligned and padded on the right with blanks.

ADDRINFO

Input parameter. The address of a set of address information structures returned by the GETADDRINFO RES argument.

Parameter values returned to the application

Keyword

Description

ERRNO

Output parameter. A fullword binary field. If **RETCODE** is negative, **ERRNO** contains a valid error number. Otherwise, ignore the **ERRNO** field.

See Appendix A, "Return codes," on page 267 for information about **ERRNO** return codes.

RETCODE

Output parameter. A fullword binary field that returns one of the following values:

```
Value
```

Description

0

Successful call.

-1

Check **ERRNO** for an error code.

GETADDRINFO

The GETADDRINFO call translates either the name of a service location (for example, a host name), a service name, or both, and returns a set of socket addresses and associated information to be used in creating a socket with which to address the specified service or sending a datagram to the specified service.

Table 12. GETADDRINFO call requirements		
Condition	Requirement	
Authorization:	Supervisor state or problem state, any PSW key.	
Dispatchable unit mode:	Task.	
Cross memory mode:	PASN = HASN.	
Amode:	31-bit or 24-bit.	
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.	
ASC mode:	Primary address space control (ASC) mode.	
Interrupt status:	Enabled for interrupts.	
Locks:	Unlocked.	
Control parameters:	All parameters must be addressable by the caller and in the primary address space.	

Figure 22 on page 70 shows an example of GETADDRINFO call instructions.

WURKING-STURAGE SECTION 01 SOC-FUNCTION 01 NODE 01 NODELEN 01 SERVICE 01 SERVLEN 01 AI-PASSIVE 01 AI-CANONNAMEOK 01 AI-NUMERICHOST 01 AI-NUMERICSERV 01 AI-NUMERICSERV 01 AI-V4MAPPED 01 AI-ALL 01 AI-ADDRCONFIG 01 AI-ALL 01 AI-ADDRCONFIG 01 AI-EXTFLAGS 01 HINTS 01 RES 01 CANNLEN 01 ERRNO 01 RETCODE	<pre>N. PIC X(16) VALUE IS 'GETADDRINFO'. PIC X(255). PIC 9(8) BINARY. PIC 7(32). PIC 9(8) BINARY VALUE 1. PIC 9(8) BINARY VALUE 1. PIC 9(8) BINARY VALUE 2. PIC 9(8) BINARY VALUE 4. PIC 9(8) BINARY VALUE 8. PIC 9(8) BINARY VALUE 16. PIC 9(8) BINARY VALUE 16. PIC 9(8) BINARY VALUE 32. PIC 9(8) BINARY VALUE 4. PIC 9(8) BINARY VALUE 4. PIC 9(8) BINARY VALUE 128. USAGE IS POINTER. USAGE IS POINTER. PIC 9(8) BINARY. PIC 9(8) BINARY. PIC 9(8) BINARY.</pre>
LINKAGE SECTION. 01 HINTS-ADDRINFO 03 FLAGS 03 AF 03 SOCTYPE 03 PROTO 03 FILLER 03 FILLER 03 FILLER 03 FILLER 03 FILLER 03 FILLER 03 FILLER 03 FILLER 03 FILLER 03 FLAGS 01 RES-ADDRINFO. 03 FLAGS 01 RES-ADDRINFO. 03 FLAGS 03 AF 03 SOCTYPE 03 PROTO 03 NAMELEN 03 FILLER 03 FILLER 03 FILLER 03 FILLER 03 FILLER 03 FILLER 03 NAME 03 FILLER 03 NAME 03 FILLER 03 NAME 03 FILLER 03 NAME 03 FILLER 03 NEXT 03 FILLER	PIC 9(8) BINARY. PIC 9(8) BINARY. PIC 9(8) BINARY. PIC 9(8) BINARY. PIC 9(8) BINARY. PIC X(4). PIC X(4). PIC 9(8) BINARY. PIC 10(4). USAGE IS POINTER. PIC X(4). USAGE IS POINTER. PIC X(4). USAGE IS POINTER. PIC 9(8) BINARY.
PROCEDURE DIVISION. MOVE 'www.hostr MOVE 16 TO HOS MOVE 'TELNET' MOVE 6 TO SERVI SET HINTS TO AI CALL 'EZASOKET RES CANN	name.com' TO NODE. TLEN. TO SERVICE. LEN. DDRESS OF HINTS-ADDRINFO. ' USING SOC-FUNCTION NODE NODELEN SERVICE SERVLEN HINTS LEN ERRNO RETCODE.

Figure 22. GETADDRINFO call instruction example

Parameter values set by the application

Keyword

Description

SOC-FUNCTION

A 16-byte character field containing GETADDRINFO. The field is left-aligned and padded on the right with blanks.

NODE

An input parameter. Storage up to 255 bytes long that contains the host name being queried. If the AI-NUMERICHOST flag is specified in the storage pointed to by the HINTS field, then NODE should contain the IP address of the queried host in presentation form. This is an optional field but if

specified you must also code NODELEN. The NODE name being queried will consist of up to NODELEN or up to the first binary 0.

You can append scope information to the host name, using the format *node%scope information*. The combined information must be 255 bytes or less. For more information, see <u>z/OS Communications</u> Server: IPv6 Network and Application Design Guide.

NODELEN

An input parameter. A fullword binary field set to the length of the host name specified in the NODE field and should not include extraneous blanks. This is an optional field but if specified you must also code NODE.

SERVICE

An input parameter. Storage up to 32 bytes long that contains the service name being queried. If the AI-NUMERICSERV flag is specified in the storage pointed to by the HINTS field, then SERVICE should contain the queried port number in presentation form. This is an optional field but if specified you must also code SERVLEN. The SERVICE name being queried will consist of up to SERVLEN or up to the first binary 0.

SERVLEN

An input parameter. A fullword binary field set to the length of the service name specified in the SERVICE field and should not include extraneous blanks. This is an optional field but if specified you must also code SERVICE.

HINTS

An input parameter. If the HINTS argument is specified, it contains the address of an addrinfo structure containing input values that might direct the operation by providing options and limiting the returned information to a specific socket type, address family, or protocol. If the HINTS argument is not specified, then the information returned will be as if it referred to a structure containing the value 0 for the FLAGS, SOCTYPE and PROTO fields, and AF_UNSPEC for the AF field. Include the EZBREHST resolver macro so that your assembler program will contain the assembler mappings for the ADDR_INFO structure. The EZBREHST assembler macro is stored in the SYS1.MACLIB library. The macro defines the resolver hostent (host entry), address information (addrinfo) mappings, and services return codes. Copy definitions from the EZACOBOL sample module to your COBOL program for mapping the ADDRINFO structure. The EZACOBOL sample module is stored in the *hlq*.SEZAINST library. Copy definitions from the CBLOCK sample module to your PL/I program for mapping the ADDRINFO structure. The CBLOCK sample module is stored in *hlq*.SEZAINST library.

This is an optional field.

The address information structure has the following fields:

Field

Description

FLAGS

A fullword binary field. Must have the value of 0 or the bitwise OR of one or more of the following values:

AI-PASSIVE (X'00000001') or the decimal value 1.

- Specifies how to fill in the NAME pointed to by the returned RES.
- If this flag is specified, then the returned address information will be suitable for use in binding a socket for accepting incoming connections for the specified service (for example, the BIND call). In this case, if the NODE argument is not specified, then the IP address portion of the socket address structure pointed to by the returned RES will be set to INADDR_ANY for an IPv4 address or to the IPv6 unspecified address (in6addr_any) for an IPv6 address.
- If this flag is not set, the returned address information will be suitable for the CONNECT call (for a connection-mode protocol) or for a CONNECT, SENDTO, or SENDMSG call (for a connectionless protocol). In this case, if the NODE argument is not specified, then the IP address portion of the socket address structure pointed to by the returned RES will be set

to the default loopback address for an IPv4 address (127.0.0.1) or the default loopback address for an IPv6 address (::1).

• This flag is ignored if the NODE argument is specified.

AI-CANONNAMEOK (X'0000002') or the decimal value 2.

• If this flag is specified and the NODE argument is specified, then the GETADDRINFO call attempts to determine the canonical name corresponding to the NODE argument.

AI-NUMERICHOST (X'00000004') or the decimal value 4.

• If this flag is specified then the NODE argument must be a numeric host address in presentation form. Otherwise, an error of host not found [EAI_NONAME] is returned.

AI-NUMERICSERV (X'00000008') or the decimal value 8.

• If this flag is specified, the SERVICE argument must be a numeric port in presentation form. Otherwise, an error [EAI_NONAME] is returned.

AI-V4MAPPED (X'00000010') or the decimal value 16.

- If this flag is specified along with the AF field with the value of AF_INET6 or a value of AF_UNSPEC when IPv6 is supported, the caller accepts IPv4-mapped IPv6 addresses.
 - If the AF field is AF_INET6, a query for IPv4 addresses is made if the AI-ALL flag is specified or if no IPv6 addresses are found. Any IPv4 addresses that are found are returned as IPv4-mapped IPv6 addresses.
 - If the AF field is AF_UNSPEC, queries are made for both IPv6 and IPv4 addresses. If IPv4 addresses are found and IPv6 is supported, the IPv4 addresses are returned as IPv4-mapped IPv6 addresses.
- Otherwise, this flag is ignored.

AI-ALL (X'0000020') or the decimal value 32.

- If the AF field has a value of AF_INET6 and AI-ALL is set, the AI-V4MAPPED flag must also be set to indicate that the caller will accept all addresses: IPv6 and IPv4-mapped IPv6 addresses.
- If the AF field has a value of AF_UNSPEC, AI-ALL is accepted, but has no impact on the processing. No matter if AI-ALL is specified or not, the caller accepts both IPv6 and IPv4 addresses. A query is first made for IPv6 addresses and if successful, the IPv6 addresses are returned. Another query is then made for IPv4 addresses:
 - If AI-V4MAPPED is also specified and the system supports IPv6, the IPv4 addresses are returned as IPv4-mapped IPv6 addresses.
 - If AI-V4MAPPED is not specified or the system does not support IPv6, the IPv4 addresses are returned.
- Otherwise, this flag is ignored.

AI-ADDRCONFIG (X'00000040') or the decimal value 64.

If this flag is specified, then a query on the name in NODE will occur if the Resolver determines whether either of the following values is true:

- If the system is IPv6 enabled and has at least one IPv6 interface, then the Resolver will make a query for IPv6 (AAAA or A6 DNS) records.
- If the system is IPv4 enabled and has at least one IPv4 interface, then the Resolver will make a query for IPv4 (A DNS) records.

The loopback address is not considered in this case as a valid interface.

AI-EXTFLAGS (X'0000080') or the decimal value 128.

Specifies this flag to request the extended form of the getaddrinfo function. The extended form allows additional hints to be passed to the resolver for determining the order of destination addresses that are returned. If this flag is specified, the EFLAGS field is required.

Tip: To perform the binary OR'ing of the flags above in a COBOL program, simply add the necessary COBOL statements as in the example below. Note that the value of the FLAGS field after the COBOL ADD is a decimal 80 or an X'00000050', which is the sum of OR'ing AI_V4MAPPED and AI_ADDRCONFIG or X'00000010' and X'00000040':

```
01 AI-V4MAPPED PIC 9(8) BINARY VALUE 16.
01 AI-ADDRCONFIG PIC 9(8) BINARY VALUE 64.
ADD AI-V4MAPPED TO FLAGS.
ADD AI-ADDRCONFG TO FLAGS.
```

AF

A fullword binary field. Used to limit the returned information to a specific address family. The value of AF_UNSPEC means that the caller will accept any protocol family. The value of a decimal 0 indicates AF_UNSPEC. The value of a decimal 2 indicates AF_INET, and the value of a decimal 19 indicates AF_INET6.

SOCTYPE

A fullword binary field. Used to limit the returned information to a specific socket type. A value of 0 means that the caller will accept any socket type. If a specific socket type is not given (for example, a value of 0) then information on all supported socket types will be returned.

The following table shows the acceptable socket types:

Type name	Decimal value	Description
SOCK_STREAM	1	for stream socket
SOCK_DGRAM	2	for datagram socket
SOCK_RAW	3	for raw-protocol interface

Anything else will fail with return code EAI_SOCTYPE. Note that although SOCK_RAW will be accepted, it will be valid only when SERVICE is numeric (for example, SERVICE=23). A lookup for a SERVICE name will never occur in the appropriate services file (for example, *hlq*.ETC.SERVICES) using any protocol value other than SOCK_STREAM or SOCK_DGRAM.

If PROTO is not 0 and SOCTYPE is 0, then the only acceptable input values for PROTO are IPPROTO_TCP and IPPROTO_UDP. Otherwise, the GETADDRINFO call will be failed with return code of EAI_BADFLAGS.

If SOCTYPE and PROTO are both specified as 0, then GETADDRINFO will proceed as follows:

- If SERVICE is null, or if SERVICE is numeric, then any returned addrinfos will default to a specification of SOCTYPE as SOCK_STREAM.
- If SERVICE is specified as a service name (for example, SERVICE=FTP), the GETADDRINFO call will search the appropriate services file (for example, *hlq*.ETC.SERVICES) twice. The first search will use SOCK_STREAM as the protocol, and the second search will use SOCK_DGRAM as the protocol. No default socket type provision exists in this case.

If both SOCTYPE and PROTO are specified as nonzero, then they should be compatible, regardless of the value specified by SERVICE. In this context, *compatible* means one of the following values:

- SOCTYPE=SOCK_STREAM and PROTO=IPPROTO_TCP
- SOCTYPE=SOCK_DGRAM and PROTO=IPPROTO_UDP
- SOCTYPE is specified as SOCK_RAW, in which case PROTO can be anything

PROTO

A fullword binary field. Used to limit the returned information to a specific protocol. A value of 0 means that the caller will accept any protocol.

The following table shows the acceptable protocols:

Protocol name	Decimal value	Description
IPPROTO_TCP	6	ТСР
IPPROTO_UDP	17	user datagram

If SOCTYPE is 0 and PROTO is nonzero, the only acceptable input values for PROTO are IPPROTO_TCP and IPPROTO_UDP. Otherwise, the GETADDRINFO call will be failed with return code of EAI_BADFLAGS.

If PROTO and SOCTYPE are both specified as 0, then GETADDRINFO will proceed as follows:

- If SERVICE is null, or if SERVICE is numeric, then any returned addrinfos will default to a specification of SOCTYPE as SOCK_STREAM.
- If SERVICE is specified as a service name (for example, SERVICE=FTP), the GETADDRINFO will search the appropriate services file (for example, *hlq*.ETC.SERVICE) twice. The first search will use SOCK_STREAM as the protocol, and the second search will use SOCK_DGRAM as the protocol. No default socket type provision exists in this case.

If both PROTO and SOCTYPE are specified as nonzero, they should be compatible, regardless of the value specified by SERVICE. In this context, *compatible* means one of the following values:

- SOCTYPE=SOCK_STREAM and PROTO=IPPROTO_TCP
- SOCTYPE=SOCK_DGRAM and PROTO=IPPROTO_UDP
- SOCTYPE=SOCK_RAW, in which case PROTO can be anything

If the lookup for the value specified in SERVICE fails [for example, the service name does not appear in an appropriate service file (such as, *hlq*.ETC.SERVICES) using the input protocol], then the GETADDRINFO call will be failed with return code of EAI_SERVICE.

NAMELEN

A fullword binary field followed by 8 padding bytes. On input, this field must be 0.

CANONNAME

A fullword binary field followed by 4 padding bytes. On input, this field must be 0.

NAME

A fullword binary field followed by 4 padding bytes. On input, this field must be 0.

NEXT

A fullword binary field. On input, this field must be 0.

EFLAGS

A fullword binary field that specifies the source IPv6 address selection preferences. This field is required if the value AI_EXTFLAGS is specified in the FLAGS field.

This field must contain the value 0 or the bitwise OR of one or more of the following values:

IPV6_PREFER_SRC_HOME (X'00000001') or the decimal value 1

Indicates that home source IPv6 addresses are preferred over care-of source IPv6 addresses.

IPV6_PREFER_SRC_COA (X'00000002') or the decimal value 2

Indicates that care-of source IPv6 addresses are preferred over home source IPv6 addresses.

IPV6_PREFER_SRC_TMP (X'00000004') or the decimal value 4

Indicates that temporary source IPv6 addresses are preferred over public source IPv6 addresses.

IPV6_PREFER_SRC_PUBLIC (X'00000008') or the decimal value 8

Indicates that public source IPv6 addresses are preferred over temporary source IPv6 addresses.

IPV6_PREFER_SRC_CGA (X'00000010') or the decimal value 16

Indicates that cryptographically generated source IPv6 addresses are preferred over noncryptographically generated source IPv6 addresses.

IPV6_PREFER_SRC_NONCGA (X'00000020') or the decimal value 32

Indicates that non-cryptographically generated source IPv6 addresses are preferred over cryptographically generated source IPv6 addresses.

Guidelines:

- If contradictory EFLAGS (for example, IPV6_PREFER_SRC_TMP and IPV6_PREFER_SRC_PUBLIC) or invalid EFLAGS (for example, X'0000040' or the decimal value 64) are specified, then the GETADDRINFO call fails with RETCODE -1 and ERRNO EAI_BADEXTFLAGS (decimal value 11).
- The COBOL constants for EFLAGS use hyphens instead of underscores.

RES

Initially a fullword binary field. On a successful return, this field contains a pointer to a chain of one or more address information structures. The structures are allocated in the key of the calling application. The structures that are returned on a GETADDRINFO call are serially reusable storage for the z/OS UNIX process. They can be used or referenced between process threads, but should not be used or referenced between processes. When you finish using the structures, explicitly release their storage by specifying the returned pointer on a FREEADDRINFO call. Include the EZBREHST resolver macro so that your assembler program contains the assembler mappings for the ADDR_INFO structure. The EZBREHST assembler macro is stored in the SYS1.MACLIB library. Copy definitions from the EZACOBOL sample module to your COBOL program for mapping the ADDRINFO structure. The EZACOBOL sample module is stored in the *hlq*.SEZAINST library. Copy definitions from the CBLOCK sample module to your PL/I program for mapping the ADDRINFO structure. The CBLOCK sample module is stored in the *hlq*.SEZAINST library.

The address information structure contains the following fields:

Field

Description

FLAGS

A fullword binary field that is not used as output.

AF

A fullword binary field. The value returned in this field can be used as the AF argument on the SOCKET call to create a socket suitable for use with the returned address NAME.

SOCTYPE

A fullword binary field. The value returned in this field can be used as the SOCTYPE argument on the SOCKET call to create a socket suitable for use with the returned address NAME.

PROTO

A fullword binary field. The value returned in this field can be used as the PROTO argument on the SOCKET call to create a socket suitable for use with the returned address ADDR.

NAMELEN

A fullword binary field followed by 8 padding bytes. The length of the NAME socket address structure.

CANONNAME

A fullword binary field followed by 4 padding bytes. The canonical name for the value specified by NODE. If the NODE argument is specified, and if the AI-CANONNAMEOK flag was specified by the HINTS argument, then the CANONNAME field in the first returned address information structure will contain the address of storage containing the canonical name corresponding to the input NODE argument. If the canonical name is not available, then the CANONNAME field will refer to the NODE argument or a string with the same contents. The CANNLEN field contains the length of the returned canonical name.

NAME

A fullword binary field followed by 4 padding bytes. The address of the returned socket address structure. The value returned in this field can be used as the arguments for the CONNECT, BIND, or BIND2ADDRSEL call with such a socket, according to the AI-PASSIVE flag.

NEXT

A fullword binary field. Contains the address of the next address information structure on the list, or 0's if it is the last structure on the list.

EFLAGS

A fullword binary field that is not used as output.

CANNLEN

Initially an input parameter. A fullword binary field used to contain the length of the canonical name returned by the RES CANONNAME field. This is an optional field.

ERRNO

Output parameter. A fullword binary field. If RETCODE is negative, ERRNO contains a valid error number. Otherwise, ignore the ERRNO field.

See Appendix A, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

Output parameter. A fullword binary field that returns one of the following values:

Value

Description

0

Successful call.

-1

Check ERRNO for an error code.

The ADDRINFO structure uses indirect addressing to return a variable number of NAMES. If you are coding in PL/I or assembly language, this structure can be processed in a relatively straight-forward manner. If you are coding in COBOL, this structure might be difficult to interpret. You can use the subroutine EZACIC09 to simplify interpretation of the information returned by the GETADDRINFO calls.

GETCLIENTID

GETCLIENTID call returns the identifier by which the calling application is known to the TCP/IP address space in the calling program. The CLIENT parameter is used in the GIVESOCKET and TAKESOCKET calls. See "GIVESOCKET" on page 111 for a discussion of the use of GIVESOCKET and TAKESOCKET calls.

Do not be confused by the terminology; when GETCLIENTID is called by a server, the identifier of the *caller* (not necessarily the *client*) is returned.

Table 13. GETCLIENTID call requirements		
Condition	Requirement	
Authorization:	Supervisor state or problem state, any PSW key.	
Dispatchable unit mode:	Task.	
Cross memory mode:	PASN = HASN.	
Amode:	31-bit or 24-bit.	
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.	
ASC mode:	Primary address space control (ASC) mode.	
Interrupt status:	Enabled for interrupts.	
Locks:	Unlocked.	
Control parameters:	All parameters must be addressable by the caller and in the primary address space.	

Figure 23 on page 77 shows an example of GETCLIENTID call instructions.

```
WORKING-STORAGE SECTION.
    01 SOC-FUNCTION
                         PIC X(16) VALUE IS 'GETCLIENTID'.
    01 CLIENT.
        03 DOMAIN
                         PIC 9(8) BINARY.
        03 NAME
                         PIC X(8).
        03
            TASK
                         PIC X(8).
        03 RESERVED
                         PIC X(20).
                         PIC 9(8) BINARY.
PIC S9(8) BINARY.
       ERRNO
    01
    01 RETCODE
PROCEDURE DIVISION.
     CALL 'EZASOKET' USING SOC-FUNCTION CLIENT ERRNO RETCODE.
```

Figure 23. GETCLIENTID call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing GETCLIENTID. The field is left-aligned and padded to the right with blanks.

Parameter values returned to the application

CLIENT

A client-ID structure that describes the application that issued the call.

DOMAIN

This is a fullword binary number specifying the domain of the client. On input this is an optional parameter for AF_INET, and required parameter for AF_INET6 to specify the domain of the client. For TCP/IP the value is a decimal 2, indicating AF_INET, or a decimal 19, indicating AF_INET6. On output, this is the returned domain of the client.

NAME

An 8-byte character field set to the caller's address space name.

TASK

An 8-byte field set to the task identifier of the caller.

RESERVED

Specifies a 20-byte character reserved field. This field is required, but not used.

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See <u>Appendix A</u>, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

0

Successful call.

-1

Check ERRNO for an error code.

GETHOSTBYADDR

The GETHOSTBYADDR call returns the domain name and alias name of a host whose IPv4 IP address is specified in the call. A given TCP/IP host can have multiple alias names and multiple host IPv4 IP addresses. The address resolution attempted depends on how the resolver is configured and if any

local host tables exist. See <u>z/OS</u> Communications Server: IP Configuration Guide for information about configuring the resolver and how local host tables can be used.

	·	
Condition	Requirement	
Authorization:	Supervisor state or problem state. The PSW key must match the key in which the MVS application task was attached	
Dispatchable unit mode:	Task.	
Cross memory mode:	PASN = HASN.	
Amode:	31-bit or 24-bit.	
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.	
ASC mode:	Primary address space control (ASC) mode.	
Interrupt status:	Enabled for interrupts.	
Locks:	Unlocked.	
Control parameters:	All parameters must be addressable by the caller and in the primary address space.	

Table 14. GETHOSTBYADDR call requirements

Figure 24 on page 78 shows an example of GETHOSTBYADDR call instructions.

WORKING-STORAGE SECTION. 01 SOC-FUNCTION PIC X(16) VALUE IS 'GETHOSTBYADDR'. 01 HOSTADDR PIC 9(8) BINARY. 01 HOSTENT PIC 9(8) BINARY. 01 RETCODE PIC S9(8) BINARY. PROCEDURE DIVISION. CALL 'EZASOKET' USING SOC-FUNCTION HOSTADDR HOSTENT RETCODE.



For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing GETHOSTBYADDR. The field is left-aligned and padded on the right with blanks.

HOSTADDR

A fullword binary field set to the IP address (specified in network byte order) of the host whose name is being sought. See <u>Appendix A</u>, "Return codes," on page 267 for information about ERRNO return codes.

Parameter values returned to the application

HOSTENT

A fullword containing the address of the HOSTENT structure.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

0

Successful call.

-1

Check **ERRNO** for an error code.

GETHOSTBYADDR returns the HOSTENT structure shown in Figure 25 on page 79.



Figure 25. HOSTENT structure that is returned by the GETHOSTBYADDR call

GETHOSTBYADDR returns the HOSTENT structure shown in figure Figure 25 on page 79. The HOSTENT structure is a tasks's serially reusable storage area. It should not be used or referenced between MVS tasks. The storage is freed when the task terminates. The assembler mapping of the structure is defined in macro EZBREHST, which is installed in the data set specified on your SMP/E DDDEF for MACLIB. The EZBREHST assembler macro is stored in the SYS1.MACLIB library. The macro defines the resolver hostent (host entry), address information (addrinfo) mappings, and services return codes. This structure contains:

- The address of the host name that is returned by the call. The name length is variable and is ended by X'00'.
- The address of a list of addresses that point to the alias names returned by the call. This list is ended by the pointer X'00000000'. Each alias name is a variable length field ended by X'00'.
- The value returned in the FAMILY field is always 2 for AF_INET.
- The length of the host IP address returned in the HOSTADDR_LEN field is always 4 for AF_INET.

• The address of a list of addresses that point to the host IP addresses returned by the call. The list is ended by the pointer X'00000000'. If the call cannot be resolved, the HOSTENT structure contains the ERRNO 10214.

The HOSTENT structure uses indirect addressing to return a variable number of alias names and IP addresses. If you are coding in PL/I or assembly language, this structure can be processed in a relatively straight-forward manner. If you are coding in COBOL, this structure might be difficult to interpret. You can use the subroutine EZACIC08 to simplify interpretation of the information returned by the GETHOSTBYADDR and GETHOSTBYNAME calls. For more information about EZACIC08, see <u>"EZACIC08"</u> on page 192.

GETHOSTBYNAME

The GETHOSTBYNAME call returns the alias name and the IPv4 IP address of a host whose domain name is specified in the call. A given TCP/IP host can have multiple alias names and multiple host IPv4 IP addresses.

The name resolution attempted depends on how the resolver is configured and if any local host tables exist. See <u>z/OS</u> Communications Server: IP Configuration Guide for information about configuring the resolver and how local host tables can be used.

Tuble 13. de most biname cui requirements	
Condition	Requirement
Authorization:	Supervisor state or problem state. The PSW key must match the key in which the MVS application task was attached.
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN.
Amode:	31-bit or 24-bit.
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.
ASC mode:	Primary address space control (ASC) mode.
Interrupt status:	Enabled for interrupts.
Locks:	Unlocked.
Control parameters:	All parameters must be addressable by the caller and in the primary address space

Table 15. GETHOSTBYNAME call requirements

Figure 26 on page 80 shows an example of GETHOSTBYNAME call instructions.

WORKING-STORAGE SEC	FION.
01 SOC-FUNCTION	N PIC X(16) VALUE IS 'GETHOSTBYNAME
01 NAMELEN	PIC 9(8) BINARY.
01 NAME	PIC X(255).
01 HOSTENT	PIC 9(8) BINARY.
01 RETCODE	PIC S9(8) BINARY.
PROCEDURE DIVISION. CALL 'EZASOKET	USING SOC-FUNCTION NAMELEN NAME
	HOSTENT RETCODE.

Figure 26. GETHOSTBYNAME call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.
Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing GETHOSTBYNAME. The field is left-aligned and padded on the right with blanks.

NAMELEN

A value set to the length of the host name. The maximum length is 255.

NAME

A character string, up to 255 characters, set to a host name. Any trailing blanks will be removed from the specified name prior to trying to resolve it to an IP address. This call returns the address of the HOSTENT structure for this name.

Parameter values returned to the application

HOSTENT

A fullword binary field that contains the address of the HOSTENT structure.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

0

Successful call.

-1

An error occurred.



Figure 27. HOSTENT structure returned by the GETHOSTYBYNAME call

GETHOSTBYNAME returns the HOSTENT structure shown in Figure 27 on page 82. The HOSTENT structure is a tasks's serially reusable storage area. It should not be used or referenced between MVS tasks. The storage is freed when the task terminates. The assembler mapping of the structure is defined in macro EZBREHST, which is installed in the data set specified on your SMP/E DDDEF for MACLIB. The EZBREHST assembler macro is stored in the SYS1.MACLIB library. The macro defines the resolver hostent (host entry), address information (addrinfo) mappings, and services return codes. This structure contains:

- The address of the host name that is returned by the call. The name length is variable and is ended by X'00'.
- The address of a list of addresses that point to the alias names returned by the call. This list is ended by the pointer X'00000000'. Each alias name is a variable length field ended by X'00'.
- The value returned in the FAMILY field is always 2 for AF_INET.
- The length of the host IP address returned in the HOSTADDR_LEN field is always 4 for AF_INET.
- The address of a list of addresses that point to the host IP addresses returned by the call. The list is ended by the pointer X'00000000'. If the call cannot be resolved, the HOSTENT structure contains the ERRNO 10214.

The HOSTENT structure uses indirect addressing to return a variable number of alias names and IP addresses. If you are coding in PL/I or assembly language, this structure can be processed in a relatively straight-forward manner. If you are coding in COBOL, this structure might be difficult to interpret. You can use the subroutine EZACIC08 to simplify interpretation of the information returned by the GETHOSTBYADDR and GETHOSTBYNAME calls. For more information about EZACIC08, see <u>"EZACIC08"</u> on page 192.

GETHOSTID

The GETHOSTID call returns the 32-bit IP address for the current host.

Table 16. GETHOSTID call requirements		
Condition	Requirement	
Authorization:	Supervisor state or problem state, any PSW key.	
Dispatchable unit mode:	Task.	
Cross memory mode:	PASN = HASN.	
Amode:	31-bit or 24-bit.	
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.	
ASC mode:	Primary address space control (ASC) mode.	
Interrupt status:	Enabled for interrupts.	
Locks:	Unlocked.	
Control parameters:	All parameters must be addressable by the caller and in the primary address space.	

Figure 28 on page 83 shows an example of GETHOSTID call instructions.

```
WORKING-STORAGE SECTION.

01 SOC-FUNCTION PIC X(16) VALUE IS 'GETHOSTID'.

01 RETCODE PIC S9(8) BINARY.

PROCEDURE DIVISION.

CALL 'EZASOKET' USING SOC-FUNCTION RETCODE.
```

Figure 28. GETHOSTID call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing GETHOSTID. The field is left-aligned and padded on the right with blanks.

RETCODE

Returns a fullword binary field containing the 32-bit IP address of the host. There is no ERRNO parameter for this call.

GETHOSTNAME

The GETHOSTNAME call returns the domain name of the local host.

Note: The host name returned is the host name the TCPIP stack learned at startup from the TCPIP.DATA file that was found.

Condition	Requirement	
Authorization:	Supervisor state or problem state, any PSW key.	
Dispatchable unit mode:	Task.	

Condition	Requirement	
Cross memory mode:	PASN = HASN.	
Amode:	31-bit or 24-bit.	
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.	
ASC mode:	Primary address space control (ASC) mode.	
Interrupt status:	Enabled for interrupts.	
Locks:	Unlocked.	
Control parameters:	All parameters must be addressable by the caller and in the primary address space.	

Figure 29 on page 84 shows an example of GETHOSTNAME call instructions.

WORKING-STORAGE SECTION. 01 SOC-FUNCTION PIC X(16) VALUE IS 'GETHOSTNAME'. 01 NAMELEN PIC 9(8) BINARY. 01 NAME PIC X(24). 01 ERRNO PIC 9(8) BINARY. 01 RETCODE PIC S9(8) BINARY. PROCEDURE DIVISION. CALL 'EZASOKET' USING SOC-FUNCTION NAMELEN NAME ERRNO RETCODE.

Figure 29. GETHOSTNAME call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing GETHOSTNAME. The field is left-aligned and padded on the right with blanks.

NAMELEN

A fullword binary field set to the length of the NAME field. The minimum length of the NAME field is 1 character. The maximum length of the NAME field is 255 characters.

Parameter values returned to the application

NAME

Indicates the receiving field for the host name. If the host name is shorter than the NAMELEN value, the NAME field is filled with binary zeros after the host name. If the host name is longer than the NAMELEN value, the name is truncated.

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See <u>Appendix A</u>, "Return codes," on page 267 for information about ERRNO return codes.

A fully

A fullword binary field that returns one of the following values:

Value

Description

0

Successful call.

-1

Check ERRNO for an error code.

GETIBMOPT

The GETIBMOPT call returns the number of TCP/IP images installed on a given MVS system and their status, versions, and names. With this information, the caller can dynamically choose the TCP/IP image with which to connect by using the INITAPI call. The GETIBMOPT call is optional. If you do not use the GETIBMOPT call, follow the standard method to determine the connecting TCP/IP image:

- Connect to the TCP/IP specified by TCPIPJOBNAME in the active TCPIP.DATA file.
- Locate TCPIP.DATA using the search order described in the z/OS Communications Server: IP Configuration Reference.

Table 17. GETIBMOPT call requirements		
Condition	Requirement	
Authorization:	Supervisor state or problem state, any PSW key.	
Dispatchable unit mode:	Task.	
Cross memory mode:	PASN = HASN.	
Amode:	31-bit or 24-bit.	
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.	
ASC mode:	Primary address space control (ASC) mode.	
Interrupt status:	Enabled for interrupts.	
Locks:	Unlocked.	
Control parameters:	All parameters must be addressable by the caller and in the primary address space.	

Figure 30 on page 85 shows an example of GETIBMOPT call instructions.

WORKING-STORAGE SECTION. 01 SOC-FUNCTION PIC X(16) VALUE IS 'GETIBMOPT'. 01 COMMAND PIC 9(8) BINARY VALUE IS 1. 01 BUF. 03 NUM-IMAGES PIC 9(8) COMP. 03 TCP-IMAGE OCCURS 8 TIMES. 05 TCP-IMAGE-STATUS PIC 9(4) BINARY. 05 TCP-IMAGE-VERSION PIC 9(4) BINARY. 05 TCP-IMAGE-NAME PIC X(8) NO PIC 9(8) BINARY. ERRNO 01 01 RETCODE PIC S9(8) BINARY. PROCEDURE DIVISION. CALL 'EZASOKET' USING SOC-FUNCTION COMMAND BUF ERRNO RETCODE.

Figure 30. GETIBMOPT call instruction example

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing GETIBMOPT. The field is left-aligned and padded on the right with blanks.

COMMAND

A value or the address of a fullword binary number specifying the command to be processed. The only valid value is 1.

Parameter values returned to the application

BUF

A 100-byte buffer into which each active TCP/IP image status, version, and name are placed.

On successful return, these buffer entries contain the status, names, and versions of up to eight active TCP/IP images. The following layout shows the BUF field upon completion of the call.

The NUM_IMAGES field indicates how many entries of TCP_IMAGE are included in the total BUF field. If the NUM_IMAGES returned is 0, there are no TCP/IP images present.

The status field can have a combination of the following information:

Sta	tus	field	
	Me	aning	5

X'8xxx' Active

X'4xxx'

Terminating

X'2xxx'

Down

X'1xxx'

Stopped or stopping

Note: In the preceding status fields, xxx is reserved for IBM use and can contain any value.

When the status field is returned with a combination of Down and Stopped, TCP/IP abended. Stopped, when returned alone, indicates that TCP/IP was stopped.

The version field is:

Version	Field
TCP/IP z/OS Communications Server V1R4	X'0614'
TCP/IP z/OS Communications Server V1R5	X'0615'
TCP/IP z/OS Communications Server V1R6	X'0616'
TCP/IP z/OS Communications Server V1R7	X'0617'
TCP/IP z/OS Communications Server V1R8	X'0618'
TCP/IP z/OS Communications Server V1R9	X'0619'
TCP/IP z/OS Communications Server V1R10	X'061A'
TCP/IP z/OS Communications Server V1R11	X'061B'

The name field is the PROC name, left-aligned, and padded with blanks.

NUM_IN (4 b)	VIAGES /tes)	
Status	Version	Name
(2 bytes)	(2 bytes)	(8 bytes)
Status	Version	Name
(2 bytes)	(2 bytes)	(8 bytes)
Status	Version	Name
(2 bytes)	(2 bytes)	(8 bytes)
Status	Version	Name
(2 bytes)	(2 bytes)	(8 bytes)
Status	Version	Name
(2 bytes)	(2 bytes)	(8 bytes)
Status	Version	Name
(2 bytes)	(2 bytes)	(8 bytes)
Status	Version	Name
(2 bytes)	(2 bytes)	(8 bytes)
Status	Version	Name
(2 bytes)	(2 bytes)	(8 bytes)

Figure 31. Example of name field

ERRNO

A fullword binary field. If RETCODE is negative, this field contains an error number. See <u>Appendix A</u>, <u>"Return codes," on page 267</u> for information about ERRNO return codes.

RETCODE

A fullword binary field with the following values:

Value

Description

-1

Call returned error. See ERRNO field.

0

Successful call.

GETNAMEINFO

The GETNAMEINFO call returns the node name and service location of a socket address that is specified in the call. On successful completion, GETNAMEINFO returns the node and service named, if requested, in the buffers provided.

Table 18. GETNAMEINFO call requirements		
Condition Requirement		
Authorization:	Supervisor state or problem state, any PSW key.	
Dispatchable unit mode:	Task.	
Cross memory mode:	PASN = HASN.	

Figure 32. GETNAMEINFO call instruction example

Parameter values set by the application

Keyword Description

SOC-FUNCTION

A 16-byte character field containing GETNAMEINFO. The field is left-aligned and padded on the right with blanks.

NAME

An input parameter. A socket address structure to be translated which has the following fields:

The IPv4 socket address structure must specify the following fields:

Field

Description

FAMILY

A halfword binary number specifying the IPv4 addressing family. For TCP/IP the value is a decimal 2, indicating AF_INET.

PORT

A halfword binary number specifying the port number.

IP-ADDRESS

A fullword binary number specifying the 32-bit IPv4 IP address.

RESERVED

An 8-byte reserved field. This field is required, but is not used.

The IPv6 socket address structure specifies the following fields:

Field

Description

FAMILY

A halfword binary field specifying the IPv6 addressing family. For TCP/IP the value is a decimal 19, indicating AF_INET6.

PORT

A halfword binary number specifying the port number.

FLOWINFO

A fullword binary field specifying the traffic class and flow label. This value of this field is undefined.

IP-ADDRESS

A 16-byte binary field specifying the 128-bit IPv6 IP address, in network byte order.

SCOPE-ID

A fullword binary field that identifies a set of interfaces as appropriate for the scope of the address carried in the IPv6-ADDRESS field. For a link-local scope IPv6-ADDRESS, SCOPE-ID contains the interface index for the IPv6-ADDRESS. For all other address scopes, SCOPE-ID is undefined and is ignored by the resolver.

NAMELEN

An input parameter. A fullword binary field. The length of the socket address structure pointed to by the NAME argument.

HOST

On input, storage capable of holding the returned resolved host name, which can be up to 255 bytes long, for the input socket address. If inadequate storage is specified to contain the resolved host name, then the resolver will return the host name up to the storage specified and truncation might occur. If the host name cannot be located, the numeric form of the host address is returned instead of its name. However, if the NI_NAMEREQD option is specified and no host name is located then an error is returned. HOST is an optional field, but if you specify it, you also must code HOSTLEN. One or both of the following groups of parameters are required:

- The HOST and HOSTLEN parameters
- The SERVICE and SERVLEN parameters

Otherwise, an error occurs.

If the IPv6-ADDRESS value is a link-local address, and the SCOPE-ID interface index is nonzero, scope information is appended to the resolved host name in the format *host%scope information*. The scope information can be either the numeric form of the SCOPE-ID interface index or the interface name associated with the SCOPE-ID interface index. Use the NI_NUMERICSCOPE option to select which form should be returned. The combined host name and scope information will still be at most 255 bytes long. For more information about scope information and GETNAMEINFO processing, see <u>z/OS</u> Communications Server: IPv6 Network and Application Design Guide.

HOSTLEN

An output parameter. A fullword binary field that contains the length of the host storage used to contain the returned resolved host name. The HOSTLEN value must be equal to or greater than the length of the longest host name, or host name and scope information combination, to be returned. The GETNAMEINFO call returns the host name, or host name and scope information combination, up to the length specified by the HOSTLEN value. On output, the HOSTLEN value contains the length of the returned resolved host name or host name and scope information combination. If HOSTLEN is 0 on input, then the resolved host name is not returned. HOSTLEN is an optional field but if specified you must also code the HOST value. One or both of the following groups of parameters are required:

- The HOST and HOSTLEN parameters
- The SERVICE and SERVLEN parameters

Otherwise, an error occurs.

SERVICE

On input, storage capable of holding the returned resolved service name, which can be up to 32 bytes long, for the input socket address. If inadequate storage is specified to contain the resolved service name, then the resolver will return the service name up to the storage specified and truncation might occur. If the service name cannot be located, or if NI_NUMERICSERV was specified in the FLAGS operand, then the numeric form of the service address is returned instead of its name. SERVICE is an optional field, but if you specify it, you also must code the SERVLEN value. One or both of the following groups of parameters are required:

- The HOST and HOSTLEN parameters
- The SERVICE and SERVLEN parameters

Otherwise, an error occurs.

SERVLEN

An output parameter. A fullword binary field. The length of the SERVICE storage used to contain the returned resolved service name. SERVLEN must be equal to or greater than the length of the longest service name to be returned. GETNAMEINFO will return the service name up to the length specified by SERVLEN. On output, SERVLEN will contain the length of the returned resolved service name. If SERVLEN is 0 on input, then the service name information will not be returned. SERVLEN is an optional field, but if you specify it, you also must code the SERVICE value. One or both of the following groups of parameters are required:

- The HOST and HOSTLEN parameters
- The SERVICE and SERVLEN parameters

Otherwise, an error occurs.

FLAGS

An input parameter. A fullword binary field. FLAGS is an optional field. The FLAGS field must contain either a binary value or decimal value, depending on the programming language used:

Flag name	Binary value	Decimal value	Description
'NI_NOFQDN'	X'00000001'	1	Return the NAME portion of the fully qualified domain name.
'NI_NUMERICHOST'	X'00000002'	2	Return only the numeric form of host's address.
'NI_NAMEREQD'	X'00000004'	4	Return an error if the host's name cannot be located.

Flag name	Binary value	Decimal value	Description
'NI_NUMERICSERV'	X'0000008'	8	Return only the numeric form of the service address.
'NI_DGRAM'	X'00000010'	16	Indicates that the service is a datagram service. The default behavior is to assume that the service is a stream service.
'NI_NUMERICSCOPE'	X'00000020'	32	Return only the numeric form of the scope information, when applicable

ERRNO

Output parameter. A fullword binary field. If RETCODE is negative, ERRNO contains a valid error number. Otherwise, ignore the ERRNO field.

See Appendix A, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

Output parameter. A fullword binary field that returns one of the following values:

Value

Description

0

Successful call.

-1

Check **ERRNO** for an error code.

GETPEERNAME

•

The GETPEERNAME call returns the name of the remote socket to which the local socket is connected.

·			
Table 19. GETPEERNAME call requirement			
Condition	Requirement		
Authorization:	Supervisor state or problem state, any PSW key.		
Dispatchable unit mode:	Task.		
Cross memory mode:	PASN = HASN.		
Amode:	31-bit or 24-bit.		
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.		
ASC mode:	Primary address space control (ASC) mode.		
Interrupt status:	Enabled for interrupts.		
Locks:	Unlocked.		
Control parameters: All parameters must be addressable by the caller and in the address space.			

Figure 33 on page 92 shows an example of GETPEERNAME call instructions.

```
WORKING-STORAGE SECTION.
    01 SOC-FUNCTION PIC X(16) VALUE IS 'GETPEERNAME'.
                           PIC 9(4) BINARY.
    01 S
* IPv4 socket structure.
    01 NAME.
         03 FAMILY
                           PIC 9(4) BINARY.
        03 PORT PIC 9(4) BINARY.
03 IP-ADDRESS PIC 9(8) BINARY.
         03 RESERVED PIC X(8).
* IPv6 socket structure.
    01 NAME.
             FAMILY
                           PIC 9(4) BINARY.
         03
         03 PORT
                           PIC 9(4) BINARY.
        03PORTPIC 9(4) BINARY.03FLOWINFOPIC 9(8) BINARY.
         03 IP-ADDRESS.
        10 FILLERPIC 9(16) BINARY.10 FILLERPIC 9(16) BINARY.03 SCOPE-IDPIC 9(8) BINARY.
    01 ERRNO
                           PIC 9(8) BINARY.
                           PIC S9(8) BINARY.
    01 RETCODE
PROCEDURE DIVISION.
     CALL 'EZASOKET' USING SOC-FUNCTION S NAME ERRNO RETCODE.
```



For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing GETPEERNAME. The field is left-aligned and padded on the right with blanks.

S

A halfword binary number set to the socket descriptor of the local socket connected to the remote peer whose address is required.

Parameter Values Returned to the Application

NAME

An IPv4 socket address structure to contain the peer name. The structure that is returned is the socket address structure for the remote socket connected to the local socket specified in field **S**.

FAMILY

A halfword binary field containing the connection peer's IPv4 addressing family. The call always returns the value decimal 2, indicating AF_INET.

PORT

A halfword binary field set to the connection peer's port number.

IP-ADDRESS

A fullword binary field set to the 32-bit IPv4 IP address of the connection peer's host machine.

RESERVED

Specifies an 8-byte reserved field. This field is required, but not used.

An IPv6 socket address structure to contain the peer name. The structure that is returned is the socket address structure for the remote socket that is connected to the local socket specified in field S.

FAMILY

A halfword binary field containing the connection peer's IPv6 addressing family. The call always returns the value decimal 19, indicating AF_INET6.

PORT

A halfword binary field set to the connection peer's port number.

FLOWINFO

A fullword binary field specifying the traffic class and flow label. This value of this field is undefined.

IP-ADDRESS

A 16-byte binary field set to the 128-bit IPv6 IP address of the connection peer's host machine.

SCOPE-ID

A fullword binary field which identifies a set of interfaces as appropriate for the scope of the address carried in the IPv6-ADDRESS field. For a link scope IPv6-ADDRESS, SCOPE-ID contains the link index for the IPv6-ADDRESS. For all other address scopes, SCOPE-ID is undefined.

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix A, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

0

Successful call.

-1

Check ERRNO for an error code.

GETSOCKNAME

The GETSOCKNAME call returns the address currently bound to a specified socket. If the socket is not currently bound to an address, the call returns with the FAMILY field set, and the rest of the structure set to 0.

Because a stream socket is not assigned a name until after a successful call to either BIND, CONNECT, or ACCEPT, the GETSOCKNAME call can be used after an implicit bind to discover which port was assigned to the socket.

Table 20. GETSOCKNAME call requirements		
Condition	Requirement	
Authorization:	Supervisor state or problem state, any PSW key.	
Dispatchable unit mode:	Task.	
Cross memory mode:	PASN = HASN.	
Amode:	31-bit or 24-bit.	
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.	
ASC mode:	Primary address space control (ASC) mode.	
Interrupt status:	Enabled for interrupts.	
Locks:	Unlocked.	
Control parameters:	All parameters must be addressable by the caller and in the primary address space.	

Figure 34 on page 94 shows an example of GETSOCKNAME call instructions.

```
WORKING-STORAGE SECTION.
    01 SOC-FUNCTION PIC X(16) VALUE IS 'GETSOCKNAME'.
    01 S
                           PIC 9(4) BINARY.
* IPv4 socket address structure.
    01 NAME.
         03 FAMILY
                           PIC 9(4) BINARY.
        03 PORT PIC 9(4) BINARY.
03 IP-ADDRESS PIC 9(8) BINARY.
         03 RESERVED PIC X(8).
* IPv6 socket address structure.
    01 NAME.
             FAMILY
                           PIC 9(4) BINARY.
         03
         03 PORT
                           PIC 9(4) BINARY.
         03PORTPIC 9(4) BINARY.03FLOWINFOPIC 9(8) BINARY.
         03 IP-ADDRESS.
        10 FILLER PIC 9(16) BINARY.
10 FILLER PIC 9(16) BINARY.
03 SCOPE-ID PIC 9(8) BINARY.
                          PIC 9(8) BINARY.
    01 ERRNO
                         PIC S9(8) BINARY.
    01 RETCODE
PROCEDURE DIVISION.
     CALL 'EZASOKET' USING SOC-FUNCTION S NAME ERRNO RETCODE.
```

Figure 34. GETSOCKNAME call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing GETSOCKNAME. The field is left-aligned and padded on the right with blanks.

S

A halfword binary number set to the descriptor of a local socket whose address is required.

Parameter values returned to the application

NAME

Specifies the IPv4 socket address structure returned by the call.

FAMILY

A halfword binary field containing the IPv4 addressing family. The call always returns the value decimal 2, indicating AF_INET.

PORT

A halfword binary field set to the port number bound to this socket. If the socket is not bound, 0 is returned.

IP-ADDRESS

A fullword binary field set to the 32-bit IP address of the local host machine.

RESERVED

Specifies 8 bytes of binary zeros. This field is required but not used.

NAME

Specifies the IPv6 socket address structure returned by the call.

FAMILY

A halfword binary field containing the IPv6 addressing family. The call always returns the value decimal 19, indicating AF_INET6.

PORT

A halfword binary field set to the port number bound to this socket. If the socket is not bound, 0 is returned.

FLOWINFO

A fullword binary field specifying the traffic class and flow label. This value of this field is undefined.

IP-ADDRESS

A 16 byte binary field set to the 128-bit IPv6 IP address in network byte order, of the local host machine.

SCOPE-ID

A fullword binary field which identifies a set of interfaces as appropriate for the scope of the address carried in the IPv6-ADDRESS field. For a link scope IPv6-ADDRESS, SCOPE-ID contains the link index for the IPv6-ADDRESS. For all other address scopes, SCOPE-ID is undefined.

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See <u>Appendix A</u>, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

0

Successful call.

-1

Check **ERRNO** for an error code.

GETSOCKOPT

The GETSOCKOPT call queries the options that are set by the SETSOCKOPT call.

Several options are associated with each socket. These options are described in Table 22 on page 96. You must specify the option to be queried when you issue the GETSOCKOPT call.

Condition	Requirement
Authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN.
Amode:	31-bit or 24-bit.
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.
ASC mode:	Primary address space control (ASC) mode.
Interrupt status:	Enabled for interrupts.
Locks:	Unlocked.
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Figure 35 on page 96 shows an example of GETSOCKOPT call instructions.

WORKING-STORAGE SECTION	I.
01 SOC-FUNCTION	PIC X(16) VALUE IS 'GETSOCKOPT'
01 S	PIC 9(4) BINARY.
01 OPTNAME	PIC 9(8) BINARY.
01 OPTVAL	PIC 9(8) BINARY.
If OPNAME = SO-LING	ER then
01 OPTVAL	PIC X(16).
01 OPTLEN	PIC 9(8) BINARY.
01 ERRNO	PIC 9(8) BINARY.
01 RETCODE	PIC S9(8) BINARY.
PROCEDURE DIVISION. CALL 'EZASOKET' US OPT	SING SOC-FUNCTION S OPTNAME VAL OPTLEN ERRNO RETCODE.

Figure 35. GETSOCKOPT call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
IP_ADD_MEMBERSHIP Use this option to enable an application to join a multicast group on a specific interface. An interface has to be specified with this option. Only applications that want to receive multicast datagrams need to join multicast groups. This is an IPv4-only socket option.	Contains the IP_MREQ structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 interface address. See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ. See SEZAINST(EZACOBOL) for the COBOL example of IP- MREQ	N/A
IP_ADD_SOURCE_MEMBERSHIP Use this option to enable an application to join a source multicast group on a specific interface and a specific source address. You must specify an interface and a source address with this option. Applications that want to receive multicast datagrams need to join source multicast groups. This is an IPv4-only socket option.	Contains the IP_MREQ_SOURCE structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ_SOURCE structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 source address and a 4- byte IPv4 interface address. See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE. See SEZAINST(EZACOBOL) for the COBOL example of IP- MREQ-SOURCE.	N/A

Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT

Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)			
OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)	
IP_BLOCK_SOURCE Use this option to enable an application to block multicast packets that have a source address that matches the given IPv4 source address. You must specify an interface and a source address with this option. The specified multicast group must have been joined previously.	Contains the IP_MREQ_SOURCE structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ_SOURCE structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 source address and a 4- byte IPv4 interface address.	N/A	
This is an IPv4-only socket option.	See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE.		
	See SEZAINST(EZACOBOL) for the COBOL example of IP- MREQ-SOURCE.		
IP_DROP_MEMBERSHIP Use this option to enable an application to exit a multicast group or to exit all sources for a multicast group. This is an IPv4-only socket option.	Contains the IP_MREQ structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 interface address.	N/A	
	See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ. See SEZAINST(EZACOBOL) for the COBOL example of IP- MREQ.		
IP_DROP_SOURCE_MEMBERSHIP Use this option to enable an application to exit a source multicast group. This is an IPv4-only socket option.	Contains the IP_MREQ_SOURCE structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ_SOURCE structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 source address and a 4- byte IPv4 interface address.	N/A	
	See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE. See SEZAINST(EZACOBOL) for the COBOL example of IP- MREQ-SOURCE.		

Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)			
OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)	
IP_MULTICAST_IF	A 4-byte binary field containing an IPv4 interface address.	A 4-byte binary field containing an IPv4 interface address.	
Use this option to set or obtain the IPv4 interface address used for sending outbound multicast datagrams from the socket application.			
This is an IPv4-only socket option.			
Note: Multicast datagrams can be transmitted only on one interface at a time.			
IP_MULTICAST_LOOP	A 1-byte binary field.	A 1-byte binary field.	
Use this option to control or determine	To enable, set to 1.	If enabled, will contain a 1.	
whether a copy of multicast datagrams are looped back for multicast datagrams sent to a group to which the sending host itself belongs. The default is to loop the datagrams back.	To disable, set to 0.	If disabled, will contain a 0.	
This is an IPv4-only socket option.			
IP_MULTICAST_TTL	A 1-byte binary field containing	A 1-byte binary field containing	
Use this option to set or obtain the IP time-to-live of outgoing multicast datagrams. The default value is '01'x meaning that multicast is available only to the local subnet.	the value of '00'x to 'FF'x.	the value of '00'x to 'FF'x.	
This is an IPv4-only socket option.			
IP_UNBLOCK_SOURCE	Contains the IP_MREQ_SOURCE structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ_SOURCE structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 source address and a 4- byte IPv4 interface address.		
Use this option to enable an application to unblock a previously blocked source for a given IPv4 multicast group. You must specify an interface and a source address with this option. This is an IPv4-only socket option.			
	See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE.		
	See SEZAINST(EZACOBOL) for the COBOL example of IP- MREQ-SOURCE.		

Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
IPV6_ADDR_PREFERENCES Use this option to query or set IPv6 address preferences of a socket. The default source address selection algorithm considers these preferences when it selects an IP address that is appropriate to communicate with a given destination address.	Contains the 4-byte flags field IPV6_ADDR_PREFERENCES_ FLAGS that is defined in SYS1.MACLIB(BPXYSOCK) with the following flags:	Contains the 4-byte flags field IPV6_ADDR_PREFERENCES_ FLAGS that is defined in SYS1.MACLIB(BPXYSOCK) with the following flags:
	IPV6_PREFER_SRC_HOME (X'00000001') Prefer home address	IPV6_PREFER_SRC_HOME (X'00000001') Prefer home address
This is an AF_INET6-only socket option.	IPV6_PREFER_SRC_COA (X'00000002')	IPV6_PREFER_SRC_COA (X'00000002')
Result: These flags are only preferences. The stack could assign a source IP address that does not conform to the IPV6_ADDR_PREFERENCES flags that you specify.	Prefer care-of address IPV6_PREFER_SRC_TMP (X'00000004') Prefer temporary address	Prefer care-of address IPV6_PREFER_SRC_TMP (X'00000004') Prefer temporary address
Guideline: Use the INET6_IS_SRCADDR function to test whether the source IP address matches one or more IPV6_ADDR_PREFERENCES flags.	IPV6_PREFER_SRC_PUBLIC (X'00000008') Prefer public address	IPV6_PREFER_SRC_PUBLIC (X'0000008') Prefer public address
	IPV6_PREFER_SRC_CGA (X'00000010') Prefer cryptographically generated address	IPV6_PREFER_SRC_CGA (X'00000010') Prefer cryptographically generated address
	IPV6_PREFER_SRC_NONCGA (X'00000020') Prefer non-cryptographically generated address	IPV6_PREFER_SRC_ NONCGA (X'00000020') Prefer non-cryptographically generated address
	Some of these flags are contradictory. Combining contradictory flags, such as IPV6_PREFER_SRC_CGA and IPV6_PREFER_SRC_NONCGA, results in error code EINVAL.	See IPV6_ADDR_ PREFERENCES and Mapping of GAI_HINTS/GAI_ADDRINFO EFLAGS in SEZAINST(CBLOCK) for the PL/I example of the OPTNAME and flag definitions.
	See IPV6_ADDR_PREFERENCES and Mapping of GAI_HINTS/ GAI_ADDRINFO EFLAGS in SEZAINST(CBLOCK) for the PL/I example of the OPTNAME and flag definitions.	See IPV6_ADDR_PREFERENCES and AI_EFLAGS mappings in SEZAINST(EZACOBOL) for the COBOL example of the OPTNAME and flag definitions.
	See IPV6_ADDR_PREFERENCES and AI_EFLAGS mappings in SEZAINST(EZACOBOL) for the COBOL example of the OPTNAME and flag definitions.	

Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)			
OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)	
IPV6_JOIN_GROUP Use this option to control the reception of multicast packets and specify that the socket join a multicast group. This is an IPv6-only socket option.	Contains the IPV6_MREQ structure as defined in SYS1.MACLIB(BPXYSOCK). The IPV6_MREQ structure contains a 16-byte IPv6 multicast address followed by a 4-byte IPv6 interface index number. If the interface index number is 0, then the stack chooses the local interface. See the SEZAINST(CBLOCK) for the PL/I example of IPV6_MREQ. See SEZAINST(EZACOBOL) for the COBOL example of IPV6- MREQ.	N/A	
IPV6_LEAVE_GROUP Use this option to control the reception of multicast packets and specify that the socket leave a multicast group. This is an IPv6-only socket option.	Contains the IPV6_MREQ structure as defined in SYS1.MACLIB(BPXYSOCK). The IPV6_MREQ structure contains a 16-byte IPv6 multicast address followed by a 4-byte IPv6 interface index number. If the interface index number is 0, then the stack chooses the local interface. See the SEZAINST(CBLOCK) for the PL/I example of IPV6_MREQ. See SEZAINST(EZACOBOL) for the COBOL example of IPV6- MREQ.	N/A	
IPV6_MULTICAST_HOPS Use to set or obtain the hop limit used for outgoing multicast packets. This is an IPv6-only socket option.	Contains a 4-byte binary value specifying the multicast hops. If not specified, then the default is 1 hop. -1 indicates use stack default. 0 – 255 is the valid hop limit range. Note: An application must be APF authorized to enable it to set the hop limit value above the system defined hop limit value. CICS applications cannot execute as APF authorized.	Contains a 4-byte binary value in the range 0 – 255 indicating the number of multicast hops.	

Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)			
OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)	
IPV6_MULTICAST_IF	Contains a 4-byte binary field containing an IPv6 interface index number.	Contains a 4-byte binary field containing an IPv6 interface index number.	
Use this option to set or obtain the index of the IPv6 interface used for sending outbound multicast datagrams from the socket application.			
This is an IPv6-only socket option.			
IPV6_MULTICAST_LOOP	A 4-byte binary field.	A 4-byte binary field.	
Use this option to control or determine	To enable, set to 1.	If enabled, contains a 1.	
back on the outgoing interface by the IP layer for local delivery when datagrams are sent to a group to which the sending host itself belongs. The default is to loop multicast datagrams back.	To disable, set to 0.	If disabled, contains a 0.	
This is an IPv6-only socket option.			
IPV6_UNICAST_HOPS Use this option to set or obtain the hop limit used for outgoing unicast IPv6 packets	Contains a 4-byte binary value specifying the unicast hops. If not specified, then the default is 1 hop.	Contains a 4-byte binary value in the range 0 – 255 indicating the number of unicast hops.	
This is an IPv6-only socket option.	-1 indicates use stack default.		
	0 – 255 is the valid hop limit range.		
	Note: APF authorized applications are permitted to set a hop limit that exceeds the system configured default. CICS applications cannot execute as APF authorized.		
IPV6_V60NLY	A 4-byte binary field.	A 4-byte binary field.	
Use this option to set or determine whether the socket is restricted to send and receive only IPv6 packets. The default is to not restrict the sending and receiving of only IPv6 packets.	To enable, set to 1.	If enabled, contains a 1.	
	To disable, set to 0.	If disabled, contains a 0.	
This is an IPv6-only socket option.			

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
MCAST_BLOCK_SOURCE Use this option to enable an application to block multicast packets that have a source address that matches the given source address. You must specify an interface index and a source address with this option. The specified multicast group must have been joined previously.	Contains the GROUP_SOURCE_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_SOURCE_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address and a socket address structure of the source address.	N/A
	See SEZAINST(CBLOCK) for the PL/I example of GROUP_SOURCE_REQ.	
	See SEZAINST(EZACOBOL) for the COBOL example of GROUP- SOURCE-REQ.	
MCAST_JOIN_GROUP Use this option to enable an application to join a multicast group on a specific interface. You must specify an interface index. Applications that want to receive multicast datagrams must join multicast groups.	Contains the GROUP_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address.	N/A
	See SEZAINST(CBLOCK) for the PL/I example of GROUP_REQ.	
	See SEZAINST(EZACOBOL) for the COBOL example of GROUP- REQ.	
MCAST_JOIN_SOURCE_GROUP Use this option to enable an application to join a source multicast group on a specific interface and a source address. You must specify an interface index and the source address. Applications that want to receive multicast datagrams only from specific source addresses need to join source multicast groups.	Contains the GROUP_SOURCE_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_SOURCE_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address and a socket address structure of the source address.	N/A
	See SEZAINST(CBLOCK) for the PL/I example of GROUP_SOURCE_REQ.	
	See SEZAINST(EZACOBOL) for the COBOL example of GROUP- SOURCE-REQ.	

Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)		
OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
MCAST_LEAVE_GROUP Use this option to enable an application to exit a multicast group or exit all sources for a given multicast groups.	Contains the GROUP_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address.	N/A
	See SEZAINST(CBLOCK) for the PL/I example of GROUP_REQ.	
	See SEZAINST(EZACOBOL) for the COBOL example of GROUP- REQ.	
MCAST_LEAVE_SOURCE_GROUP Use this option to enable an application to exit a source multicast group.	Contains the GROUP_SOURCE_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_SOURCE_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address and a socket address structure of the source address. See SEZAINST(CBLOCK) for the PL/I example of GROUP_SOURCE_REQ. See SEZAINST(EZACOBOL) for the COBOL example of GROUP- SOURCE-REQ.	N/A
MCAST_UNBLOCK_SOURCE Use this option to enable an application to unblock a previously blocked source for a given multicast group. You must specify an interface index and a source address with this option.	Contains the GROUP_SOURCE_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_SOURCE_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address and a socket address structure of the source address. See SEZAINST(CBLOCK) for the PL/I example of GROUP_SOURCE_REQ. See SEZAINST(EZACOBOL) for the COBOL example of GROUP- SOURCE-REQ.	N/A

Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)		
OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
S0_ASCII	To enable, set to ON.	If enabled, contains ON.
Use this option to set or determine the	To disable, set to OFF.	If disabled, contains OFF.
SO_ASCII is set, data is translated to ASCII. When SO_ASCII is not set, data is not translated to or from ASCII.	Note: The <i>optvalue</i> is returned and is optionally followed by the name of the translation table that is used if translation is	Note: The <i>optvalue</i> is returned and is optionally followed by the name of the translation table that is used if translation is
Note: This is a REXX-only socket option.		
SO_BROADCAST	A 4-byte binary field.	A 4-byte field.
Use this option to set or determine	To enable, set to 1 or a positive	If enabled, contains a 1.
messages over the socket to destinations that can receive datagram messages. The default is disabled.	To disable, set to 0.	If disabled, contains a 0.
Note: This option has no meaning for stream sockets.		
SO_DEBUG	To enable, set to ON.	If enabled, contains ON.
Use SO_DEBUG to set or determine the status of the debug option. The default is <i>disabled</i> . The debug option controls the recording of debug information.	To disable, set to OFF.	If disabled, contains OFF.
Note:		
1. This is a REXX-only socket option.		
2. This option has meaning only for stream sockets.		
SO_EBCDIC	To enable, set to ON.	If enabled, contains ON.
Use this option to set or determine the	To disable, set to OFF.	If disabled, contains OFF.
SO_EBCDIC is set, data is translated to EBCDIC. When SO_EBCDIC is not set, data is not translated to or from EBCDIC. This option is ignored by EBCDIC hosts.	Note: The <i>optvalue</i> is returned and is optionally followed by the name of the translation table that is used if translation is applied to the data.	Note: The <i>optvalue</i> is returned and is optionally followed by the name of the translation table that is used if translation is applied to the data.
Note: This is a REXX-only socket option.		
SO_ERROR	N/A	A 4-byte binary field containing
Use this option to request pending errors on the socket or to check for asynchronous errors on connected datagram sockets or for other errors that are not explicitly returned by one of the socket calls. The error status is clear afterwards.		socket.

Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)		
OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
SO_KEEPALIVE	A 4-byte binary field.	A 4-byte binary field.
Use this option to set or determine whether the keep alive mechanism periodically sends a packet on an otherwise idle connection for a stream socket.	To enable, set to 1 or a positive	If enabled, contains a 1.
	value. To disable, set to 0.	If disabled, contains a 0.
The default is disabled.		
When activated, the keep alive mechanism periodically sends a packet on an otherwise idle connection. If the remote TCP does not respond to the packet or to retransmissions of the packet, the connection is terminated with the error ETIMEDOUT.		
SO_LINGER	Contains an 8-byte field	Contains an 8-byte field
Use this option to control or determine how TCP/IP processes data that has not	fields.	fields.
been transmitted when a CLOSE is issued for the socket. The default is disabled.	Assembler coding:	Assembler coding:
Note:	ONOFF DS F LINGER DS F	ONOFF DS F LINGER DS F
 This option has meaning only for stream sockets. 	COBOL coding:	COBOL coding:
2. If you set a zero linger time, the connection cannot close in an orderly manner, but stops, resulting in a RESET segment being sent to the connection partner. Also, if the aborting socket is in nonblocking mode, the close call is treated as though no linger option had been set.	ONOFF PIC 9(8) BINARY. LINGER PIC 9(8) BINARY.	ONOFF PIC 9(8) BINARY. LINGER PIC 9(8) BINARY.
	Set ONOFF to a nonzero value to enable and set to 0 to disable this option. Set LINGER to the number of seconds that TCP/IP lingers after the CLOSE is issued.	A nonzero value returned in ONOFF indicates enabled, a 0 indicates disabled. LINGER indicates the number of seconds that TCP/IP will try to send data
When SO_LINGER is set and CLOSE is called, the calling program is blocked until the data is successfully transmitted or the connection has timed out.		aller the CLOSE is issued.
When SO_LINGER is not set, the CLOSE returns without blocking the caller, and TCP/IP continues to attempt to send data for a specified time. This usually allows sufficient time to complete the data transfer.		
Use of the SO_LINGER option does not guarantee successful completion because TCP/IP waits only the amount of time specified in OPTVAL for SO_LINGER.		

Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)		
OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
SO_OOBINLINE	A 4-byte binary field.	A 4-byte binary field.
Use this option to control or determine whether out-of-band data is received.	To enable, set to 1 or a positive value.	If enabled, contains a 1.
Note: This option has meaning only for stream sockets.	To disable, set to 0.	
When this option is set, out-of-band data is placed in the normal data input queue as it is received and is available to a RECV or a RECVFROM even if the OOB flag is not set in the RECV or the RECVFROM.		
When this option is disabled, out-of-band data is placed in the priority data input queue as it is received and is available to a RECV or a RECVFROM only when the OOB flag is set in the RECV or the RECVFROM.		
S0_RCVBUF	A 4-byte binary field.	A 4-byte binary field.
Use this option to control or determine the size of the data portion of the TCP/IP receive buffer.	To enable, set to a positive value specifying the size of the data portion of the TCP/IP receive	If enabled, contains a positive value indicating the size of the data portion of the TCP/IP
The size of the data portion of the receive	buffer.	receive buffer.
following values prior to any SETSOCKOPT call:	To disable, set to a 0.	If disabled, contains a 0.
 TCPRCVBufrsize keyword on the TCPCONFIG statement in the PROFILE.TCPIP data set for a TCP Socket 		
 UDPRCVBufrsize keyword on the UDPCONFIG statement in the PROFILE.TCPIP data set for a UDP Socket 		
• The default of 65535 for a raw socket		

Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
SO_RCVTIMEO	This option requires a TIMEVAL	This option stores a TIMEVAL
Use this option to control or determine the maximum length of time that a receive-type function can wait before it completes.	structure, which is defined in SYS1.MACLIB(BPXYRLIM) macro. The TIMEVAL structure contains the number of seconds and microseconds specified	structure that is defined in the SYS1.MACLIB(BPXYRLIM) macro. The TIMEVAL structure contains the number of seconds and microseconds, which are
If a receive-type function has blocked for the maximum length of time that was specified without receiving data, control is returned with an errno set to EWOULDBLOCK. The default value for this option is 0, which indicates that a receive- type function does not time out. When the MSG_WAITALL flag (stream	as fullword binary numbers. The seconds can be a value in the range 0 - 2678400 (equal to 31 days), and the microseconds can be a value in the range 0 - 1000000 (equal to 1 second). Although TIMEVAL value can be specified	specified as fullword binary numbers. The number of seconds value that is returned is in the range 0 - 2678400 (equal to 31 days). The number of microseconds value that is returned is in the range 0 - 1000000.
sockets only) is specified, the timeout takes precedence. The receive-type function can return the partial count. See the explanation of that operation's MSG_WAITALL flag parameter.	the internal TCP/IP timers that are used to implement this function have a granularity of approximately 100 milliseconds.	
The following receive-type functions are supported:		
• READ		
• READV		
• RECV		
RECVFROM		
RECVMSG		

Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)		
OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
S0_REUSEADDR	A 4-byte binary field.	A 4-byte binary field.
Use this option to control or determine whether local addresses are reused. The default is disabled. This alters the normal algorithm used with BIND. The normal BIND algorithm allows each Internet address and port combination to be bound only once. If the address and port have been already bound, then a subsequent BIND will fail and result error will be EADDRINUSE.	To enable, set to 1 or a positive value. To disable, set to 0.	If enabled, contains a 1. If disabled, contains a 0.
When this option is enabled, the following situations are supported:		
• A server can BIND the same port multiple times as long as every invocation uses a different local IP address and the wildcard address INADDR_ANY is used only one time per port.		
• A server with active client connections can be restarted and can bind to its port without having to close all of the client connections.		
• For datagram sockets, multicasting is supported so multiple bind() calls can be made to the same class D address and port number.		
• If you require multiple servers to BIND to the same port and listen on INADDR_ANY, see the SHAREPORT option on the PORT statement in TCPIP.PROFILE.		
S0_SNDBUF	A 4-byte binary field.	A 4-byte binary field.
Use this option to control or determine the size of the data portion of the TCP/IP send buffer. The size of the TCP/IP send buffer is protocol specific and is based on the following values:	To enable, set to a positive value specifying the size of the data portion of the TCP/IP send buffer.	If enabled, contains a positive value indicating the size of the data portion of the TCP/IP send buffer.
 The TCPSENDBufrsize keyword on the TCPCONFIG statement in the PROFILE.TCPIP data set for a TCP socket 	To disable, set to a 0.	in disabled, contains a 0.
 The UDPSENDBufrsize keyword on the UDPCONFIG statement in the PROFILE.TCPIP data set for a UDP socket 		
• The default of 65535 for a raw socket		

Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
S0_SNDTIME0	This option requires a TIMEVAL	This option stores a TIMEVAL
Use this option to control or determine the maximum length of time that a send- type function can remain blocked before it completes.	structure, which is defined in the SYS1.MACLIB(BPXYRLIM) macro. The TIMEVAL structure contains the number of seconds and microseconds specified as	structure that is defined in SYS1.MACLIB(BPXYRLIM). The TIMEVAL structure contains the number of seconds and microseconds, which are
If a send-type function has blocked for this length of time, it returns with a partial count or, if no data is sent, with an errno set to EWOULDBLOCK. The default value for this is 0, which indicates that a send- type function does not time out.	fullword binary numbers. The seconds value is in the range 0 - 2678400 (equal to 31 days), and the microseconds value is in the range 0 - 1000000 (equal to 1 second). Although the	specified as fullword binary numbers. The number of seconds value that is returned is in the range 0 - 2678400 (equal to 31 days). The microseconds value that is returned is in the
For a SETSOCKOPT, the following send-	using microsecond granularity,	range 0 - 1000000.
• SEND	the internal TCP/IP timers that are used to implement this	
SENDMSG	function have a granularity of approximately 100 milliseconds	
• SENDTO		
• WRITE		
• WRITEV		
S0_TYPE	N/A	A 4-byte binary field indicating the socket type:
Use this option to return the socket type.		X'1' indicates SOCK_STREAM.
		X'2' indicates SOCK_DGRAM
		X'3' indicates SOCK_BAW
TCP_KEEPALIVE	A 4-byte binary field.	A 4-byte binary field.
Use this option to set or determine whether a socket-specific timeout value (in seconds) is to be used in place of	To enable, set to a value in the range of 1 – 2147460.	If enabled, contains the specific timer value (in seconds) that is
a configuration-specific value whenever keep alive timing is active for that socket	To disable, set to a value of 0.	In effect for the given socket. If disabled, contains a 0
When activated, the socket-specified timer value remains in effect until respecified by SETSOCKOPT or until the socket is closed. See the <u>z/OS</u> <u>Communications Server: IP Programmer's</u> <u>Guide and Reference for more information</u> about the socket option parameters.		indicating keep alive timing is not active.

Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)		
OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
TCP_NODELAY	A 4-byte binary field.	A 4-byte binary field.
Use this option to set or determine	To enable, set to a 0.	If enabled, contains a 0.
subject to the Nagle algorithm (RFC 896).	To disable, set to a 1 or nonzero.	If disabled, contains a 1.
Under most circumstances, TCP sends data when it is presented. When this option is enabled, TCP will wait to send small amounts of data until the acknowledgment for the previous data sent is received. When this option is disabled, TCP will send small amounts of data even before the acknowledgment for the previous data sent is received.		
Note: Use the following to set TCP_NODELAY OPTNAME value for COBOL programs:		
 01 TCP-NODELAY-VAL PIC 9(10) COMP VALUE 2147483649. 01 TCP-NODELAY-REDEF REDEFINES TCP-NODELAY-VAL. 05 FILLER PIC 9(6) BINARY. 05 TCP-NODELAY PIC 9(8) BINARY. 		

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing GETSOCKOPT. The field is left-aligned and padded on the right with blanks.

S

A halfword binary number specifying the socket descriptor for the socket requiring options.

OPTNAME

Set **OPTNAME** to the required option before you issue GETSOCKOPT. See the following table for a list of the options and their unique requirements.

See the GETSOCKOPT command values information in <u>z/OS</u> Communications Server: IP Sockets Application Programming Interface Guide and Reference for the numeric values of **OPTNAME**.

Note: COBOL programs cannot contain field names with the underscore character. Fields representing the option name should contain dashes instead.

OPTLEN

Input parameter. A fullword binary field containing the length of the data returned in **OPTVAL**. See the following table for determining on what to base the value of **OPTLEN**.

Parameter values returned to the application

OPTVAL

For the GETSOCKOPT API, **OPTVAL** will be an output parameter. See the following table for a list of the options and their unique requirements.

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See <u>Appendix A</u>, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

```
Value
```

Description

0

Successful call.

-1

Check ERRNO for an error code.

GIVESOCKET

The GIVESOCKET call is used to pass a socket from one process to another.

UNIX-based platforms use a command called FORK to create a new child process that has the same descriptors as the parent process. You can use this new child process in the same way that you used the parent process.

TCP/IP normally uses GETCLIENTID, GIVESOCKET, and TAKESOCKET calls in the following sequence:

- 1. A process issues a GETCLIENTID call to get the job name of its region and its MVS subtask identifier. This information is used in a GIVESOCKET call.
- 2. The process issues a GIVESOCKET call to prepare a socket for use by a child process.
- 3. The child process issues a TAKESOCKET call to get the socket. The socket now belongs to the child process, and can be used by TCP/IP to communicate with another process.

Note: The TAKESOCKET call returns a new socket descriptor in RETCODE. The child process must use this new socket descriptor for all calls that use this socket. The socket descriptor that was passed to the TAKESOCKET call must not be used.

- 4. After issuing the GIVESOCKET command, the parent process issues a SELECT command that waits for the child to get the socket.
- 5. When the child gets the socket, the parent receives an exception condition that releases the SELECT command.
- 6. The parent process closes the socket.

The original socket descriptor can now be reused by the parent.

Sockets that have been given, but not taken for a period of four days, will be closed and will no longer be available for taking. If a select for the socket is outstanding, it will be posted.

Table 23. GIVESOCKET call requirements		
Condition	Requirement	
Authorization:	Supervisor state or problem state, any PSW key.	
Dispatchable unit mode:	Task.	
Cross memory mode:	PASN = HASN.	
Amode:	31-bit or 24-bit.	
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.	
ASC mode:	Primary address space control (ASC) mode.	
Interrupt status:	Enabled for interrupts.	
Locks:	Unlocked.	

Table 23. GIVESOCKET call requirements (continued)

Condition

Requirement

Control parameters: All parameters must be addressable by the caller and in the primary address space.

Figure 36 on page 112 shows an example of GIVESOCKET call instructions.

WORKING	-STORAGE SECT	ION.
01	SOC-FUNCTION	I PIC X(16) VALUE IS 'GIVESOCKET'.
01	S	PIC 9(4) BINARY.
01	CLIENT.	
	03 DOMAIN	PIC 9(8) BINARY.
	03 NAME	PIC X(8).
	03 TASK	PIC X(8).
	03 RESERVED) PIC X(20).
01	ERRNO	PIC 9(8) BINARY.
01	RETCODE	PIC S9(8) BINARY.
PROCEDU	RE DIVISION.	
CA	LL 'EZASOKET'	USING SOC-FUNCTION S CLIENT ERRNO RETCODE.

Figure 36. GIVESOCKET call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing GIVESOCKET. The field is left-aligned and padded on the right with blanks.

S

A halfword binary number set to the socket descriptor of the socket to be given.

CLIENT

A structure containing the identifier of the application to which the socket should be given.

DOMAIN

A fullword binary number that must be set to decimal 2, indicating AF_INET, or decimal 19 indicating AF_INET6.

Note: A socket given by GIVESOCKET can be taken only by a TAKESOCKET with the same DOMAIN (AF_INET or AF_INET6).

NAME

Specifies an 8-character field, left-aligned, padded to the right with blanks, that can be set to the name of the MVS address space that will contain the application that is going to take the socket.

- If the socket-taking application is in the *same* address space as the socket-giving application (as in CICS), NAME can be specified. The socket-giving application can determine its own address space name by issuing the GETCLIENTID call.
- If the socket-taking application is in a *different* MVS address space (as in IMS), this field should be set to blanks. When this is done, any MVS address space that requests the socket can have it.

TASK

Specifies an 8-byte field that can be set to blanks, or to the identifier of the socket-taking MVS subtask. If this field is set to blanks, any subtask in the address space specified in the NAME field can take the socket.

- As used by IMS and CICS, the field should be set to blanks.
- If TASK identifier is non-blank, the socket-receiving task should already be in execution when the GIVESOCKET is issued.

RESERVED

A 20-byte reserved field. This field is required, but not used.

Parameter values returned to the application

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See <u>Appendix A</u>, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

0

Successful call.

-1

Check ERRNO for an error code.

INET6_IS_SRCADDR

The INET6_IS_SRCADDR call verifies whether the input IP address matches an IP address in the node that conforms to all IPV6_ADDR_PREFERENCES flags specified in the call. You can use this call with IPv6 addresses or with IPv4-mapped IPv6 addresses.

You can use this call to test local IP addresses to verify whether these addresses have the characteristics that are required by your application.

See RFC 5014 *IPv6 Socket API for Source Address Selection* for more information about the INET6_IS_SRCADDR call. See <u>Appendix B</u>, "Related protocol specifications," on page 279 for information about accessing RFCs.

Table 24. INET6_IS_SRCADDR call requirements		
Condition	Requirement	
Authorization:	Supervisor state or problem state, any PSW key.	
Dispatchable unit mode:	Task.	
Cross memory mode:	PASN = HASN.	
Amode:	31-bit or 24-bit.	
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.	
ASC mode:	Primary address space control (ASC) mode.	
Interrupt status:	Enabled for interrupts.	
Locks:	Unlocked.	
Control parameters:	All parameters must be addressable by the caller and in the primary address space.	

Table 24. INET6 IS SRCADDR call requirements

Figure 37 on page 114 shows an example of INET6_IS_SRCADDR call instructions.

WORKING-STORAGE SECTION 01 SOC-FUNCTION PIC X(16) VALUE IS 'INET6 IS SRCADDR'. * IPv6 socket address structure. 01 NAME. 03 FAMILY PIC 9(4) BINARY. PIC 9(4) BINARY. 03 PORT 03 FLOWINFO PIC 9(8) BINARY. 03 IP-ADDRESS. PIC 9(16) BINARY. 10 FILLER PIC 9(16) BINARY. PIC 9(8) BINARY. 10 FILLER 03 SCOPE-ID PIC 9(8) BINARY 01 FLAGS 88 IPV6-PREFER-SRC-HOME PIC 9(8) BINARY VALUE 1. PIC 9(8) BINARY VALUE 2. 88 IPV6-PREFER-SRC-COA 2. 88 IPV6-PREFER-SRC-TMP PIC 9(8) BINARY VALUE 4. 88 IPV6-PREFER-SRC-PUBLIC PIC 9(8) BINARY VALUE 8. PIC 9(8) BINARY VALUE 16. 88 IPV6-PREFER-SRC-CGA 88 IPV6-PREFER-SRC-NONCGA PIC 9(8) BINARY VALUE 32. PIC 9(8) BINARY. 01 ERRNO 01 RETCODE PIC S9(8) BINARY. PROCEDURE DIVISION. CALL 'EZASOKET' USING SOC-FUNCTION NAME FLAGS ERRNO RETCODE.

Figure 37. INET6_IS_SRCADDR call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing INET6_IS_SRCADDR.

NAME

Specifies the AF_INET6 socket address structure for the address that is to be tested.

Requirement: You must specify an AF_INET6 address. You can specify an IPv6 address, or an IPv4mapped IPv6 address.

The IPv6 socket address structure specifies the following fields:

FAMILY

A halfword binary field that specifies the IPv6 addressing family. For TCP/IP the value is the decimal value 19, indicating AF_INET6.

PORT

A halfword binary field. This field is ignored by INET6_IS_SRCADDR processing.

FLOWINFO

A fullword binary field that specifies the traffic class and flow label. This field is ignored by INET6_IS_SRCADDR processing.

IP-ADDRESS

A 16-byte binary field that is set to the 128-bit IPv6 IP address (network byte order) of the IP address to be tested.

Rule: Specify an IPv4 address by using its IPv4-mapped IPv6 address format.

SCOPE-ID

A fullword binary field that identifies a set of appropriate interfaces for the scope of the address that is specified in the IP-ADDRESS field. The value 0 indicates that the SCOPE-ID field does not identify the set of interfaces to be used.

Requirements:

- If the IP address is a link-local address, this field must be set to a nonzero value.
- If the IP address is not a link-local address, this field must be set to 0.

FLAGS

A fullword binary field that contains one or more valid IPV6_ADDR_PREFERENCES flags.

Flag name	Binary value	Decimal value	Description
IPV6_PREFER_SRC_HOME	X'0000001'	1	Test whether the input IP address is a home address. ¹
IPV6_PREFER_SRC_COA	X'0000002'	2	Test whether the input IP address is a care-of address. ²
IPV6_PREFER_SRC_TMP	X'00000004'	4	Test whether the input IP address is a temporary address.
IPV6_PREFER_SRC_PUBLIC	X'0000008'	8	Test whether the input IP address is a public address.
IPV6_PREFER_SRC_CGA	X'00000010'	16	Test whether the input IP address is cryptographically generated. ²
IPV6_PREFER_SRC_NONCGA	X'00000020'	32	Test whether the input IP address is not cryptographically generated. ¹

Note:

1. Any valid IP address that is known to the stack satisfies this flag.

2. z/OS Communications Server does not support this type of address. The call always returns FALSE if this flag is specified with a valid IP address that is known to the stack.

Tips:

- The SEZAINST(EZACOBOL) and SEZAINST(CBLOCK) samples contain mappings for these flags. For assembler programs, the flags are defined in the system maclib member BPXYSOCK.
- Some of these flags are contradictory, for example:
 - The flag IPV6_PREFER_SRC_HOME contradicts the flag IPV6_PREFER_SRC_COA.
 - The flag IPV6_PREFER_SRC_CGA contradicts the flag IPV6_PREFER_SRC_NONCGA.
 - The flag IPV6_PREFER_SRC_TMP contradicts the flags IPV6_PREFER_SRC_PUBLIC.

Result: If you specify contradictory flags in the call, the result is FALSE.

Parameter values returned to the application

ERRNO

A fullword binary field. If the RETCODE value is negative, the field contains an error number. See Appendix A, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

0

FALSE

The call was successful and the result is FALSE. The input AF_INET6 address corresponds to an IP address on the node, but does not conform to one or more IPV6_ADDR_PREFERENCES flags that are specified in the call.

1

TRUE

The call was successful and the result is TRUE. The input AF_INET6 address corresponds to an IP address on the node and conforms to all IPV6_ADDR_PREFERENCES flags that are specified in the call.

-1

Check **ERRNO** for an error code.

INITAPI

The INITAPI call connects an application to the TCP/IP interface. Almost all sockets programs that are written in COBOL, PL/I, or assembler language must issue the INITAPI socket command before they issue other socket commands.

The exceptions to this rule are the following calls, which, when issued first, will generate a default INITAPI call.

- GETCLIENTID
- GETHOSTID
- GETHOSTNAME
- GETIBMOPT
- SELECT
- SELECTEX
- SOCKET
- TAKESOCKET

Table	25	INITAPI	call red	nuirements
rubic	20.		cunrec	junchichus

1			
Condition	Requirement		
Authorization:	Supervisor state or problem state, any PSW key.		
Dispatchable unit mode:	Task.		
Cross memory mode:	PASN = HASN.		
Amode:	31-bit or 24-bit.		
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.		
ASC mode:	Primary address space control (ASC) mode.		
Interrupt status:	Enabled for interrupts.		
Locks:	Unlocked.		
Control parameters:	All parameters must be addressable by the caller and in the primary address space.		

Figure 38 on page 117 shows an example of INITAPI call instructions.
```
WORKING-STORAGE SECTION.
   01 SOC-FUNCTION PIC X(16) VALUE IS 'INITAPI'.
    01 MAXSOC
                        PIC 9(4) BINARY.
    01 IDENT.
       02 TCPNAME
02 ADSNAME
                        PIC X(8).
                        PIC X(8).
    01 SUBTASK
                        PIC X(8).
                        PIC 9(8) BINARY.
PIC 9(8) BINARY.
    01 MAXSNO
    01 FRRNO
    01 RETCODE
                        PIC S9(8) BINARY.
PROCEDURE DIVISION.
     CALL 'EZASOKET' USING SOC-FUNCTION MAXSOC IDENT SUBTASK
     MAXSNO ERRNO RETCODE.
```

Figure 38. INITAPI call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing INITAPI. The field is left-aligned and padded on the right with blanks.

MAXSOC

A halfword binary field set to the maximum number of sockets this application will ever have open at one time. The maximum number is 65535 and the minimum number is 50. This value is used to determine the amount of memory that is allocated for socket control blocks and buffers. If less than 50 are requested, MAXSOC defaults to 50.

IDENT

A structure containing the identities of the TCP/IP address space and the calling program's address space. Specify IDENT on the INITAPI call from an address space.

TCPNAME

An 8-byte character field that should be set to the MVS job name of the TCP/IP address space with which you are connecting.

ADSNAME

An 8-byte character field set to the identity of the calling program's address space. It is the name of the CICS startup job. For explicit-mode IMS server programs, use the TIMSrvAddrSpc field passed in the TIM. If ADSNAME is not specified, the system derives a value from the MVS control block structure.

SUBTASK

Indicates an 8-byte field that contains a unique subtask identifier, which is used to distinguish between multiple subtasks within a single address space. Use your own job name as part of your subtask name. This ensures that, if you issue more than one INITAPI command from the same address space, each SUBTASK parameter is unique.

Restriction: EZASOKET calls outside of the CICS environment are not reentrant. If EZASOKET is to be used by a multithread or multitask application, a separate copy must be loaded for each thread or task. See <u>z/OS Communications Server: IP CICS Sockets Guide</u> for information about use in the CICS environment.

Parameter values returned to the application

MAXSNO

A fullword binary field that contains the highest socket number assigned to this application. The lowest socket number is 0. If you have 50 sockets, they are numbered from 0 to 49. If MAXSNO is not specified, the value for MAXSNO is 49.

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See <u>Appendix A</u>, <u>"Return codes," on page 267</u> for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

0

Successful call.

-1

Check **ERRNO** for an error code.

IOCTL

The IOCTL call is used to control certain operating characteristics for a socket.

Before you issue an IOCTL socket command, you must load a value that represents the characteristic that you want to control into the COMMAND field.

The variable length parameters REQARG and RETARG are arguments that are passed to and returned from IOCTL. The length of REQARG and RETARG is determined by the value that you specify in COMMAND. See Table 27 on page 125 for information about REQARG and RETARG.

Table 26. IOCTL call requirements	
Condition	Requirement
Authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN.
Amode:	31-bit or 24-bit.
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.
ASC mode:	Primary address space control (ASC) mode.
Interrupt status:	Enabled for interrupts.
Locks:	Unlocked.
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Figure 39 on page 119 shows an example of IOCTL call instructions.

WORKING-STORAGE SECTION. 01 SOC-FUNCTION 01 S 01 COMMAND	PIC X(16) VALUE 'IOCTL'. PIC 9(4) BINARY. PIC 9(8) BINARY.
01 IFREQ. 03 NAME 03 FAMILY 03 PORT 03 ADDRESS 03 RESERVED	PIC X(16). PIC 9(4) BINARY. PIC 9(4) BINARY. PIC 9(8) BINARY. PIC X(8).
01 IFREQOUT. 03 NAME 03 FAMILY 03 PORT 03 ADDRESS 03 RESERVED	PIC X(16). PIC 9(4) BINARY. PIC 9(4) BINARY. PIC 9(8) BINARY. PIC X(8).
01 GRP-IOCTL-TABLE. 02 IOCTL-ENTRY OCCURS 100 03 NAME 03 FAMILY 03 PORT 03 ADDRESS 03 NULLS	TIMES. PIC X(16). PIC 9(4) BINARY. PIC 9(4) BINARY. PIC 9(8) BINARY. PIC X(8).
01 IOCTL-REQARG 01 IOCTL-RETARG 01 ERRNO 01 RETCODE	USAGE IS POINTER. USAGE IS POINTER. PIC 9(8) BINARY. PIC 9(8) BINARY.
PROCEDURE DIVISION. CALL 'EZASOKET' USING SOO RETARG ERRNO RETCOI	C-FUNCTION S COMMAND REQARG DE.

Figure 39. IOCTL call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing IOCTL. The field is left-aligned and padded to the right with blanks.

S

A halfword binary number set to the descriptor of the socket to be controlled.

COMMAND

To control an operating characteristic, set this field to one of the following symbolic names. A value in a bit mask is associated with each symbolic name. By specifying one of these names, you are turning on a bit in a mask which communicates the requested operating characteristic to TCP/IP.

FIONBIO

Sets or clears blocking status.

FIONREAD

Returns the number of immediately readable bytes for the socket.

SIOCATMARK

Determines whether the current location in the data input is pointing to out-of-band data.

SIOCGHOMEIF6

Requests all IPv6 home interfaces. To request OSM interfaces the application must have READ authorization to the EZB.OSM.sysname.tcpname resource.

• When the SIOCGHOMEIF6 IOCTL is issued, the REGARQ must contain a Network Configuration Header. The NETCONFHDR is defined in the SYS1.MACLIB(BPXYIOC6) for assembly language. The following fields are input fields and must be filled out:

NchEyeCatcher

Contains eye catcher '6NCH'

NchIoctl

Contains the command code

NchBufferLength

Buffer length large enough to contain all the IPv6 interface records. Each interface record is length of HOME-IF-ADDRESS. If buffer is not large enough, then errno will be set to ERANGE and the NchNumEntryRet will be set to number of interfaces. Based on NchNumEntryRet and size of HOME-IF-ADDRESS, calculate the necessary storage to contain the entire list.

NchBufferPtr

This is a pointer to an array of HOME-IF structures returned on a successful call. The size will depend on the number of qualifying interfaces returned.

NchNumEntryRet

If return code is 0 this will be set to number of HOME-IF-ADDRESS returned. If errno is ERANGE, then will be set to number of qualifying interfaces. No interfaces are returned. Recalculate The NchBufferLength based on this value times the size of HOME-IF-ADDRESS.

REQARG and RETARG

Point to the arguments that are passed between the calling program and IOCTL. The length of the argument is determined by the COMMAND request. REQARG is an input parameter and is used to pass arguments to IOCTL. RETARG is an output parameter and is used for arguments returned by IOCTL. For the lengths and meanings of REQARG and RETARG for each COMMAND type, see Table 27 on page 125.

Working-Storage Section. 01 STOCGHOMETE6-VAL pic s9(10) binary value 3222599176	
01 SIOCGHOMEIF6-REDEF REDEFINES SIOCGHOMEIF6-VAL.	•
05 SIOCGHOMEIF6 PIC 9(8) COMP.	
01 IOCTL-RETARG USAGE IS POINTER. 01 NET-CONF-HDR.	
05 NCH-EYE-CATCHER PIC X(4) VALUE '6NCH'.	
05 NCH-DUFFER-LENTH PIC 9(8) BINARY.	
05 NCH-BUFFER-PTR USAGE IS POINTER. 05 NCH-NUM-ENTRY-RET PIC 9(8) BINARY.	
01 HOME-IF.	
05 FILLER PIC 9(16) BINARY.	
Linkage Section.	
01 L1.	
03 NetConfHdr. 05 NchEyeCatcher pic x(4).	
05 NchIoctl pic 9(8) binary. 05 NchBufferLength pic 9(8) binary	
05 NchBufferPtr usage is pointer.	
 WChNumEntryRet pic 9(8) binary. * Allocate storage based on your need. 	
03 Allocated-Storage pic x(nn).	
Procedure Division using L1.	
set NchBufferPtr to address of Allocated-Storage.	
 * Set NchBufferLength to the length of your allocated storage. move nn to NchBufferLength. 	
move SIOCGHOMEIF6 to Nchloctl.	
SIOCGHOMEIF6	
NETCONFHDR NETCONFHDR errno retcode	

Figure 40. COBOL language example for SIOCGHOMEIF6

SIOCGIFADDR

Requests the IPv4 network interface address for a given interface name. For assembler, see the IOCN_IFNAME field in the SYS1.MACLIB(BPXYIOCC) API. For COBOL, see the IFR-NAME field in the SEZAINST(EZACOBOL) API. For PL/I, see the IFR_NAME field in the SEZAINST(CBLOCK) API.

SIOCGIFBRDADDR

Requests the IPv4 network interface broadcast address for a given interface name. For assembler, see the IOCN_IFNAME field in the SYS1.MACLIB(BPXYIOCC) API. For COBOL, see the IFR-NAME field in the SEZAINST(EZACOBOL) API. For PL/I, see the IFR_NAME field in the SEZAINST(CBLOCK) API.

SIOCGIFCONF

Requests the IPv4 network interface configuration. The configuration is a variable number of 32-byte structures. For assembler, see the IOCN_IFREQ field in the SYS1.MACLIB(BPXYIOCC) API for the structure format. For COBOL, see the IFREQ field in the SEZAINST(EZACOBOL) API for the structure format. For PL/I, see the IFREQ field in the SEZAINST(CBLOCK) API for the structure format.

- When IOCTL is issued, REQARG must contain the length of the array to be returned. To determine the length of REQARG, multiply the structure length (array element) by the number of interfaces requested. The maximum number of array elements that TCP/IP can return is 100.
- When IOCTL is issued, RETARG must be set to the beginning of the storage area that you have defined in your program for the array to be returned.

SIOCGIFDSTADDR

Requests the network interface destination address for a given interface name. For assembler, see the IOCN_IFNAME field in the SYS1.MACLIB(BPXYIOCC) API. For COBOL, see the IFR-NAME field in the SEZAINST(EZACOBOL) API. For PL/I, see the IFR_NAME field in the SEZAINST(CBLOCK) API.

SIOCGIFMTU

Requests the IPv4 network interface MTU (maximum transmission unit) for a given interface name. For assembler, see the IOCN_IFNAME field in the SYS1.MACLIB(BPXYIOCC) API. For COBOL, see the IFR-NAME field in the SEZAINST(EZACOBOL) API. For PL/I, see the IFR_NAME field in the SEZAINST(CBLOCK) API.

SIOCGIFNAMEINDEX

Requests all interface names and interface indexes including local loopback but excluding VIPAs. Information is returned for both IPv4 and IPv6 interfaces whether they are active or inactive. For IPv6 interfaces, information is returned for an interface only if it has at least one available IP address. To request OSM interfaces the application must have READ authorization to the EZB.OSM.sysname.tcpname resource.

The configuration consists of IF_NAMEINDEX structure, which is defined in SYS1.MACLIB(BPX1IOCC) for the assembly language.

- When the SIOCGIFNAMEINDEX IOCTL is issued, the first word in REQARG must contain the length (in bytes) to contain an IF-NAME-INDEX structure to return the interfaces. The formula to compute this length is as follows:
 - 1. Determine the number of interfaces expected to be returned upon successful completion of this command.
 - 2. Multiply the number of interfaces by the array element (size of IF-NIINDEX, IF-NINAME, and IF-NIEXT) to get the size of the array element.
 - 3. Add the size of the IF-NITOTALIF and IF-NIENTRIES to the size of the array to get the total number of bytes needed to accommodate the name and index information returned.
- When IOCTL is issued, RETARG must be set to the address of the beginning of the area in your program's storage that is reserved for the IF-NAMEINDEX structure that is to be returned by IOCTL.
- The command 'SIOCGIFNAMEINDEX' returns a variable number of all the qualifying network interfaces.

WORKING-STORAGE SECTION. 01 SIOCGIFNAMEINDEX-VAL pic 9(10) binary value 1073804803. 01 SIOCGIFNAMEINDEX-REDEF REDEFINES SIOCGIFNAMEINDEX-VAL. 05 FILLER PIC 9(6) COMP. 05 SIOCGIFNAMEINDEX PIC 9(8) COMP. reqarg pic 9(8) binary. 01 reqarg 01 reqarg-header-only pic 9(8) binary. 01 IF-NIHEADER. 05 IF-NITOTALIF PIC 9(8) BINARY. **05 IF-NIENTRIES** PIC 9(8) BINARY. 01 IF-NAME-INDEX-ENTRY. PIC 9(8) BINARY. 05 IF-NIINDEX 05 IF-NINAME PIC X(16). 05 IF-NINAMETERM PIC X(1). 05 IF-NIRESV1 PIC X(3) 01 OUTPUT-STORAGE PIC X(500). Procedure Division. move 8 to reqarg-header-only. Call 'EZASOKET' using soket-ioctl socket-descriptor SIOCGIFNAMEINDEX REOARG-HEADER-ONLY IF-NIHEADER errno retcode. move 500 to reqarg. Call 'EZASOKET' using soket-ioctl socket-descriptor SIOCGIFNAMEINDEX REOARG OUTPUT-STORAGE errno retcode.

Figure 41. COBOL language example for SIOCGIFNAMEINDEX

SIOCGIPMSFILTER

Requests a list of the IPv4 source addresses that comprise the source filter, with the current mode on a given interface and a multicast group for a socket. The source filter can include or exclude the set of source address, depending on the filter mode (MCAST_INCLUDE or MCAST EXCLUDE). When the SIOCGIPMSFILTER IOCTL is issued, the REQARG parameter must contain a IP MSFILTER structure, which is defined in SYS1.MACLIB(BPXYIOCC) for assembly language, in SEZAINST(CBLOCK) for PL/I, and in SEZAINST(EZACOBOL) for COBOL. The IP MSFILTER must include an interface address (input), a multicast address (input), filter mode (output), the number of source addresses in the following array (input and output), and an array of source addresses (output). On input, the number of source addresses is the number of source addresses that will fit in the input array. On output, the number of source addresses contains the total number of source filters in the output array. If the application does not know the size of the source list prior to processing, it can make a reasonable guess (for example, 0), and if when the call completes the number of source addresses is a greater value, the IOCTL can be repeated with a buffer that is large enough. That is, on output, the number of source addresses is always updated to be the total number of sources in the filter, but the array holds as many source addresses as will fit, up to the minimum of the array size passed in as the input number.

Calculate the size of IF_MSFILTER value as follows:

- 1. Determine the number of expected source addresses.
- 2. Multiply the number of source addresses by the array element (size of the IMSF_SrcEntry value) to determine the size of all array elements.
- 3. Add the size of all array elements to the size of the IMSF_Header value to determine the total number of bytes needed to accommodate the source addresses information that is returned.

SIOCGMSFILTER

Requests a list of the IPv4 or IPv6 source addresses that comprise the source filter, with the current mode on a given interface index and a multicast group for a socket. The source filter can include or exclude the set of source address, depending on the filter mode (MCAST_INCLUDE or MCAST_EXCLUDE). When the SIOCGMSFILTER IOCTL is issued, the REQARG parameter must contain a GROUP_FILTER structure, which is defined in SYS1.MACLIB(BPXYIOCC) for assembler , in SEZAINST(CBLOCK) for PL/I, and in SEZAINST(EZACOBOL) for COBOL. The GROUP_FILTER option must include an interface index (input), a socket address structure of the multicast address (input), filter mode (output), the number of source addresses in the following array (output), and an array of the socket address structure of source addresses (input and output). On input, the

number of source addresses is the number of source addresses that will fit in the input array. On output, the number of source addresses contains the total number of source filters in the output array. If the application does not know the size of the source list prior to processing, it can make a reasonable guess (for example, 0), and if when the call completes the number of source addresses is a greater value, the IOCTL can be repeated with a buffer that is large enough. That is, on output, the number of source addresses is always updated to be the total number of sources in the filter, but the array holds as many source addresses as will fit, up to the minimum of the array size passed in as the input number.

Calculate the size of the GROUP_FILTER value as follows:

- 1. Determine the number of source addresses expected.
- 2. Multiply the number of source addresses by the array element (size of the GF_SrcEntry value) to determine the size of all array elements.
- 3. Add the size of all array elements to the size of the GF_Header value to determine the total number of bytes needed to accommodate the source addresses information returned.

SIOCGPARTNERINFO

Provides an interface for an application to retrieve security information about its partner. When you issue the SIOCGPARTNERINFO IOCTL, the REQARG parameter must contain a PartnerInfo structure. The PartnerInfo structure is defined in members within SEZANMAC; EZBPINF1 defines the PL/I layout, EZBPINFA defines the assembler layout, and EZBPINFB defines the COBOL layout. For more information about using the <u>SIOCGPARTNERINFO</u> IOCTL, see <u>z/OS</u> Communications Server: IP Programmer's Guide and Reference.

SIOCSAPPLDATA

The SIOCSAPPLDATA IOCTL enables an application to set 40 bytes of user-specified application data against a socket endpoint. You can also use this application data to identify socket endpoints in interfaces such as Netstat, SMF, or network management applications. When the SIOCSAPPLDATA IOCTL is issued, the REQARG parameter must contain a SetApplData structure as defined by the EZBYAPPL macro. See the CBLOCK and the EZACOBOL samples for the equivalent SetApplData and SetADcontainer structure definitions for PL/I and COBOL programming environments. See z/OS Communications Server: IP Programmer's Guide and Reference for more information about programming the SIOCSAPPLDATA IOCTL.

SetAD_buffer: The user-defined application data is 40 bytes of data that identifies the endpoint with the application. You can obtain this application data from the following sources:

- Netstat reports. The information is displayed in the ALL/-A report. If you use the APPLDATA modifier, then the information also is displayed on the ALLConn/-a and COnn/-c reports.
- The SMF 119 TCP connection termination record. See <u>TCP connection termination record</u> (subtype 2) in <u>z/OS Communications Server: IP Programmer's Guide and Reference</u> for more information.
- Network management interfaces. See <u>Network management interfaces</u> in <u>z/OS Communications</u> Server: IP Programmer's Guide and Reference for more information.

Consider the following guidelines:

- The application must document the content, format and meaning of the ApplData strings that it associates with the sockets that it owns.
- The application should uniquely identify itself with printable EBCDIC characters at the beginning of the string. Strings beginning with 3-character IBM product identifiers, such as TCP/IP's EZA or EZB, are reserved for IBM use. IBM product identifiers begin with a letter in the range A-I.
- Use printable EBCDIC characters for the entire string to enable searching with Netstat filters.

Tip: Separate application data elements with a blank for easier reading.

SIOCSIPMSFILTER

Sets a list of the IPv4 source addresses that comprise the source filter, with the current mode on a given interface and a multicast group for a socket. The source filter can include or exclude the set of source address, depending on the filter mode (MCAST_INCLUDE or MCAST_EXCLUDE). When

the SIOCSIPMSFILTER IOCTL is issued, the REQARG parameter must contain a IP_MSFILTER structure, which is defined in SYS1.MACLIB(BPXYIOCC) for assembler, in SEZAINST(CBLOCK) for PL/I and in SEZAINST(EZACOBOL) for COBOL. The IP_MSFILTER option must include an interface address, a multicast address, filter mode, the number of source addresses in the following array, and an array of source addresses.

Calculate the size of the IF_MSFILTER value as follows:

- 1. Determine the number of expected source addresses.
- 2. Multiply the number of source addresses by the array element (size of the IMSF_SrcEntry value) to determine the size of all array elements.
- 3. Add the size of all array elements to the size of the IMSF_Header value to determine the total number of bytes needed to accommodate the source addresses information that is returned.

SIOCSMSFILTER

Sets a list of the IPv4 or IPv6 source addresses that comprise the source filter, along with the current mode on a given interface index and a multicast group for a socket. The source filter can include or exclude the set of source address, depending on the filter mode (INCLUDE or EXCLUDE). When the SIOCSMSFILTER IOCTL is issued, the REQARG parameter must contain a GROUP_FILTER structure which is defined in SYS1.MACLIB(BPXYIOCC) for assembler, in SEZAINST(CBLOCK) for PL/I, and in SEZAINST(EZACOBOL) for COBOL. The GROUP_FILTER option must include an interface index, a socket address structure of the multicast address, filter mode, the number of source addresses in the following array, and an array of the socket address structure of source addresses.

Calculate the size of GROUP_FILTER as follows:

- 1. Determine the number of source addresses expected.
- 2. Multiply the number of source addresses by the array element (size of the GF_SrcEntry value) to get the size of all array elements.
- 3. Add the size of all array elements to the size of the GF_Header value to get the total number of bytes needed to accommodate the source addresses information returned.

SIOCSPARTNERINFO

The SIOCSPARTNERINFO IOCTL sets an indicator to retrieve the partner security credentials during connection setup and saves the information, enabling an application to issue a SIOCGPARTNERINFO IOCTL without suspending the application, or at least minimizing the time it takes to retrieve the information. The SIOCSPARTNERINFO IOCTL must be issued prior to the SIOCGPARTNERINFO IOCTL. When you issue the SIOCSPARTNERINFO IOCTL, the REQARG parameter must contain a constant value, PI_REQTYPE_SET_PARTNERDATA. This constant is defined in members within SEZANMAC; EZBPINF1 defines the PL/I layout, EZBPINFA defines the assembler layout, and EZBPINFB defines the COBOL layout. For more information about using the <u>SIOCSPARTNERINFO</u> IOCTL, see <u>z/OS</u> Communications Server: IP Programmer's Guide and Reference.

SIOCTTLSCTL

Controls Application Transparent Transport Layer Security (AT-TLS) for the connection. REQARG and RETARG must contain a TTLS_IOCTL structure. If a partner certificate is requested, the TTLS_IOCTL must include a pointer to additional buffer space and the length of that buffer. Information is returned in the TTLS_IOCTL structure. If a partner certificate is requested and one is available, it is returned in the additional buffer space. The TTLS_IOCTL structure is defined in members within SEZANMAC. EZBZTLS1 defines the PL/I layout, EZBZTLSP defines the assembler layout, and EZBZTLSB defines the COBOL layout. For more usage details, see the <u>Application Transparent TLS (AT-TLS)</u> information in z/OS Communications Server: IP Programmer's Guide and Reference.

Restriction: Use of this ioctl for functions other than query requires that the AT-TLS policy mapped to the connection be defined with the ApplicationControlled parameter set to On.

REQARG and **RETARG**

Points to arguments that are passed between the calling program and IOCTL. The length of the argument is determined by the COMMAND request. REQARG is an input parameter or an output parameter and is used to pass and receive arguments to and from IOCTL. RETARG is an output parameter and receives arguments from IOCTL. The REQARG and RETARG parameters are described in Table 27 on page 125.

COMMAND/CODE	SIZE	REQARG	SIZE	RETARG
FIONBIO X'8004A77E'	4	Set socket mode to: X'00'=blocking, X'01'=nonblocking.	0	Not used.
FIONREAD X'4004A77F'	0	Not used.	4	Number of characters available for read.
SIOCATMARK	0	Not used.	4	X'00'= not at OOB data
X'4004A707'				X'01'= at OOB data.
SIOCGHOMEIF6 X'C014F608'	20	NetConfHdr		See <u>Figure 40 on page 120</u> NetConfHdr.
SIOCGIFADDR X'CO2OA7OD'	32	First 16 bytes - interface name. Last 16 bytes - not used.	32	Network interface address. For assembler, see the IOCN_SADDRIF field in the SYS1.MACLIB(BPXYIOCC) API. For COBOL, see the IFR-ADDR field in the SEZAINST(EZACOBOL) API. For PL/I, see the IFR_ADDR field in the SEZAINST(CBLOCK) API.
SIOCGIFBRDADDR X'C020A712'	32	First 16 bytes - interface name. Last 16 bytes - not used.	32	Network interface address. For assembler, see the IOCN_SADDRIFBROADCAST field in the SYS1.MACLIB(BPXYIOCC) API. For COBOL, see the IFR- BROADADDR field in the SEZAINST(EZACOBOL) API. For PL/I, see the IFR_BROADADDR field in the SEZAINST(CBLOCK) API.
SIOCGIFCONF X'C008A714'	8	Size of RETARG.	See note ¹ .	
SIOCGIFDSTADDR X'C020A70F'	32	First 16 bytes - interface name. Last 16 bytes - not used.	32	Destination interface address. For assembler, see the IOCN_SADDRIFDEST field in the SYS1.MACLIB(BPXYIOCC) API. For COBOL, see the IFR-DSTADDR field in the SEZAINST(EZACOBOL) API. For PL/I, see the IFR_DSTADDR field in the SEZAINST(CBLOCK) API.

Table 27. IOCTL call arguments

COMMAND/CODE	SIZE	REQARG	SIZE	RETARG
SIOCGIFMTU X'C020A726'	32	First 16 bytes - interface name. Last 16 bytes - not used.	32	IPv4 interface MTU (maximum transmission unit). For assembler, see the IOCN_MTUSIZE field in the SYS1.MACLIB(BPXYIOCC) API. For COBOL, see the IFR-MTU field in the SEZAINST(EZACOBOL) API. For PL/I, see the IFR_MTU field in the SEZAINST(CBLOCK) API.
SIOCGIFNAMEINDEX X'4000F603'	4	First 4 bytes size of return buffer.		See Figure 41 on page 122 IF- NAMEINDEX .
SIOCGIPMSFILTER X'C000A724'	-	See IP_MSFILTER structure in macro BPXYIOCC. See note 2.	0	Not used
SIOCGMSFILTER X'C000F610'	-	See GROUP_FILTER structure in macro BPXYIOCC. See note 3	0	Not used
SIOCGPARTNERINFO X'C000F612'	-	For the PartnerInfo structure layout, see SEZANMAC(EZBPINFA) for assembler, SEZANMAC(EZBPINF1) for PL/I, and SEZANMAC(EZBPINFB) for COBOL. See note 4.	0	Not used
SIOCSAPPLDATA X'8018D90C'	_	See SETAPPLDATA structure in macro EZBYAPPL	0	Not used
SIOCSIPMSFILTER X'8000A725'	_	See IP_MSFILTER structure in macro BPXYIOCC. See note 2.	0	Not used
SIOCSMSFILTER X'8000F611'	-	See GROUP_FILTER structure in macro BPXYIOCC. See note 3	0	Not used
SIOCSPARTNERINFO X'8004F613'	4	See PI_REQTYPE_SET_PARTNERDATA in SEZANMAC(EZBPINFA) for assembler, SEZANMAC(EZBPINF1) for PL/I, and SEZANMAC(EZBPINFB) for COBOL.	0	Not used
SIOCTTLSCTL X'C038D90B'	56	For IOCTL structure layout, see SEZANMAC(EZBZTLS1) for PL/I, SEZANMAC(EZBZTLSP) for assembler, and SEZANMAC(EZBZTLSB) for COBOL.	56	For IOCTL structure layout, see SEZANMAC(EZBZTLS1) for PL/I, SEZANMAC(EZBZTLSP) for assembler, and SEZANMAC(EZBZTLSB) for COBOL.

Table 27. IOCTL call arguments (continued)

COMMAND/CODE SIZE REQARG

SIZE RETARG

Note:

- 1. When you call IOCTL with the SIOCGIFCONF command set, REQARG should contain the length in bytes of RETARG. Each interface is assigned a 32-byte array element and REQARG should be set to the number of interfaces times 32. TCP/IP Services can return up to 100 array elements.
- 2. The size of the IP_MSFILTER structure must be equal to or greater than the size of the IMSF_Header value.
- 3. The size of the GROUP_FILTER structure must be equal to or greater than the size of GF_Header value.
- 4. The size of the PartnerInfo structure must be equal to or greater than the PI_FIXED_SIZE value.

Parameter values returned to the application

RETARG

Returns an array whose size is based on the value in COMMAND. See <u>Table 27 on page 125</u> for information about REQARG and RETARG.

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See <u>Appendix A</u>, <u>"Return codes," on page 267</u> for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

0

Successful call.

-1

Check ERRNO for an error code.

The COMMAND SIOGIFCONF returns a variable number of network interface configurations. Figure 42 on page 127 contains an example of a COBOL II routine that can be used to work with such a structure.

Note: This call can be programmed only in languages that support address pointers. Figure 42 on page 127 shows a COBOL II example for SIOCGIFCONF.

```
WORKING-STORAGE SECTION.
      REQARG
                           PIC 9(8) COMP.
  77
77
                           PIC 9(8) COMP VALUE max number of interfaces.
        COUNT
LINKAGE SECTION.
  01
         RETARG.
         05
               IOCTL-TABLE OCCURS 1 TO max TIMES DEPENDING ON COUNT.
                       NAME
                                  PIC X(16).
PIC 9(4) BINARY.
               10
               10
                       FAMILY
                                  PIC 9(4) BINARY.
PIC 9(8) BINARY.
                       PORT
               10
               10
                       ADDR
               10
                       NULLS
                                  PIC X(8).
PROCEDURE DIVISION.
MULTIPLY COUNT BY 32 GIVING REQARQ.
CALL 'EZASOKET' USING SOC-FUNCTION S COMMAND
        REQARG RETARG ERRNO RETCODE.
```

Figure 42. COBOL II example for SIOCGIFCONF

LISTEN

The LISTEN call:

- Completes the bind, if BIND has not already been called for the socket.
- Creates a connection-request queue of a specified length for incoming connection requests.

Note: The LISTEN call is not supported for datagram sockets or raw sockets.

The LISTEN call is typically used by a server to receive connection requests from clients. When a connection request is received, a new socket is created by a subsequent ACCEPT call, and the original socket continues to listen for additional connection requests. The LISTEN call converts an active socket to a passive socket and conditions it to accept connection requests from clients. Once a socket becomes passive it cannot initiate connection requests.

	5
Condition	Requirement
Authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN.
Amode:	31-bit or 24-bit.
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.
ASC mode:	Primary address space control (ASC) mode.
Interrupt status:	Enabled for interrupts.
Locks:	Unlocked.
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Table 20 IICTEN call requirements

Figure 43 on page 128 shows an example of LISTEN call instructions.

```
WORKING-STORAGE SECTION.
    01 SOC-FUNCTION
                          PIC X(16) VALUE IS 'LISTEN'.
    01 S
01 BACKLOG
                          PIC 9(4) BINARY.
PIC 9(8) BINARY.
                          PIC 9(8) BINARY.
    01 RETCODE
                          PIC S9(8) BINARY.
PROCEDURE DIVISION.
CALL 'EZASOKET' USING SOC-FUNCTION S BACKLOG ERRNO RETCODE.
```

Figure 43. LISTEN call instruction example

For equivalent PL/I and assembly language declarations, see "Converting parameter descriptions" on page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing LISTEN. The field is left-aligned and padded to the right with blanks.

S

A halfword binary number set to the socket descriptor.

BACKLOG

A fullword binary number set to the number of communication requests to be queued.

Rule: The BACKLOG value specified on the LISTEN call is limited to the value configured by the SOMAXCONN statement in the stack's TCPIP PROFILE (default=10); no error is returned if a larger backlog is requested. SOMAXCONN might need to be updated if a larger backlog is desired. see z/OS Communications Server: IP Configuration Reference for details.

Parameter values returned to the application

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See <u>Appendix A</u>, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

0

Successful call.

-1

Check ERRNO for an error code.

NTOP

The NTOP call converts an IP address from its numeric binary form into a standard text presentation form. On successful completion, NTOP returns the converted IP address in the buffer provided.

Table 29. NTOP call requirements	
Condition	Requirement
Authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN.
Amode:	31-bit or 24-bit.
	Note: See the addressability mode (Amode) considerations under <u>"CALL instruction API environmental restrictions and programming requirements" on page 51</u> .
ASC mode:	Primary address space control (ASC) mode.
Interrupt status:	Enabled for interrupts.
Locks:	Unlocked.
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Figure 44 on page 130 shows an example of NTOP call instructions.

WORKING-STORAGE SECTION. PIC X(16) VALUE IS 'ACCEPT'. PIC X(16) VALUE IS 'NTOP'. PIC 9(4) BINARY. 01 SOC-ACCEPT-FUNCTION 01 SOC-NTOP-FUNCTION 01 S * IPv4 socket structure. 01 NAME. NAMELPIC 9(4) BINARY.03FAMILYPIC 9(4) BINARY.03IP-ADDRESSPIC 9(8) BINARY.03RESERVEDPIC X(8). * IPv6 socket structure. 01 NAME. 03FAMILYPIC 9(4) BINARY.03PORTPIC 9(4) BINARY.03FLOWINFOPIC 9(8) BINARY. 03 IP-ADDRESS. PIC 9(16) BINARY. 10 FILLER 10 FILLER PIC 9(16) BINARY. 03 SCOPE-ID PIC 9(8) BINARY. 01 NTOP-FAMILY PIC 9(8) BINARY. 01 ERRNO PIC 9(8) BINARY. 01 ERRNO PIC 9(8) BINARY. 01 PETCODE PIC 99(8) BINARY. PIC S9(8) BINARY. 01 RETCODE 01 PRESENTABLE-ADDRESS PIC X(45). 01 PRESENTABLE-ADDRESS-LEN PIC 9(4) BINARY. PROCEDURE DIVISION. CALL 'EZASOKET' USING SOC-ACCEPT-FUNCTION S NAME ERRNO RETCODE. CALL 'EZASOKET' USING SOC-NTOP-FUNCTION NTOP-FAMILY IP-ADDRESS PRESENTABLE-ADDRESS PRESENTABLE-ADDRESS-LEN ERRNO RETURN-CODE.

Figure 44. NTOP call instruction example

Parameter values set by the application

Keyword

Description

FAMILY

The addressing family for the IP address being converted. The value of decimal 2 must be specified for AF_INET and 19 for AF_INET6.

IP-ADDRESS

A field containing the numeric binary form of the IPv4 or IPv6 address being converted. For an IPv4 address this field must be a fullword and for an IPv6 address this field must be 16 bytes. The address must be in network byte order.

Parameter values returned to the application

Keyword

Description

PRESENTABLE-ADDRESS

A field used to receive the standard text presentation form of the IPv4 or IPv6 address being converted. For IPv4 the address will be in dotted-decimal format and for IPv6 the address will be in colon-hex format. The size of the IPv4 address will be a maximum of 15 bytes and the size of the converted IPv6 address will be a maximum of 45 bytes. Consult the value returned in PRESENTABLE-ADDRESS-LEN for the actual length of the value in PRESENTABLE-ADDRESS.

PRESENTABLE-ADDRESS-LEN

Initially, an input parameter. The address of a binary halfword field that is used to specify the length of DSTADDR field on input and upon a successful return will contain the length of converted IP address.

ERRNO

Output parameter. A fullword binary field. If RETCODE is negative, ERRNO contains a valid error number. Otherwise, ignore the ERRNO field.

See Appendix A, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

0

Successful call.

-1

Check ERRNO for an error code.

PTON

The PTON call converts an IP address in its standard text presentation form to its numeric binary form. On successful completion, PTON returns the converted IP address in the buffer provided.

Table 30. PTON call requirements	
Condition	Requirement
Authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN.
Amode:	31-bit or 24-bit.
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.
ASC mode:	Primary address space control (ASC) mode.
Interrupt status:	Enabled for interrupts.
Locks:	Unlocked.
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Figure 45 on page 132 shows an example of PTON call instructions.

WORKING-STORAGE SECTION. PIC X(16) VALUE IS 'BIND'. PIC X(16) VALUE IS 'PTON'. PIC 9(4) BINARY. 01 SOC-BIND-FUNCTION 01 SOC-PTON-FUNCTION 01 S * IPv4 socket structure. 01 NAME. NAMELPIC 9(4) BINARY.03FAMILYPIC 9(4) BINARY.03IP-ADDRESSPIC 9(8) BINARY.03RESERVEDPIC X(8). * IPv6 socket structure. 01 NAME. NAIL:PIC 9(4) BINARY.03 FORTPIC 9(4) BINARY.03 FLOWINFOPIC 9(4) BINARY.03 FLOWINFOPIC 9(8) BINARY. 03 IP-ADDRESS. 10FILLERPIC 9(16)BINARY.10FILLERPIC 9(16)BINARY.03SCOPE-IDPIC 9(8)BINARY. PIC 9(8) BINARY VALUE 2. PIC 9(8) BINARY VALUE 19. 01 AF-INET 01 AF-INET6 * IPv4 address. 01 PRESENTABLE-ADDRESS PIC X(45). 01 PRESENTABLE-ADDRESS-IPV4 REDEFINES PRESENTABLE-ADDRESS. 05 PRESENTABLE-IPV4-ADDRESS PIC X(15) VALUE '192.26.5.19'. 05 FILLER PIC X(30). 01 PRESENTABLE-ADDRESS-LEN PIC 9(4) BINARY VALUE 11. * IPv6 address. 01 PRESENTABLE-ADDRESS PIC X(45) VALUE '12f9:0:0:c30:123:457:9cb:1112' 01 PRESENTABLE-ADDRESS-LEN PIC 9(4) BINARY VALUE 29. * IPv4-mapped IPv6 address. 01 PRESENTABLE-ADDRESS PIC X(45) VALUE '12f9:0:0:c30:123:457:192.26.5.19' 01 PRESENTABLE-ADDRESS-LEN PIC 9(4) BINARY VALUE 32. 01 ERRNO PIC 9(8) BINARY. 01 RETCODE PIC S9(8) BINARY. PROCEDURE DIVISION. * IPv4 address. CALL 'EZASOKET' USING SOC-PTON-FUNCTION AF-INET PRESENTABLE-ADDRESS PRESENTABLE-ADDRESS-LEN IP-ADDRESS ERRNO RETURN-CODE. * IPv6 address. CALL 'EZASOKET' USING SOC-PTON-FUNCTION AF-INET6 PRESENTABLE-ADDRESS PRESENTABLE-ADDRESS-LEN IP-ADDRESS ERRNO RETURN-CODE.

CALL 'EZASOKET' USING SOC-BIND-FUNCTION S NAME ERRNO RETURN-CODE.

Figure 45. PTON call instruction example

Parameter values set by the application

Keyword

Description

FAMILY

The addressing family for the IP address being converted. The value of decimal 2 must be specified for AF_INET and 19 for AF_INET6.

PRESENTABLE-ADDRESS

A field containing the standard text presentation form of the IPv4 or IPv6 address being converted. For IPv4 the address will be in dotted-decimal format and for IPv6 the address will be in colon-hex format.

PRESENTABLE-ADDRESS-LEN

Input parameter. The address of a binary halfword field that must contain the length of the IP address to be converted.

Parameter values returned to the application

Keyword

Description

IP-ADDRESS

A field containing the numeric binary form of the IPv4 or IPv6 address being converted. For an IPv4 address this field must be a fullword and for an IPv6 address this field must be 16 bytes. The address must be in network byte order.

ERRNO

Output parameter. A fullword binary field. If RETCODE is negative, ERRNO contains a valid error number. Otherwise, ignore the ERRNO field.

See Appendix A, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

0

Successful call.

-1

Check ERRNO for an error code.

READ

The READ call reads the data on socket s. This is the conventional TCP/IP read data operation. If a datagram packet is too long to fit in the supplied buffer, datagram sockets discard extra bytes.

For stream sockets, data is processed as streams of information with no boundaries separating the data. For example, if programs A and B are connected with a stream socket and program A sends 1000 bytes, each call to this function can return any number of bytes, up to the entire 1000 bytes. The number of bytes returned will be contained in RETCODE. Therefore, programs using stream sockets should place this call in a loop that repeats until all data has been received.

Note: See "EZACIC05" on page 189 for a subroutine that will translate ASCII input data to EBCDIC.

Condition	Requirement
Authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN.
Amode:	31-bit or 24-bit.
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.
ASC mode:	Primary address space control (ASC) mode.
Interrupt status:	Enabled for interrupts.
Locks:	Unlocked.
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Table 31. READ call requirements

Figure 46 on page 134 shows an example of READ call instructions.

```
WORKING-STORAGE SECTION.

01 SOC-FUNCTION PIC X(16) VALUE IS 'READ'.

01 S PIC 9(4) BINARY.

01 NBYTE PIC 9(8) BINARY.

01 BUF PIC X(length of buffer).

01 ERRNO PIC 9(8) BINARY.

01 RETCODE PIC S9(8) BINARY.

PROCEDURE DIVISION.

CALL 'EZASOKET' USING SOC-FUNCTION S NBYTE BUF

ERRNO RETCODE.
```

Figure 46. READ call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing READ. The field is left-aligned and padded to the right with blanks.

S

A halfword binary number set to the socket descriptor of the socket that is going to read the data.

NBYTE

A fullword binary number set to the size of BUF. READ does not return more than the number of bytes of data in NBYTE even if more data is available.

Parameter values returned to the application

BUF

On input, a buffer to be filled by completion of the call. The length of BUF must be at least as long as the value of NBYTE.

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See <u>Appendix A</u>, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

0

A 0 return code indicates that the connection is closed and no data is available.

>0

A positive value indicates the number of bytes copied into the buffer.

-1

Check ERRNO for an error code.

READV

The READV function reads data on a socket and stores it in a set of buffers. If a datagram packet is too long to fit in the supplied buffers, datagram sockets discard extra bytes.

Table 32. READV call requirements		
Condition	Requirement	
Authorization:	Supervisor state or problem state, any PSW key.	

Table 32. READV call requirements (continued)		
Condition	Requirement	
Dispatchable unit mode:	Task.	
Cross memory mode:	PASN = HASN.	
Amode:	31-bit or 24-bit.	
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.	
ASC mode:	Primary address space control (ASC) mode.	
Interrupt status:	Enabled for interrupts.	
Locks:	Unlocked.	
Control parameters:	All parameters must be addressable by the caller and in the primary address space.	

Figure 47 on page 135 shows an example of READV call instructions.

WORKING-STORAGE SECTION. 01 SOC-FUNCTION 01 S PIC X(16) VALUE 'READV'. PIC 9(4) BINARY. PIC 9(8) BINARY. 01 IOVCNT 01 IOV. 03 BUFFER-ENTRY OCCURS N TIMES. 05BUFFER-POINTERUSAGE IS POINTER.05RESERVEDPIC X(4).05BUFFER_LENGTHPIC 9(8) BINARY. 01 ERRNO 01 RETCODE PIC 9(8) BINARY. PIC 9(8) BINARY. PROCEDURE DIVISION. SET BUFFER-POINTER(1) TO ADDRESS OF BUFFER1. SET BUFFER-LENGTH(1) TO LENGTH OF BUFFER1. SET BUFFER-DOINTER(2) TO ADDRESS OF BUFFER2. SET BUFFER-LENGTH(2) TO LENGTH OF BUFFER2. н н ш SET BUFFER-POINTER(n) TO ADDRESS OF BUFFERn. SET BUFFER-LENGTH(n) TO LENGTH OF BUFFERn. Call 'EZASOCKET' USING SOC-FUNCTION S IOV IOVCNT ERRNO RETCODE.

Figure 47. READV call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing READV. The field is left-aligned and padded to the right with blanks.

S

A value or the address of a halfword binary number specifying the descriptor of the socket into which the data is to be read.

ΙΟΥ

An array of tripleword structures with the number of structures equal to the value in IOVCNT and the format of the structures as follows:

Fullword 1

Pointer to the address of a data buffer, which is filled in on completion of the call

Fullword 2

Reserved

Fullword 3

The length of the data buffer referenced in fullword one

IOVCNT

A fullword binary field specifying the number of data buffers provided for this call.

Parameter values returned to the application

ERRNO

A fullword binary field. If RETCODE is negative, this contains an error number. See Appendix A, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

0

A 0 return code indicates that the connection is closed and no data is available.

>0

A positive value indicates the number of bytes copied into the buffer.

-1

Check ERRNO for an error code.

RECV

The RECV call, like READ, receives data on a socket with descriptor S. RECV applies only to connected sockets. If a datagram packet is too long to fit in the supplied buffers, datagram sockets discard extra bytes.

For additional control of the incoming data, RECV can:

- · Peek at the incoming message without having it removed from the buffer
- Read out-of-band data

For stream sockets, data is processed as streams of information with no boundaries separating the data. For example, if programs A and B are connected with a stream socket and program A sends 1000 bytes, each call to this function can return any number of bytes, up to the entire 1000 bytes. The number of bytes returned will be contained in RETCODE. Therefore, programs using stream sockets should place RECV in a loop that repeats until all data has been received.

If data is not available for the socket, and the socket is in blocking mode, RECV blocks the caller until data arrives. If data is not available and the socket is in nonblocking mode, RECV returns a -1 and sets ERRNO to 35 (EWOULDBLOCK). See "FCNTL" on page 66 or "IOCTL" on page 118 for a description of how to set nonblocking mode.

For raw sockets, RECV adds a 20-byte header.

Table 22 DEOV and a subserve and

Note: See "EZACIC05" on page 189 for a subroutine that will translate ASCII input data to EBCDIC.

Tuble 33. RECV Cull requirements	
Condition	Requirement
Authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task.

Table 33. RECV call requirem	nents (continued)
Condition	Requirement
Cross memory mode:	PASN = HASN.
Amode:	31-bit or 24-bit.
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.
ASC mode:	Primary address space control (ASC) mode.
Interrupt status:	Enabled for interrupts.
Locks:	Unlocked.
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Figure 48 on page 137 shows an example of RECV call instructions.



Figure 48. RECV call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing RECV. The field is left-aligned and padded to the right with blanks.

S

A halfword binary number set to the socket descriptor of the socket to receive the data.

FLAGS

A fullword binary field with values as follows:

Literal Value	Binary Value	Description
NO-FLAG	X'0000000'	Read data.
MSG-OOB	X'0000001'	Receive out-of-band data (stream sockets only). Even if the OOB flag is not set, out-of-band data can be read if the SO-OOBINLINE option is set for the socket.

Literal Value	Binary Value	Description
MSG-PEEK	X'0000002'	Peek at the data, but do not destroy data. If the peek flag is set, the next receive operation reads the same data.
MSG-WAITALL	X'0000040'	Requests that the function block until the full amount of data that was requested can be returned (stream sockets only). The function might return a smaller amount of data if the connection is closed, if an error is pending, or if the SO_RCVTIMEO field is set and the timer has expired for the socket.

NBYTE

A value or the address of a fullword binary number set to the size of BUF. RECV does not receive more than the number of bytes of data in NBYTE even if more data is available.

Parameter values returned to the application

BUF

The input buffer to receive the data.

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See <u>Appendix A</u>, <u>"Return codes," on page 267</u> for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

0

The socket is closed.

>0

A positive return code indicates the number of bytes copied into the buffer.

-1

Check ERRNO for an error code.

RECVFROM

The RECVFROM call receives data on a socket with descriptor S and stores it in a buffer. The RECVFROM call applies to both connected and unconnected sockets. The socket address is returned in the NAME structure. If a datagram packet is too long to fit in the supplied buffers, datagram sockets discard extra bytes.

For datagram protocols, RECVFROM returns the source address associated with each incoming datagram. For connection-oriented protocols like TCP, GETPEERNAME returns the address associated with the other end of the connection.

If NAME is nonzero, the call returns the address of the sender. The NBYTE parameter should be set to the size of the buffer.

On return, NBYTE contains the number of data bytes received.

For stream sockets, data is processed as streams of information with no boundaries separating the data. For example, if programs A and B are connected with a stream socket and program A sends 1000 bytes, each call to this function can return any number of bytes, up to the entire 1000 bytes. The number of bytes returned will be contained in RETCODE. Therefore, programs using stream sockets should place RECVFROM in a loop that repeats until all data has been received. For raw sockets, RECVFROM adds a 20-byte header.

If data is not available for the socket, and the socket is in blocking mode, RECVFROM blocks the caller until data arrives. If data is not available and the socket is in nonblocking mode, RECVFROM returns a -1 and sets ERRNO to 35 (EWOULDBLOCK). See <u>"FCNTL" on page 66</u> or <u>"IOCTL" on page 118</u> for a description of how to set nonblocking mode.

Note: See "EZACIC05" on page 189 for a subroutine that will translate ASCII input data to EBCDIC.

Table 34. RECVFROM call requirements		
Condition	Requirement	
Authorization:	Supervisor state or problem state, any PSW key.	
Dispatchable unit mode:	Task.	
Cross memory mode:	PASN = HASN.	
Amode:	31-bit or 24-bit.	
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.	
ASC mode:	Primary address space control (ASC) mode.	
Interrupt status:	Enabled for interrupts.	
Locks:	Unlocked.	
Control parameters:	All parameters must be addressable by the caller and in the primary address space.	

.

Figure 49 on page 139 shows an example of RECVFROM call instructions.

WORKING- 01	STORAGE SECTION	PIC X(16) VALUE IS 'RECVFROM
01	S	PIC 9(4) BINARY.
01	FLAGS 88 NO-FLAG	VALUE IS 0.
	88 00B	VALUE IS 1.
01	NBYTE	PIC 9(8) BINARY.
01	BUF	PIC X(length of buffer).
* IPv4 s	ocket address s	tructure.
01	NAME. 03 FAMTLY	PTC Q(A) BTNARY
	03 PORT	PIC 9(4) BINARY.
	03 IP-ADDRESS 03 RESERVED	PIC 9(8) BINARY.
	US RESERVED	
* IPv6 s 01	socket address s NAMF.	tructure.
	03 FAMILY	PIC 9(4) BINARY.
	03 FLOWINFO	PIC 9(4) BINARY. PIC 9(8) BINARY.
	03 IP-ADDRESS.	
	10 FILLER 10 FILLER	PIC 9(16) BINARY. PIC 9(16) BINARY.
	03 SCOPE-ID	PIC 9(8) BINARY.
01	ERRNO	PIC 9(8) BINARY.
01	RETCODE	PIC S9(8) BINARY.
PROCEDUF CAL	RE DIVISION. LL 'EZASOKET' US NB	ING SOC-FUNCTION S FLAGS YTE BUF NAME ERRNO RETCODE.

Figure 49. RECVFROM call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing RECVFROM. The field is left-aligned and padded to the right with blanks.

S

A halfword binary number set to the socket descriptor of the socket to receive the data.

FLAGS

A fullword binary field containing flag values as follows:

Literal Value	Binary Value	Description
NO-FLAG	X'00000000'	Read data.
MSG-OOB	X'00000001'	Receive out-of-band data (stream sockets only). Even if the OOB flag is not set, out-of-band data can be read if the SO-OOBINLINE option is set for the socket.
MSG-PEEK	X'00000002'	Peek at the data, but do not destroy data. If the peek flag is set, the next receive operation reads the same data.
MSG-WAITALL	X'0000040'	Requests that the function block until the requested amount of data can be returned (stream sockets only). The function might return a smaller amount of data if the connection is closed, if an error is pending, or if the SO_RCVTIMEO field is set and the timer has expired for the socket.

NBYTE

A fullword binary number specifying the length of the input buffer.

Parameter values returned to the application

BUF

Defines an input buffer to receive the input data.

NAME

An IPv4 socket address structure containing the address of the socket that sent the data. The structure is as follows:

FAMILY

A halfword binary number specifying the IPv4 addressing family. The value is always decimal 2, indicating AF_INET.

PORT

A halfword binary number specifying the port number of the sending socket.

IP-ADDRESS

A fullword binary number specifying the 32-bit IPv4 IP address of the sending socket.

RESERVED

An 8-byte reserved field. This field is required, but is not used.

An IPv6 socket address structure containing the address of the socket that sent the data. The structure is as follows:

Field

Description

FAMILY

A halfword binary number specifying the IPv6 addressing family. The value is decimal 19, indicating AF_INET6.

PORT

A halfword binary number specifying the port number of the sending socket.

FLOWINFO

A fullword binary field specifying the traffic class and flow label. This value of this field is undefined.

IP-ADDRESS

A 16-byte binary field set to the 128-bit IPv6 IP address of the sending socket.

SCOPE-ID

A fullword binary field which identifies a set of interfaces as appropriate for the scope of the address carried in the IPv6-ADDRESS field. For a link scope IPv6-ADDRESS, SCOPE-ID contains the link index for the IPv6-ADDRESS. For all other address scopes, SCOPE-ID is undefined.

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See <u>Appendix A</u>, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

0

The socket is closed.

>0

A positive return code indicates the number of bytes of data transferred by the read call.

-1

Check **ERRNO** for an error code.

RECVMSG

The RECVMSG call receives messages on a socket with descriptor S and stores them in an array of message headers. If a datagram packet is too long to fit in the supplied buffers, datagram sockets discard extra bytes.

For datagram protocols, RECVMSG returns the source address associated with each incoming datagram. For connection-oriented protocols like TCP, GETPEERNAME returns the address associated with the other end of the connection.

Table 35. RECVMSG call requirements		
Condition	Requirement	
Authorization:	Supervisor state or problem state, any PSW key.	
Dispatchable unit mode:	Task.	
Cross memory mode:	PASN = HASN.	

Table 35. RECVMSG call requirements (continued)		
Condition	Requirement	
Amode:	31-bit or 24-bit.	
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.	
ASC mode:	Primary address space control (ASC) mode.	
Interrupt status:	Enabled for interrupts.	
Locks:	Unlocked.	
Control parameters:	All parameters must be addressable by the caller and in the primary address space.	

RECVMSG call instruction example

WORKING-STORAGE 01 01 01	E SECTION. SOC-FUNCTION S MSG-HDR. 03 MSG-NAME 03 MSG-NAME 03 IOV 03 IOV 03 IOVCNT 03 MSG-ACCR 03 MSG-ACCR	PIC PIC -LEN PIGHTS IGHTS-LE	X(16) 9(4) PIC 9(8 USAGE I USAGE I USAGE I N USAGE	VALUE IS BINARY. S POINTE) COMP. S POINTE S POINTE IS POINTE IS POIN	G 'RECVMSG'. R. R. R. R. R. ITER.
01 01 01	FLAGS 88 NO-FLAG 88 OOB 88 PEEK ERRNO RETCODE	PIC PIC PIC	9(8) 9(8) S9(8)	BINARY. VALUE VALUE VALUE BINARY. BINARY.	IS 0. IS 1. IS 2.
LINKAGE 01	SECTION. L1. 03 RECVMSG-IO 05 IOV1A 05 IOV1AL 05 IOV2A 05 IOV2A 05 IOV2AL 05 IOV2AL 05 IOV3A 05 IOV3AL 05 IOV3L	VECTOR.	US. PI US. PI US. PI PI	AGE IS F C 9(8) C AGE IS F C 9(8) C C 9(8) C AGE IS F C 9(8) C C 9(8) C C 9(8) C	POINTER. COMP. POINTER. COMP. COMP. POINTER. COMP. COMP.
03 03 03 03	RECVMSG-BUFFE RECVMSG-BUFFE RECVMSG-BUFFE RECVMSG-BUFNO	R1 P R2 P R3 P P P	IC X(16 IC X(16 IC X(16 IC 9(8)).).). COMP.	
* IPv4 soc 03 NAM 05 05 05 05	ket address s IE. FAMILY PORT IP-ADDRESS RESERVED	Tructure PIC 9(4) PIC 9(4) PIC 9(8) PIC X(8)	BINARY BINARY BINARY		
* IPv6 sock 03 NAM 05 53 05 05	tet address st IE. FAMILY PORT FLOWINFO IP-ADDRESS. 10 FILLER 10 FILLER SCOPE-ID	PIC 9(4) PIC 9(4) PIC 9(4) PIC 9(8) PIC 9(16 PIC 9(16 PIC 9(8)	BINARY BINARY BINARY) BINAR) BINARY BINARY	Y.	

PROCEDURE DIVISION USING L1.

SET MSG-NAME TO ADDRESS OF NAME. MOVE LENGTH OF NAME TO MSG-NAME-LEN SET IOV TO ADDRESS OF RECVMSG-IOVECTOR. MOVE 3 TO RECVMSG-BUFNO. SET IOVCNT TO ADDRESS OF RECVMSG-BUFNO. SET IOV1A TO ADDRESS OF RECVMSG-BUFFER1. MOVE 0 TO IOV1AL MOVE LENGTH OF RECVMSG-BUFFER1 TO IOV1L. SET IOV2A TO ADDRESS OF RECVMSG-BUFFER2. MOVE 0 TO IOV2AL MOVE LENGTH OF RECVMSG-BUFFER2 TO IOV2L. SET IOV3A TO ADDRESS OF RECVMSG-BUFFER3. MOVE 0 TO IOV3AL MOVE LENGTH OF RECVMSG-BUFFER3 TO IOV3L. SET MSG-ACCRIGHTS TO NULLS. SET MSG-ACCRIGHTS-LEN TO NULLS. MOVE 0 TO FLAGS. MOVE SPACES TO RECVMSG-BUFFER1. MOVE SPACES TO RECVMSG-BUFFER2. MOVE SPACES TO RECVMSG-BUFFER3.

CALL 'EZASOKET' USING SOC-FUNCTION S MSG-HDR FLAGS ERRNO RETCODE.

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

S

A value or the address of a halfword binary number specifying the socket descriptor.

MSG

On input, a pointer to a message header into which the message is received upon completion of the call.

Field

Description

NAME

On input, a pointer to a buffer where the sender address is stored upon completion of the call. The storage being pointed to should be for an IPv4 socket address or an IPv6 socket address. The IPv4 socket address structure contains the following fields:

Field

Description

FAMILY

Output parameter. A halfword binary number specifying the IPv4 addressing family. The value for IPv4 socket descriptor (S parameter) is decimal 2, indicating AF_INET.

PORT

Output parameter. A halfword binary number specifying the port number of the sending socket.

IP-ADDRESS

Output parameter. A fullword binary number specifying the 32-bit IPv4 IP address of the sending socket.

RESERVED

Output parameter. An 8-byte reserved field. This field is required, but is not used.

The IPv6 socket address structure contains the following fields:

Field

Description

FAMILY

Output parameter. A halfword binary number specifying the IPv6 addressing family. The value for IPv6 socket descriptor (S parameter) is decimal 19, indicating AF_INET6.

PORT

Output parameter. A halfword binary number specifying the port number of the sending socket.

FLOWINFO

A fullword binary field specifying the traffic class and flow label. This value of this field is undefined.

IP-ADDRESS

Output parameter. A 16 byte binary field specifying the 128–bit IPv6 IP address, in network byte order, of the sending socket.

SCOPE-ID

A fullword binary field which identifies a set of interfaces as appropriate for the scope of the address carried in the IPv6-ADDRESS field. For a link scope IPv6-ADDRESS, SCOPE-ID contains the link index for the IPv6-ADDRESS. For all other address scopes, SCOPE-ID is undefined.

NAME-LEN

On input, a pointer to the size of the NAME.

IOV

On input, a pointer to an array of tripleword structures with the number of structures equal to the value in IOVCNT and the format of the structures as follows:

Fullword 1

A pointer to the address of a data buffer. This data buffer must be in the home address space.

Fullword 2

Reserved. This storage will be cleared.

Fullword 3

A pointer to the length of the data buffer referenced in fullword 1.

In COBOL, the IOV structure must be defined separately in the Linkage section, as shown in the example.

IOVCNT

On input, a pointer to a fullword binary field specifying the number of data buffers provided for this call.

ACCRIGHTS

On input, a pointer to the access rights received. This field is ignored.

ACCRLEN

On input, a pointer to the length of the access rights received. This field is ignored.

FLAGS

A fullword binary field with values as follows:

Literal Value	Binary Value	Description
NO-FLAG	X'0000000'	Read data.
MSG-OOB	X'0000001'	Receive out-of-band data (stream sockets only). Even if the OOB flag is not set, out-of-band data can be read if the SO-OOBINLINE option is set for the socket.
MSG-PEEK	X'0000002'	Peek at the data, but do not destroy data. If the peek flag is set, the next receive operation reads the same data.

Literal Value	Binary Value	Description
MSG-WAITALL	X'0000040'	Requests that the function block until the requested amount of data can be returned (stream sockets only). The function might return a smaller amount of data if the connection is closed, if an error is pending, or if the SO_RCVTIMEO field is set and the timer has expired for the socket.

Parameter values returned to the application

ERRNO

A fullword binary field. If RETCODE is negative, this contains an error number. See <u>Appendix A</u>, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field with the following values:

Value

Description

<0

Call returned error. See ERRNO field.

0

Connection partner has closed connection.

>0

Number of bytes read.

SELECT

In a process where multiple I/O operations can occur it is necessary for the program to be able to wait on one or several of the operations to complete.For example, consider a program that issues a READ to multiple sockets whose blocking mode is set. Because the socket would block on a READ call, only one socket could be read at a time. Setting the sockets nonblocking would solve this problem, but would require polling each socket repeatedly until data became available. The SELECT call allows you to test several sockets and to execute a subsequent I/O call only when one of the tested sockets is ready, thereby ensuring that the I/O call will not block.

To use the SELECT call as a timer in your program, do one of the following actions:

- Set the read, write, and exception arrays to zeros.
- Specify MAXSOC <= 0.

Table 36. SELECT call requirements

Table 50. Select call requirements		
Condition	Requirement	
Authorization:	Supervisor state or problem state, any PSW key.	
Dispatchable unit mode:	Task.	
Cross memory mode:	PASN = HASN.	
Amode:	31-bit or 24-bit.	
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.	
ASC mode:	Primary address space control (ASC) mode.	

Table 36. SELECT call requirements (continued)	
Condition	Requirement
Interrupt status:	Enabled for interrupts.
Locks:	Unlocked.
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Defining which sockets to test

The SELECT call monitors for read operations, write operations, and exception operations:

- When a socket is ready to read, one of the following conditions has occurred:
 - A buffer for the specified sockets contains input data. If input data is available for a given socket, a
 read operation on that socket will not block.
 - A connection has been requested on that socket.
- When a socket is ready to write, TCP/IP can accommodate additional output data. If TCP/IP can accept additional output for a given socket, a write operation on that socket will not block.
- When an exception condition has occurred on a specified socket it is an indication that a TAKESOCKET has occurred for that socket.
- A timeout occurs on the SELECT call. The timeout period can be specified when the SELECT call is issued.

Each socket descriptor is represented by a bit in a bit string. The length of this bit-mask array is dependent on the value of the MAXSOC parameter and must be a multiple of 4 bytes.

For information about <u>selecting requests in a concurrent server program</u>, see <u>z/OS Communications</u> Server: IP Sockets Application Programming Interface Guide and Reference.

Note: To simplify string processing in COBOL, you can use the program EZACIC06 to convert each bit in the string to a character. For more information, see "EZACIC06 " on page 190.

Read operations

Read operations include ACCEPT, READ, READV, RECV, RECVFROM, or RECVMSG calls. A socket is ready to be read when data has been received for it or when a connection request has occurred.

To test whether any of several sockets is ready for reading, set the appropriate bits in RSNDMSK to one before issuing the SELECT call. When the SELECT call returns, the corresponding bits in the RRETMSK indicate sockets are ready for reading.

Write operations

A socket is selected for writing (ready to be written) when:

- TCP/IP can accept additional outgoing data.
- The socket is marked nonblocking and a previous CONNECT did not complete immediately. In this case, CONNECT returned an ERRNO with a value of 36 (EINPROGRESS). This socket will be selected for write when the CONNECT completes.

A call to WRITE, SEND, or SENDTO blocks when the amount of data to be sent exceeds the amount of data TCP/IP can accept. To avoid this, you can precede the write operation with a SELECT call to ensure that the socket is ready for writing. Once a socket is selected for WRITE, the program can determine the amount of TCP/IP buffer space available by issuing the GETSOCKOPT call with the SO-SNDBUF option.

To test whether any of several sockets is ready for writing, set the WSNDMSK bits representing those sockets to 1 before issuing the SELECT call. When the SELECT call returns, the corresponding bits in the WRETMSK indicate sockets are ready for writing.

Exception operations

For each socket to be tested, the SELECT call can check for an existing exception condition. Two exception conditions are supported:

- The calling program (concurrent server) has issued a GIVESOCKET command and the target child server has successfully issued the TAKESOCKET call. When this condition is selected, the calling program (concurrent server) should issue CLOSE to dissociate itself from the socket.
- A socket has received out-of-band data. On this condition, a READ will return the out-of-band data ahead of program data.

To test whether any of several sockets have an exception condition, set the ESNDMSK bits representing those sockets to 1. When the SELECT call returns, the corresponding bits in the ERETMSK indicate sockets with exception conditions.

MAXSOC parameter

The SELECT call must test each bit in each string before returning results. For efficiency, the MAXSOC parameter can be used to specify the largest socket descriptor number that needs to be tested for any event type. The SELECT call tests only bits that are in the range 0 through the MAXSOC value minus 1.

Example: If MAXSOC is set to 50, the range is 0 - 49.

TIMEOUT parameter

If the time specified in the TIMEOUT parameter elapses before any event is detected, the SELECT call returns, and the RETCODE is set to 0.

Figure 50 on page 147 shows an example of SELECT call instructions.

```
WORKING-STORAGE SECTION.
                              PIC X(16) VALUE IS 'SELECT'.
PIC 9(8) BINARY.
     01 SOC-FUNCTION
     01 MAXSOC
    01 TIMEOUT.
         03 TIMEOUT-SECONDS PIC 9(8) BINARY.
03 TIMEOUT-MICROSEC PIC 9(8) BINARY.
     01 RSNDMSK
                             PIC X(*).
                            PIC X(*).
PIC X(*).
     01 WSNDMSK
    01 ESNDMSK
                            PIC X(*).
     01 RRETMSK
                            PIC X(*).
PIC X(*).
     01
         WRETMSK
    01 ERETMSK
                             PIC 9(8) BINARY.
PIC S9(8) BINARY.
     01 ERRNO
    01 RETCODE
PROCEDURE DIVISION.
      CALL 'EZASOKET' USING SOC-FUNCTION MAXSOC TIMEOUT
                         RSNDMSK WSNDMSK ESNDMSK
RRETMSK WRETMSK ERETMSK
                         ERRNO RETCODE.
```

* The bit mask lengths can be determined from the expression:

((maximum socket number +32)/32 (drop the remainder))*4

Figure 50. SELECT call instruction example

Bit masks are 32-bit fullwords with one bit for each socket. Up to 32 sockets fit into one 32-bit mask [PIC X(4)]. If you have 33 sockets, you must allocate two 32-bit masks [PIC X(8)].

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing SELECT. The field is left-aligned and padded on the right with blanks.

MAXSOC

A fullword binary field that specifies the largest socket descriptor value that is being checked. The SELECT call tests only bits that are in the range 0 through the MAXSOC value minus 1. For example, if you set the MAXSOC value to 50, the range is 0 - 49.

TIMEOUT

If TIMEOUT is a positive value, it specifies the maximum interval to wait for the selection to complete. If TIMEOUT-SECONDS is a negative value, the SELECT call blocks until a socket becomes ready. To poll the sockets and return immediately, specify the TIMEOUT value to be 0.

TIMEOUT is specified in the two-word TIMEOUT as follows:

- TIMEOUT-SECONDS, word one of the TIMEOUT field, is the seconds component of the timeout value.
- TIMEOUT-MICROSEC, word two of the TIMEOUT field, is the microseconds component of the timeout value (0-999999).

For example, if you want SELECT to time out after 3.5 seconds, set TIMEOUT-SECONDS to 3 and TIMEOUT-MICROSEC to 500000.

RSNDMSK

A bit string sent to request read event status.

- For each socket to be checked for pending read events, the corresponding bit in the string should be set to 1.
- For sockets to be ignored, the value of the corresponding bit should be set to 0.

If this parameter is set to all zeros, the SELECT will not check for read events.

WSNDMSK

A bit string sent to request write event status.

- For each socket to be checked for pending write events, the corresponding bit in the string should be set to 1.
- For sockets to be ignored, the value of the corresponding bit should be set to 0.

If this parameter is set to all zeros, the SELECT will not check for write events.

ESNDMSK

A bit string sent to request exception event status.

- For each socket to be checked for pending exception events, the corresponding bit in the string should be set to 1.
- For each socket to be ignored, the corresponding bit should be set to 0.

If this parameter is set to all zeros, the SELECT will not check for exception events.

Parameter values returned to the application

RRETMSK

A bit string returned with the status of read events. The length of the string should be equal to the maximum number of sockets to be checked. For each socket that is ready to read, the corresponding bit in the string will be set to 1; bits that represent sockets that are not ready to read will be set to 0.

WRETMSK

A bit string returned with the status of write events. The length of the string should be equal to the maximum number of sockets to be checked. For each socket that is ready to write, the corresponding

bit in the string will be set to 1; bits that represent sockets that are not ready to be written will be set to 0.

ERETMSK

A bit string returned with the status of exception events. The length of the string should be equal to the maximum number of sockets to be checked. For each socket that has an exception status, the corresponding bit will be set to 1; bits that represent sockets that do not have exception status will be set to 0.

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See <u>Appendix A</u>, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

>0

Indicates the sum of all ready sockets in the three masks.

0

Indicates that the SELECT time limit has expired.

-1

Check **ERRNO** for an error code.

SELECTEX

The SELECTEX call monitors a set of sockets, a time value, and an ECB. It completes when either one of the sockets has activity, the time value expires, or one of the ECBs is posted.

To use the SELECTEX call as a timer in your program, do either of the following tasks:

- Set the read, write, and exception arrays to zeros.
- Specify MAXSOC ≤ 0.

Table 37. SELECTEX call requirements

Condition	Requirement
Authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN.
Amode:	31-bit or 24-bit.
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.
ASC mode:	Primary address space control (ASC) mode.
Interrupt status:	Enabled for interrupts.
Locks:	Unlocked.
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Figure 51 on page 150 shows an example of SELECTEX call instructions.

If an application intends to pass a single ECB on the SELECTEX call, then the corresponding working storage definitions and CALL instruction should be coded as shown in the following example:

WORKING-STORAGE SECTION. PIC X(16) VALUE IS 'SELECTEX'. 01 SOC-FUNCTION 01 MAXSOC PIC 9(8) BINARY. 01 TIMEOUT. 03 TIMEOUT-SECONDS PIC 9(8) BINARY. PIC 9(8) BINARY. 03 TIMEOUT-MINUTES 01 RSNDMSK PIC X(*). 01 WSNDMSK PIC X(*). 01 ESNDMSK 01 RRETMSK PIC X(*). PIC X(*). 01 WRETMSK PIC X(*). ERETMSKPIC X(*).SELECBPIC X(4).ERRNOPIC 9(8)RETCODEPIC S9(8) 01 01 01 BINARY. 01 BINARY. Where * is the size of the select mask PROCEDURE DIVISION. CALL 'EZASOKET' USING SOC-FUNCTION MAXSOC TIMEOUT RSNDMSK WSNDMSK ESNDMSK RRETMSK WRETMSK ERETMSK SELECB ERRNO RETCODE.

However, if the application intends to pass the address of an ECB list on the SELECTEX call, then the application must set the high order bit in the ECB list address and pass that address using the BY VALUE option as documented in the following example. The remaining parameters must be set back to the default by specifying BY REFERENCE before ERRNO:

WORKING-STORAGE SECTION. SOC-FUNCTION PIC X(16) 01 VALUE IS 'SELECTEX'. 01 MAXSOC PIC 9(8) BTNARY. 01 TIMEOUT. 03 TIMEOUT-SECONDS 03 TIMEOUT-MINUTES PIC 9(8) BINARY. PIC 9(8) BINARY. RSNDMSK PIC X(*). 01 01 WSNDMSK PIC X(*). RRETMSK WRETMSK ERETMSK ECPL TOT PIC X(*). 01 ESNDMSK 01 PIC X(*). 01 PIC X(*). PIC X(*) 01 EREIMSK ECBLIST-PTR ERRNO RETCODE USAGE IS POINTER. 01 PIC 9(8) BINARY. PIC S9(8) BINARY. 01 01 RETCODE Where * is the size of the select mask PROCEDURE DIVISION. CALL 'EZASOKET' USING SOC-FUNCTION MAXSOC TIMEOUT RSNDMSK WSNDMSK ESNDMSK RRETMSK WRETMSK ERETMSK BY VALUE ECBLIST-PTR BY REFERENCE ERRNO RETCODE.

* The bit mask lengths can be determined from the expression:

((maximum socket number +32)/32 (drop the remainder))*4

Figure 51. SELECTEX call instruction example

Defining which sockets to test

The SELECTEX call monitors for read operations, write operations, and exception operations:

- When a socket is ready to read, one of the following conditions has occurred:
 - A buffer for the specified sockets contains input data. If input data is available for a given socket, a
 read operation on that socket will not block.

- A connection has been requested on that socket.
- When a socket is ready to write, TCP/IP can accommodate additional output data. If TCP/IP can accept additional output for a given socket, a write operation on that socket will not block.
- When an exception condition has occurred on a specified socket it is an indication that a TAKESOCKET has occurred for that socket.
- A timeout occurs on the SELECTEX call. The timeout period can be specified when the SELECTEX call is issued.
- The ECB (or one of the ECBs in the ECB list) passed on the SELECTEX call has been posted.

Each socket descriptor is represented by a bit in a bit string. The length of this bit-mask array is dependent on the value of the MAXSOC parameter and must be a multiple of 4 bytes.

For information about <u>selecting requests in a concurrent server program</u>, see <u>z/OS Communications</u> Server: IP Sockets Application Programming Interface Guide and Reference.

Note: To simplify string processing in COBOL, you can use the program EZACIC06 to convert each bit in the string to a character. For more information, see <u>"EZACIC06 " on page 190</u>.

Read operations

Read operations include ACCEPT, READ, READV, RECV, RECVFROM, or RECVMSG calls. A socket is ready to be read when data has been received for it or when a connection request has occurred.

To test whether any of several sockets is ready for reading, set the appropriate bits in RSNDMSK to one before issuing the SELECTEX call. When the SELECTEX call returns, the corresponding bits in the RRETMSK indicate sockets are ready for reading.

Write operations

A socket is selected for writing (ready to be written) when:

- TCP/IP can accept additional outgoing data.
- The socket is marked nonblocking and a previous CONNECT did not complete immediately. In this case, CONNECT returned an ERRNO with a value of 36 (EINPROGRESS). This socket will be selected for write when the CONNECT completes.

A call to WRITE, SEND, or SENDTO blocks when the amount of data to be sent exceeds the amount of data TCP/IP can accept. To avoid this, you can precede the write operation with a SELECTEX call to ensure that the socket is ready for writing. Once a socket is selected for WRITE, the program can determine the amount of TCP/IP buffer space available by issuing the GETSOCKOPT call with the SO-SNDBUF option.

To test whether any of several sockets is ready for writing, set the WSNDMSK bits representing those sockets to 1 before issuing the SELECTEX call. When the SELECTEX call returns, the corresponding bits in the WRETMSK indicate sockets are ready for writing.

Exception operations

For each socket to be tested, the SELECTEX call can check for an existing exception condition. Two exception conditions are supported:

- The calling program (concurrent server) has issued a GIVESOCKET command and the target child server has successfully issued the TAKESOCKET call. When this condition is selected, the calling program (concurrent server) should issue CLOSE to dissociate itself from the socket.
- A socket has received out-of-band data. On this condition, a READ will return the out-of-band data ahead of program data.

To test whether any of several sockets have an exception condition, set the ESNDMSK bits representing those sockets to 1. When the SELECTEX call returns, the corresponding bits in the ERETMSK indicate sockets with exception conditions.

MAXSOC parameter

The SELECTEX call must test each bit in each string before returning results. For efficiency, the MAXSOC parameter can be used to specify the largest socket descriptor number that needs to be tested for any event type. The SELECTEX call tests only bits that are in the range 0 through the MAXSOC value minus 1.

Example: If MAXSOC is set to 50, the range is 0 - 49.

TIMEOUT parameter

If the time specified in the TIMEOUT parameter elapses before any event is detected, the SELECTEX call returns, and the RETCODE is set to 0.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing SELECT. The field is left-aligned and padded on the right with blanks.

MAXSOC

A fullword binary field that specifies the largest socket descriptor value that is being checked. The SELECTEX call tests only bits that are in the range 0 through the MAXSOC value minus 1. For example, if you set the MAXSOC value to 50, the range is 0 - 49.

TIMEOUT

If TIMEOUT is a positive value, it specifies a maximum interval to wait for the selection to complete. If TIMEOUT-SECONDS is a negative value, the SELECTEX call blocks until a socket becomes ready or an ECB or ECB in a list is posted. To poll the sockets and return immediately, set TIMEOUT to be zeros.

TIMEOUT is specified in the two-word TIMEOUT as follows:

- TIMEOUT-SECONDS, word one of the TIMEOUT field, is the seconds component of the timeout value.
- TIMEOUT-MICROSEC, word two of the TIMEOUT field, is the microseconds component of the timeout value (0-999999).

For example, if you want SELECTEX to time out after 3.5 seconds, set TIMEOUT-SECONDS to 3 and TIMEOUT-MICROSEC to 500000.

RSNDMSK

The bit-mask array to control checking for read interrupts. If this parameter is not specified or the specified bit-mask is zeros, the SELECT will not check for read interrupts. The length of this bit-mask array is dependent on the value in MAXSOC.

WSNDMSK

The bit-mask array to control checking for write interrupts. If this parameter is not specified or the specified bit-mask is zeros, the SELECT will not check for write interrupts. The length of this bit-mask array is dependent on the value in MAXSOC.

ESNDMSK

The bit-mask array to control checking for exception interrupts. If this parameter is not specified or the specified bit-mask is zeros, the SELECT will not check for exception interrupts. The length of this bit-mask array is dependent on the value in MAXSOC.

SELECB

An ECB which, if posted, causes completion of the SELECTEX.

ECBLIST-PTR

A pointer to an ECB list. The application must set the high order bit in the ECB list address and pass that address using the BY VALUE option. The remaining parameters must be set back to the default by specifying BY REFERENCE before ERRNO.
Parameter values returned to the application

ERRNO

A fullword binary field; if RETCODE is negative, this contains an error number. See <u>Appendix A</u>, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field

Value

Meaning

>0

The number of ready sockets.

0

Either the SELECTEX time limit has expired (ECB value is 0) or one of the caller's ECBs has been posted (ECB value is nonzero and the caller's descriptor sets is set to 0). The caller must initialize the ECB values to 0 before issuing the SELECTEX socket command.

-1

Check ERRNO for an error code.

RRETMSK

The bit-mask array returned by the SELECT if RSNDMSK is specified. The length of this bit-mask array is dependent on the value in MAXSOC.

WRETMSK

The bit-mask array returned by the SELECT if WSNDMSK is specified. The length of this bit-mask array is dependent on the value in MAXSOC.

ERETMSK

The bit-mask array returned by the SELECT if ESNDMSK is specified. The length of this bit-mask array is dependent on the value in MAXSOC.

SEND

The SEND call sends data on a specified connected socket.

The FLAGS field allows you to:

- Send out-of-band data, such as interrupts, aborts, and data marked urgent. Only stream sockets created in the AF_INET address family support out-of-band data.
- Suppress use of local routing tables. This implies that the caller takes control of routing and writing network software.

For datagram sockets, SEND transmits the entire datagram if it fits into the receiving buffer. Extra data is discarded.

For stream sockets, data is processed as streams of information with no boundaries separating the data. For example, if a program is required to send 1000 bytes, each call to this function can send any number of bytes, up to the entire 1000 bytes, with the number of bytes sent returned in RETCODE. Therefore, programs using stream sockets should place this call in a loop, reissuing the call until all data has been sent.

Note: See "EZACICO4" on page 188 for a subroutine that will translate EBCDIC input data to ASCII.

Condition	Requirement
Authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN.

Table 38. SEND call requirements

Table 38. SEND call requirements (continued)	
Condition	Requirement
Amode:	31-bit or 24-bit.
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.
ASC mode:	Primary address space control (ASC) mode.
Interrupt status:	Enabled for interrupts.
Locks:	Unlocked.
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Figure 52 on page 154 shows an example of SEND call instructions.

```
WORKING-STORAGE SECTION.

01 SOC-FUNCTION PIC X(16) VALUE IS 'SEND'.

01 S PIC 9(4) BINARY.

01 FLAGS PIC 9(8) BINARY.

88 NO-FLAG VALUE IS 0.

88 00B VALUE IS 1.

88 DONT-ROUTE VALUE IS 4.

01 NBYTE PIC 9(8) BINARY.

01 BUF PIC X(length of buffer).

01 ERRNO PIC 9(8) BINARY.

01 RETCODE PIC S9(8) BINARY.

PROCEDURE DIVISION.

CALL 'EZASOKET' USING SOC-FUNCTION S FLAGS NBYTE

BUF ERRNO RETCODE.
```

Figure 52. SEND call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing SEND. The field is left-aligned and padded on the right with blanks.

S

A halfword binary number specifying the socket descriptor of the socket that is sending data.

FLAGS

A fullword binary field with values as follows:

Literal Value	Binary Value	Description
NO-FLAG	X'00000000'	No flag is set. The command behaves like a WRITE call.
MSG-OOB	X'0000001'	Send out-of-band data. (Stream sockets only.) Even if the OOB flag is not set, out-of-band data can be read if the SO-OOBINLINE option is set for the socket.
MSG-DONTROUTE	X'0000004'	Do not route. Routing is provided by the calling program.

NBYTE

A fullword binary number set to the number of bytes of data to be transferred.

BUF

The buffer containing the data to be transmitted. BUF should be the size specified in NBYTE.

Parameter values returned to the application

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix A, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

≥0

A successful call. The value is set to the number of bytes transmitted.

-1

Check ERRNObe for an error code.

SENDMSG

The SENDMSG call sends messages on a socket with descriptor S passed in an array of messages.

Table 39. SENDMSG call requirements		
Condition	Requirement	
Authorization:	Supervisor state or problem state, any PSW key.	
Dispatchable unit mode:	Task.	
Cross memory mode:	PASN = HASN.	
Amode:	31-bit or 24-bit.	
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.	
ASC mode:	Primary address space control (ASC) mode.	
Interrupt status:	Enabled for interrupts.	
Locks:	Unlocked.	
Control parameters:	All parameters must be addressable by the caller and in the primary address space.	

"SENDMSG call instruction example (part 1 of 2)" on page 155 and "SENDMSG call instruction example (part 2 of 2)" on page 157 show an example of SENDMSG call instructions.

SENDMSG call instruction example (part 1 of 2)

WORKING-STORAGE	SECT	ION.					
01	SOC-	FUNCTION	PIC	X(16)	VALUE	IS	'SENDMSG'.
01	S		PIC	9(4)	BINARY		
01	MSG-	HDR.					
	03	MSG-NAME		USAGE 1	IS POIN	TER	
	03	MSG-NAME-LEI	N	PIC 9(8	8) BINA	RY.	
	03	IOV		USAGE 1	IS POIN	TER	
	03	IOVCNT		USAGE 1	IS POIN	TER	
	03	MSG-ACCRIGH	TS	USAGE 1	IS POIN	TER	

03 MSG-ACCRIGHTS-LEN USAGE IS POINTER.

01	FLAGS 88 NO-FLAG 88 OOB 88 DONTROUTE	PIC 9(8)	BINARY. VALUE VALUE VALUE	IS IS IS	0. 1. 4.
01	ERRNO	PIC 9(8)	BINARY.		
01	RETCODE	PIC S9(8)	BINARY.		
01 01	SENDMSG-IPV4ADD SENDMSG-IPV6ADD	R PIC 9(8) R.	BINARY.		
	05 FILLER	PIC9(16)	BINARY.		
	05 FILLER	PIC9(16)	BINARY.		

LINKAGE SECTION. 01 L1. 03 SENDMSG-IOVECTOR. USAGE IS POINTER. PIC 9(8) COMP. PIC 9(8) COMP. USAGE IS POINTER. PIC 9(8) COMP. PIC 9(8) COMP. USAGE IS POINTER. PIC 9(8) COMP. PIC 9(8) COMP. PIC 9(8) COMP. 05 IOV1A 05 IOV1AL 05 IOV1L 05 IOV2A 05 IOV2AL 05 IOV2L 05 IOV3A 05 IOV3AL 05 IOV3L PIC X(16). PIC X(16). PIC X(16). PIC 9(8) COMP. PIC X(16). PIC X(16). PIC X(16). 03 SENDMSG-BUFFER1 03 SENDMSG-BUFFER2 03 SENDMSG-BUFFER3 03 SENDMSG-BUFNO

* IPv4 socket address structure.

03 NAME.

FAMILY	PIC	9(4)	BINARY
PORT	PIC	9(4)	BINARY.
IP-ADDRESS	PIC	9(8)	BINARY
RESERVED	PIC	X(8)	BINARY.
	FAMILY PORT IP-ADDRESS RESERVED	FAMILYPICPORTPICIP-ADDRESSPICRESERVEDPIC	FAMILYPIC 9(4)PORTPIC 9(4)IP-ADDRESSPIC 9(8)RESERVEDPIC X(8)

* IPv6 socket address structure.

03	NAMI	E.			
	05	FAMILY	PIC	9(4)	BINARY.
	05	PORT	PIC	9(4)	BINARY.
	05	FLOWINFO	PIC	9(8)	BINARY.
	05	IP-ADDRESS.			
		10 FILLER	PIC	9(16)) BINARY.
		10 FILLER	PIC	9(16)) BINARY.
	05	SCOPE-ID	PIC	9(8)	BINARY.

PROCEDURE DIVISION USING L1.

```
* For IPv6.
MOVE 19 TO FAMILY.
MOVE 1234 TO PORT.
MOVE 0 TO FLOWINFO.
MOVE SENDMSG-IPV6ADDR TO IP-ADDRESS.
MOVE 0 TO SCOPE-ID.
* For IPv4.
MOVE 2 TO FAMILY.
MOVE 1234 TO PORT.
MOVE SENDMSG-IPV4ADDR TO IP-ADDRESS.
```

SET MSG-NAME TO ADDRESS OF NAME. MOVE LENGTH OF NAME TO MSG-NAME-LEN. SET IOV TO ADDRESS OF SENDMSG-IOVECTOR. MOVE 3 TO SENDMSG-BUFNO. SET MSG-IOVCNT TO ADDRESS OF SENDMSG-BUFNO. SET IOV1A TO ADDRESS OF SENDMSG-BUFFER1. MOVE 0 TO IOV1AL.

```
MOVE LENGTH OF SENDMSG-BUFFER1 TO IOV1L.
SET IOV2A TO ADDRESS OF SENDMSG-BUFFER2.
MOVE 0 TO IOV2AL.
MOVE LENGTH OF SENDMSG-BUFFER2 TO IOV2L.
SET IOV3A TO ADDRESS OF SENDMSG-BUFFER3.
MOVE 0 TO IOV3AL.
MOVE 0 TO IOV3AL.
SET MSG-ACCRIGHTS TO NULLS.
SET MSG-ACCRIGHTS TO NULLS.
SET MSG-ACCRIGHTS-LEN TO NULLS.
MOVE 0 TO FLAGS.
MOVE 'MESSAGE TEXT 1 ' TO SENDMSG-BUFFER1.
MOVE 'MESSAGE TEXT 2 ' TO SENDMSG-BUFFER2.
MOVE 'MESSAGE TEXT 3 ' TO SENDMSG-BUFFER3.
```

CALL 'EZASOKET' USING SOC-FUNCTION S MSG-HDR FLAGS ERRNO RETCODE.

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing SENDMSG. The field is left-aligned and padded on the right with blanks.

S

A value or the address of a halfword binary number specifying the socket descriptor.

MSG

A pointer to an array of message headers from which messages are sent.

Field

Description

NAME

On input, a pointer to a buffer where the sender's address is stored upon completion of the call. The storage being pointed to should be for an IPv4 socket address or an IPv6 socket address. The IPv4 socket address structure contains the following fields:

Field

Description

FAMILY

Output parameter. A halfword binary number specifying the IPv4 addressing family. The value for IPv4 socket descriptor (S parameter) is decimal 2, indicating AF_INET.

PORT

Output parameter. A halfword binary number specifying the port number of the sending socket.

IP-ADDRESS

Output parameter. A fullword binary number specifying the 32-bit IPv4 IP address of the sending socket.

RESERVED

Output parameter. An 8-byte reserved field. This field is required, but is not used.

The IPv6 socket address structure contains the following fields:

Field

Description

FAMILY

Output parameter. A halfword binary number specifying the IPv6 addressing family. The value for IPv6 socket descriptor (S parameter) is decimal 19, indicating AF_INET6.

PORT

Output parameter. A halfword binary number specifying the port number of the sending socket.

FLOWINFO

A fullword binary field specifying the traffic class and flow label. This field must be set to 0.

IP-ADDRESS

Output parameter. A 16-byte binary field set to the 128-bit IPv6 IP address of the sending socket.

SCOPE-ID

A fullword binary field which identifies a set of interfaces as appropriate for the scope of the address carried in the IPv6-ADDRESS field. A value of 0 indicates the SCOPE-ID field does not identify the set of interfaces to be used, and can be specified for any address types and scopes. For a link scope IPv6-ADDRESS, SCOPE-ID can specify a link index which identifies a set of interfaces. For all other address scopes, SCOPE-ID must be set to 0.

NAME-LEN

On input, a pointer to the size of the address buffer.

IOV

On input, a pointer to an array of three fullword structures with the number of structures equal to the value in IOVCNT and the format of the structures as follows:

Fullword 1

A pointer to the address of a data buffer.

Fullword 2

Reserved.

Fullword 3

A pointer to the length of the data buffer referenced in Fullword 1.

In COBOL, the IOV structure must be defined separately in the Linkage section, as shown in the example.

IOVCNT

On input, a pointer to a fullword binary field specifying the number of data buffers provided for this call.

ACCRIGHTS

On input, a pointer to the access rights received. This field is ignored.

ACCRIGHTS-LEN

On input, a pointer to the length of the access rights received. This field is ignored.

FLAGS

A fullword field containing the following information:

Literal Value	Binary Value	Description
NO-FLAG	X'0000000'	No flag is set. The command behaves like a WRITE call.
MSG-OOB	X'0000001'	Send out-of-band data. (Stream sockets only.)
MSG-DONTROUTE	X'00000004'	Do not route. Routing is provided by the calling program.

Parameter values returned to the application

ERRNO

A fullword binary field. If RETCODE is negative, this contains an error number. See <u>Appendix A</u>, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

≥0

A successful call. The value is set to the number of bytes transmitted.

-1

Check ERRNO for an error code.

SENDTO

SENDTO is similar to SEND, except that it includes the destination address parameter. The destination address allows you to use the SENDTO call to send datagrams on a UDP socket, regardless of whether the socket is connected.

The FLAGS parameter allows you to:

- · Send out-of-band data, such as interrupts, aborts, and data marked as urgent.
- Suppress use of local routing tables. This implies that the caller takes control of routing, which requires writing network software.

For datagram sockets, SENDTO transmits the entire datagram if it fits into the receiving buffer. Extra data is discarded.

For stream sockets, data is processed as streams of information with no boundaries separating the data. For example, if a program is required to send 1000 bytes, each call to this function can send any number of bytes, up to the entire 1000 bytes, with the number of bytes sent returned in RETCODE. Therefore, programs using stream sockets should place SENDTO in a loop that repeats the call until all data has been sent.

Note: See "EZACICO4" on page 188 for a subroutine that will translate EBCDIC input data to ASCII.

Condition	Requirement	
Authorization:	Supervisor state or problem state, any PSW key.	
Dispatchable unit mode:	Task.	
Cross memory mode:	PASN = HASN.	
Amode:	31-bit or 24-bit.	
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.	
ASC mode:	Primary address space control (ASC) mode.	
Interrupt status:	Enabled for interrupts.	
Locks:	Unlocked.	
Control parameters:	All parameters must be addressable by the caller and in the primary address space.	

Table 40. SENDTO call requirements

Figure 53 on page 160 shows an example of SENDTO call instructions.

WORKING-STORAGE SECTION. O1SOC-FUNCTIONPIC X(16)VALUE IS 'SENDTO'.O1SPIC 9(4)BINARY.O1FLAGS.PIC 9(8)BINARY.88NO-FLAGVALUE IS 0.8800BVALUE IS 1. 88 DONT-ROUTE VALUE IS 4. 01 NBYTE PIC 9(8) BINARY. PIC X(length of buffer). 01 BUF * IPv4 socket address structure. 01 NAME 03FAMILYPIC 9(4) BINARY.03PORTPIC 9(4) BINARY.03IP-ADDRESSPIC 9(8) BINARY.03RESERVEDPIC X(8). * IPv6 socket address structure. 01 NAME 03 FAMILY PIC 9(4) BINARY. 03PORTPIC 9(4) BINARY.03FLOWINFOPIC 9(8) BINARY. 03 IP-ADDRESS. 10 FILLER PIC 9(16) BINARY. 10 FILLER PIC 9(16) BINARY. 03 SCOPE-ID PIC 9(8) BINARY. 01 ERRNO PIC 9(8) BINARY. PIC S9(8) BINARY. 01 RETCODE PROCEDURE DIVISION. CALL 'EZASOKET' USING SOC-FUNCTION S FLAGS NBYTE BUF NAME ERRNO RETCODE.

Figure 53. SENDTO call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing SENDTO. The field is left-aligned and padded on the right with blanks.

S

A halfword binary number set to the socket descriptor of the socket sending the data.

FLAGS

A fullword field that returns one of the following information:

Literal Value	Binary Value	Description
NO-FLAG	X'0000000'	No flag is set. The command behaves like a WRITE call.
MSG-OOB	X'0000001'	Send out-of-band data. (Stream sockets only.)
MSG-DONTROUTE	X'0000004'	Do not route. Routing is provided by the calling program.

NBYTE

A fullword binary number set to the number of bytes to transmit.

BUF

Specifies the buffer containing the data to be transmitted. BUF should be the size specified in NBYTE.

NAME

Specifies the IPv4 socket address structure as follows:

FAMILY

A halfword binary field containing the IPv4 addressing family. For TCP/IP the value must be decimal 2, indicating AF_INET.

PORT

A halfword binary field containing the port number bound to the socket.

IP-ADDRESS

A fullword binary field containing the socket's 32-bit IPv4 IP address.

RESERVED

Specifies an 8-byte reserved field. This field is required, but not used.

Specifies the IPv6 socket address structure as follows:

FAMILY

A halfword binary field containing the IPv6 addressing family. For TCP/IP the value is decimal 19, indicating AF_INET6.

PORT

A halfword binary field containing the port number bound to the socket.

FLOWINFO

A fullword binary field specifying the traffic class and flow label. This field must be set to 0.

IP-ADDRESS

A 16-byte binary field set to the 128-bit IPv6 IP address, in network byte order.

SCOPE-ID

A fullword binary field which identifies a set of interfaces as appropriate for the scope of the address carried in the IPv6-ADDRESS field. A value of 0 indicates the SCOPE-ID field does not identify the set of interfaces to be used, and can be specified for any address types and scopes. For a link scope IPv6-ADDRESS, SCOPE-ID can specify a link index which identifies a set of interfaces. For all other address scopes, SCOPE-ID must be set to 0.

Parameter values returned to the application

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See <u>Appendix A</u>, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

≥0

A successful call. The value is set to the number of bytes transmitted.

-1

Check ERRNO for an error code.

SETSOCKOPT

The SETSOCKOPT call sets the options associated with a socket. SETSOCKOPT can be called only for sockets in the AF_INET or AF_INET6 domains.

The OPTVAL and OPTLEN parameters are used to pass data used by the particular set command. The OPTVAL parameter points to a buffer containing the data needed by the set command. The OPTLEN parameter must be set to the size of the data pointed to by OPTVAL.

Table 41. SETSOCKOPT call requirements

Condition	Requirement
Authorization:	Supervisor state or problem state, any PSW key.

Table 41. SETSOCKOPT call requirements (continued)	
--	--

Condition	Requirement
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN.
Amode:	31-bit or 24-bit.
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.
ASC mode:	Primary address space control (ASC) mode.
Interrupt status:	Enabled for interrupts.
Locks:	Unlocked.
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Figure 54 on page 162 shows an example of SETSOCKOPT call instructions.

WORKING	-STORAGE SECTION	۷.					
01	SOC-FUNCTION	PIC	X(16)	VALUE	IS	'SETSO	CKOPT'
01	S	PIC	9(4) I	BINARY.			
01	OPTNAME	PIC	9(8) I	BINARY.			
01	OPTVAL	PIC	9(16)	BINARY			
01	OPTLEN	PIC	9(8) I	BINARY.			
01	ERRNO	PIC	9(8) I	BINARY.			
01	RETCODE	PIC	S9(8)	BINARY			
01	OPTVAL	PIC	9(16)	BINARY			
01	OPTLEN	PIC	9(8)	BINARY			
01	ERRNO	PIC	9(8)	BINARY			
01	RETCODE	PIC	S9(8)	BINARY			
PROCEDU	RE DIVISION						
CA	LL 'EZASOKET' US	SING S	SOC-FUI	NCTION S	S OF	PTNAME	
	0P	FVAL (OPTLEN	ERRNO I	RETO	CODE.	

Figure 54. SETSOCKOPT call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Table 42. OPTNAME options for GETSOCKOPT and SETSOCKOPT

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
IP_ADD_MEMBERSHIP Use this option to enable an application to join a multicast group on a specific interface. An interface has to be specified with this option. Only applications that want to receive multicast datagrams need to join multicast groups. This is an IPv4-only socket option.	Contains the IP_MREQ structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 interface address. See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ. See SEZAINST(EZACOBOL) for the COBOL example of IP- MREQ.	N/A

Table 42. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)				
OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)		
IP_ADD_SOURCE_MEMBERSHIP Use this option to enable an application to join a source multicast group on a specific interface and a specific source address. You must specify an interface and a source address with this option. Applications that want to receive multicast datagrams need to join source multicast groups. This is an IPv4-only socket option.	Contains the IP_MREQ_SOURCE structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ_SOURCE structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 source address and a 4- byte IPv4 interface address. See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE. See SEZAINST(EZACOBOL) for the COBOL example of IP- MREQ-SOURCE.	N/A		
IP_BLOCK_SOURCE Use this option to enable an application to block multicast packets that have a source address that matches the given IPv4 source address. You must specify an interface and a source address with this option. The specified multicast group must have been joined previously. This is an IPv4-only socket option.	Contains the IP_MREQ_SOURCE structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ_SOURCE structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 source address and a 4- byte IPv4 interface address. See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE. See SEZAINST(EZACOBOL) for the COBOL example of IP- MREQ-SOURCE.	N/A		
IP_DROP_MEMBERSHIP Use this option to enable an application to exit a multicast group or to exit all sources for a multicast group. This is an IPv4-only socket option.	Contains the IP_MREQ structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 interface address. See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ. See SEZAINST(EZACOBOL) for the COBOL example of IP- MREQ.	N/A		

Table 42. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)				
OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)		
IP_DROP_SOURCE_MEMBERSHIP	Contains the IP_MREQ_SOURCE	N/A		
Use this option to enable an application to exit a source multicast group. This is an IPv4-only socket option.	structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREO_SOURCE structure			
	contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 source address and a 4- byte IPv4 interface address.			
	See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE.			
	See SEZAINST(EZACOBOL) for the COBOL example of IP- MREQ-SOURCE.			
IP_MULTICAST_IF	A 4-byte binary field containing	A 4-byte binary field containing		
Use this option to set or obtain the IPv4 interface address used for sending outbound multicast datagrams from the socket application.	an IPv4 interface address.	an IPv4 interface address.		
This is an IPv4-only socket option.				
Note: Multicast datagrams can be transmitted only on one interface at a time.				
IP_MULTICAST_LOOP	A 1-byte binary field.	A 1-byte binary field.		
Use this option to control or determine	To enable, set to 1.	If enabled, will contain a 1.		
whether a copy of multicast datagrams are looped back for multicast datagrams sent to a group to which the sending host itself belongs. The default is to loop the datagrams back.	To disable, set to 0.	If disabled, will contain a 0.		
This is an IPv4-only socket option.				
IP_MULTICAST_TTL	A 1-byte binary field containing	A 1-byte binary field containing		
Use this option to set or obtain the IP time-to-live of outgoing multicast datagrams. The default value is '01'x meaning that multicast is available only to the local subnet.	the value of '00'x to 'FF'x.	the value of '00'x to 'FF'x.		
This is an IPv4-only socket option.				

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
IP_UNBLOCK_SOURCE Use this option to enable an application to unblock a previously blocked source for a given IPv4 multicast group. You must specify an interface and a source address with this option. This is an IPv4-only socket option.	Contains the IP_MREQ_SOURCE structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ_SOURCE structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 source address and a 4- byte IPv4 interface address. See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE. See SEZAINST(EZACOBOL) for the COBOL example of IP- MREQ-SOURCE.	

Table 42. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)	
IPV6_ADDR_PREFERENCES Use this option to query or set IPv6 address preferences of a socket. The default source address selection algorithm considers these preferences	Contains the 4-byte flags field IPV6_ADDR_PREFERENCES_ FLAGS that is defined in SYS1.MACLIB(BPXYSOCK) with the following flags:	Contains the 4-byte flags field IPV6_ADDR_PREFERENCES_ FLAGS that is defined in SYS1.MACLIB(BPXYSOCK) with the following flags:	
when it selects an IP address that is appropriate to communicate with a given destination address.	IPV6_PREFER_SRC_HOME (X'00000001') Prefer home address	IPV6_PREFER_SRC_HOME (X'00000001') Prefer home address	
This is an AF_INET6-only socket option.	IPV6_PREFER_SRC_COA (X'00000002')	IPV6_PREFER_SRC_COA (X'00000002')	
Result: These flags are only preferences. The stack could assign a source IP address that does not conform to the IPV6_ADDR_PREFERENCES flags that you specify	Prefer care-of address IPV6_PREFER_SRC_TMP (X'00000004') Prefer temporary address	Prefer care-of address IPV6_PREFER_SRC_TMP (X'00000004') Prefer temporary address	
Guideline: Use the INET6_IS_SRCADDR function to test whether the source IP address matches one or more IPV6_ADDR_PREFERENCES flags.	IPV6_PREFER_SRC_PUBLIC (X'00000008') Prefer public address	IPV6_PREFER_SRC_PUBLIC (X'00000008') Prefer public address	
	IPV6_PREFER_SRC_CGA (X'00000010') Prefer cryptographically generated address	IPV6_PREFER_SRC_CGA (X'00000010') Prefer cryptographically generated address	
	IPV6_PREFER_SRC_NONCGA (X'00000020') Prefer non-cryptographically generated address	IPV6_PREFER_SRC_ NONCGA (X'00000020') Prefer non-cryptographically generated address	
	Some of these flags are contradictory. Combining contradictory flags, such as IPV6_PREFER_SRC_CGA and IPV6_PREFER_SRC_NONCGA, results in error code EINVAL.	See IPV6_ADDR_ PREFERENCES and Mapping of GAI_HINTS/GAI_ADDRINFO EFLAGS in SEZAINST(CBLOCK) for the PL/I example of the OPTNAME and flag definitions.	
	See IPV6_ADDR_PREFERENCES and Mapping of GAI_HINTS/ GAI_ADDRINFO EFLAGS in SEZAINST(CBLOCK) for the PL/I example of the OPTNAME and flag definitions.	See IPV6_ADDR_PREFERENCES and AI_EFLAGS mappings in SEZAINST(EZACOBOL) for the COBOL example of the OPTNAME and flag definitions.	
	See IPV6_ADDR_PREFERENCES and AI_EFLAGS mappings in SEZAINST(EZACOBOL) for the COBOL example of the OPTNAME and flag definitions.		

Table 42. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)				
OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)		
IPV6_JOIN_GROUP Use this option to control the reception of multicast packets and specify that the socket join a multicast group. This is an IPv6-only socket option.	Contains the IPV6_MREQ structure as defined in SYS1.MACLIB(BPXYSOCK). The IPV6_MREQ structure contains a 16-byte IPv6 multicast address followed by a 4-byte IPv6 interface index number. If the interface index number is 0, then the stack chooses the local interface. See the SEZAINST(CBLOCK) for the PL/I example of IPV6_MREQ. See SEZAINST(EZACOBOL) for the COBOL example of IPV6- MREQ.	N/A		
IPV6_LEAVE_GROUP Use this option to control the reception of multicast packets and specify that the socket leave a multicast group. This is an IPv6-only socket option.	Contains the IPV6_MREQ structure as defined in SYS1.MACLIB(BPXYSOCK). The IPV6_MREQ structure contains a 16-byte IPv6 multicast address followed by a 4-byte IPv6 interface index number. If the interface index number is 0, then the stack chooses the local interface. See the SEZAINST(CBLOCK) for the PL/I example of IPV6_MREQ. See SEZAINST(EZACOBOL) for the COBOL example of IPV6- MREQ.	N/A		
IPV6_MULTICAST_HOPS Use to set or obtain the hop limit used for outgoing multicast packets. This is an IPv6-only socket option.	Contains a 4-byte binary value specifying the multicast hops. If not specified, then the default is 1 hop. -1 indicates use stack default. 0 – 255 is the valid hop limit range. Note: An application must be APF authorized to enable it to set the hop limit value above the system defined hop limit value. CICS applications cannot execute as APF authorized.	Contains a 4-byte binary value in the range 0 – 255 indicating the number of multicast hops.		

Table 42. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)				
OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)		
IPV6_MULTICAST_IF	Contains a 4-byte binary field	Contains a 4-byte binary field		
Use this option to set or obtain the index of the IPv6 interface used for sending outbound multicast datagrams from the socket application.	containing an IPv6 interface index number.	containing an IPv6 interface index number.		
This is an IPv6-only socket option.				
IPV6_MULTICAST_LOOP	A 4-byte binary field.	A 4-byte binary field.		
Use this option to control or determine	To enable, set to 1.	If enabled, contains a 1.		
whether a multicast datagram is looped back on the outgoing interface by the IP layer for local delivery when datagrams are sent to a group to which the sending host itself belongs. The default is to loop multicast datagrams back.	To disable, set to 0.	If disabled, contains a 0.		
This is an IPv6-only socket option.				
IPV6_UNICAST_HOPS	Contains a 4-byte binary value	Contains a 4-byte binary value in		
Use this option to set or obtain the hop limit used for outgoing unicast IPv6 packets.	specifying the unicast hops. If not specified, then the default is 1 hop.	the range 0 – 255 indicating the number of unicast hops.		
This is an IPv6-only socket option.	-1 indicates use stack default.			
	0 – 255 is the valid hop limit range.			
	Note: APF authorized applications are permitted to set a hop limit that exceeds the system configured default. CICS applications cannot execute as APF authorized.			
IPV6_V60NLY	A 4-byte binary field.	A 4-byte binary field.		
Use this option to set or determine whether the socket is restricted to sock	To enable, set to 1.	If enabled, contains a 1.		
and receive only IPv6 packets. The default is to not restrict the sending and receiving of only IPv6 packets.	To disable, set to 0.	If disabled, contains a 0.		
This is an IPv6-only socket option.				

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
MCAST_BLOCK_SOURCE Use this option to enable an application to block multicast packets that have a source address that matches the given source address. You must specify an interface index and a source address with this option. The specified multicast group must have been joined previously.	Contains the GROUP_SOURCE_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_SOURCE_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address and a socket address structure of the source address.	N/A
	See SEZAINST(CBLOCK) for the PL/I example of GROUP_SOURCE_REQ.	
	See SEZAINST(EZACOBOL) for the COBOL example of GROUP- SOURCE-REQ.	
MCAST_JOIN_GROUP Use this option to enable an application to join a multicast group on a specific interface. You must specify an interface index. Applications that want to receive multicast datagrams must join multicast groups.	Contains the GROUP_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address.	N/A
	See SEZAINST(CBLOCK) for the PL/I example of GROUP_REQ.	
	See SEZAINST(EZACOBOL) for the COBOL example of GROUP- REQ.	
MCAST_JOIN_SOURCE_GROUP Use this option to enable an application to join a source multicast group on a specific interface and a source address. You must specify an interface index and the source address. Applications that want to receive multicast datagrams only from specific source addresses need to join source multicast groups.	Contains the GROUP_SOURCE_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_SOURCE_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address and a socket address structure of the source address.	N/A
	See SEZAINST(CBLOCK) for the PL/I example of GROUP_SOURCE_REQ.	
	See SEZAINST(EZACOBOL) for the COBOL example of GROUP- SOURCE-REQ.	

Table 42. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)		
OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
MCAST_LEAVE_GROUP Use this option to enable an application to exit a multicast group or exit all sources for a given multicast groups.	Contains the GROUP_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address.	N/A
	See SEZAINST(CBLOCK) for the PL/I example of GROUP_REQ.	
	See SEZAINST(EZACOBOL) for the COBOL example of GROUP- REQ.	
MCAST_LEAVE_SOURCE_GROUP Use this option to enable an application to exit a source multicast group.	Contains the GROUP_SOURCE_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_SOURCE_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address and a socket address structure of the source address. See SEZAINST(CBLOCK) for	N/A
	the PL/I example of GROUP_SOURCE_REQ.	
	See SEZAINST(EZACOBOL) for the COBOL example of GROUP- SOURCE-REQ.	
MCAST_UNBLOCK_SOURCE Use this option to enable an application to unblock a previously blocked source for a given multicast group. You must specify an interface index and a source address with this option.	Contains the GROUP_SOURCE_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_SOURCE_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address and a socket address structure of the source address.	N/A
	See SEZAINST(CBLOCK) for the PL/I example of GROUP_SOURCE_REQ.	
	See SEZAINST(EZACOBOL) for the COBOL example of GROUP- SOURCE-REQ.	

Table 42. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)		
OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
S0_ASCII	To enable, set to ON.	If enabled, contains ON.
Use this option to set or determine the	To disable, set to OFF.	If disabled, contains OFF.
SO_ASCII is set, data is translated to ASCII. When SO_ASCII is not set, data is not translated to or from ASCII.	Note: The <i>optvalue</i> is returned and is optionally followed by the name of the translation table that is used if translation is	Note: The <i>optvalue</i> is returned and is optionally followed by the name of the translation table that is used if translation is
Note: This is a REXX-only socket option.		
SO_BROADCAST	A 4-byte binary field.	A 4-byte field.
Use this option to set or determine whether a program can send broadcast	To enable, set to 1 or a positive	If enabled, contains a 1.
messages over the socket to destinations that can receive datagram messages. The default is disabled.	Value. To disable, set to 0.	If disabled, contains a 0.
Note: This option has no meaning for stream sockets.		
SO_DEBUG	To enable, set to ON.	If enabled, contains ON.
Use SO_DEBUG to set or determine the status of the debug option. The default is <i>disabled</i> . The debug option controls the recording of debug information.	To disable, set to OFF.	If disabled, contains OFF.
Notes:		
1. This is a REXX-only socket option.		
2. This option has meaning only for stream sockets.		
SO_EBCDIC	To enable, set to ON.	If enabled, contains ON.
Use this option to set or determine the	To disable, set to OFF.	If disabled, contains OFF.
SO_EBCDIC is set, data is translated to EBCDIC. When SO_EBCDIC is not set, data is not translated to or from EBCDIC. This option is ignored by EBCDIC hosts.	Note: The <i>optvalue</i> is returned and is optionally followed by the name of the translation table that is used if translation is applied to the data.	Note: The <i>optvalue</i> is returned and is optionally followed by the name of the translation table that is used if translation is applied to the data.
Note: This is a REXX-only socket option.		
SO_ERROR	N/A	A 4-byte binary field containing
Use this option to request pending errors on the socket or to check for asynchronous errors on connected datagram sockets or for other errors that are not explicitly returned by one of the socket calls. The error status is clear afterwards.		socket.

Table 42. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)		
OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
SO_KEEPALIVE	A 4-byte binary field.	A 4-byte binary field.
Use this option to set or determine	To enable, set to 1 or a positive	If enabled, contains a 1.
periodically sends a packet on an otherwise idle connection for a stream socket.	value. To disable, set to 0.	If disabled, contains a 0.
The default is disabled.		
When activated, the keep alive mechanism periodically sends a packet on an otherwise idle connection. If the remote TCP does not respond to the packet or to retransmissions of the packet, the connection is terminated with the error ETIMEDOUT.		
SO_LINGER	Contains an 8-byte field	Contains an 8-byte field
Use this option to control or determine how TCP/IP processes data that has not	fields.	fields.
been transmitted when a CLOSE is issued for the socket. The default is disabled.	Assembler coding:	Assembler coding:
Notes:	ONOFF DS F LINGER DS F	ONOFF DS F LINGER DS F
 This option has meaning only for stream sockets. 	COBOL coding:	COBOL coding:
2. If you set a zero linger time, the connection cannot close in an orderly manner, but stops, resulting in a RESET segment being sent to the connection partner. Also, if the aborting socket is in nonblocking mode, the close call is treated as though no linger option had been set.	ONOFF PIC 9(8) BINARY. LINGER PIC 9(8) BINARY.	ONOFF PIC 9(8) BINARY. LINGER PIC 9(8) BINARY.
	Set ONOFF to a nonzero value to enable and set to 0 to disable this option. Set LINGER to the number of seconds that TCP/IP lingers after the CLOSE is issued.	A nonzero value returned in ONOFF indicates enabled, a 0 indicates disabled. LINGER indicates the number of seconds that TCP/IP will try to send data
When SO_LINGER is set and CLOSE is called, the calling program is blocked until the data is successfully transmitted or the connection has timed out.		alter the CLOSE is issued.
When SO_LINGER is not set, the CLOSE returns without blocking the caller, and TCP/IP continues to attempt to send data for a specified time. This usually allows sufficient time to complete the data transfer.		
Use of the SO_LINGER option does not guarantee successful completion because TCP/IP waits only the amount of time specified in OPTVAL for SO_LINGER.		

Table 42. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)		
OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
SO_OOBINLINE	A 4-byte binary field.	A 4-byte binary field.
Use this option to control or determine whether out-of-band data is received.	To enable, set to 1 or a positive value.	If enabled, contains a 1.
Note: This option has meaning only for stream sockets.	To disable, set to 0.	in disabled, contains a 0.
When this option is set, out-of-band data is placed in the normal data input queue as it is received and is available to a RECV or a RECVFROM even if the OOB flag is not set in the RECV or the RECVFROM.		
When this option is disabled, out-of-band data is placed in the priority data input queue as it is received and is available to a RECV or a RECVFROM only when the OOB flag is set in the RECV or the RECVFROM.		
S0_RCVBUF	A 4-byte binary field.	A 4-byte binary field.
Use this option to control or determine the size of the data portion of the TCP/IP receive buffer.	To enable, set to a positive value specifying the size of the data portion of the TCP/IP receive	If enabled, contains a positive value indicating the size of the data portion of the TCP/IP
The size of the data portion of the receive	buffer.	receive buffer.
following values prior to any SETSOCKOPT call:	To disable, set to a 0.	If disabled, contains a 0.
 TCPRCVBufrsize keyword on the TCPCONFIG statement in the PROFILE.TCPIP data set for a TCP Socket 		
 UDPRCVBufrsize keyword on the UDPCONFIG statement in the PROFILE.TCPIP data set for a UDP Socket 		
• The default of 65535 for a raw socket		

Table 42. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
SO_RCVTIMEO	This option requires a TIMEVAL	This option stores a TIMEVAL
Use this option to control or determine the maximum length of time that a receive-type function can wait before it completes.	structure, which is defined in SYS1.MACLIB(BPXYRLIM) macro. The TIMEVAL structure contains the number of seconds and microseconds specified	structure that is defined in the SYS1.MACLIB(BPXYRLIM) macro. The TIMEVAL structure contains the number of seconds and microseconds, which are
If a receive-type function has blocked for the maximum length of time that was specified without receiving data, control is returned with an errno set to EWOULDBLOCK. The default value for this option is 0, which indicates that a receive- type function does not time out. When the MSG_WAITALL flag (stream	as fullword binary numbers. The seconds can be a value in the range 0 - 2678400 (equal to 31 days), and the microseconds can be a value in the range 0 - 1000000 (equal to 1 second). Although TIMEVAL value can be specified	specified as fullword binary numbers. The number of seconds value that is returned is in the range 0 - 2678400 (equal to 31 days). The number of microseconds value that is returned is in the range 0 - 1000000.
sockets only) is specified, the timeout takes precedence. The receive-type function can return the partial count. See the explanation of that operation's MSG_WAITALL flag parameter.	the internal TCP/IP timers that are used to implement this function have a granularity of approximately 100 milliseconds.	
The following receive-type functions are supported:		
• READ		
• READV		
• RECV		
RECVFROM		
RECVMSG		

Table 42. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)		
OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
S0_REUSEADDR	A 4-byte binary field.	A 4-byte binary field.
Use this option to control or determine whether local addresses are reused. The default is disabled. This alters the normal algorithm used with BIND. The normal BIND algorithm allows each Internet address and port combination to be bound only once. If the address and port have been already bound, then a subsequent BIND will fail and result error will be EADDRINUSE.	To enable, set to 1 or a positive value. To disable, set to 0.	If enabled, contains a 1. If disabled, contains a 0.
When this option is enabled, the following situations are supported:		
• A server can BIND the same port multiple times as long as every invocation uses a different local IP address and the wildcard address INADDR_ANY is used only one time per port.		
• A server with active client connections can be restarted and can bind to its port without having to close all of the client connections.		
• For datagram sockets, multicasting is supported so multiple bind() calls can be made to the same class D address and port number.		
• If you require multiple servers to BIND to the same port and listen on INADDR_ANY, see the SHAREPORT option on the PORT statement in TCPIP.PROFILE.		
S0_SNDBUF	A 4-byte binary field.	A 4-byte binary field.
Use this option to control or determine the size of the data portion of the TCP/IP send buffer. The size is of the TCP/IP send buffer is protocol specific and is based on the following values:	To enable, set to a positive value specifying the size of the data portion of the TCP/IP send buffer.	If enabled, contains a positive value indicating the size of the data portion of the TCP/IP send buffer.
 The TCPSENDBufrsize keyword on the TCPCONFIG statement in the PROFILE.TCPIP data set for a TCP socket 	To disable, set to a 0.	IT disabled, contains a 0.
 The UDPSENDBufrsize keyword on the UDPCONFIG statement in the PROFILE.TCPIP data set for a UDP socket 		
• The default of 65535 for a raw socket		

Table 42. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
S0_SNDTIME0	This option requires a TIMEVAL	This option stores a TIMEVAL
Use this option to control or determine the maximum length of time that a send- type function can remain blocked before it completes.	structure, which is defined in the SYS1.MACLIB(BPXYRLIM) macro. The TIMEVAL structure contains the number of seconds and microseconds specified as	structure that is defined in SYS1.MACLIB(BPXYRLIM). The TIMEVAL structure contains the number of seconds and microseconds, which are
If a send-type function has blocked for this length of time, it returns with a partial count or, if no data is sent, with an errno set to EWOULDBLOCK. The default value for this is 0, which indicates that a send- type function does not time out.	fullword binary numbers. The seconds value is in the range 0 - 2678400 (equal to 31 days), and the microseconds value is in the range 0 - 1000000 (equal to 1 second). Although the	specified as fullword binary numbers. The number of seconds value that is returned is in the range 0 - 2678400 (equal to 31 days). The microseconds value that is returned is in the
For a SETSOCKOPT, the following send- type functions are supported:	TIMEVAL value can be specified using microsecond granularity,	range 0 - 1000000.
• SEND	are used to implement this	
• SENDMSG	function have a granularity of	
• SENDTO	approximately 100 milliseconds.	
• WRITE		
• WRITEV		
S0_TYPE	N/A	A 4-byte binary field indicating the socket type:
ose this option to return the socket type.		X'1' indicates SOCK STREAM.
		– X'2' indicates SOCK_DGRAM.
		X'3' indicates SOCK_RAW
	A 4-byte binary field.	A 4-byte binary field.
Use this option to set or determine whether a socket-specific timeout value (in seconds) is to be used in place of	To enable, set to a value in the range of 1 – 2 147460.	If enabled, contains the specific timer value (in seconds) that is
a configuration-specific value whenever keep alive timing is active for that socket.	To disable, set to a value of 0.	If disabled, contains a 0
When activated, the socket-specified timer value remains in effect until respecified by SETSOCKOPT or until the socket is closed. See the <u>z/OS</u> <u>Communications Server: IP Programmer's</u> <u>Guide and Reference</u> for more information about the socket option parameters.		indicating keep alive timing is not active.

Table 42. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)		
OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
TCP_NODELAY	A 4-byte binary field.	A 4-byte binary field.
Use this option to set or determine	To enable, set to a 0.	If enabled, contains a 0.
subject to the Nagle algorithm (RFC 896).	To disable, set to a 1 or nonzero.	If disabled, contains a 1.
Under most circumstances, TCP sends data when it is presented. When this option is enabled, TCP will wait to send small amounts of data until the acknowledgment for the previous data sent is received. When this option is disabled, TCP will send small amounts of data even before the acknowledgment for the previous data sent is received.		
Note: Use the following to set TCP_NODELAY OPTNAME value for COBOL programs:		
 01 TCP-NODELAY-VAL PIC 9(10) COMP VALUE 2147483649. 01 TCP-NODELAY-REDEF REDEFINES TCP-NODELAY-VAL. 05 FILLER PIC 9(6) BINARY. 05 TCP-NODELAY PIC 9(8) BINARY. 		

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing SETSOCKOPT. The field is left-aligned and padded to the right with blanks.

S

A halfword binary number set to the socket whose options are to be set.

OPTNAME

Input parameter. See the table below for a list of the options and their unique requirements.

See the GETSOCKOPT command values information in <u>z/OS</u> Communications Server: IP Sockets Application Programming Interface Guide and Reference for the numeric values of **OPTNAME**.

Note: COBOL programs cannot contain field names with the underbar character. Fields representing the option name should contain dashes instead.

OPTVAL

Contains data which further defines the option specified in OPTNAME. For the SETSOCKOPT API, OPTVAL will be an input parameter. See the table below for a list of the options and their unique requirements.

OPTLEN

Input parameter. A fullword binary field containing the length of the data returned in OPTVAL. See the table below for determining on what to base the value of OPTLEN.

Parameter values returned to the application

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See <u>Appendix A</u>, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

```
Value
```

Description

0

Successful call.

-1

Check ERRNO for an error code.

SHUTDOWN

One way to terminate a network connection is to issue the CLOSE call which attempts to complete all outstanding data transmission requests prior to breaking the connection. The SHUTDOWN call can be used to close one-way traffic while completing data transfer in the other direction. The HOW parameter determines the direction of traffic to shutdown.

When the CLOSE call is used, the SETSOCKOPT OPTVAL LINGER parameter determines the amount of time the system will wait before releasing the connection. For example, with a LINGER value of 30 seconds, system resources (including the IMS or CICS transaction) will remain in the system for up to 30 seconds after the CLOSE call is issued. In high volume, transaction-based systems like CICS and IMS, this can impact performance severely.

If the SHUTDOWN call is issued when the CLOSE call is received, the connection can be closed immediately, rather than waiting for the 30-second delay.

If you issue SHUTDOWN for a socket that currently has outstanding socket calls pending, see the Effect of shutdown socket call table in the z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference to determine the effects of this operation on the outstanding socket calls.

Table 43. SHUTDOWN call requirements	
Condition	Requirement
Authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN.
Amode:	31-bit or 24-bit.
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.
ASC mode:	Primary address space control (ASC) mode.
Interrupt status:	Enabled for interrupts.
Locks:	Unlocked.
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Figure 55 on page 179 shows an example of SHUTDOWN call instructions.

```
WORKING-STORAGE SECTION.
    01 SOC-FUNCTION PIC X(16) VALUE IS 'SHUTDOWN'.
                           PIC 9(4) BINARY.
PIC 9(8) BINARY.
    01 S
    01 HOW
        88 END-FROM
88 END-TO
                             VALUE 0.
                             VALUE
                                    1.
        88 END-BOTH
                             VALUE 2.
                           PIC 9(8) BINARY.
PIC S9(8) BINARY.
    01 ERRNO
    01 RETCODE
PROCEDURE DIVISION.
     CALL 'EZASOKET' USING SOC-FUNCTION S HOW ERRNO RETCODE.
```

Figure 55. SHUTDOWN call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing SHUTDOWN. The field is left-aligned and padded on the right with blanks.

S

A halfword binary number set to the socket descriptor of the socket to be shutdown.

HOW

A fullword binary field. Set to specify whether all or part of a connection is to be shut down. The following values can be set:

Value

Description

0 (END-FROM)

Ends further receive operations.

1 (END-TO)

Ends further send operations.

2 (END-BOTH)

Ends further send and receive operations.

Parameter values returned to the application

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See <u>Appendix A</u>, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

0

Successful call.

-1

Check **ERRNO** for an error code.

SOCKET

The SOCKET call creates an endpoint for communication and returns a socket descriptor representing the endpoint.

Table 44. SOCKET call requirements	
Condition	Requirement
Authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN.
Amode:	31-bit or 24-bit.
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.
ASC mode:	Primary address space control (ASC) mode.
Interrupt status:	Enabled for interrupts.
Locks:	Unlocked.
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Figure 56 on page 180 shows an example of SOCKET call instructions.

WORKING-STORAGE SECTION 01 SOC-FUNCTION * AF INFT	PIC X(16) VALUE IS 'SOCKET'
01 AF	PIC 9(8) COMP VALUE 2.
* AF_INET6 01 AF 01 SOCTYPE 88 STREAM 88 DATAGRAM 88 RAW 01 PROTO 01 ERRNO 01 RETCODE	PIC 9(8) COMP VALUE 19. PIC 9(8) BINARY. VALUE 1. VALUE 2. VALUE 3. PIC 9(8) BINARY. PIC 9(8) BINARY. PIC S9(8) BINARY.
PROCEDURE DIVISION. CALL 'EZASOKET' US: PRO	ING SOC-FUNCTION AF SOCTYPE TO ERRNO RETCODE.

Figure 56. SOCKET call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing SOCKET. The field is left-aligned and padded on the right with blanks.

AF

A fullword binary field set to the addressing family. For TCP/IP the value is set to decimal 2 for AF_INET, or decimal 19, indicating AF_INET6.

SOCTYPE

A fullword binary field set to the type of socket required. The types are:

Value

Description

1

Stream sockets provide sequenced, two-way byte streams that are reliable and connectionoriented. They support a mechanism for out-of-band data.

2

Datagram sockets provide datagrams, which are connectionless messages of a fixed maximum length whose reliability is not guaranteed. Datagrams can be corrupted, received out of order, lost, or delivered multiple times.

3

Raw sockets provide the interface to internal protocols (such as IP and ICMP).

PROTO

A fullword binary field set to the protocol to be used for the socket. If this field is set to 0, the default protocol is used. For streams, the default is TCP; for datagrams, the default is UDP.

PROTO numbers are found in the *hlq*.etc.proto data set. For IPv6 raw sockets, PROTO cannot be set to the following values:

Protocol name Numeric value **IPROTO_HOPOPTS** 0 **IPPROTO_TCP** 6 **IPPROTO_UDP** 17 **IPPROTO_IPV6** 41 **IPPROTO_ROUTING** 43 **IPPROTO FRAGMENT** 44 **IPPROTO ESP** 50 **IPPROTO_AH** 51 **IPPROTO NONE** 59 **IPPROTO_DSTOPTS** 60

Parameter values returned to the application

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See <u>Appendix A</u>, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

> or = 0

Contains the new socket descriptor.

-1

Check **ERRNO** for an error code.

TAKESOCKET

The TAKESOCKET call acquires a socket from another program and creates a new socket. Typically, a child server issues this call using client ID and socket descriptor data that it obtained from the concurrent server. See <u>"GIVESOCKET" on page 111</u> for a discussion of the use of GETSOCKET and TAKESOCKET calls.

Note: When TAKESOCKET is issued, a new socket descriptor is returned in RETCODE. You should use this new socket descriptor in subsequent calls such as GETSOCKOPT, which require the S (socket descriptor) parameter.

Condition Requirement Authorization: Supervisor state or problem state, any PSW key. Task. Dispatchable unit mode: PASN = HASN. Cross memory mode: 31-bit or 24-bit. Amode: **Note:** See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51. ASC mode: Primary address space control (ASC) mode. Interrupt status: Enabled for interrupts. Locks: Unlocked. Control parameters: All parameters must be addressable by the caller and in the primary address space.

Table 45. TAKESOCKET call requirements

Figure 57 on page 182 shows an example of TAKESOCKET call instructions.

```
WORKING-STORAGE SECTION.
    01 SOC-FUNCTION
01 SOCRECV
                        PIC X(16) VALUE IS 'TAKESOCKET'.
                        PIC 9(4) BINARY.
    01 CLIENT.
        03 DOMAIN
                        PIC 9(8) BINARY.
        03 NAME
                        PIC X(8).
       03 TASK
03 RESERVED
                        PIC X(8).
                        PIC X(20)
    01 ERRNO
                        PIC 9(8) BINARY.
    01 RETCODE
                        PIC S9(8) BINARY.
PROCEDURE DIVISION.
     CALL 'EZASOKET' USING SOC-FUNCTION SOCRECV CLIENT
                     ERRNO RETCODE.
```

Figure 57. TAKESOCKET call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing TAKESOCKET. The field is left-aligned and padded to the right with blanks.

SOCRECV

A halfword binary field set to the descriptor of the socket to be taken. The socket to be taken is passed by the concurrent server.

CLIENT

Specifies the client ID of the program that is giving the socket. In CICS and IMS, these parameters are passed by the Listener program to the program that issues the TAKESOCKET call.

- In CICS, the information is obtained using EXEC CICS RETRIEVE.
- In IMS, the information is obtained by issuing GU TIM.

DOMAIN

A fullword binary field set to the domain of the program giving the socket. It is decimal 2, indicating AF_INET, or decimal 19, indicating AF_INET6.

Note: The TAKESOCKET can acquire only a socket of the same address family from a GIVESOCKET.

NAME

Specifies an 8-byte character field set to the MVS address space identifier of the program that gave the socket.

TASK

Specifies an 8-byte field set to the task identifier of the task that gave the socket.

RESERVED

A 20-byte reserved field. This field is required, but not used.

Parameter values returned to the application

ERRNO

A fullword binary field. If the value of RETCODE is negative, the field contains an error number. See Appendix A, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

≥ 0

Contains the new socket descriptor.

-1

Check ERRNO for an error code.

TERMAPI

This call terminates the session created by INITAPI.

Table 46. TERMAPI call requirements	
Condition	Requirement
Authorization:	Supervisor state or problem state, any PSW key.
Dispatchable unit mode:	Task.
Cross memory mode:	PASN = HASN.
Amode:	31-bit or 24-bit.
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.
ASC mode:	Primary address space control (ASC) mode.
Interrupt status:	Enabled for interrupts.

Table 46. TERMAPI call requirements (continued)

Condition	Requirement
Locks:	Unlocked.
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Figure 58 on page 184 shows an example of TERMAPI call instructions.

```
WORKING-STORAGE SECTION.
                       PIC X(16) VALUE IS 'TERMAPI'.
    01 SOC-FUNCTION
PROCEDURE DIVISION.
     CALL 'EZASOKET' USING SOC-FUNCTION.
```

Figure 58. TERMAPI call instruction example

For equivalent PL/I and assembly language declarations, see "Converting parameter descriptions" on page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing TERMAPI. The field is left-aligned and padded to the right with blanks.

WRITE

The WRITE call writes data on a connected socket. This call is similar to SEND, except that it lacks the control flags available with SEND.

For datagram sockets the WRITE call writes the entire datagram if it fits into the receiving buffer.

Stream sockets act like streams of information with no boundaries separating data. For example, if a program wants to send 1000 bytes, each call to this function can send any number of bytes, up to the entire 1000 bytes. The number of bytes sent will be returned in RETCODE. Therefore, programs using stream sockets should place this call in a loop, calling this function until all data has been sent.

See "EZACICO4" on page 188 for a subroutine that will translate EBCDIC output data to ASCII.

Table 47. WRITE call requirem	ents						
Condition	Requirement						
Authorization:	Supervisor state or problem state, any PSW key.						
Dispatchable unit mode:	Task.						
Cross memory mode:	PASN = HASN.						
Amode:	31-bit or 24-bit.						
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.						
ASC mode:	Primary address space control (ASC) mode.						
Interrupt status: Enabled for interrupts.							
Locks:	Unlocked.						
Control parameters:	All parameters must be addressable by the caller and in the primary address space.						

T . I . I .

Figure 59 on page 185 shows an example of WRITE call instructions.

```
WORKING-STORAGE SECTION.

01 SOC-FUNCTION PIC X(16) VALUE IS 'WRITE'.

01 S PIC 9(4) BINARY.

01 NBYTE PIC 9(8) BINARY.

01 BUF PIC X(length of buffer).

01 ERRNO PIC 9(8) BINARY.

01 RETCODE PIC S9(8) BINARY.

PROCEDURE DIVISION.

CALL 'EZASOKET' USING SOC-FUNCTION S NBYTE BUF

ERRNO RETCODE.
```

Figure 59. WRITE call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing WRITE. The field is left-aligned and padded on the right with blanks.

S

A halfword binary field set to the socket descriptor.

NBYTE

A fullword binary field set to the number of bytes of data to be transmitted.

BUF

Specifies the buffer containing the data to be transmitted.

Parameter values returned to the application

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See <u>Appendix A</u>, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following values:

Value

Description

≥0

A successful call. A return code greater than 0 indicates the number of bytes of data written.

-1

Check ERRNO for an error code.

WRITEV

The WRITEV function writes data on a socket from a set of buffers.

Table 48. WRITEV call requirements							
Condition	Requirement						
Authorization:	Supervisor state or problem state, any PSW key.						
Dispatchable unit mode:	Task.						
Cross memory mode:	PASN = HASN.						

Table 48. WRITEV call requirements (continued)					
Condition	Requirement				
Amode:	31-bit or 24-bit.				
	Note: See the addressability mode (Amode) considerations under "CALL instruction API environmental restrictions and programming requirements" on page 51.				
ASC mode:	Primary address space control (ASC) mode.				
Interrupt status:	Enabled for interrupts.				
Locks:	Unlocked.				
Control parameters:	All parameters must be addressable by the caller and in the primary address space.				

Figure 60 on page 186 shows an example of WRITEV call instructions.

```
WORKING-STORAGE SECTION.
01 SOC-FUNCTION
                           PIC X(16) VALUE 'WRITEV'.
                              PIC 9(4) BINARY.
PIC 9(8) BINARY.
01 S
01 IOVCNT
01 IOV.
    03 BUFFER-ENTRY OCCURS N TIMES.
      05 BUFFER-POINTER USAGE IS POINTER.
                             PIC X(4).
PIC 9(8) USAGE IS BINARY.
      05 RESERVED
      05 BUFFER-LENGTH
01 ERRNO
                              PIC 9(8) BINARY.
01 RETCODE
                              PIC 9(8) BINARY.
PROCEDURE DIVISION.
    SET BUFFER-POINTER(1) TO ADDRESS OF BUFFER1.
    SET BUFFER-LENGTH(1) TO LENGTH OF BUFFER1.
    SET BUFFER-POINTER(2) TO ADDRESS OF BUFFER2.
SET BUFFER-LENGTH(2) TO LENGTH OF BUFFER2.
    SET BUFFER-POINTER(n) TO ADDRESS OF BUFFERn.
    SET BUFFER-LENGTH(n) TO LENGTH OF BUFFERn.
    CALL 'EZASOKET' USING SOC-FUNCTION S IOV IOVCNT ERRNO RETCODE.
```

Figure 60. WRITEV call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

S

A value or the address of a halfword binary number specifying the descriptor of the socket from which the data is to be written.

ΙΟΥ

An array of tripleword structures with the number of structures equal to the value in IOVCNT and the format of the structures as follows:

Fullword 1

The address of a data buffer.

Fullword 2

Reserved.

Fullword 3

The length of the data buffer referenced in Fullword 1.

IOVCNT

A fullword binary field specifying the number of data buffers provided for this call.

Parameters returned by the application

ERRNO

A fullword binary field. If RETCODE is negative, this contains an error number. See <u>Appendix A</u>, "Return codes," on page 267 for information about ERRNO return codes.

RETCODE

A fullword binary field.

```
Value
```

Meaning

<0

Check **ERRNO** for an error code.

0

Connection partner has closed connection.

>0

Number of bytes sent.

Using data translation programs for socket call interface

In addition to the socket calls, you can use utility programs to translate data.

Assembly language utility programs call format

The following example shows the assembly language call format for utility programs:

```
>>__CALL EZACIC04,(Inbuf, Inbuf_Length),VL__><</pre>
```

Data translation

TCP/IP hosts and networks use ASCII data notation; MVS TCP/IP and its subsystems use EBCDIC data notation. In situations where data must be translated from one notation to the other, you can use the following utility programs:

- EZACIC04 translates EBCDIC data to ASCII data using the translation table documented in the <u>z/OS</u> Communications Server: IP Configuration Reference.
- EZACIC05 translates ASCII data to EBCDIC data using the translation table documented in the <u>z/OS</u> Communications Server: IP Configuration Reference.
- EZACIC14 provides an alternative to EZACIC04 and translates EBCDIC data to ASCII data using the translation table documented in Figure 69 on page 197.
- EZACIC15 provides an alternative to EZACIC05 and translates ASCII data to EBCDIC data using the translation table documented in Figure 71 on page 199.

Bit-string processing

In C-language, bit strings are often used to convey flags, switch settings, and so on; TCP/IP makes frequent uses of bit strings. However, because bit strings are difficult to decode in COBOL, TCP/IP includes the following information:

- EZACIC06 translates bit-masks into character arrays and character arrays into bit-masks.
- EZACIC08 interprets the variable length address list in the HOSTENT structure returned by GETHOSTBYNAME or GETHOSTBYADDR.

• EZACIC09 interprets the ADDRINFO structure returned by GETADDRINFO.

EZACIC04

The EZACIC04 program is used to translate EBCDIC data to ASCII data. <u>Figure 61 on page 188</u> shows how EZACIC04 translates a byte of EBCDIC data.

ASCII		 	se	ecor	nd l	nex	dig	git	of	byt	te d	of E	BCI	DIC	dat	a	
EZACICO	EZACIC04		1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	0	00	01	02	03	1A	09	1A	7F	1A	1A	1A	0B	00	0D	0E	0F
	1	10	11	12	13	1A	0A	08	1A	18	19	1A	1A	10	1D	1E	1F
	2	1A	1A	10	1A	1A	0A	17	1B	1A	1A	1A	1A	1A	05	06	07
	3	1A	1A	16	1A	1A	1E	1A	04	1A	1A	1A	1A	14	15	1A	1A
	4	20	A6	E1	80	EB	90	9F	E2	AB	8B	9B	2E	30	28	2B	7C
	5	26	A9	AA	90	DB	A5	99	E3	A8	9E	21	24	2A	29	3B	5E
	6	2D	2F	DF	DC	9A	DD	DE	98	9D	AC	BA	2C	25	5F	3E	3F
	7	D7	88	94	BO	B1	B2	FC	D6	FB	60	3A	23	40	27	3D	22
	8	F8	61	62	63	64	65	66	67	68	69	96	A4	F3	AF	AE	C5
	9	80	6A	6B	6C	6D	6E	6F	70	71	72	97	87	CE	93	F1	FE
	A	C8	7E	73	74	75	76	77	78	79	7A	EF	CO	DA	5B	F2	AE
	В	B5	B6	FD	B7	B8	B9	E6	BB	BC	BD	8D	D9	BF	5D	D8	C4
	C	7B	41	42	43	44	45	46	47	48	49	СВ	CA	BE	E8	EC	ED
	D	7D	4A	4B	4C	4D	4E	4F	50	51	52	A1	AD	F5	F4	A3	8F
	E	5C	E7	53	54	55	56	57	58	59	5A	AO	85	8E	E9	E4	D1
	F	30	31	32	33	34	35	36	37	38	39	В3	F7	F0	FA	A7	FF

Figure 61. EZACIC04 EBCDIC-to-ASCII table

Figure 62 on page 188 shows an example of EZACIC04 call instructions.

```
WORKING-STORAGE SECTION.

01 OUT-BUFFER PIC X(length of output).

01 LENGTH PIC 9(8) BINARY.

PROCEDURE DIVISION.

CALL 'EZACIC04' USING OUT-BUFFER LENGTH.

IF RETURN-CODE > 0

THEN

DISPLAY 'TRANSLATION FAILED ' RETURN-CODE.
```

Figure 62. EZACIC04 call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

OUT-BUFFER

A buffer that contains the following information:

- When called, EBCDIC data
- Upon return, ASCII data

LENGTH

Specifies the length of the data to be translated.
RETURN-CODE

Upon return, register 15 contains a return code value, which indicates if the data translation occurred successfully. The return code can be one of the following values:

0

The data translation occurred.

8

Too many parameters passed, translation did not occur.

12

Zero buffer length passed, translation did not occur.

16

Zero buffer address passed, translation did not occur.

EZACIC05

The EZACIC05 program is used to translate ASCII data to EBCDIC data. EBCDIC data is required by COBOL, PL/I, and assembly language programs. Figure 63 on page 189 shows how EZACIC05 translates a byte of ASCII data.

EBCDIC	 1V	 9 	eco	nd l	nex	dig	git	of	byt	te d	of /	ASCI	EI (data	 a 	· ·
EZACICO	5	0 1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	0	000101	. 02	03	37	2D	2E	2F	16	05	25	0B	00	0D	0E	0F
	1	10 11	12	13	3C	3D	32	26	18	19	3F	27	22	1D	35	1F
	2	40 54	17F	+ 7В	5B	60	50	7D	4D	5D	50	4E	6B	60	4B	61
	3	F0 F1	+	+	+ F4	F5	F6	F	F8	F9	7A	5E	4C	+ 7E	6E	6F
	4	++- 7C C1	+	+ C3	+ C4	 C5	+ C6	 C7	C8	 C9	 D1	 D2	D3	+ D4	++ D5	D6
	5	++- D7 D8	+ 8 D9	+ E2	+ E3	E4	+ E5	E6	E7	E8	E9	AD	E0	+ BD	++ 5F	6D
first	6	++- 79 81	82	+ 83	84	85	86	87	88	89	91	92	93	+ 94	95	96
nex digit		++- 97 98	199	+ A2	A3	A4	A5	A6	A7	A8	A9	C0	4F	+	A1	07
byte	8	00 01	02	+ 03	37	2D	2E	2F	16	05	25	0B	00	+	0E	0F
ASCII	9	++- 10 11	12	+ 13	3C	3D	32	26	18	19	3F	27	22	+ 1D	35	1F
αατα 		++- 40 54	+	+ 7B	5B	6C	50	7D	4D	5D	5C	4E	6B	60	AF	61
	 B	++- F0 F1	+	+ F3	F4	F5	F6	F7	F8	F9	7A	5E	4C	+ 7E	6E	6F
	C	7C C1	+	+ C3	C4	C5	C6	C7	C8	C9	D1	D2	D3	+ D4	D5	D6
	D	++- D7 D8	+	E2	E3	E4	E5	E6	E7	E8	E9	AD	E0	+ BD	+ 5F	6D
	E	++- 79 81	82	83	84	85	86	87	88	89	91	92	93	94	95	96
	F	++- 97 98	99	A2	A3	A4	A5	A6	Α7	A8	A9	C0	4F	D0	A1	07

Figure 63. EZACIC05 ASCII-to-EBCDIC table

Figure 64 on page 189 shows an example of EZACIC05 call instructions.

```
WORKING-STORAGE SECTION.

01 IN-BUFFER PIC X(length of output)

01 LENGTH PIC 9(8) BINARY VALUE

PROCEDURE DIVISION.

CALL 'EZACICO5' USING IN-BUFFER LENGTH. IF RETURN-CODE > 0

THEN

DISPLAY 'TRANSLATION FAILED ' RETURN-CODE.
```

Figure 64. EZACIC05 call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

IN-BUFFER

A buffer that contains the following information:

- When called, ASCII data
- Upon return, EBCDIC data

LENGTH

Specifies the length of the data to be translated.

RETURN-CODE

Upon return, register 15 contains a return code value, which indicates if the data translation occurred successfully. The return code can be one of the following values:

0

The data translation occurred.

8

Too many parameters passed, translation did not occur.

12

Zero buffer length passed, translation did not occur.

16

Zero buffer address passed, translation did not occur.

EZACIC06

The SELECT and SELECTEX call uses bit strings to specify the sockets to test and to return the results of the test. Because bit strings are difficult to manage in COBOL, you might want to use the EZACICO6 utility program to translate them to character strings to be used with the SELECT or SELECTEX call.

Figure 65 on page 190 shows an example of EZACIC06 call instructions.

```
WORKING-STORAGE SECTION.
   01 CHAR-MASK.
       05 CHAR-STRING
                                     PIC X(nn).
   01 CHAR-ARRAY
                                    REDEFINES CHAR-MASK.
       05 CHAR-ENTRY-TABLE
                                     OCCURS nn TIMES.
            10 CHAR-ENTRY
                                    PIC X(1).
   01 BIT-MASK.
        05 BIT-ARRAY-FWDS
                                     OCCURS (nn+31)/32 TIMES.
          10 BIT_ARRAY_WORD
                                    PIC 9 (8) COMP.
   01 BIT-FUNCTION-CODES.
                                    PIC X(4) VALUE 'CTOB'.
PIC X(4) VALUE 'BTOC'.
       05 CTOB
05 BTOC
   01 CHAR-MASK-LENGTH
                                     PIC 9(8) COMP VALUE nn.
   PROCEDURE CALL (to convert from character to binary)
     CALL 'EZACICO6' USING CTOB
                           BTT-MASK
                           CHAR-MASK
                           CHAR-MASK-LENGTH
                           RETCODE.
   PROCEDURE CALL (to convert from binary to character)
     CALL 'EZACICO6' USING BTOC
                           BIT-MASK
                           CHAR-MASK
                           CHAR-MASK-LENGTH
                           RETCODE.
```

Figure 65. EZACIC06 call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

CHAR-MASK

Specifies the character array where *nn* is the maximum number of sockets in the array. The first character in the array represents socket 0, the second represents socket 1, and so on. Note that the index is 1 greater than the socket number [for example, CHAR-ENTRY(1) represents socket 0, CHAR-ENTRY (2) represents socket 1, and so on.]

BIT-MASK

Specifies the bit string to be translated for the SELECT call. Within each fullword of the bit string, the bits are ordered right to left. The rightmost bit in the first fullword represents socket 0 and the leftmost bit represents socket 31. The rightmost bit in the second fullword represents socket 32 and the leftmost bit represents socket 63. The number of fullwords in the bit string should be calculated by dividing the sum of 31 and the character array length by 32 (truncate the remainder).

COMMAND

BTOC specifies bit string to character array translation.

CTOB specifies character array to bit string translation.

CHAR-MASK-LENGTH

Specifies the length of the character array. This field should be no greater than 1 plus the MAXSNO value returned on the INITAPI (which is usually the same as the MAXSOC value specified on the INITAPI).

RETCODE

A binary field that returns one of the following values:

Value

Description

0

Successful call.

-1

Check ERRNO for an error code.

Examples

```
MOVE ZEROS TO CHAR-STRING.
MOVE '1' TO CHAR-ENTRY(1), CHAR-ENTRY(6), CHAR-ENTRY(33).
CALL 'EZACIC06' USING TOKEN CTOB BIT-MASK CH-MASK
CHAR-MASK-LENGTH RETCODE.
MOVE BIT-MASK TO ....
```

When the select call returns and you want to check the bit-mask string for socket activity, enter the following instructions.

MOVE TO BIT-MASK. CALL 'EZACIC06' USING TOKEN BTOC BIT-MASK CH-MASK CHAR-MASK-LENGTH RETCODE. PERFORM TEST-SOCKET THRU TEST-SOCKET-EXIT VARYING IDX FROM 1 BY 1 UNTIL IDX EQUAL CHAR-MASK-LENGTH. TEST-SOCKET. IF CHAR-ENTRY(IDX) EQUAL '1' THEN PERFORM SOCKET-RESPONSE THRU SOCKET-RESPONSE-EXIT ELSE NEXT SENTENCE.

EZACIC08

The GETHOSTBYNAME and GETHOSTBYADDR calls were derived from C socket calls that return a structure known as HOSTENT. A given TCP/IP host can have multiple alias names and host IP addresses.

TCP/IP uses indirect addressing to connect the variable number of alias names and IP addresses in the HOSTENT structure that are returned by the GETHOSTBYADDR and GETHOSTBYNAME calls.

If you are coding in PL/I or assembly language, the HOSTENT structure can be processed in a relatively straight-forward manner. However, if you are coding in COBOL, HOSTENT can be more difficult to process and you should use the EZACIC08 subroutine to process it for you.

It works as follows:

- 1. GETHOSTBYADDR or GETHOSTBYNAME returns a HOSTENT structure that indirectly addresses the lists of alias names and IP addresses.
- 2. Upon return from GETHOSTBYADDR or GETHOSTBYNAME, your program calls EZACIC08 and passes it the address of the HOSTENT structure. EZACIC08 processes the structure and returns the following information:
 - The length of host name, if present
 - The host name
 - The number of alias names for the host
 - The alias name sequence number
 - The length of the alias name
 - The alias name
 - The host IP address type, always 2 for AF_INET
 - The host IP address length, always 4 for AF_INET
 - The number of host IP addresses for this host
 - The host IP address sequence number
 - The host IP address
- If the GETHOSTBYADDR or GETHOSTBYNAME call returns more than one alias name or host IP address, the application program should repeat the call to EZACIC08 until all alias names and host IP addresses have been retrieved.

Figure 66 on page 193 shows an example of EZACIC08 call instructions.

WORKING-STORAGE SECTION.

01 01 01 01 01 01 01 01 01 01 01	HOSTENT-ADI HOSTNAME-LI HOSTNAME-V/ HOSTALIAS-(HOSTALIAS-) HOSTALIAS-) HOSTALIAS-) HOSTADDR-T HOSTADDR-LI HOSTADDR-C(HOSTADDR-SI HOSTADDR-V/ RETURN-CODI	DR F ENGTH F COUNT F LENGTH F VALUE F YPE F ENGTH F DUNT F EQ F ALUE F	PIC 9(8) PIC 9(4) PIC X(255 PIC 9(4) PIC 9(4) PIC 9(4) PIC 9(4) PIC 9(4) PIC 9(4) PIC 9(4) PIC 9(4) PIC 9(8) PIC 9(8) PIC 9(8)	BINARY. BINARY. BINARY. BINARY. BINARY. BINARY. BINARY. BINARY. BINARY. BINARY. BINARY. BINARY.		
PROCEDU	RE DIVISION	•				
CALL	'EZASOKET'	USING 'GH HOSTADDR RETCODE.	ETHOSTBY/ HOSTENT	ADDR ' ADDR		
CALL	'EZASOKET'	USING 'GE NAMELEN F RETCODE.	ETHOSTBYN NAME HOS ⁻	IAME ' TENT - ADDF	2	
CALL	'EZACIC08'	USING HOS HOSTNAME HOSTALIAS HOSTADDR HOSTADDR	STENT-ADI -VALUE HO S-LENGTH -TYPE HOS -SEQ HOS ⁻	DR HOSTNA DSTALIAS HOSTALIA STADDR-LE TADDR-VAL	ME-LENGTH •COUNT HOSTALIAS-SEQ AS-VALUE ENGTH HOSTADDR-COUNT LUE RETURN-CODE.	

Figure 66. EZAZIC08 call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application

HOSTENT-ADDR

This fullword binary field must contain the address of the HOSTENT structure (as returned by the GETHOSTBY*xxxx* call). This variable is the same as the variable HOSTENT in the GETHOSTBYADDR and GETHOSTBYNAME socket calls.

HOSTALIAS-SEQ

This halfword field is used by EZACIC08 to index the list of alias names. When EZACIC08 is called, it adds 1 to the current value of HOSTALIAS-SEQ and uses the resulting value to index into the table of alias names. Therefore, for a given instance of GETHOSTBYxxxx, this field should be set to 0 for the initial call to EZACIC08. For all subsequent calls to EZACIC08, this field should contain the HOSTALIAS-SEQ number returned by the previous invocation.

HOSTADDR-SEQ

This halfword field is used by EZACIC08 to index the list of IP addresses. When EZACIC08 is called, it adds 1 to the current value of HOSTADDR-SEQ and uses the resulting value to index into the table of IP addresses. Therefore, for a given instance of GETHOSTBYxxxx, this field should be set to 0 for the initial call to EZACIC08. For all subsequent calls to EZACIC08, this field should contain the HOSTADDR-SEQ number returned by the previous call.

Parameter values returned to the application

HOSTNAME-LENGTH

This halfword binary field contains the length of the host name (if host name was returned).

HOSTNAME-VALUE

This 255-byte character string contains the host name (if host name was returned).

HOSTALIAS-COUNT

This halfword binary field contains the number of alias names returned.

HOSTALIAS-SEQ

This halfword binary field is the sequence number of the alias name currently found in HOSTALIAS-VALUE.

HOSTALIAS-LENGTH

This halfword binary field contains the length of the alias name currently found in HOSTALIAS-VALUE.

HOSTALIAS-VALUE

This 255-byte character string contains the alias name returned by this instance of the call. The length of the alias name is contained in HOSTALIAS-LENGTH.

HOSTADDR-TYPE

This halfword binary field contains the type of host address. For FAMILY type AF_INET, HOSTADDR-TYPE is always 2.

HOSTADDR-LENGTH

This halfword binary field contains the length of the host IP address currently found in HOSTADDR-VALUE. For FAMILY type AF_INET, HOSTADDR-LENGTH is always set to 4.

HOSTADDR-COUNT

This halfword binary field contains the number of host IP addresses returned by this instance of the call.

HOSTADDR-SEQ

This halfword binary field contains the sequence number of the host IP address currently found in HOSTADDR-VALUE.

HOSTADDR-VALUE

This fullword binary field contains a host IP address.

RETURN-CODE

This fullword binary field contains the EZACIC08 return code:

Value

Description

0

Successful completion.

-1

HOSTENT address is not valid.

A value of HOSTALIAS-SEQ is not valid.

-3

A value of HOSTADDR-SEQ is not valid.

EZACIC09

The GETADDRINFO call was derived from the C socket call that return a structure known as RES. A given TCP/IP host can have multiple sets of NAMES. TCP/IP uses indirect addressing to connect the variable number of NAMES in the RES structure that is returned by the GETADDRINFO call. If you are coding in PL/I or assembly language, the RES structure can be processed in a relatively straight-forward manner. However, if you are coding in COBOL, RES can be more difficult to process and you should use the EZACIC09 subroutine to process it for you. It works as follows:

1. GETADDRINFO returns a RES structure that indirectly addresses the lists of socket address structures.

- 2. Upon return from GETADDRINFO, your program calls EZACICO9 and passes it the address of the next address information structure as referenced by the NEXT argument. EZACICO9 processes the structure and returns the following information: a. The socket address structure b. The next address information structure.
- 3. If the GETADDRINFO call returns more than one socket address structure the application program should repeat the call to EZACIC09 until all socket address structures have been retrieved.

Figure 67 on page 195 shows an example of EZACIC09 call instructions.

```
WORKING-STORAGE SECTION.
          * Variables used for the GETADDRINFO call
           01 getaddrinfo-parms.
                02 node-name pic x(255).
02 node-name-len pic 9(8) binary.
                02 service-name pic x(32)
                02 service-name-len pic 9(8) binary.
                02 canonical-name-len pic 9(8) binary.
02 ai-passive pic 9(8) binary value 1.
                02 ai-canonnameok pic 9(8) binary value 2.
02 ai-numerichost pic 9(8) binary value 4.
                02 ai-numericisery pic 9(8) binary value 4.
02 ai-v4mapped pic 9(8) binary value 16.
02 ai-all pic 9(8) binary value 32.
02 ai-addrconfig pic 9(8) binary value 64.
          * Variables used for the EZACIC09 call
          *
           01 ezacic09-parms.
                02 res usage is pointer.
                02 res-name-len pic 9(8) binary.
02 res-canonical-name pic x(256).
                02 res-name usage is pointer.
                02 res-next-addrinfo usage is pointer.
          * Socket address structure
           01 server-socket-address.
05 server-family pic 9(4) Binary Value 19.
05 server-port pic 9(4) Binary Value 9997.
05 server-flowinfo pic 9(8) Binary Value 0.
05 server-ipaddr.
                10 filler pic 9(16) binary value 0.
10 filler pic 9(16) binary value 0.
05 server-scopeid pic 9(8) Binary Value 0.
        LINKAGE SECTION.
           01 L1.
                03 HINTS-ADDRINFO.
                     05 HINTS-AI-FLAGS PIC 9(8) BINARY.
05 HINTS-AI-FAMILY PIC 9(8) BINARY.
05 HINTS-AI-SOCKTYPE PIC 9(8) BINARY.
                     05 HINTS-AI-PROTOCOL PIC 9(8) BINARY.
                     05 FILLER PIC 9(8) BINARY.
05 FILLER PIC 9(8) BINARY.
                     05 FILLER PIC 9(8) BINARY.
05 FILLER PIC 9(8) BINARY.
                03 HINTS-ADDRINFO-PTR USAGE IS POINTER.
                03 RES-ADDRINFO-PTR USAGE IS POINTER.
            RESULTS ADDRESS INFO
          *
           01 RESULTS-ADDRINFO.
                05 RESULTS-AI-FLAGS PIC 9(8) BINARY.
05 RESULTS-AI-FAMILY PIC 9(8) BINARY
                05 RESULTS-AI-SOCKTYPE PIC 9(8) BINARY.
                05 RESULTS-AI-PROTOCOL PIC 9(8) BINARY.
05 RESULTS-AI-ADDR-LEN PIC 9(8) BINARY.
```



05 RESULTS-AI-CANONICAL-NAME USAGE IS POINTER. 05 RESULTS-AI-ADDR-PTR USAGE IS POINTER. 05 RESULTS-AI-NEXT-PTR USAGE IS POINTER.

```
* SOCKET ADDRESS STRUCTURE FROM EZACIC09.
*
01 OUTPUT-NAME-PTR USAGE IS POINTER.
01 OUTPUT-IP-NAME
    03 OUTPUT-IP-FAMILY PIC 9(4) BINARY.
   03 OUTPUT-IP-PORT PIC 9(4) BINARY.
   03 OUTPUT-IP-SOCK-DATA PIC X(24)
    03 OUTPUT-IPV4-SOCK-DATA REDEFINES OUTPUT-IP-SOCK-DATA.
       05 OUTPUT-IPV4-IPADDR PIC 9(8) BINARY.
       05 FILLER PIC X(20)
    03 OUTPUT-IPV6-SOCK-DATA REDEFINES OUTPUT-IP-SOCK-DATA.
       05 OUTPUT-IPV6-FLOWINFO PIC 9(8) BINARY.
       05 OUTPUT-IPV6-IPADDR.
          10 FILLER PIC 9(16) BINARY.
10 FILLER PIC 9(16) BINARY.
       05 OUTPUT-IPV6-SCOPEID PIC 9(8) BINARY.
PROCEDURE DIVISION USING L1.
 Get and address from the resolver.
*
*
     move 'yournodename' to node-name.
     move 12 to node-name-len.
     move spaces to service-name.
     move 0 to service-name-len.
     move af-inet6 to hints-ai-family.
    move 49 to hints-ai-flags
    move 0 to hints-ai-socktype.
move 0 to hints-ai-protocol.
     set address of results-addrinfo to res-addrinfo-ptr.
     set hints-addrinfo-ptr to address of hints-addrinfo.
     call 'EZASOKET' using soket-getaddrinfo
                            node-name node-name-len
                            service-name service-name-len
                            hints-addrinfo-ptr
                            res-addrinfo-ptr
                            canonical-name-len
                            errno retcode.
* Use EZACIC09 to extract the IP address
     set address of results-addrinfo to res-addrinfo-ptr.
     set res to address of results-addrinfo.
    move zeros to res-name-len.
    move spaces to res-canonical-name.
     set res-name to nulls.
     set res-next-addrinfo to nulls.
     call 'EZACIC09' using res
                            res-name-len
                            res-canonical-name
                            res-name
                            res-next-addrinfo
                            retcode.
     set address of output-ip-name to res-name.
     move output-ipv6-ipaddr to server-ipaddr.
```

Figure 68. EZACIC09 call instruction example (Part 2 of 2)

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

Parameter values set by the application:

RES

This fullword binary field must contain the address of the ADDRINFO structure (as returned by the GETADDRINFO call). This variable is the same as the RES variable in the GETADDRINFO socket call.

RES-NAME-LEN

A fullword binary field that will contain the length of the socket address structure as returned by the GETADDRINFO call.

Parameter values returned to the application: Description

RES-CANONICAL-NAME

A field large enough to hold the canonical name. The maximum field size is 256 bytes. The canonical name length field will indicate the length of the canonical name as returned by the GETADDRINFO call.

RES-NAME

The address of the subsequent socket address structure.

RES-NEXT

The address of the next address information structure.

RETURN-CODE

CODE This fullword binary field contains the EZACIC09 return code:

Value

Description

0

Successful call.

-1

Invalid RES address.

EZACIC14

The EZACIC14 program is an alternative to EZACIC04, which translates EBCDIC data to ASCII data. Figure 69 on page 197 shows how EZACIC14 translates a byte of EBCDIC data.

ASCII	 v	second hex digit of byte of EBCDIC data
EZACIC14	1	0 1 2 3 4 5 6 7 8 9 A B C D E F
	0	0000102039C09867F978D8E0B0C0000E0F
	1	10 11 12 13 9D 85 08 87 18 19 92 8F 1C 1D 1E 1F
	2	80 81 82 83 84 0A 17 1B 88 89 8A 8B 8C 05 06 07
	3	9091169394959604989998914159E1A
	4	20 A0 E2 E4 E0 E1 E3 E5 E7 F1 A2 2E 3C 28 2B 7C
	5	26 E9 EA EB E8 ED EE EF EC DF 21 24 2A 29 3B 5E
first	6	2D 2F C2 C4 C0 C1 C3 C5 C7 D1 A6 2C 25 5F 3E 3F
nex digit	7	F8 C9 CA CB C8 CD CE CF CC 60 3A 23 40 27 3D 22
byte	8	D8 61 62 63 64 65 66 67 68 69 AB BB F0 FD FE B1
OI EBCDIC	9	B0 6A 6B 6C 6D 6E 6F 70 71 72 AA BA E6 B8 C6 A4
data	A	B5 7E 73 74 75 76 77 78 79 7A A1 BF D0 5B DE AE
	В	AC A3 A5 B7 A9 A7 B6 BC BD BE DD A8 AF 5D B4 D7
	C	7B 41 42 43 44 45 46 47 48 49 AD F4 F6 F2 F3 F5
	D	7D 4A 4B 4C 4D 4E 4F 50 51 52 B9 FB FC F9 FA FF
	Е	5C F7 53 54 55 56 57 58 59 5A B2 D4 D6 D2 D3 D5
	F	+-++++++++++++++++++++++++++++++++++++



Figure 70 on page 198 shows an example of EZACIC14 call instructions.

```
WORKING-STORAGE SECTION.

01 OUT-BUFFER PIC X(length of output).

01 LENGTH PIC 9(8) BINARY.

PROCEDURE DIVISION.

CALL 'EZACIC14' USING OUT-BUFFER LENGTH. IF RETURN-CODE > 0

THEN

DISPLAY 'TRANSLATION FAILED ' RETURN-CODE.
```

Figure 70. EZACIC14 call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

OUT-BUFFER

A buffer that contains the following information:

- When called, EBCDIC data
- Upon return, ASCII data

LENGTH

Specifies the length of the data to be translated.

RETURN-CODE

Upon return, register 15 contains a return code value, which indicates if the data translation occurred successfully. The return code can be one of the following values:

0

The data translation occurred.

8

Too many parameters passed, translation did not occur.

12

Zero buffer length passed, translation did not occur.

16

Zero buffer address passed, translation did not occur.

EZACIC15

The EZACIC15 program is an alternative to EZACIC05, which translates ASCII data to EBCDIC data. Figure 71 on page 199 shows how EZACIC15 translates a byte of ASCII data.

EBCDIC		 	se	ecor	nd ł	nex	di	git	of	byt	e d	of A	\SC]	LI (lata	a	
EZACIC1	5 5	0	1	2	3	4	5	6	7	8	9	A	В	С	D	E	F
	0	00	01	02	03	37	2D	2E	2F	16	05	25	0B	0C	0D	0E	0F
	1	10	11	12	13	3C	3D	32	26	18	19	3F	27	10	1D	1E	1F
	2	40	5A	7F	7B	5B	60	50	7D	4D	5D	5C	4E	6B	60	4B	61
	3	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	7A	5E	4C	7E	6E	6F
	4	7C	C1	C2	C3	C4	C5	C6	C7	C8	C9	D1	D2	D3	D4	D5	D6
	5	+	D8	D9	E2	E3	E4	E5	E6	E7	E8	E9	AD	E0	BD	5F	6D
 first	6	79	81	82	83	84	85	86	87	88	89	91	92	93	94	95	96
digit	7	97	98	99	A2	A3	A4	A5	A6	A7	A8	A9	C0	4F	D0	A1	07
byte	8	20	21	22	23	24	15	06	17	28	29	2A	2B	20	09	0A	1B
ASCII	9	30	31	1A	33	34	35	36	08	38	39	3A	3B	04	14	3E	F F
	A	41	AA	4A	B1	9F	B2	6A	B5	BB	B4	9A	8A	B0	CA	AF	BC
	B	90	8F	EA	FA	BE	A0	B6	B3	9D	DA	9B	8B	B7	B8	B9	A9
	C	64	65	62	66	63	67	9E	68	74	71	72	73	78	75	76	77
	D	AC	69	ED	EE	EB	E F	EC	BF	80	FD	FE	FB	FC	BA	AE	59
	E	44	45	42	46	43	47	90	48	54	51	52	53	58	55	56	57
	F	80	49	CD	CE	СВ	CF		E1	70	DD	DE	DB	DC	8D	8E	DF

Figure 71. EZACIC15 ASCII-to-EBCDIC table

Figure 72 on page 199 shows an example of EZACIC15 call instructions.

```
WORKING-STORAGE SECTION.

01 OUT-BUFFER PIC X(length of output).

01 LENGTH PIC 9(8) BINARY.

PROCEDURE DIVISION.

CALL 'EZACIC15' USING OUT-BUFFER LENGTH. IF RETURN-CODE > 0

THEN

DISPLAY 'TRANSLATION FAILED ' RETURN-CODE.
```

Figure 72. EZACIC15 call instruction example

For equivalent PL/I and assembly language declarations, see <u>"Converting parameter descriptions" on</u> page 54.

OUT-BUFFER

A buffer that contains the following infomation:

- When called, ASCII data
- Upon return, EBCDIC data

LENGTH

Specifies the length of the data to be translated.

RETURN-CODE

Upon return, register 15 contains a return code value, which indicates if the data translation occurred successfully. The return code can be one of the following values:

0

The data translation occurred.

8

Too many parameters passed, translation did not occur.

12

Zero buffer length passed, translation did not occur.

16

Zero buffer address passed, translation did not occur.

Call interface sample programs

This information provides sample programs for the call interface that you can use for a PL/I or COBOL application program.

The following are the sample programs that are available in the SEZAINST data set:

Program	Description
EZASOKPS	PL/I call interface sample IPv4 server program
EZASOKPC	PL/I call interface sample IPv4 client program
EZASO6PS	PL/I call interface sample IPv6 server program
EZASO6PC	PL/I call interface sample IPv6 client program
CBLOCK	PL/I common variables
EZACOBOL	COBOL common variables
EZASO6CS	COBOL call interface sample IPv6 server program
EZASO6CC	COBOL call interface sample IPv6 client program

Sample code for IPv4 server program

The EZASOKPS PL/I sample program is a server program that shows you how to use the following calls:

- ACCEPT
- BIND
- CLOSE
- GETSOCKNAME
- INITAPI
- LISTEN
- READ
- SOCKET
- TERMAPI
- WRITE

```
MODULE NAME: EZASOKPS - THIS IS A VERY SIMPLE IPV4 SERVER
**********
  Copyright:
              Licensed Materials - Property of IBM
               "Restricted Materials of IBM"
               5694-A01
               (C) Copyright IBM Corp. 1994, 2005
               US Government Users Restricted Rights -
Use, duplication or disclosure restricted by
GSA ADP Schedule Contract with IBM Corp.
               CSV1R7
/*
  Status:
                                                               */
     EZASOKPS: PROC OPTIONS(MAIN);
/* INCLUDE CBLOCK - common variables
                                                               */
```

% include CBLOCK;

ID.TCPNAME = 'TCPIP'; ID.ADSNAME = 'EZASOKPS'; /* Set TCP to use */ /* and address space name open file(driver); /* Execute INITAPI /* ******/ /* Uncomment this code to set max sockets to the maximum */ /* */ /* MAXSOC_INPUT = 65535; . /* MAXSOC_FWD = MAXSOC_INPUT; ***** call ezasoket(INITAPI, MAXSOC, ID, SUBTASK, MAXSNO, ERRNO, RETCODE); if retcode < 0 then do; msg = 'FAIL: initapi' || errno; write file(driver) from (msg); goto getout; end; /* Execute SOCKET */ call ezasoket(SOCKET, AF_INET, TYPE_STREAM, PROTO, ERRNO, RETCODE); if retcode < 0 then do; /* clear field */ write file(driver) from (msg); goto getout; end: else sock_stream = retcode; /* Execute BIND */ /* **/ name_id.port = 8888; /* internet address */ if retcode < 0 then do; msg = blank; msg = 'FAIL: bind' || errno; write file(driver) from (msg); /* clear field */ goto getout; end; /********************** */ /* Execute GETSOCKNAME */ **/ /* internet address */ /* clear field */ msg = blam, if retcode < 0 then do; msg = 'FAIL: getsockname, stream, internet' || errno; write file(driver) from (msg); end; else do; se d0, msg = 'getsockname = ' || name_id.address; write file(driver) from (msg); end: /* * /* Execute LISTEN */ /* backlog = 5; call ezasoket(LISTEN, SOCK_STREAM, BACKLOG, ERRNO, RETCODE); if retcode < 0 then do; msg = blank; /* clear field */

```
msg = 'FAIL: listen w/ backlog = 5' || errno;
  write file(driver) from (msg);
  goto getout;
end;
/*
                                                      */
/* Execute ACCEPT
                                                      */
/*
/* internet address
                                                      */
msg = blank;
if retcode < 0 then do;
msg = 'FAIL: accept' || errno;
                             /* clear field
                                                      */
  write file(driver) from (msg);
end;
else do;
  accpsock = retcode;
msg = 'accept socket = ' || accpsock;
  write file(driver) from (msg);
end:
/*
                                                      */
/
/* Execute READ
                                                      */
NBYTE, BUFIN, ERRNO, RETCODE);
msg = blank;
if retcode < 0 then do;
msg = 'FAIL: read' || errno;
write file(driver) from (msg);
                             /* clear field
                                                      */
end;
else do;
  msg = 'read = ' || bufin;
write file(driver) from (msg);
  bufout = bufin;
nbyte = retcode;
end;
******/
/*
                                                      */
/* Execute WRITE
                                                      */
call ezasoket(WRITE, ACCPSOCK, NBYTE, BUFOUT,
                ERRNO, RETCODE);
                             /* clear field
msg = blank;
                                                      */
if retcode < 0 then do;
msg = 'FAIL: write' || errno;
write file(driver) from (msg);
end;
else do;
  msg = 'write = ' || bufout;
  write file(driver) from (msg);
end:
/*
/* Execute CLOSE accept socket
/*
call ezasoket(CLOSE, ACCPSOCK,
                ERRNO, RETCODE);
if retcode < 0 then do;
  msg = blank; /* clean
msg = 'FAIL: close, accept sock' || errno;
write file(driver) from (msg);
                              /* clear field
                                                      */
end;
/*
                                                      */
/* Execute TERMAPI
                                                      */
getout:
call ezasoket(TERMAPI);
close file(driver);
end ezasokps;
```

Figure 73. EZASOKPS PL/1 sample server program for IPv4

Sample program for IPv4 client program

The EZASOKPC PL/I sample program is a client program that shows you how to use the following calls provided by the call socket interface:

- CONNECT
- GETPEERNAME
- INITAPI
- READ
- SHUTDOWN
- SOCKET
- TERMAPI
- WRITE

```
/*
                                                      */
   MODULE NAME: EZASOKPC - THIS IS A VERY SIMPLE IPV4 CLIENT
/*
                                                      */
/*
                                                      */
             Licensed Materials - Property of IBM
                                                      */
*/
  Copyright:
"Restricted Materials of IBM"
                                                      * * * * * * * * *
             5694-A01
             (C) Copyright IBM Corp. 1994, 2002
             US Government Users Restricted Rights -
Use, duplication or disclosure restricted by
GSA ADP Schedule Contract with IBM Corp.
/*
                                                      */
/*
  Status:
             CSV1R4
                                                       */
**/
EZASOKPC: PROC OPTIONS(MAIN);
/* INCLUDE CBLOCK - common variables
                                                      */
% include CBLOCK;
ID.TCPNAME = 'TCPIP';
ID.ADSNAME = 'EZASOKPC';
                              /* Set TCP to use
                             /* and address space name
open file(driver);
/* Execute INITAPI
                                                      */
call ezasoket(INITAPI, MAXSOC, ID, SUBTASK,
MAXSNO, ERRNO, RETCODE);
if retcode < 0 then do;
msg = 'FAIL: initapi' || errno;
write file(driver) from (msg);
  goto getout;
end:
*/
/* Execute SOCKET
                                                      */
call ezasoket(SOCKET, AF_INET, TYPE_STREAM, PROTO,
ERRNO, RETCODE);
if retcode < 0 then do;
  */
  goto getout;
end:
sock_stream = retcode;
                             /* save socket descriptor
                                                      */
 /* Execute CONNECT
name_id.port = 8888;
name_id.address = '01234567'BX; /* int
call ezasoket(CONNECT, SOCK_STREAM, NAME_ID,
ERRNO, RETCODE);
                              /* internet address
                                                      */
```

if retcode < 0 then do; /* clear field */ write file(driver) from (msg); goto getout; end: /* /* Execute GETPEERNAME 1* *****/ call ezasoket(GETPEERNAME, SOCK_STREAM, NAME_ID, ERRNO, RETCODE); msg = blank; if retcode < 0 then do; msg = 'FAIL: getpeername' || errno; /* clear field */ write file(driver) from (msg); end; else do; msg = 'getpeername =' || name_id.address; write file(driver) from (msg); end; /* */ /* Execute WRITE */ . /* . bufout = message; nbyte = length(message); call ezasoket(WRITE, SOCK_STREAM, NBYTE, BUFOUT, ERRNO, RETCODE); /* clear f msg = blank; if retcode < 0 then do; msg = 'FAIL: write' || errno; write file(driver) from (msg); /* clear field */ end; else do; msg = 'write = ' || bufout; write file(driver) from (msg); end: /* /* Execute READ */ /* nbyte = length(bufin); call ezasoket(READ, SOCK_STREAM, NBYTE, BUFIN, ERRNO, RETCODE); /* clear field msg = blank; if retcode < 0 then do; msg = 'FAIL: read' || errno; write file(driver) from (msg); */ end; else do; msg = 'read = ' || bufin; write file(driver) from (msg); end: /* */ /* Execute SHUTDOWN from/to /* getout: how = 2; call ezasoket(SHUTDOWN, SOCK_STREAM, HOW, ERRNO, RETCODE); if retcode < 0 then do; msg = blank; msg = 'FAIL: shutdown' || errno; /* clear field */ write file(driver) from (msg); end: /* */ /* Execute TERMAPI call ezasoket(TERMAPI); close file(driver); end ezasokpc;

Figure 74. EZASOKPC PL/1 sample client program for IPv4

Sample code for IPv6 server program

The EZASO6PS PL/I sample program is a server program that shows you how to use the following calls provided by the call socket interface:

- ACCEPT
- BIND
- CLOSE
- EZACIC09
- FREEADDRINFO
- GETADDRINFO
- GETHOSTNAME
- GETSOCKNAME
- INITAPI
- LISTEN
- NTOP
- PTON
- READ
- SOCKET
- TERMAPI
- WRITE

```
/*
                                                                */
.
/*
    MODULE NAME: EZASO6PS - THIS IS A VERY SIMPLE IPV6 SERVER
                                                                */
                                                                */
/*
/* Copyright:
/*
/*
/*
/*
/*
/*
/*
               Licensed Materials - Property of IBM
                                                                * * * * * * * * * *
               "Restricted Materials of IBM"
               5694-A01
               (C) Copyright IBM Corp. 2002, 2005
               US Government Users Restricted Rights -
Use, duplication or disclosure restricted by
GSA ADP Schedule Contract with IBM Corp.
/*
/*
                                                                */
/*
                                                                */
*/
/* Status:
               CSV1R7
                        ******
EZASO6PS: PROC OPTIONS(MAIN);
/* INCLUDE CBLOCK - common variables
                                                                */
% include CBLOCK;
ID.TCPNAME = 'TCPCS';
ID.ADSNAME = 'EZASO6PS';
                                  /* Set TCP to use
/* and address space name
                                                                */
open file(driver);
/* Execute INITAPI
                                                                */
**/
*/
/* Uncomment this code to set max sockets to the maximum.
                                                                */
/*
                                                                */
/* MAXSOC_INPUT = 65535;
/* MAXSOC_FWD = MAXSOC_INPUT;
                                                                */
call ezasoket(INITAPI, MAXSOC, ID, SUBTASK,
MAXSNO, ERRNO, RETCODE);
if retcode < 0 then do;
  msg = 'FAIL: initapi' || errno;
  write file(driver) from (msg);
goto getout;
end;
```

```
*/
/*
  Execute SOCKET
                                                     */
if retcode < 0 then do;
  */
  goto getout;
end;
else sock_stream = retcode;
*/
/*
/* Execute PTON
                                                     */
         *****
/*****
ERRNO, RETCODE);
if retcode < 0 then do;
  msg = blank;
msg = 'FAIL: pton' || errno;
                             /* clear field
                                                     */
  write file(driver) from (msg);
  goto getout;
end;
name6_id.address = NUMERIC_ADDR;
                             /* IPV6 internet address
**/
1*
                                                     */
/* Execute GETHOSTNAME
                                                     */
call ezasoket(GETHOSTNAME, HOSTNAME_LEN, HOSTNAME,
ERRNO, RETCODE);
msg = blank;
if retcode < 0 then do;
msg = 'FAIL: gethostname' || errno;
write file(driver) from (msg);
                             /* clear field
                                                     */
  goto getout;
end;
else do;
  se do;
msg = 'gethostname = ' || HOSTNAME;
write file(driver) from (msg);
GAI_NODE = HOSTNAME; /* Set ho
                     /* Set host name for getaddrinfo to use */
end;
/*
/* Execute GETADDRINFO
                                                     */
/*
GAI_SERVLEN = 0;
GAI_HINTS.FLAGS = ai_CANONNAMEOK;
HINTS = ADDR(GAI_HINTS);
                           /* set service length
                                                     */
                            /* Request canonical name
                                                     */
                            /* Set results pointer
call ezasoket(GETADDRINFO,
                GAI_NODE, GAI_NODELEN,
GAI_SERVICE, GAI_SERVLEN,
                HINTS, RES,
CANONNAME_LEN,
ERRNO, RETCODE);
msg = blank;
if retcode < 0 then do;
msg = 'FAIL: getaddrinfo' || errno;
write file(driver) from (msg);
                             /* clear field
                                                     */
end:
else do;
                            /* process returned RES
                                                     */
/* Call EZACIC09 to format the returned result address information
                                                     */
******/
call ezacic09(RES, OPNAMELEN, OPCANON, OPNAME, OPNEXT,
            RETCODE);
msg = blank;
                             /* clear field
                                                     */
if retcode ^= 0 then do;
  msg = 'FAIL: EZACIC09' || RETCODE;
  write file(driver) from (msg);
end;
else do;
msg = 'OPCANON = ' || OPCANON;
  write file(driver) from (msg);
end:
/* Execute FREEADDRINFO
```

```
call ezasoket(FREEADDRINFO, RES,
ERRNO, RETCODE);
msg = blank; /*
if retcode < 0 then do;
msg = 'FAIL: freeaddrinfo' || errno;
write file(driver) from (msg);</pre>
                           /* clear field
                                                  */
end;
     /* end from getaddrinfo */
end:
/*
/* Execute BIND
/*
name6_id.port = 8888;
call ezasoket(BIND, SOCK_STREAM, NAME6_ID,
ERRNO, RETCODE);
if retcode < 0 then do;
  msg = blank;
msg = 'FAIL: bind' || errno;
                           /* clear field
                                                  */
  write file(driver) from (msg);
  goto getout;
end;
/*
                                                  */
/* Execute GETSOCKNAME
*/
end;
/*
                                                  */
/* Execute LISTEN
                                                   */
backlog = 5;
call ezasoket(LISTEN, SOCK_STREAM, BACKLOG,
ERRNO, RETCODE);
if retcode < 0 then do;
  msg = blank; /* clear field
msg = 'FAIL: listen w/ backlog = 5' || errno;
write file(driver) from (msg);
                                                  */
  goto getout;
end;
******************
/*
                                                  */
/* Execute ACCEPT
                                                  */
call ezasoket(ACCEPT, SOCK_STREAM,
NAME6_ID, ERRNO, RETCODE);
msg = blank;
if retcode < 0 then do;
msg = 'FAIL: accept' || errno;
write file(driver) from (msg);
                           /* clear field
                                                  */
end:
else do;
  accpsock = retcode;
  msg = 'accept socket = ' || accpsock;
  write file(driver) from (msg);
end:
/* Execute NTOP
                                                  */
/*
msg = blank;
if retcode < 0 then do;
msg = 'FAIL: ntop' || errno;
write file(driver) from (msg);
                           /* clear field
                                                  */
  goto getout;
end;
else do;
      'presentable address = ' || PRESENTABLE_ADDR;
  msg =
  write file(driver) from (msg);
```

```
end:
                                 /*
                                                           */
              /* Execute READ
                                                           */
/*
nbyte = length(bufin);
call ezasoket(READ, ACCPSOCK,
                  NBYTE, BUFIN, ERRNO, RETCODE);
msg = blank;
if retcode < 0 then do;
msg = 'FAIL: read' || errno;
write file(driver) from (msg);
                               /* clear field
                                                           */
end;
else do;
   msg = 'read = ' || bufin;
   write file(driver) from (msg);
  bufout = bufin;
  nbyte = retcode;
end:
   /* Execute WRITE
                                                           */
+*1
call ezasoket(WRITE, ACCPSOCK, NBYTE, BUFOUT,
                  ERRNO, RETCODE);
msg = blank;
if retcode < 0 then do;
msg = 'FAIL: write' || errno;
write file(driver) from (msg);
                                /* clear field
                                                           */
end;
else do;
msg = 'write = ' || bufout;
  write file(driver) from (msg);
end;
**/
/*
                                                           */
/* Execute CLOSE accept socket
                                                           */
call ezasoket(CLOSE, ACCPSOCK,
                  ERRNO, RETCODE);
if retcode < 0 then do;
  msg = blank; /* clean
msg = 'FAIL: close, accept sock' || errno;
write file(driver) from (msg);
                                 /* clear field
                                                           */
end:
/*
  Execute TERMAPI
/*
/**
             getout:
call ezasoket(TERMAPI);
close file(driver);
end EZASO6PS;
```

Figure 75. EZASO6PS PL/1 sample server program for IPv6

Sample program for IPv6 client program

The EZASO6PC PL/I sample program is a client program that shows you how to use the following calls provided by the call socket interface:

- CONNECT
- GETNAMEINFO
- GETPEERNAME
- INITAPI
- PTON
- READ
- SHUTDOWN

- SOCKET
- TERMAPI
- WRITE

```
/*
/*
   MODULE NAME: EZASO6PC - THIS IS A VERY SIMPLE IPV6 CLIENT
                                                    */
                                                    */
/*
  Copyright:
            Licensed Materials - Property of IBM
                                                    */
/******
                                                    * * * * * * * * *
             "Restricted Materials of IBM"
             5694-A01
             (C) Copyright IBM Corp. 2002
             US Government Users Restricted Rights -
Use, duplication or disclosure restricted by
GSA ADP Schedule Contract with IBM Corp.
/*
/*
                                                    */
                                                    */
            CSV1R4
/* Status:
                                                    */
/*
                                                    */
     EZASO6PC: PROC OPTIONS(MAIN);
/* INCLUDE CBLOCK - common variables
                                                    */
% include CBLOCK;
ID.TCPNAME = 'TCPCS';
ID.ADSNAME = 'EZASO6PS';
                             /* Set TCP to use
                            /* and address space name
open file(driver);
/* Execute INITAPI
/*
call ezasoket(INITAPI, MAXSOC, ID, SUBTASK,
                MAXSNO, ERRNO, RETCODE);
if retcode < 0 then do;
    msg = 'FAIL: initapi' || errno;
    write file(driver) from (msg);
  goto getout;
end;
*/
/* Execute SOCKET
                                                    */
if retcode < 0 then do;
  */
goto getout;
end;
sock_stream = retcode;
                             /* save socket descriptor
                                                    */
*******/
/* Execute PTON
/*
PRÈSENTABLE_COLL_
ERRNO, RETCODE); /* clear field
msg = blank;
if retcode < 0 then do;
msg = 'FAIL: pton' || errno;
write file(driver) from (msg);
                                                    */
  goto getout;
end;
msg = 'SUCCESS: pton converted ' || PRESENTABLE_ADDR;
name6_id.address = NUMERIC_ADDR; /* IPV6 interne
                                                    */
                            /* IPV6 internet address
   /* Execute CONNECT
name6_id.port = 8888;
call ezasoket(CONNECT, SOCK_STREAM, NAME6_ID,
                ÉRRNO, RETCODE);
if retcode < 0 then do;
```

```
*/
  goto getout;
end:
/*
                                                    */
′/*
    Execute GETPEERNAME
*************/
call ezasoket(GETPEERNAME, SOCK_STREAM,
NAME6_ID, ERRNO, RETCODE);
                            /* clear field
msg = blank;
                                                    */
if retcode < 0 then do;
msg = 'FAIL: getpeername' || errno;
write file(driver) from (msg);
end:
  /*
                                                    */
′/*
    Execute GETNAMEINFO
                                                    */
                       /* Set length of NAME
/* Clear Host name
*/
                                                    */
                                                    */
msg = blank;
if retcode < 0 then do;
msg = 'FAIL: getnameinfo' || errno;
                                                    */
  write file(driver) from (msg);
end;
else do;
  se do;
msg = 'getnameinfo host=' || GNI_HOST ;
write file(driver) from (msg);
msg = 'getnameinfo service=' || GNI_SERVICE ;
write file(driver) from (msg);
end:
/* Execute WRITE
                                                    */
/*
msg = blank;
if retcode < 0 then do;
msg = 'FAIL: write' || errno;
                                                    */
  write file(driver) from (msg);
end;
else do;
msg = 'write = ' || bufout;
  write file(driver) from (msg);
end;
/*
                                                    */
/* Execute READ
                                                    */
nbyte = length(bufin);
call ezasoket(READ, SOCK_STREAM,
NBYTE, BUFIN, ERRNO, RETCODE);
msg = blank;
if retcode < 0 then do;
msg = 'FAIL: read' || errno;
write file(driver) from (msg);
                            /* clear field
                                                    */
end;
else do;
msg = 'read = ' || bufin;
  write file(driver) from (msg);
end;
/*
                                                    */
/* Execute SHUTDOWN from/to
                                                    */
```

```
getout:
how = 2;
call ezasoket(SHUTDOWN, SOCK_STREAM, HOW,
              ERRNO, RETCODE);
if retcode < 0 then do;
  msg = blank;
msg = 'FAIL: shutdown' || errno;
                           /* clear field
                                                 */
  write file(driver) from (msg);
end;
/*
  Execute TERMAPI
call ezasoket(TERMAPI);
close file(driver);
end ezaso6pc;
```

Figure 76. EZASO6PC PL/1 sample client program for IPv6

Common variables used in PL/I sample programs

The CBLOCK common storage area contains the variables that are used in the PL/I programs in this section.

```
*/
      MODULE NAME: CBLOCK - SOKET COMMON VARIABLES
/*
                                                                                */
                                                                                */
/*
/*
   Copyright:
                   Licensed Materials - Property of IBM
/******
                    "Restricted Materials of IBM"
                                                                                */ */ */
                    5694-A01
                    Copyright IBM Corp. 1994, 2010
 /*
                    US Government Users Restricted Rights -
/*
                                                                                */
*/
*/
/*
/*
                    Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
.
/*
   Part Type:
                    Enterprise PL/1 for z/OS
                                                                                 */
                                                                                */
                    CSV1R12
/* Status:
                                                                                */
                                                                                 */
*/
   Change Activity:
   Flag Reason Release Date
                                     Origin
                                                   Description
                                                                                */
   $A1= PH34590 HIP6240 210323 tevaller: Identify internal use
/*
                                                                                */
                                                   only interfaces with
                                                                                */
                                                   IFF_RESTRICTED
 /* SOKET COMMON VARIABLES
DCL ABS BUILTIN;
DCL ADDR BUILTIN;
DCL ACCEPT CHAR(16) INIT('ACCEPT');
DCL ACCEPI CHAR(16) INII('ACCEPI');
DCL ACCPSOCK FIXED BIN(15); /* temporary ACCEPT sc
DCL AF_INET FIXED BIN(31) INIT(2); /* internet domain
DCL AF_INET6 FIXED BIN(31) INIT(19); /* internet v6 domain
DCL AF_IUCV FIXED BIN(31) INIT(17); /* iucv domain
/* Mapping of GAI_HINTS/GAI_ADDRINFO FLAGS
DCL ai_PASSIVE BIT(32) INIT('00000001'BX);
                                            /* temporary ACCEPT socket
                                                                                */
                                                                                */
*/
                                                                                */
                                                                                */
/* flag: getaddrinfo hints
DCL ai_CANONNAMEOK BIT(32) INIT('000000002'BX);
                                                                                */
                                             /* flag: getaddrinfo hints
                                                                                */
DCL ai_NUMERICHOST BIT(32) INIT('00000004'BX);
/* flag: getaddrinfo hints
DCL ai_NUMERICSERV BIT(32) INIT('00000008'BX);
                                                                                */
                                              * flag: getaddrinfo hints
                                                                                */
DCL ai_V4MAPPED
                      BIT(32) INIT('00000010'BX)
                                             /* flag: getaddrinfo hints
                                                                                */
                      BIT(32) INIT('00000020'BX);
DCL ai_ALL
/* flag: getaddrinfo hints
DCL ai_ADDRCONFIG BIT(32) INIT('00000040'BX);
                                                                                */
                                             /* flag: getaddrinfo hints
                                                                                */
DCL ai_EXTFLAGS BIT(32) INIT('00000080'BX);
/* flag: getaddrinfo hints
DCL ai_ALLFLAGMASK BIT(32) INIT('FFFFFF00'BX);
                                                                                */
DCL ALĪAS CHAR(255);
                                             /* alternate NAME
                                                                                */
```

DCL APITYPE FIXED BIN(15) INIT(2); /* defaul DCL BACKLOG FIXED BIN(31); /* max lei DCL BADNAME CHAR(20); /* tempor DCL BIND CHAR(16) INIT('BIND'); DCL BIND2ADDRSEL CHAR(16) INIT('BIND2ADDRSEL'); DCL BIT BUILTIN; // bit co /* default API type */
/* max length of pending queue*/ /* temporary name DCL BITZERO BIT(1); /* bit zero value DCL BLANK255 CHAR(255) INIT(' '); DCL BLANK CHAR(100) INIT(' '); DCL BUF CHAR(80) INIT(' '); /* /* */ DCL BLANK CHAR(100) INIT(''); /* macro READ/WRITE buffer */
DCL BUFF CHAR(80) INIT(''); /* short buffer */
DCL BUFF CHAR(15) INIT(''); /* BUFFER */
DCL BUFFER CHAR(32767) INIT(''); /* Read buffer */
DCL BUFFUN CHAR(32767) INIT(''); /* Read buffer */
DCL BUFFUN CHAR(32767) INIT(''); /* Read buffer */
DCL BUFFUN CHAR(32767) INIT(''); /* BUFFER */
DCL CANONNAME_LEN FIXED BIN(31);/* getaddrinfo canonical name length*/
DCL 1 CLIENT, /* socket addr of connection peer */
2 DOMAIN FIXED BIN(31) INIT(2), /* domain of client (AF_INET) */
2 NAME CHAR(8) INIT(''), /* task identifier for client */
2 TASK CHAR(8) INIT(''); /* reserved */
DCL CLOSE CHAR(16) INIT('CLOSE');
DCL COMMAND FIXED BIN(31) INIT(3); /* Query FNDELAY flag */
DCL CONNECT CHAR(16) INIT('CONNECT'); /* temporary datagram socket */ /* macro READ/WRITE buffer DCL DATA_SOCK FIXED BIN(31) INIT(100); /* temporary datagram socket DCL DATA_SOCK FIXED BIN(15); /* temporary datagram socket DCL DEF FIXED BIN(31) INIT(0); /* default protocol DCL DONE_SENDING CHAR(1); /* ready flag DCL DRIVER FILE OUTPUT UNBUF ENV(FB RECSIZE(100)) RECORD; DCL ERETMSK CHAR(4); /* indicate exception events /* temporary datagram socket */ */ FIXED BIN(31);
FIXED BIN(31) INIT(0); /* error number variable DCL ERR */ DCL ERRNO /* error number /* check for pending /* exception events */ DCL ESNDMSK CHAR(4); */ */ /* common exit point DCL EXIT LABEL; DCL EXIT LABEL; /* common exit point */ DCL EZACIC05 ENTRY OPTIONS(ASM,INTER) EXT; /* translate ascii>ebcdic*/ DCL EZACIC09 ENTRY OPTIONS(ASM,INTER) EXT; /* format getaddrinfo res*/ DCL EZASOKET ENTRY OPTIONS(ASM,INTER) EXT; /* socket call */ DCL FCNTL CHAR(16) INIT('FCNTL'); DCL FIONBIO BIT(32) INIT('8004A77F'BX); /* flag: nonblocking */ DCL FIONBIO BIT(32) INIT('4004A77F'BX); /* flag: #readable bytes */ DCL FLAGS FIXED BIN(31) INIT(0); /* default: no flags */ /* 1 = 00B, SEND OUT-OF-BAND*/ /* 4 = DON'T ROUTE */ DCL FEPEEADDRINEO_CHAR(16) INIT('FCEEADDRINEC'): DCL FREEADDRINFO CHAR(16) INIT('FREEADDRINFO'); DCL GAI_NODE CHAR(25) INIT(' '); /* getaddrinfo node DCL GAI_NODE CHAR(255) INIT(' '); /* getaddrinfo node DCL GAI_SERVICE CHAR(32) INIT('255);/* getaddrinfo service DCL GAI_SERVICE CHAR(32) INIT(' '); /* getaddrinfo service DCL GAI_SERVLEN FIXED BIN(31) INIT(32); /* getaddrinfo service /* length */ */ /* getaddrinfo hints addrinfo */ DCL 1 GAI HINTS, 2 FLAGS FIXED BIN(31) INIT(0), FIXED BIN(31) INIT(0), FIXED BIN(31) INIT(0), /* hints family /* hints socket type 2 AF */ 2 SOCTYPE */ /* hints protocol 2 PROTO 2 NAMELEN FIXED BIN(31) INIT(0), CHAR(4), 2 * 2 * CHAR(4)2 CANONNAME FIXED BIN(31) INIT(0), CHAR(4) 2 2 NAME FIXED BIN(31) INIT(0), 2 * CHAR(4)FIXED BIN(31) INIT(0), FIXED BIN(31) INIT(0); 2 NEXT 2 EFLAGS /* see definitions after */ /* IPV6_ADDR_PREFERENCES */ DCL 1 GAI_ADDRINFO BASED(RES), /* getaddrinfo RES addrinfo 2 FLAGS 2 AF 2 SOCTYPE FIXED BIN(31), /* see ai_PASSIVE & following defns*/ FIXED BIN(31), FIXED BIN(31), FIXED BIN(31), 2 PROTO 2 NAMELEN FIXED BIN(31), /* RES socket address struct length*/ 2 * CHAR(4), CHAR(4) 2 + 2 CANONNAME POINTER, /* RES canonical name */ CHAR(4), 2 2 NAME POINTER. /* RES socket address structure */ CHAR(4), 2 2 NEXT /* RES next addrinfo, zero if none.*/ POINTER FIXED BIN(31); /* see definitions that follow the */
 /* IPV6_ADDR_PREFERENCES definition*/ 2 EFLAGS DCL 1 GAI_NAME_ID BASED(GAI_ADDRINFO.NAME), 2 LEN BIT(8), 2 FAMILY BIT(8), 2 PORT BIT(16), 2 ADDRESS BIT(32) 2 RESERVED1 CHAR(8); DCL 1 GAI_NAME6_ID BASED(GAI_ADDRINF0.NAME), 2 LEN BIT(8), 2 FAMILY BIT(8), 2 PORT BIT(16), 2 FLOWINFO FIXED BIN(31), 2 ADDRECC CHAR(4(1)) 2 ADDRESS CHAR(16),

2 SCOPEID FIXED BIN(31); DCL GETADDRINFO CHAR(16) INIT('GETADDRINFO'); DCL GETCLIENTID CHAR(16) INIT('GETCLIENTID'); DCL GETHOSTBYADDR CHAR(16) INIT('GETHOSTBYADR'); DCL GETHOSTBYNAME CHAR(16) INIT('GETHOSTBYNAME'); DCL GETHOSTDYNAME CHAR(16) INIT('GETHOSTNAME'); DCL GETHOSTID CHAR(16) INIT('GETHOSTID'); DCL GETIBMOPT CHAR(16) INIT('GETIBMOPT'); DCL GETIBMOPT CHAR(16) INIT('GETNAMEINFO'); DCL GETIBMOPT CHAR(16) INIT('GETIBMOPT'); DCL GETNAMEINFO CHAR(16) INIT('GETNAMEINFO'); DCL GETPEERNAME CHAR(16) INIT('GETPEERNAME'); DCL GETSOCKNAME CHAR(16) INIT('GETSOCKNAME'); DCL GETSOCKOPT CHAR(16) INIT('GETSOCKOPT'); DCL GIVESOCKET CHAR(16) INIT('GIVESOCKET'); DCL GLOBAL CHAR(16) INIT('GLOBAL'); DCL GLOBAL CHAR(16) INIT('GLOBAL'); DCL GNI_FLAGS FIXED BIN(31); /* getnameinfo flags DCL GNI_HOST CHAR(255); /* getnameinfo host DCL GNI_HOST CHAR(255); /* getnameinfo host length DCL GNI_SERVICE CHAR(32); /* getnameinfo service */ */ /* getnameinfo service length /* Group_Filter for IPv4 DCL GNI_SERVLEN FIXED BIN(31); DCL 1 GROUP_FILTER4 BASED, 2 GF4_HEADER, /* Header portion */ GF4_HEADER, 3 GF4_INTERFACE FIXED BIN(31), /* Interface index 3 * CHAR(4), /* Padding 3 GF4_GROUP, /* Group Multi Address BIT(8), /* Socket len /* Socket family 4 GF4_SOCK_LEN 4 GF4_SOCK_FAMILY BIT(8) 4 GF4_SOCK_SIN_PORT BIT(16) 4 GF4_SOCK_SIN_ADDR BIT(32), BIT(16), /* Socket port
/* Socket address *********** /* Unused 4 GF4_RESERVED1 CHAR(8) CHAR(112) /* 3 GF4_FMODE FIXED BIN(31), /* Filter mode 3 GF4_NUMSRC 2 GF4 SLIST FIXED BIN(31), /* Num of sources
/* Source list CHAR(0); DCL 1 GF4_SRCENTRY BASED, /* Source Entry 2 GF4_SRCADDR, /* Source IP address GF4_SCCADDR, 3 GF4_SOCK_LEN BIT(8), 3 GF4_SOCK_FAMILY BIT(8), 3 GF4_SOCK_SIN_PORT BIT(16), 3 GF4_SOCK_SIN_PORT BIT(16), 3 GF4_SOCK_SIN_ADDR BIT(32), /* Socket len /* Socket family BIT(8), /* Socket port
/* Socket address 3 GF4_RESERVED1 CHAR(8), /* Unused CHAR(112); */ 3 /* DCL 1 GROUP_FILTER6 BASED, 2 GF6_HEADER, */ */ */ /* Group_Filter for IPv6 ROUP_FILTER6 BASED, /* Group_Filter for IPv6 GF6_HEADER, /* Header portion 3 GF6_INTERFACE FIXED BIN(31), /* Interface index 3 * CHAR(4), /* Padding 3 GF6_GROUP, /* Group Multi Address 4 GF6_SOCK_LEN BIT(8), /* Socket len 4 GF6_SOCK_FAMILY BIT(8), /* Socket family 4 GF6_SOCK_SIN6_PORT BIT(16), /* Socket family 4 GF6_SOCK_SIN6_FLOWINFO FIXED BIN(31), /* flow info 4 GF6_SOCK_SIN6_ADDRESS CHAR(16), /* Socket address 4 GF6_SOCK_SIN6_SIN6_SCHED_FIXED BIN(31), /* Socket score */ */ */ 4 GF6_SOCK_SIN6_SCOPEID FIXED BIN(31), /* Socket scopeid 4 * CHAR(100), /* GF6_FMODE FIXED BIN(31), /* Filter mode */ /* /* Filter mode /* Num of sources */ 3 GF6_FMODE */ FIXED BIN(31), 3 GF6_NUMSRC 2 GF6_SLIST */ */ CHAR(0); /* Source list DCL 1 GF6_SRCENTRY BASED, /* Source Entry GF6_SRCADDR, /* SOURCE Entry 2 GF6_SRCADDR, /* Source IP address 3 GF6_SOCK_LEN BIT(8), /* Socket Ien 3 GF6_SOCK_FAMILY BIT(8), /* Socket family 3 GF6_SOCK_SIN6_PORT BIT(16), /* Socket port 3 GF6_SOCK_SIN6_FLOWINFO FIXED BIN(31), /* flow info 3 GF6_SOCK_SIN6_ADDRESS CHAR(16), /* Socket address 3 GF6_SOCK_SIN6_SCOPEID FIXED BIN(31), /* Socket scopeid 3 * CHAR(100) /* /* */ */ */ */ */ */ */ */ CHAR(100); DCL 1 GROUP_REQ4 BASED, /* Group_Req for IPv4 2 GR4_INTERFACE FIXED BIN(31), /* Interface index /* Padding /* Socket len /* Socket family CHAR(4), BIT(8), 2 GR4_SOCK_LEN GR4_SOCK_FAMILY BIT(8), GR4_SOCK_SIN_PORT BIT(16) GR4_SOCK_SIN_ADDR BIT(32), /* Socket port /* Socket address BIT(16), 2 2 2 GR4_RESERVED1 CHAR(8), /* Unused /* CHAR(112); * /* Group_Req for IPv6 /* Interface index DCL 1 GROUP REQ6 BASED, 2 GR6_INTERFACE FIXED BIN(31), GR6_INTERFACE FIXED BIN(31), /* Interface index * CHAR(4), /* Padding GR6_SOCK_LEN BIT(8), /* Socket len GR6_SOCK_SIN6_PORT BIT(16), /* Socket family GR6_SOCK_SIN6_FLOWINFO FIXED BIN(31), /* flow info GR6_SOCK_SIN6_ADDRESS CHAR(16), /* Socket address GR6_SOCK_SIN6_SCOPEID FIXED BIN(31), /* Socket scopeid * CHAR(100): /* 2 2 2 */ 2 2 * CHAR(100); /* Sucket scopeid GROUP_SOURCE_REQ4 BASED, /* Group_Source_Req for IPv4 2 GSR4_INTERFACE FIXED BIN(31), /* Interface index 2 * CHAR(4), /* Padding 2 GSR4_GROUP, // Multicett 2 * CHAR(100); DCL 1 GROUP_SOURCE_REQ4 BASED, /* Multicast group addr 3 GSR4_SOCK_LEN BIT(8), /* Socket len 3 GSR4_SOCK_FAMILY BIT(8), /* Socket family 3 GSR4_SOCK_SIN_PORT BIT(16), /* Socket port 3 GSR4_SOCK_SIN_ADDR BIT(32), /* Socket address */ */ */

3 GSR4_RESERVED1 CHAR(8), /* Unused CHAR(112), /* 3 * 2 GSR4_SOURCE, /* Source IP address BIT(8), /* Socket len 3 GSR4_SOCK_LEN 3 GSR4_SOCK_FAMILY BIT(8), /* Socket family 3 GSR4_SOCK_SIN_PORT BIT(16), /* Socket port 3 GSR4_SOCK_SIN_ADDR BIT(32), /* Socket address 3 GSR4_RESERVED1 CHAR(8), /* Unused */ CHAR(112); /* 3 * CHAR(112); /* DCL 1 GROUP_SOURCE_REQ6 BASED, /* Group_Source_Req for IPv6 2 GSR6_INTERFACE FIXED BIN(31), /* Interface index 2 * CHAR(4), /* Padding 2 GSR6_GROUP, /* Multicast group addr 3 GSR6_SOCK_LEN BIT(8), /* Socket Ien 3 GSR6_SOCK_LEN BIT(8) /* Socket Ien 3 GSR6_SOCK_LEN 3 GSR6_SOCK_FAMILY 3 GSR6_SOCK_LEN BIT(6), /* Socket len 3 GSR6_SOCK_FAMILY BIT(8), /* Socket port 3 GSR6_SOCK_SIN6_PORT BIT(16), /* Socket port 3 GSR6_SOCK_SIN6_FLOWINFO FIXED BIN(31), /* flow info 3 GSR6_SOCK_SIN6_ADDRESS CHAR(16), /* Socket address 3 GSR6_SOCK_SIN6_SCOPEID FIXED BIN(31), /* Socket scopeid 3 * */ CHAR(100), /* 3 * 2 GSR6_SOURCE, /* Source IP address

 GSR6_SOURCE,
 /* Source IP address
 */

 3 GSR6_SOCK_LEN
 BIT(8),
 /* Socket lan
 */

 3 GSR6_SOCK_FAMILY
 BIT(8),
 /* Socket family
 */

 3 GSR6_SOCK_SIN6_PORT
 BIT(16),
 /* Socket port
 */

 3 GSR6_SOCK_SIN6_FLOWINFO FIXED BIN(31),
 /* flow info
 */

 3 GSR6_SOCK_SIN6_ADDRESS
 CHAR(16),
 /* Socket address
 */

 3 GSR6_SOCK_SIN6_SCOPEID FIXED BIN(31),
 /* flow info
 */

 3 GSR6_SOCK_SIN6_ADDRESS
 CHAR(160);
 /* Socket address
 */

 3 GSR6_SOCK_SIN6_SCOPEID FIXED BIN(31),
 /* Socket scopeid
 */

 3 *
 CHAR(100);
 /*
 /*

 7 S POINTER;
 /*getaddrinfo hints addrinfo pointer*/
 */

 METE
 /* Home_Interface_Structure_*/
 */

 DCL HINTS POINTER; DCL 1 HOMEIF, 2 ADDRESS CHAR(16); /* Home Interface Structure */ /* Home Interface Address */ DCL HOSTADDR BIT(32); DCL HOSTNAME CHAR(24); /* host internet address /* host name from GETHOSTNAME */ DCL HOSTNAME_LEN FIXED BIN(31) INIT(24); /* host name length GETHOSTNAME */ * host name length GETHOSTNAME */
/* how shutdown is to be done */
/* Host entry */
/* Official name of host */
/* Alias list address */
/* Host address type */
/* Length of address from */
/* List of addresses from */
/* name server */
/* loop index */ DCL HOW FIXED BIN(31) INIT(2); Dcl 1 HOSTENT Based, 3 H_NAME 3 H_ALIASES POINTER. POINTER, 3 H_ADDRTYPE BIT(32), H_LENGTH FIXED BIN(31), 3 H_ADDR_LIST POINTER; DCL I FIXED BIN(15); /* loop index FIXED BIN(31) INIT(2); DCL ICMP /* prototype icmp ??? */ DCL 1 ID, 2 TCPNAME CHAR(8) INIT('TCPIP'), /* remote address space 2 ADSNAME CHAR(8) INIT('USER9'); /* local address space DENT POINTER; /* TCP/IP Addr Space */ DCL IDENT POINTER; DCL IFCONF CHAR(255); */ /* configuration structure 1 IF_NAMEINDEX, DCL 2 IF_NIHEADER, 3 IF_NITOTALIF FIXED BIN(31), /*Total Active Interfaces on Sys. */ 3 IF_NIENTRIES FIXED BIN(31), /* Number of entries returned */ 2 IF_NITABLE(10) CHAR(24); 2 IF_NITABLE(10, SINK)(21, 7) DCL 1 IF_NAMEINDEXENTRY, 2 IF_NIINDEX FIXED BIN(31), 2 IF_NIAME CHAR(16), /* Interface Index /* Interface Name, blank padded */ 2 IF_NIEXT, 3 IF_NINAMETERM CHAR(1), 3 IF_RESERVED CHAR(3); /* Null for C for Name len=16 */ /* Reserved */ DCL 1 IFREQ, 2 IFR_NAME CHAR(16), 2 IFR_IFR_UNION, /* Interface Structure /* Interface Name, blank padded */ 3 IFR ADDR, /* Interface IP Address /* 4 IFR_ADDR_LEN BIT(8), Socket Len */ 4 IFR_ADDR_FAMILY BIT(8), /* Socket Family */ 4 IFR_ADDR_PORT BIT(16), 4 IFR_ADDR_ADDR BIT(32), 4 IFR_ADDR_RSVD CHAR(8), /* Socket Port . /* /* Socket Address Socket Reserved */ 3 IFR_DSTADDR, /* Interface Dest IP Addr */ BIT(8), 4 IFR_DSTADDR_LEN /* Socket Len 4 IFR_DSTADDR_FAMILY BIT(8), 4 IFR_DSTADDR_PORT BIT(16), 4 IFR_DSTADDR_ADDR BIT(32), /* Socket Family /* /* Socket Port Socket Address Socket Reserved */ 4 IFR_DSTADDR_RSVD CHAR(8), /* 3 IFR_BROADADDR, /* Interface Broadcast IP Addr*/ /* Interface Brv 4 IFR_BROADADDR_LEN BIT(8), /* Socket Lu 4 IFR_BROADADDR_FAMILY BIT(8), /* Socket Fa 4 IFR_BROADADDR_PORT BIT(16), /* Socket Port 4 IFR_BROADADDR_ADDR BIT(32), /* Socket Ac 4 IFR_BROADADDR_RSVD CHAR(8), /* Socket Pa IFR_FLAGS BTT(14) Socket Len Socket Family */ */ Socket Address Socket Reserved 4 IFR_BROADS BIT(16), 3 IFR_FLAGS BIT(16), 3 IFR_METRIC FIXED BIN(31), 3 IFR_DATA FIXED BIN(31), 5 TER MTU FIXED BIN(31); /* Interface Flags /* Interface Metric /* Interface Data */ /* Interface MTU /* The following constants are for use with the IFR_FLAGS field */ /* in structure IFREQ. */

since we don't allow BIT(16) INIT('0008'BX); /* software loopback @A1A*/ DCL IFF_LOOPBACK DCL IFF_POINTOPOINT BIT(16) INIT('0008'BX); /* software loopback DCL IFF_POINTOPOINT BIT(16) INIT('0010'BX); /* point-to-point link DCL IFF_NOTRAILERS BIT(16) INIT('0020'BX); /* avoid use trailers DCL IFF_RUNNING BIT(16) INIT('0040'BX); /* resources allocated DCL IFF_PROMISC BIT(16) INIT('0080'BX); /* no ARP DCL IFF_PROMISC BIT(16) INIT('0100'BX); /* receive all packets DCL IFF_ALLMULTI BIT(16) INIT('0200'BX); /* multicast packets DCL IFF_MULTICAST BIT(16) INIT('0400'BX); /* multicast capable DCL IFF_POINTOWULTIPT BIT(16) INIT('0400'BX); /* multicast capable */ */ DCL IFF_POINTOMULTIPT BIT(16) INIT('0400 BX); /* multicast capable DCL IFF_POINTOMULTIPT BIT(16) INIT('0800'BX); /* pt-to-multipt DCL IFF_BRIDGE BIT(16) INIT('1000'BX); /* support token ring DCL IFF_SNAP BIT(16) INIT('2000'BX); /* support extended SAP DCL IFF_VIRTUAL BIT(16) INIT('4000'BX); /* virtual interface DCL IFF_SAMEHOST BIT(16) INIT('8000'BX); /* Samehost */ BUILTIN; DCL INDEX DCL IOCTL CHAR(16) INIT('IOCTL'); DCL IOCTL_CMD FIXED BIN(31); /* ioctl command DCL IOCTL_REQARG POINTER ; /* send pointer to data area*/ /* return pointer to data area*/ /* command request argument */ DCL IOCTL_RETARG DCL IOCTL_REQ00 POINTER FIXED BIN(31); DCL IOCTL_REQ04 DCL IOCTL_REQ08 FIXED BIN(31); /* command request argument FIXED BIN(31); /* command request argument DCL IOCTL_REQ32 CHAR(32) INIT('); /* command request argument */ DCL IOCTL_RET00 FIXED BIN(31); /* command return DCL IOCTL_RET04 FIXED BIN(31); /* command return DCL INET6_IS_SRCADDR CHAR(16) INIT('INET6_IS_SRCADDR'); DCL INITAPI CHAR(16) INIT('INITAPI'); /* /* command return argument /* command return argument argument */ FIXED BIN(31) INIT(1); /* prototype ip DCL IP ??? */ DCL 1 IP_MREQ, 2 IMR_MULTIADDR BIT(32), 2 IMR_INTERFACE BIT(32); /* IP multicast addr of group */ /* local IP addr of interface */ DCL 1 IPV6_MREQ, 2 IPV6MR_MULTIADDR CHAR(16), 2 IPV6MR_INTERFACE FIXED BIN(31); DCL 1 IP_MREQ_SOURCE BASED, /* Multi source API structure */ 2 IMRS_MULTIADDR BIT(32), 2 IMRS_SOURCEADDR BIT(32), /* IP multicast addr of grp /* IP source addr */ */ 2 IMRS_INTERFACE BIT(32); DCL 1 IP_MSFILTER BASED, /* local IP addr of intf /* IP_MsFilter 2 IMSF_HEADER, /* Header portion */ 3 IMSF_MULTIADDR BIT(32), 3 IMSF_INTERFACE BIT(32), 3 IMSF_FMODE FIXED BI /* Multicast address /* Interface address FIXED BIN(31), /* Filter mode FIXED BIN(31), /* Num of sources HAR(0); /* Source list 3 IMSF_NUMSRC 2 IMSF SLIST CHAR(0);*/ DCL 1 IMSF_SRCENTRY BASED /* Source Entry */ 2 IMSF_SCADDR BIT(32); /* Source DCL IP_MULTICAST_TTL BIT(32) INIT('00100003'BX); /* Source IP address /* getsockopt/setsockopt OPTNAME */ DCL IP_MULTICAST_LOOP BIT(32) INIT('00100004'BX); /* getsockopt/setsockopt OPTNAME */
DCL IP_MULTICAST_IF BIT(32) INIT('00100007'BX); /* getsockopt/setsockopt OPTNAME */ DCL IP_ADD_MEMBERSHIP BIT(32) INIT('00100005'BX); /* getsockopt/setsockopt OPTNAME */ DCL IP_DROP_MEMBERSHIP BIT(32) INIT('00100006'BX); /* getsockopt/setsockopt OPTNAME */
DCL IP_BLOCK_SOURCE BIT(32) INIT('0010000A'BX); /* getsockopt/setsockopt OPTNAME */ DCL IP_UNBLOCK_SOURCE BIT(32) INIT('0010000B'BX); /* getsockopt/setsockopt OPTNAME */
DCL IP_ADD_SOURCE_MEMBERSHIP BIT(32) INIT('0010000C'BX); DCL IP_DROP_SOURCE_MEMBERSHIP BIT(32) INIT('0010000'BX); DCL IPRES POINTER; /* FZACTOOP DESCRIPTION DCL IPRES POINTER; /* EZACICO9 RES addrinfo ptr DCL IPV6_ADDR_PREFERENCES BIT(32) INIT('00010020'BX); /* getsockopt/setsockopt OPTNAME */ /* Mapping of GAI_HINTS/GAI_ADDRINFO EFLAGS flags and /* IPV6_ADDR_PREFERENCES getsockopt, setsockopt OPTVAL flags, and /* inet6_is_srcaddr flags /* Prefer home IPv6 address over care-of IPv6 address DCL_IPV6_PREFER_SRC_HOME___BIT(32) INIT('00000001'BX); */ Prefer care-of IPv6 address over home IPv6 address IPV6_PREFER_SRC_COA BIT(32) INIT('00000002'BX); */ DCL_IPV6_PREFER_SRC_COA /* Prefer temporary IPV6 address over public IPV6 address DCL IPV6_PREFER_SRC_TMP BIT(32) INIT('00000004'BX); /* Prefer public IPv6 address over temporary IPv6 address */ */ DCL IPV6_PREFER_SRC_PUBLIC BIT(32) INIT('00000008'BX); /* Prefer cryptographic address over non-cryptographic address */ BIT(32) INIT('00000010'BX); DCL IPV6_PREFER_SRC_CGA /* Prefer non-cryptographic address over cryptographic address DCL IPV6_PREFER_SRC_NONCGA BIT(32) INIT('00000020'BX); /* Invalid EFLAGS or IPV6_ADDR_PREFERENCES OPTVAL flags DCL IPV6_PREFERENCES_FLAGS_MASKINVALID BIT(32) INIT('FFFFFC0'BX); */ */ DCL IPV6_JOIN_GROUP BIT(32) INIT('00010005'BX); /* getsockopt/setsockopt OPTNAME */ DCL IPV6_LEAVE_GROUP BIT(32) INIT('00010006'BX);

restricted interfaces

/* getsockopt/setsockopt OPTNAME */ DCL IPV6_LOOPBACK CHAR(3) INIT('::1'); DCL IPV6_MULTICAST_HOPS BIT(32) INIT('00010009'BX); getsockopt/setsockopt OPTNAME */ DCL IPV6_MULTICAST_IF BIT(32) INIT('00010007'BX); /* getsockopt/setsockopt OPTNAME */
DCL IPV6_MULTICAST_LOOP BIT(32) INIT('00010004'BX); /* getsockopt/ DCL IPV6_UNICAST_HOPS BIT(32) INIT('00010003'BX); getsockopt/setsockopt OPTNAME */ /* getsockopt/setsockopt OPTNAME */ DCL IPV6_V60NLY BIT(32) INIT('0001000A'BX); FIXED BIN(15); FIXED BIN(15); DCL J /* loop index DCL K /* loop index DCL LENGTH BUILTIN; DCL LABL CHAR(9); DCL LISTEN CHAR(16) INIT('LISTEN'); DCL MAXSNO FIXED BIN(31) INIT(0); /* r DCL 1 MAXSOC_INPUT FIXED BIN(31) INIT(0); /* max descriptor assigned */ DCL 1 MAXSOC_FWD, 2 MAXSOC_IGNORE FIXED BIN(15) INIT(0), 2 MAXSOC_FIXED BIN(15)_INIT(255); /* largest sock # checked DCL MCAST_JOIN_GROUP BIT(32) INIT(253), /* largest sock # checked */ /* getsockopt/setsockopt OPTNAME */ DCL MCAST_LEAVE_GROUP BIT(32) INIT('00100029'BX); /* getsockopt/setsockopt OPTNAME */ DCL MCAST_JOIN_SOURCE_GROUP BIT(32) INIT('0010002A'BX); /* getsockopt/setsockopt OPTNAME */
DCL MCAST_LEAVE_SOURCE_GROUP BIT(32) INIT('0010002B'BX); getsockopt/setsockopt OPTNAME */ DCL MCAST_BLOCK_SOURCE BIT(32) INIT('0010002C'BX); /* getsockopt/setsockopt OPTNAME */ DCL MCAST_UNBLOCK_SOURCE BIT(32) INIT('0010002D'BX); /* getsockopt/setsockopt OPTNAME */ /* getsockopt/setsockopt OPINAME DCL MCAST_EXCLUDE BIT(32) INIT('00000001'BX); DC1 MCAST_INCLUDE BIT(32) INIT('00000000'BX); DC1 MCAST_NUMSRC_MAX BIT(32)INIT('00000040'BX); DC1 MCAST_NUMSRC_MAX BIT(32)INIT('10ve my 1@ Rottweiler!'); /* message DC1 MSG CHAR(50) INIT('1); /* message text DC1 1 NAME ID DC1 1 NAME ID

 GG
 CHAR(100) INIT(' ');
 /* message text
 */

 NAME_ID,
 /* socket addr of connection peer */

 2 FAMILY FIXED BIN(15) INIT(2), /*addr'g family TCP/IP def */

 2 PORT
 BIT(16),
 /* system assigned port #

 2 ADDRESS
 BIT(32),
 /* 32-bit internet
 */

 DCL 1 NAME ID, 2 RESERVED CHAR(8); /* reserved DCL 1 NAME6_ID, /* socket addr of connection peer */ 2 FAMILY FIXED BIN(15) INIT(19), /* NAMELN IGNORED & FAMILY */ /* port # /* Flow info 2 PORT BIT(16) 2 FLOWINFO FIXED BIN(31), 2 ADDRESS CHAR(16), /* IPv6 internet address */ 2 SCOPEID FIXED BIN(31); /* Scope ID */ DCL NAMEL CHAR(255) VARYING: /* name field, long DCL NAMES CHAR(24) /* name field, short DCL NAMELEN FIXED BIN(31); /* length of name/alias field /* Number of bytes in buffer /* Network Configuration Hdr */ DCL NBYTE FIXED BIN(31); */ DCL 1 NETCONFHDR, */ 2 NCHEYECATCHER CHAR(4) INIT('6NCH') , /* Eye Catcher '6NCH 2 NCHIOCTL BIT(32) INIT('C014F608'BX), /* The IOCTL being processed */ */ Number of HomeIF returned via */ SIOCGHOMEIF6 or the number of*/ GRT6RtEntry's returned via */ 2 NCHNUMENTRYRET FIXED BIN(31); /* /* /* SIOCGRT6TABLE. */ DCL NI_NOFQDN FIXED BIN(31) INIT(1); /* flag: getnameinfo */ DCL NI_NUMERICHOST FIXED BIN(31) INIT(2); /* flag: getnameinfo */ DCL NI_NAMEREQD FIXED BIN(31) INIT(4); /* flag: getnameinfo DCL NI_NUMERICSERV FIXED BIN(31) INIT(8); /* flag: getnameinfo DCL NI_DGRAM FIXED BIN(31) INIT(16); /* flag: getnameinfo */ DCL NI_NUMERICSCOPE FIXED BIN(31) INIT(32); DCL NOTE(3) CHAR(25) INIT('Now is the time for 198 g', */ 'ood people to come to the', aid of their parties!'); /* socket descriptor, new DCI NS FIXED BIN(15); CHAR(16) INIT('NTOP'); DCL NTOP /* Numeric to Presentation */ DCL NULL BUILTIN; DCL 1 NUMERIC_ADDR CHAR(16); DCL OPNAMELEN FIXED BIN(31); /* NTOP/PTON Numeric address */ /* Socket address structure length */ /* Canonical name */ DCL OPCANON CHAR(256); DCL OPNAME POINTER; /* Socket address structure /* Next result address structure /* Next result address info in chain /* length of OPTVAL string /* length of OPTVAL string /* OPTNAME value (macro) /* OPTNAME value (call) DCL OPNEXT POINTER */ DCI OPTI FIXED BIN(31); DCL OPTLEN FIXED BIN(31); */ DCL OPTN CHAR(15): */ DCL OPTNAME FIXED BIN(31);

```
DCL OPTVAL CHAR(255);
                                                                                               /* GETSOCKOPT option data
 DCL OPTVALD FIXED BIN(31);
                                                                                               /* SETSOCKOPT option data
DCL OPIVALD FIALS
DCL 1 OPT STRUC,
2 ON_OFF FIXED BIN(31) INIT(1), /* enable option
2 TIME FIXED BIN(31) INIT(5); /* time-out in seconds
/* structure for option
/* structure for option
                                                                                                                                                                        */
                                                                                               /* structure for option
/* enable option
                                                                                                                                                                        */
                                                                                               /* structure for option
                                                                                                                                                                        */
                                                                                               /* used for getsockopt
/* time-out in seconds
                                                                                                                                                                        */
                2 TIMEOUT FIXED BIN(31);
                                                                                                                                                                        */
                                                                                             /* debug tool
NTOP/PTON presentable address
 DCL PLITEST BUILTIN;
 DCL PRESENTABLE_ADDR CHAR(45);
 DCL PRESENTABLE_ADDR_LEN FIXED BIN(15);
                                                                        /* NTOP/PTON presentable address length*/
/* NTOP/PT
DCL PROTO FIXED BIN(31) INIT(0); /
DCL PTON CHAR(16) INIT('PTON'); /
DCL READ CHAR(16) INIT('READ');
DCL READV CHAR(16) INIT('READV');
DCL RECV CHAR(16) INIT('READV');
DCL RECVFROM CHAR(16) INIT('RECVFROM');
                                                                                              /* prototype default
/* Presentation to numeric
 DCL RECVMSG CHAR(16) INIT('RECVMSG');
 DCL REUSE
                             FIXED BIN(31) INIT('4'); /* toggle, reuse local addr
                                                                                               /* command request argument
 DCL REQARG
                             FIXED BIN(31);
                                                                                                                                                                        */
                                                                                         /* getaddrinfo RES addrinfo ptr */
/* return code variable */
DCL REŠ
DCL RETC
                              POINTER
                              FIXED BIN(31);
DCL RETARG CHAR(255);
DCL RETCODE FIXED BIN(31) INIT(0);
                                                                                               /* return argument data area
                                                                                               /* return code
DCL RETLEN FIXED BIN(31); /* return area data length
DCL RRETMSK CHAR(4); /* indicate READ EVENTS
DCL RSNDMSK CHAR(4); /* check for pending read events
DCL RTENTRY CHAR(50) INIT('dummy table'); /* router entry
DCL SAVEFAM FIXED BIN(15); /* temporary family name
DCL SELECB CHAR(4) INIT('1');
DCL SELECT CHAR(16) INIT('SELECT');
DCL SELECT CHAR(16) INIT('SELECTEX');
DCL SELECT CHAR(16) INIT('SENDMSG');
DCL SENDTO CHAR(16) INIT('SENDTO');
DCL SENDTO CHAR(16) INIT('SENDTO');
DCL SETADEVE1 CHAR(8) INIT('SETAPPLD');
DCL SETADEVE1 CHAR(8) INIT(15) INIT(1);
DCL SETADEVER FIXED BIN(15) INIT(48);
DCL SETADBUFLEN FIXED BIN(15) INIT(40);
DCL SETAPLDATA,
 DCL RETLEN FIXED BIN(31);
                                                                                               /* return area data length
                                                                                                                                                                        */
                                                                                                                                                                        */
 DCL 1 SETAPPLDATA,
            2 SETAD_EYE1 CHAR(8)
2 SETAD_VER FIXED B
2 SETAD_LEN FIXED B
                                               FIXED BIN(15),
FIXED BIN(15),
                                               CHAR(4),
             2 SETAD_PTR64
3 SETAD_PTRHW (HAR(4),
3 SETAD_PTRHW (HAR(4),
3 SETAD_PTR POINTER;
DCL SETADEYE2 CHAR(8) INIT('APPLDATA');
DCL 1 SETADCONTAINER,
2 SETAD_EYE2 CHAR(8),
2 SETAD_BUFFER CHAR(40);
DCL SETSOCKOPT CHAR(16) INIT('SETSOCKOPT');
DCL SHUTDOWN CHAR(16) INIT('SHUTDOWN');
DCL SIOCADDRT BIT(32) INIT('4004A707'BX); /* flag: add routing entry*/
DCL SIOCADDRT BIT(32) INIT('4004A707'BX); /* flag: out-of-band data*/
DCL SIOCATMARK BIT(32) INIT('4004A707'BX); /* flag: out-of-band data*/
DCL SIOCGIFADDR BIT(32) INIT('C020A70B'BX); /* flag: network int addr*/
DCL SIOCGIFADDR BIT(32) INIT('C020A70D'BX); /* flag netw int config*/
DCL SIOCGIFADDR BIT(32) INIT('C020A712'BX); /* flag netw int config*/
DCL SIOCGIFBRDADDR BIT(32) INIT('C020A714'BX); /* flag: netw int config*/
DCL SIOCGIFFLAGS BIT(32) INIT('C020A714'BX); /* flag: net des addr*/
DCL SIOCGIFFLAGS BIT(32) INIT('C020A771'BX); /* flag: net intf flags*/
DCL SIOCGIFMETRIC BIT(32) INIT('C020A771'BX); /* flag: get rout metr*/
DCL SIOCGIFMETRIC BIT(32) INIT('C020A771'BX); /* flag: get intf mtu */
DCL SIOCGIFMAMEINDEX BIT(32) INIT('4000F603'BX);
/* flag: name and indexes */
 DCL 1 SETADCONTAINER,
/* flag: name and indexes */
DCL SIOCGIFNETMASK BIT(32) INIT('C020A715'BX); /* flag: network mask*/
DCL SIOCGIFNONSENSE BIT(32) INIT('B669FD2E'BX); /* flag: nonsense */
DCL SIOCSIFMETRIC BIT(32) INIT('B020A718'BX); /* flag: set rout metr*/
DCL SIOCSIFMETRIC BIT(32) INIT('B018D90C'BX); /* set APPLDATA */
 DCL SIOCGIPMSFILTER BIT(32) INIT('C000A724'BX);
/* flag: get multicast src filter */
DCL SIOCSIPMSFILTER BIT(32) INIT('8000A725'BX);
/* flag: set multicast src filter
DCL SIOCGMSFILTER BIT(32) INIT('C000F610'BX);
                                                                                                                                                                      */
                                                                                   /* flag: get multicast src filter
                                                                                                                                                                        */
 DCL SIOCSMSFILTER BIT(32) INIT('8000F611'BX);
                                                                                    /* flag: set multicast src filter
                                                                                                                                                                        */
 /* The following constant is defined in EZBZTLS1, but is also 
/* included here for completeness.
                                                                                                                                                                        */
                                                                                                                                                                        */
 /* DCL SIOCTTLSCTL BIT(32) INIT('C038D90B'BX)
                                                                                                                  /* flag: ttls
 /* The following constants are defined in EZBPINF1, but is also
/* included here for completeness.
/* DCL SIOCSPARTNERINFO BIT(32) INIT('8004F613'BX);
/* DCL SIOCGPARTNERINFO BIT(32) INIT('C000F612'BX);
/* flag: PartnerInfo
DCL SOCK _______/* flag: PartnerInfo
                                                                                                                                                                        */
                                                                                                                                                                        */
DCL SOCK FIXED BIN(15);
DCL SOCKET CHAR(16) INIT('SOCKET');
DCL SOCK_DATAGRAM FIXED BIN(15);
DCL SOCK_RAW FIXED BIN(15);
                                                                                               /* socket descriptor
                                                                                               /* socket descriptor datagram */
/* socket descriptor raw */
 DCL SOCK_STREAM FIXED BIN(15);
                                                                                              /* stream socket descriptor
```

DCL SOCK_STREAM_1 FIXED BIN(15); /* stream socket descriptor */ DCL SO_BROADCAST FIXED BIN(31) INIT(32); /* toggle, broadcast msg */ DCL SO_ERROR FIXED BIN(31) INIT(4103); /* check/clear async error */ DCL SO_KEEPALIVE FIXED BIN(31) INIT(8); /* request status of stream*/ DCL SO_LINGER FIXED BIN(31) INIT(128); /* toggle, linger on close */ DCL SO_OOBINLINE FIXED BIN(31) INIT(256); /* toggle, out-of-bound data*/ DCL SO_RCVTIMEO BIT(32) INIT('00001006'BX); DCL SO_REUSEADDR FIXED BIN(31) INIT(4): /* toggle_local address revert/ BIN(31) INIT(4); /* toggle, local address reuse*/ DCL SO_SNDBUF FIXED BIN(31) INIT(4097); DCL SO_SNDTIMEO BIT(32) INIT('00001005'BX); DCL SO_TYPE FIXED BIN(31) INIT(4104); /* return type of socket */ DCL STRING BUILTIN; DCL SUBSTR BUILTIN; DCL SUBTASK CHAR(8) INIT('ANYNAME'); /* task/path identifier DCL SVNC CHAR(16) INIT('SVNC'); DCL TAKESOCKET CHAR(16) INIT('TAKESOCKET'); DCL TASK CHAR(16) INIT('TASK'); DCL TERMAPI CHAR(16) INIT('TERMAPI'); /* */ */ DCL TIME BUILTIN; DCL 1IME DOTATION DCL 1 TIMEOUT, 2 TIME_SEC FIXED BIN(31), /* value in secs 2 TIME_MSEC FIXED BIN(31); /* value in millisecs */ */

 DCL TYPE_VAL,
 2
 TV_SEC BIT(32),
 /* value in secs
 */

 2
 TV_USEC BIT(32);
 /* value in microseconds
 */

 DCL TYPE_DATAGRAM FIXED BIN(31) INIT(2);/*fixed lengthconnectionless*/

 DCL TYPE_RAW
 FIXED BIN(31) INIT(3); /* internal protocol interface */

 DCL TYPE_STREAM FIXED BIN(31) INIT(1); /* two-way byte stream
 */

 DCL TYPE_STREAM FIXED BIN(31) INIT(1); /* two-way byte stream
 */

 DCL WRETMSK CHAR(4); /* indicate WRITE EVENTS */ DCL WRITE CHAR(16) INIT('WRITE'); DCL WRITEV CHAR(16) INIT('WRITEV'); DCL WSNDMSK CHAR(4); /*check for pending write events */ DCL TCP_KEEPALIVE BIT(32) INIT('80000008'BX); DCL TCP_NODELAY BIT(32) INIT('80000001'BX);

Figure 77. CBLOCK PL/1 common variables

Common variables used in COBOL sample programs

The EZACOBOL common storage area contains the variables that are used in the COBOL programs in this section.

```
MODULE NAME: EZACOBOL - COBOL COMMON VARIABLES
* Copyright:
               Licensed Materials - Property of IBM
                "Restricted Materials of IBM"
                5694-A01
                Copyright IBM Corp. 2007, 2010
                US Government Users Restricted Rights -
                Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
* Note:
                COBOL variable names can contain a maximum of
                30 characters.
                CSV1R12
* Status:
* Change Activity:
* Flag Reason Release Date Origin
                                           Description
* $A1= PH34590 HIP6240 210323 tevaller: Identify internal use *
                                           only interfaces with
                                           IFF_RESTRICTED
* COBOL COMMON VARIABLES
* Socket option values.
                                 PIC X(4) VALUE X'00100005'.

PIC X(4) VALUE X'0010000C'.

PIC X(4) VALUE X'0010000A'.

PIC X(4) VALUE X'0010000A'.

PIC X(4) VALUE X'0010000D'.

PIC X(4) VALUE X'00100004'.

PIC X(4) VALUE X'00100003'.
01 IP-ADD-MEMBERSHIP
01 IP-ADD-SOURCE-MEMBERSHIP
01 IP-BLOCK-SOURCE
01 IP-DROP-MEMBERSHIP
01 IP-DROP-SOURCE-MEMBERSHIP
01 IP-MULTICAST-IF
01 IP-MULTICAST-LOOP
01 IP-MULTICAST-TTL
```

PIC X(4) VALUE X'0010000B'. PIC X(4) VALUE X'00010020'. PIC X(4) VALUE X'00010005'. PIC X(4) VALUE X'00010006'. PIC X(4) VALUE X'00010009'. PIC X(4) VALUE X'00010004'. PIC X(4) VALUE X'00010003'. PIC X(4) VALUE X'00010003'. PIC X(4) VALUE X'0010002B'. PIC X(4) VALUE X'00001006'. PIC X(4) VALUE X'00001006'. 01 IP-UNBLOCK-SOURCE 01 IPV6-ADDR-PREFERENCES 01 IPV6-JOIN-GROUP 01 IPV6-LEAVE-GROUP 01 IPV6-MULTICAST-HOPS 01 IPV6-MULTICAST-IF 01 IPV6-MULTICAST-LOOP 01 IPV6-UNICAST-HOPS 01 IPV6-V60NLY 01 MCAST-BLOCK-SOURCE 01 MCAST-JOIN-GROUP 01 MCAST-JOIN-SOURCE-GROUP 01 MCAST-LEAVE-GROUP 01 MCAST-LEAVE-SOURCE-GROUP 01 MCAST-LEAVE-SOURCE-GROUP 01 MCAST-UNBLOCK-SOURCE 01 SO-RCVTIMEO 01 SO-SNDTIMEO * IOCTL Commands 01 SIOCGIFMTU 01 SIOCGIPMSFILTER 01 SIOCSIPMSFILTER PIC X(4) VALUE X'C020A726'. PIC X(4) VALUE X'C000A724'. PIC X(4) VALUE X'8000A725'. PIC X(4) VALUE X'C000F610'. PIC X(4) VALUE X'8000F611'. 01 SIOCGMSFILTER 01 SIOCSMSFILTER 01 SIOCSAPPLDATA PIC X(4) VALUE X'8018D90C' * * Structure allows applications to allocate space for * either form of inet socket address 01 SOCKADDR-STORAGE. 05 SS-LEN 05 SS-FAMILY PIC X(1). PIC X(1). 05 SS-DATA PIC X(126). * IP-MREQ for IP_ADD_MEMBERSHIP and IP_DROP_MEMBERSHIP 01 IP-MREO 05 IMR-MULTIADDR 05 IMR-INTERFACE PIC 9(8) BINARY. PIC 9(8) BINARY. * IP-MREQ-SOURCE for IP_ADD_SOURCE_MEMBERSHIP IP_DROP_SOURCE_MEMBERSHIP IP_BLOCK_SOURCE * IP_UNBLOCK_SOURCE 01 IP-MREQ-SOURCE PIC 9(8) BINARY. PIC 9(8) BINARY. PIC 9(8) BINARY. 05 IMR-MULTIADDR 05 IMR-SOURCEADDR 05 IMR-INTERFACE * IPV6-MREQ for IPV6_JOIN_GROUP and IPV6_LEAVE_GROUP 01 IPV6-MREQ. 05 IPV6MR-MULTIADDR. PIC 9(16) BINARY. PIC 9(16) BINARY. PIC 9(8) BINARY. 10 FILLER 10 FILLER 05 IPV6MR-INTERFACE * GROUP-REQ for MCAST_JOIN_GROUP * MCAST_LEAVE_GROUP 01 GROUP-REQ 05 GR-INTERFACE 05 FILLER PIC 9(8) BINARY. PIC X(4). PIC X(128) 05 GR-GROUP 05 GR-GROUP-R 10 GR-GROUP-SOCK-LEN REDEFINES GR-GROUP. PIC X(1). PIC X(1). 10 GR-GROUP-SOCK-FAMILY

 10
 GR-GROUP-SOCK-DATA
 PIC X(26).

 10
 GR-GROUP-SOCK-SIN
 REDEFINES GR-GRO

 15
 GR-GROUP-SOCK-SIN-PORT
 PIC 9(4)

 15
 GR-GROUP-SOCK-SIN-ADDR
 PIC 9(8)

 BINARY.
 15
 GR-GROUP-SOCK-SIN-ADDR

 REDEFINES GR-GROUP-SOCK-DATA. 15 FILLER PIC X(8) 15 FILLER PIC X(12) 10 GR-GROUP-SOCK-SIN6 REDEFINES GR-GROUP-SOCK-DATA. 15 GR-GROUP-SOCK-SIN6-PORT PIC 9(4) BINARY. 15 GR-GROUP-SOCK-SIN6-FLOWINFO PIC 9(8) BINARY. 15 GR-GROUP-SOCK-SIN6-ADDR. 20 FILLER PIC 9(16) BINARY. 20 FILLER PIC 9(16) BINARY. 15 GR-GROUP-SOCK-SING-SCOPEID PIC 9(8) BINARY. 10 FILLER PIC X(100). GROUP-SOURCE-REQ for MCAST_BLOCK_SOURCE MCAST_UNBLOCK_SOURCE MCAST_UNBLOCK_SOURCE MCAST_JOIN_SOURCE_GROUP MCAST_LEAVE_SOURCE_GROUP * * 01 GROUP-SOURCE-REQ.

```
05 GSR-INTERFACE
                                                PIC 9(8) BINARY.
                                                PIC X(4).
PIC X(128)
    05 FILLER
    05 GSR-GROUP
05 GSR-GROUP-R
                                                 REDEFINES GSR-GROUP.
                                                PIC X(1).
PIC X(1).
PIC X(26)
       10 GSR-GROUP-SOCK-LEN
10 GSR-GROUP-SOCK-FAMILY
       10 GSR-GROUP-SOCK-DATA
          15 GSR-GROUP-SOCK-SIN REDEFINES GSR-GROUP-SOCK-DATA.
15 GSR-GROUP-SOCK-SIN-PORT PIC 9(4) BINARY.
15 GSR-GROUP-SOCK-SIN-ADDR PIC 9(8) BINARY.
       10 GSR-GROUP-SOCK-SIN
                                                PIC X(8).
PIC X(12)
          15 FILLER
          15 FILLER
          15 GSR-GROUP-SOCK-SIN6 REDEFINES GSR-GROUP-SOCK-DATA.
15 GSR-GROUP-SOCK-SIN6-PORT PIC 9(4) BINARY.
15 GSR-GROUP-SOCK-SIN6-FLOWINFO PIC 9(8) BINARY.
       10 GSR-GROUP-SOCK-SIN6
          15 GSR-GROUP-SOCK-SIN6-ADDR
                                                PIC 9(16) BINARY.
PIC 9(16) BINARY.
             20 FILLER
             20 FILLER
          15 GSR-GROUP-SOCK-SIN6-SCOPEID PIC 9(8) BINARY.
                                                PIC X(100).
PIC X(128).
       10 FILLER
   05 GSR-SOURCE
05 GSR-SOURCE-R
10 GSR-SOURCE-SOCK-LEN
                                                REDEFINES GSR-SOURCE.
PIC X(1).
       10 GSR-SOURCE-SOCK-FAMILY
                                                 PIC X(1)
       10 GSR-SOURCE-SOCK-DATA
                                                PIC X(26)
       10 GSR-SOURCE-SOCK-SIN REDEFINES GSR-SOURCE-SOCK-DATA.
15 GSR-SOURCE-SOCK-SIN-PORT PIC 9(4) BINARY.
15 GSR-SOURCE-SOCK-SIN-ADDR PIC 9(8) BINARY.
                                                PIC X(8).
PIC X(12)
          15 FILLER
          15 FILLER
     10 GSR-SOURCE-SOCK-SIN6 REDEFINES GSR-SOURCE-SOCK-DATA.
15 GSR-SOURCE-SOCK-SIN6-PORT PIC 9(4) BINARY.
15 GSR-SOURCE-SOCK-SIN6-FLOWINFO PIC 9(8) BINARY.
         15 GSR-SOURCE-SOCK-SIN6-ADDR.
                                                PIC 9(16) BINARY.
           20 FILLER
        20 FILLER PIC 9(16) BINARY.
15 GSR-SOURCE-SOCK-SIN6-SCOPEID PIC 9(8) BINARY.
     10 FILLER
                                                PIC X(100).
  MULTICAST CONSTANTS
 77 MCAST-INCLUDE
                                                PIC 9(8) BINARY VALUE 0.
PIC 9(8) BINARY VALUE 1.
 77 MCAST-EXCLUDE
                                                 PIC 9(8) BINARY VALUE 64.
 77 MCAST-NUMSRC-MAX
* IP-MSFILTER
 01 IP-MSFILTER
   02 IMSF-HEADER
      03 IMSF-MULTIADDR
03 IMSF-INTERFACE
03 IMSF-FMODE
                                                PIC 9(8) BINARY.
PIC 9(8) BINARY.
                                                 PIC 9(8) BINARY.
          88 IMSF-FMODE-INCLUDE
                                                VALUE 0.
          88 IMSF-FMODE-EXCLUDE
                                                 VALUE 1
       03 IMSF-NUMSRC
                                                 PIC 9(8) BINARY.
    02 IMSF-SLIST
                                                OCCURS 1 TO 64 TIMES
DEPENDING ON IMSF-NUMSRC.
       03 IMSF-SRCENTRY
          05 IMSF-SRCADDR
                                                PIC 9(8) BINARY.
  GROUP-FILTER
*
 01 GROUP-FILTER.
   02 GF-HEADER.
03 GF-INTERFACE
                                                PIC 9(8) BINARY.
                                                PIC X(4).
PIC X(128)
       03 FILLER
       03 GF-GROUP
03 GF-GROUP-R
                                                 REDEFINES GF-GROUP.
          05 GF-GROUP-SOCK-LEN
05 GF-GROUP-SOCK-FAMILY
                                                PIC X(1).
PIC X(1).
          05 GF-GROUP-SOCK-DATA
                                                 PIC X(26)
          05 GF-GROUP-SOCK-SIN
                                                REDEFINES GF-GROUP-SOCK-DATA.
             10 GF-GROUP-SOCK-SIN-PORT PIC 9(4) BINARY.
10 GF-GROUP-SOCK-SIN-ADDR PIC 9(8) BINARY.
10 FILLER PIC X(8).
                                                PIC X(8).
PIC X(12)
             10 FILLER
          05 GF-GROUP-SOCK-SIN6
                                                REDEFINES GF-GROUP-SOCK-DATA.
             10 GF-GROUP-SOCK-SIN6-PORT PIC 9(4) BINARY.
10 GF-GROUP-SOCK-SIN6-FLOWINFO PIC 9(8) BINARY.
             10 GF-GROUP-SOCK-SIN6-ADDR
                15 FILLER
                                                PIC 9(16) BINARY.
                15 FILLER
                                                 PIC 9(16) BINARY.
             10 GF-GROUP-SOCK-SIN6-SCOPEID PIC 9(8) BINARY.
5 FILLER PIC X(100).
6 FFMODE PIC 9(8) BINARY.
       05 FILLER
03 GF-FMODE
          88 GF-FMODE-INCLUDE
88 GF-FMODE-EXCLUDE
                                                VALUE 0.
VALUE 1.
   03 GF-NUMSRC
02 GF-SLIST.
                                                 PIC 9(8) BINARY.
                                                OCCURS 1 TO 64 TIMES
       03 GF-SRCENTRY
                                                 DEPENDING ON GF-NUMSRC.
          05 GF-SRCADDR
                                                 PIC X(128)
          05 GF-SRCADDR-R
                                                 REDEFINES GF-SRCADDR.
```

10 GF-SLIST-SOCK-LEN PIC X(1)10 GF-SLIST-SOCK-FAMILY PIC X(1). 10 GF-SLIST-SOCK-DATA PIC X(26). 10 GF-SLIST-SOCK-SIN REDEFINES 9 GF-SLIST-SOCK-SIN REDEFINES GF-SLIST-SOCK-DATA. 15 GF-SLIST-SOCK-SIN-PORT PIC 9(4) BINARY. 15 GF-SLIST-SOCK-SIN-ADDR PIC 9(8) BINARY. PIC X(8) 15 FILLER GF-SLIST-SOCK-SIN6 REDEFINES GF-SLIST-SOCK-DATA. 15 GF-SLIST-SOCK-SIN6-PORT PIC 9(4) BINARY. 15 GF-SLIST-SOCK-SIN6-FLOWINFO PIC 9(8) BINARY. 15 GF-SLIST-SOCK-SIN6-ADDR. 20 FLUER 10 GF-SLIST-SOCK-SIN6 PIC 9(16) BINARY. PIC 9(16) BINARY. 20 FILLER 20 FILLER 15 GF-SLIST-SOCK-SIN6-SCOPEID PIC 9(8) BINARY. 10 FILLER PIC X(100). Structure for setting APPLDATA when using the SIOCSAPPLDATA * * ioctl. 77 SETADEYE1 PIC X(8) VALUE 'SETAPPLD'. PIC 9(4) BINARY VALUE 1. 77 SETADVER 01 SETAPPLDATA 02 SETAD-EYE1 PIC X(8). PIC 9(4) BINARY. PIC 9(4) BINARY. PIC 9(4) BINARY. PIC X(4). PIC 9(16) BINARY. 02 SETAD-VER 02 SETAD-LEN 02 FILLER 02 SETAD-PTR64 02 SETAD-PTR31 REDEFINES SETAD-PTR64. 03 SETAD-PTRHW PIC 9(8) BINARY 03 SETAD-PTR USAGE IS POINTER. * Structure for containing the actual application data being set * by the SIOCSAPPLDATA ioctl. * 77 SETADEYE2 PIC X(8) VALUE 'APPLDATA'. 01 SETADCONTAINER. PIC X(8). PIC X(40). 02 SETAD-EYE2 02 SETAD-BUFFER * TIMEVAL for SO_RCVTIMEO and SO_SNDTIMEO 01 TIMEVAL 02 TV-SEC PIC 9(8) BINARY. 02 TV-USEC PIC 9(8) BINARY. * IFREQ structure for SIOCGIFxxxx ioctls. 01 IFREQ. 05 IFR-NAME PIC X(16). 05 IFR-IFR PIC X(16) 05 IFR-ADDR REDEFINES IFR-IFR. 10 IFR-ADDR-LEN 10 IFR-ADDR-FAMILY PIC X(1). PIC X(1). 10 IFR-ADDR-PORT PIC 9(4) BINARY. PIC 9(8) BINARY. PIC X(8). 10 IFR-ADDR-ADDR 10 FILLER 05 IFR-DSTADDR 10 IFR-DSTADDR-LEN 10 IFR-DSTADDR-FAMILY REDEFINES IFR-IFR. PIC X(1) PIC X(1) PIC 9(4) BINARY. 10 IFR-DSTADDR-PORT PIC 9(8) BINARY. PIC X(8). 10 IFR-DSTADDR-ADDR 10 FILLER 05 IFR-BROADADDR REDEFINES IFR-IFR. PIC X(1). PIC X(1). PIC 9(4) BINARY. PIC 9(8) BINARY. PIC X(8). 10 IFR-BROADADDR-LEN 10 IFR-BROADADDR-FAMILY 10 IFR-BROADADDR-PORT 10 IFR-BROADADDR-ADDR 10 FILLER 05 IFR-FLAGS-R REDEFINES IFR-IFR. PIC X(2). PIC X(14)10 IFR-FLAGS 10 FILLER REDEFINES IFR-IFR. PIC 9(8) BINARY. PIC X(12). 05 IFR-METRIC-R 10 IFR-METRIC 10 FILLER 05 IFR-DATA-R REDEFINES IFR-IFR. PIC 9(8) BINARY. PIC X(12). 10 IFR-DATA 10 FILLER REDEFINES IFR-IFR. PIC 9(8) BINARY. 05 TER-MTU-R 10 IFR-MTU PIC X(12). 10 FILLER * Constants for use with the IFR_FLAGS field in structure IFREQ. * PIC X(2) VALUE X'0001' PIC X(2) VALUE X'0002' PIC X(2) VALUE X'0004' PIC X(2) VALUE X'0004' PIC X(2) VALUE X'0008' PIC X(2) VALUE X'0010' PIC X(2) VALUE X'0020' PIC X(2) VALUE X'0020' 01 IFF-UP 01 IFF-BROADCAST 01 IFF-DEBUG 01 IFF-RESTRICTED 01 IFF-LOOPBACK 01 IFF-POINTOPOINT 01 IFF-NOTRAILERS 01 IFF-RUNNING PIC X(2) VALUE X'0040'.

01 IFF-NOARP 01 IFF-PROMISC 01 IFF-ALLMULTI 01 IFF-MULTICAST 01 IFF-POINTOMULTIPT 01 IFF-BRIDGE 01 IFF-SNAP 01 IFF-VIRTUAL 01 IFF-SAMEHOST	PIC X(2) VALUE X'0080'. PIC X(2) VALUE X'0100'. PIC X(2) VALUE X'0200'. PIC X(2) VALUE X'0400'. PIC X(2) VALUE X'0800'. PIC X(2) VALUE X'1000'. PIC X(2) VALUE X'2000'. PIC X(2) VALUE X'4000'. PIC X(2) VALUE X'8000'.
* * HOSTENT structure	
 * 01 HOSTENT. * 0fficial name of host 03 H-NAME * Alias list address 03 H-ALIASES * Host address type 03 H-ADDRTYPE * Length of address 03 H-LENGTH * List of address from name s 03 H-ADDR-LIST 	PIC S9(8) BINARY. PIC S9(8) BINARY. PIC S9(8) BINARY. PIC S9(8) BINARY. erver PIC S9(8) BINARY.
<pre>* * Address information structure *</pre>	
<pre>01 ADDRINF0. * Flags 03 AI-FLAGS * Socket family 03 AI-FAMILY * Socket type 03 AI-SOCKTYPE * Protocol 03 AI-PROTOCOL * Length of AI-ADDR value 03 AI-ADDRLEN * Pad to double word boundary 03 FILLER * Canonical name 03 AI-CANONNAME 03 FILLER * Binary address, sockaddr_in(6 03 AI-ADDR 03 FILLER * Next addrinfo structure 03 AI-NEXT * Extended flags 03 AI-FEI AGS</pre>	PIC S9(8) BINARY. PIC S9(8) BINARY. PIC S9(8) BINARY. PIC S9(8) BINARY. PIC S9(8) BINARY. PIC X(4). PIC X(4). PIC S9(8) BINARY. PIC S9(8) BINARY. PIC S9(8) BINARY. PIC S9(8) BINARY. PIC S9(8) BINARY.
* * AI-FLAGS mappings	
* 77 AI-PASSIVE 77 AI-CANONNAMEOK 77 AI-CANONNAMEOK 77 AI-CANONNAMEOK-BIT 77 AI-NUMERICHOST-BIT 77 AI-NUMERICSERV 77 AI-NUMERICSERV 77 AI-NUMERICSERV-BIT 77 AI-VAMAPPED 77 AI-VAMAPPED-BIT 77 AI-ALL 77 AI-ALL-BIT 77 AI-ALL-BIT 77 AI-ADDRCONFIG 77 AI-ADDRCONFIG 77 AI-EXTFLAGS 77 AI-EXTFLAGS-BIT 77 AI-ALLFLAGMASK 77 AI-ALLFLAGMASK-BITS *	PIC X(4) VALUE X'00000001'. PIC S9(8) BINARY VALUE 1. PIC X(4) VALUE X'00000002'. PIC S9(8) BINARY VALUE 2. PIC X(4) VALUE X'00000004'. PIC S9(8) BINARY VALUE 4. PIC X(4) VALUE X'00000008'. PIC X(4) VALUE X'00000001'. PIC S9(8) BINARY VALUE 16. PIC X(4) VALUE X'00000020'. PIC S9(8) BINARY VALUE 32. PIC X(4) VALUE X'00000040'. PIC S9(8) BINARY VALUE 32. PIC X(4) VALUE X'00000040'. PIC S9(8) BINARY VALUE 32. PIC X(4) VALUE X'00000080'. PIC S9(8) BINARY VALUE 128. PIC X(4) VALUE X'FFFFFF00'. PIC S9(8) VALUE -256.
 * AI-EFLAGS mappings * Also maps OPTVAL for getsocko * OPTNAME is IPV6-ADDR-PREFEREN * Also maps FLAGS for inet6_is_ 	pt and setsockopt when CES srcaddr
77 IPV6-PREFER-SRC-HOME 77 IPV6-PREFER-SRC-COA 77 IPV6-PREFER-SRC-TMP 77 IPV6-PREFER-SRC-PUBLIC 77 IPV6-PREFER-SRC-CGA 77 IPV6-PREFER-SRC-NONCGA 77 IPV6-PREFER-SRC-INVALIDBITS *	PIC S9(8) BINARY VALUE 1. PIC S9(8) BINARY VALUE 2. PIC S9(8) BINARY VALUE 4. PIC S9(8) BINARY VALUE 8. PIC S9(8) BINARY VALUE 16. PIC S9(8) BINARY VALUE 32. PIC S9(8) BINARY VALUE -64.
* NI_FLAGS mappings * 77 NI-NOFQDN 77 NI-NUMERICHOST 77 NI-NAMEREQD 77 NI-NUMERICSERV 77 NI-DGRAM 77 NI-NUMERICSCOPE	PIC X(4) VALUE X'00000001'. PIC X(4) VALUE X'00000002'. PIC X(4) VALUE X'00000008'. PIC X(4) VALUE X'00000008'. PIC X(4) VALUE X'00000010'. PIC X(4) VALUE X'00000010'.

Figure 78. EZACOBOL COBOL common variables

COBOL call interface sample IPv6 server program

The EZASO6CS program is a server program that shows you how to use the following calls provided by the call socket interface:

- ACCEPT
- BIND
- CLOSE
- EZACIC09
- FREEADDRINFO
- GETADDRINFO
- GETCLIENTID
- GETHOSTNAME
- INITAPI
- LISTEN
- NTOP
- PTON
- READ
- SOCKET
- TERMAPI
- WRITE

```
MODULE NAME: EZASO6CS - THIS IS A VERY SIMPLE IPV6 SERVER
* Copyright: Licensed Materials - Property of IBM
              "Restricted Materials of IBM"
              5694-A01
              Copyright IBM Corp. 2002, 2008
              US Government Users Restricted Rights -
Use, duplication or disclosure restricted by
GSA ADP Schedule Contract with IBM Corp.
*
             COBOL variable names can contain a maximum of
* Note:
              30 characters.
* Status:
             CSV1R10
   LANGUAGE: COBOL
Identification Division.
Program-id. EZASO6CS.
*=========================
Environment Division.
*===============================
*===============
Data Division.
*================
```

Working-storage Section.

* * So	cket interface function (odes	
*			
01	soket-functions. 02 soket-accept	nic x(16)) value 'ACCEPT '
	02 soket-bind	pic x(16) value 'BIND '
	02 soket-close	pic x(16) value 'CLOSE
	02 soket-connect	pic $x(16)$) value 'CONNECT '
	02 soket-freeaddrinfo	pic $x(16)$) value 'FREEADDRINFO '
	02 soket-getaddrinfo	pic x(16) value 'GETADDRINFO '
	02 soket-getclientid	pic x(16) value 'GETCLIENTID '
	02 soket-gethostbyaddr	pic $x(16)$) value 'GETHOSTBYADDR '
	02 soket-gethostid	pic $x(10)$) value 'GETHOSTID
	02 soket-gethostname	pic x(16) value 'GETHOSTNAME '
	02 soket-getnameinfo	pic $x(16)$) value 'GETNAMEINFO '
	02 soket-getsockname	pic $x(10)$) value 'GETSOCKNAME '
	02 soket-getsockopt	pic x(16) value 'GETSOCKOPT
	02 soket-givesocket	pic $x(16)$) value 'GIVESOCKET '
	02 soket-ioctl	pic $x(10)$) value 'IOCTL '
	02 soket-listen	pic x(16) value 'LISTEN '
	02 soket-ntop	pic x(16) value 'NTOP '
	02 Soket-read	pic $x(16)$) value 'READ '
	02 soket-recv	pic x(16) value 'RECV '
	02 soket-recvfrom	pic x(16) value 'RECVFROM '
	02 SOKET-SELECT 02 soket-send	pic $x(16)$) VALUE 'SELECI '
	02 soket-sendto	pic x(16) value 'SENDTO '
	02 soket-setsockopt	pic x(16) value 'SETSOCKOPT '
	02 soket-shutdown 02 soket-socket	pic $x(16)$) value 'SHUIDOWN '
	02 soket-takesocket	pic $x(10)$) value 'TAKESOCKET '
	02 soket-termapi	pic x(16) value 'TERMAPI
+	02 soket-write	pic x(16) value 'WRITE '
* Wo	rk variables		
* ∩1		ni	c Q(8) bipary value zero
01	retcode	pi	c s9(8) binary value zero.
01	client-ipaddr-dotted	pi	c x(15) value space.
01	server-ipaddr-dotted	pi	c x(15) value space.
01	ezaconn-Iunction 88 CONNECTED	рі	c x value space.
01	saved-message-id	pi	c x(8) value space.
04	88 close-down-message-:	eceived	value '*CLSDWN*'.
θT	88 Opened-APT	pr	value 'A'.
	88 Opened-Socket		value 'S'.
01	saved-message-id-len	pi	c 9(8) Binary value 8.
ΘT	O2 Hour	ni	c 9(2)
	02 Minute	pi	c 9(2).
	02 Second	pi	c 9(2).
Q1	02 Hund-Sec s	pi	c 9(2).
*	5	hT	
* Va	riables used for the INI	API call	
01	maxsoc-fwd	pi	c 9(8) Binary.
01	maxsoc-rdf redefines max	(soc-fwd.	
	02 filler	pi	c x(2).
01	initani-ident	pr	C 9(4) Binary.
01	05 tcpname	pi	c x(8) Value 'TCPCS '.
	05 asname	pi	c x(8) Value space.
01 01	subtask	pi	c X(8) Value 'EZASO6CS'.
*			
* Va	riables returned by the (ETCLIENTI	D Call
01	clientid.		
	05 clientid-domain	pi	c 9(8) Binary value 19.
	05 clientid-name	pi	c x(8) value space.
	05 Cilentid-task 05 filler	pi	c x(8) value space. c x(20) value low-value
*		br	
* Va	riables used for the SOC	(ET call	
*	AF-INET	ni	c 9(8) Binarv Value 2
01	AF-INET6	pi	c 9(8) Binary Value 19.
0T	SOCK-STREAM	pi	c 9(8) Binary Value 1.
01	COOK DATION	ni	
01 01 01	SOCK-DATAGRAM	pr	C 9(8) Binary Value 2.
01 01 01 01	SOCK-DATAGRAM SOCK-RAW IPPROTO-IP	pi pi	c 9(8) Binary Value 2. c 9(8) Binary Value 3. c 9(8) Binary Value zero.
01 01 01 01 01 01	SOCK-DATAGRAM SOCK-RAW IPPROTO-IP IPPROTO-TCP	pi pi pi	c 9(8) Binary Value 2. c 9(8) Binary Value 3. c 9(8) Binary Value zero. c 9(8) Binary Value 6.
01 01 01 01 01 01 01	SOCK-DATAGRAM SOCK-RAW IPPROTO-IP IPPROTO-TCP IPPROTO-UDP IPPROTO-UDP	pi pi pi pi	c 9(8) Binary Value 2. c 9(8) Binary Value 3. c 9(8) Binary Value zero. c 9(8) Binary Value 6. c 9(8) Binary Value 17.
01 01 01 01 01 01 01 01	SOCK-DATAGRAM SOCK-RAW IPPROTO-IP IPPROTO-TCP IPPROTO-UDP IPPROTO-UPV6 Socket-descriptor	pi pi pi pi	c 9(8) Binary Value 2. c 9(8) Binary Value 3. c 9(8) Binary Value zero. c 9(8) Binary Value 6. c 9(8) Binary Value 17. c 9(8) Binary Value 41. c 9(4) Binary Value 2000
* Variables returned by the GETHOSTNAME Call 01 host-name-len pic 9(8) binary. 01 host-name pic x(24). 01 host-name-char-count pic 9(4) binary. 01 host-name-unstrung pic x(24) value spaces. * Variables used/returned by the GETADDRINFO Call pic x(255). pic 9(8) binary. 01 node-name 01 node-name-len 01 service-name pic x(32). pic 9(8) binary. service-name-len 01 canonical-name-len pic 9(8) binary. 01 01 ai-passive pic 9(8) binary value 1. pic 9(8) binary value 2. pic 9(8) binary value 4. pic 9(8) binary value 4. pic 9(8) binary value 8. 01 ai-canonnameok 01 ai-numerichost 01 ai-numericserv pic 9(8) binary value 16. pic 9(8) binary value 32. ai-v4mapped 01 01 ai-all 01 ai-addrconfig pic 9(8) binary value 64. *---* Variables used for the BIND call server-socket-address. 01 pic 9(4) Binary value 19. pic 9(4) Binary value 1031. pic 9(8) Binary value 0. 05 server-family 05 server-port 05 server-flowinfo 05 server-ipaddr. pic 9(16) Binary value 0. pic 9(16) Binary value 0. pic 9(8) Binary value 0. PIC 9(8) COMP value 80. 10 filler 10 filler 05 server-scopeid NBYTE 01 BUF PIC X(80). 01 PIC S9(8) COMP VALUE 10. 01 BACKLOG * Variables used/returned by the EZACIC09 call *usage is pointer. pic 9(8) binary. 01 input-addrinfo-ptr 01 output-name-len pic x(256). 01 output-canonical-name 01 output-name usage is pointer. 01 output-next-addrinfo usage is pointer. * Variables used for the LISTEN call pic 9(4) Binary Value zero. 01 backlog-level *- Variables used for the ACCEPT call pic 9(4) Binary Value zero. 01 socket-descriptor-new * Variables used for the NTOP/PTON call pic x(45) value '::'. 01 IN6ADDR-ANY pic x(45) value '::1'. 01 IN6ADDR-LOOPBACK 01 ntop-family 01 pton-family pic 9(8) Binary. pic 9(8) Binary. pic x(45) value spaces. 01 presentable-addr 01 presentable-addr-len pic 9(4) Binary value 45. 01 numeric-addr. pic 9(16) Binary Value 0. pic 9(16) Binary Value 0. 05 filler 05 filler * Variables used by the RECV Call 01 client-socket-address. pic 9(4) Binary Value 19. pic 9(4) Binary Value 1032. pic 9(8) Binary Value zero. 05 client-family 05 client-port 05 client-flowinfo client-ipaddr. 05 pic 9(16) Binary Value 0. pic 9(16) Binary Value 0. pic 9(8) Binary Value zero. 10 filler 10 filler 05 client-scopeid * Buffer and length field for recv and send operation pic 9(8) Binary Value zero. pic 9(8) Binary Value zero. pic x(4000) value space. 01 send-request-len 01 read-request-len 01 read-buffer filler redefines read-buffer. 01 05 message-id 05 filler pic x(8)pic x(3992). *-* recv and send flags 01 send-flag pic 9(8) Binary value zero. 01 recv-flag pic 9(8) Binary value zero. * Error message for socket interface errors

```
77 failure
                                         pic S9(8) comp.
 01 ezaerror-msg.
                                         pic x(9) Value 'Function='.
pic x(16) Value space.
     05 filler
          ezaerror-function
     05
     05
         filler
                                         pic x value '
                                         pic x(8) Value 'Retcode='.
     05
         filler
                                         pic ---99.
     05
         ezaerror-retcode
                                         pic x value ' '
         filler
     05
                                         pic x(9) Value 'Errorno='.
pic zzz99.
         filler
     05
     05
          ezaerror-errno
                                        pic 22299.
pic x value ' '.
pic x(50) value ' '.
     05
         filler
     05 ezaerror-text
 Linkage Section.
01 11.
     03
          hints-addrinfo.
          05 hints-ai-flags
05 hints-ai-family
                                         pic 9(8) binary.
              hints-ai-family
          05
                                         pic 9(8) binary.
                                         pic 9(8) binary.
pic 9(8) binary.
pic 9(8) binary.
          05
              hints-ai-socktype
             hints-ai-protocol
filler
          05
          05
             filler
                                         pic 9(8) binary.
          05
          05 filler
                                         pic 9(8) binary.
          05 filler
                                         pic 9(8) binary.
     03 hints-addrinfo-ptr
03 results-addrinfo-ptr
                                        usage is pointer.
usage is pointer.
*
 Results address info
 01
     results-addrinfo
                                        pic 9(8) binary.
pic 9(8) binary.
     05 results-ai-flags
     05 results-ai-family
        results-ai-socktype
                                         pic 9(8) binary.
     05
                                        pic 9(8) binary.
pic 9(8) binary.
     05
         results-ai-protocol
     05 results-ai-addr-len
        results-ai-canonical-name usage is pointer.
results-ai-addr-ptr usage is pointer.
     05
     05
     05 results-ai-next-ptr
                                         usage is pointer.
* Socket address structure from EZACIC09.
 01 output-name-ptr
                                         usage is pointer.
     output-ip-name.
 01
     03 output-ip-family
03 output-ip-port
                                         pic 9(4) Binary.
                                         pic 9(4) Binary.
     03 output-ip-sock-data pic 

03 output-ipv4-sock-data redefines

04 output-ip-sock-data.
                                         pic x(24).
          05 output-ipv4-ipaddr
05 filler
                                         pic 9(8) Binary.
                                         pic x(20).
     03 output-ipv6-sock-data redefines
          output-ip-sock-data.
05 output-ipv6-flowinfo
                                        pic 9(8) Binary.
              output-ipv6-ipaddr.
          05
              10 filler
10 filler
                                         pic 9(16) Binary.
pic 9(16) Binary.
                                         pic 9(8) Binary.
          05 output-ipv6-scopeid
               =================================
 Procedure Division using L1.
PROCEDURE CONTROLS
     Perform Initialize-API
Perform Get-ClientID
                                              Initialize-API-Exit.
Get-ClientID-Exit.
                                     thru
                                     thru
     Perform Sockets-Descriptor thru
                                              Sockets-Descriptor-Exit.
     Perform Presentation-To-Numeric thru
                                 Presentation-To-Numeric-Exit.
     Perform Get-Host-Name
Perform Get-Address-Info
                                     thru Get-Host-Name-Exit.
thru Get-Address-Info-Exit.
                                            Bind-Socket-Exit.
Listen-To-Socket-
     Perform Bind-Socket
                                     thru
     Perform Listen-To-Socket
                                     thru
                                              Listen-To-Socket-Exit.
     Perform Accept-Connection
                                     thru
                                              Accept-Connection-Exit.
     Move 45 to presentable-addr-len.
     Move spaces to presentable-addr.
     Move server-ipaddr to numeric-addr.
     Move 19 to ntop-family.
     Perform Numeric-TO-Presentation thru
                                   Numeric-To-Presentation-Exit.
     Perform Read-Message
                                     thru Read-Message-Exit.
                                              Write-Message-Exit.
     Perform Write-Message
                                     thru
     Perform Close-Socket
                                              Exit-Now.
                                     thru
                             * Initialize socket API
 Initialize-API.
     Move soket-initapi to ezaerror-function.
```

```
* If you want to set maxsoc to the max, uncomment the next line.*
       Move 65535 to maxsoc-fwd.
      Call 'EZASOKET' using soket-initapi maxsoc initapi-ident
      subtask maxsno errno retcode.
Move 'Initapi failed' to ezaerror-text.
If retcode < 0 move 12 to failure.
      Move 'A' to Terminate-Options.
 Initialize-API-Exit.
       Exit.
* Let us see the client-id
*-
 Get-ClientID.
      move soket-getclientid to ezaerror-function.
      Call 'EZASOKET' using soket-getclientid clientid errno
      If retcode < 0 move 24 to failure.
      Perform Return-Code-Check thru Return-Code-Exit.
 Get-ClientID-Exit.
       Exit.
* Get us a stream socket descriptor.
 Sockets-Descriptor.
      move soket-socket to ezaerror-function.
Call 'EZASOKET' using soket-socket AF-INET6 SOCK-STREAM
IPPROTO-IP errno retcode.
      Move 'Socket call failed' to ezaerror-text.
If retcode < 0 move 24 to failure.
Perform Return-Code-Check thru Return-Code-Exit.
      Move retcode to socket-descriptor.
Move 'S' to Terminate-Options.
 Sockets-Descriptor-Exit.
      Exit.
* Use PTON to create an IP address to bind to.
 Presentation-To-Numeric.
      move soket-pton to ezaerror-function.
move IN6ADDR-LOOPBACK to presentable-addr.
Call 'EZASOKET' using soket-pton AF-INET6
presentable-addr presentable-addr-len
           numeric-addr
      errno retcode.
Move 'PTON call failed' to ezaerror-text.
If retcode < 0 move 24 to failure.
      Perform Return-Code-Check thru Return-Code-Exit.
      move numeric-addr to server-ipaddr.
 Presentation-To-Numeric-Exit.
      Exit.
* Get the host name.
 Get-Host-Name.
      move soket-gethostname to ezaerror-function.
move 24 to host-name-len.
Call 'EZASOKET' using soket-gethostname
          host-name-len host-name
      errno retcode.
display 'Host name = ' host-name.
Move 'GETHOSTNAME call failed' to ezaerror-text.
      If retcode < 0 move 24 to failure.
      Perform Return-Code-Check thru Return-Code-Exit.
 Get-Host-Name-Exit.
      Exit.
                               * Get address information
                                      Get-Address-Info.
      move soket-getaddrinfo to ezaerror-function. move \boldsymbol{\theta} to host-name-char-count.
      inspect host-name tallying host-name-char-count
for characters before x'00'.
      unstring host-name delimited by x'00'
          into host-name-unstrung
count in host-name-char-count.
      string host-name-unstrung delimited by ' '
          into node-name.
      move host-name-char-count to node-name-len
display 'node-name-len: ' node-name-len.
move spaces to service-name.
move 0 to service-name-len.
```

```
move 0 to hints-ai-family.
move ai-canonnameok to hints-ai-flags
move 0 to hints-ai-socktype.
move 0 to hints-ai-protocol.
         move 0 to hints-ai-protocol.
display 'GETADDRINFO Input fields: '
display 'Node name = ' node-name.
display 'Node name length = ' node-name-len.
display 'Service name length = ' service-name.
display 'Service name length = ' service-name-len.
display 'Hints family = ' hints-ai-family.
display 'Hints flags = ' hints-ai-flags.
display 'Hints socktype = ' hints-ai-socktype.
display 'Hints protocol = ' hints-ai-protocol.
set address of results-addrinfo to results-addrinfo-ptr.
move soket-getaddrinfo to geaerror-function.
          move soket-getaddrinfo to ezaerror-function.
          set hints-addrinfo-ptr to address of hints-addrinfo.
Call 'EZASOKET' using soket-getaddrinfo
    node-name node-name-len
                service-name service-name-len
                hints-addrinfo-ptr
                results-addrinfo-ptr
               canonical-name-len
errno retcode.
         Move 'GETADDRINFO call failed' to ezaerror-text.
If retcode < 0 move 24 to failure
                  Perform Return-Code-Check thru Return-Code-Exit
          else
                  Perform Return-Code-Check thru Return-Code-Exit
                                   Address of results addrinfo is
                  display
                         results-addrinfo-ptr.
                  set address of results-addrinfo to results-addrinfo-ptr
                 set input-addrinfo-ptr to address of results-addrinfo
display 'Address of input-addrinfo-ptr is '
                  input-addrinfo-ptr.
Perform Format-Result-AI thru Format-Result-AI-Exit
                  Perform Set-Next-Addrinfo thru
Set-Next-Addrinfo-Exit until
                 output-next-addrinfo is equal to NULLS
Perform Free-Address-Info thru Free-Address-Info-Exit.
 Get-Address-Info-Exit.
          Exit.
                                              * Set next addrinfo address
 Set-Next-Addrinfo.
          display 'Setting next addrinfo address as '
         results-ai-next-ptr.
display 'Address of output-next-addrinfo as '
output-next-addrinfo.
set address of results-addrinfo to output-next-addrinfo.
          set input-addrinfo-ptr to address of results-addrinfo.
         display 'Address of input-addrinfo-ptr is '
input-addrinfo-ptr.
Perform Format-Result-AI thru Format-Result-AI-Exit.
 Set-Next-Addrinfo-Exit.
          Exit.
*-----
* Format result address information
  Format-Result-AI
         move 'EZACIC09' to ezaerror-function.
         move zeros to output-name-len.
move spaces to output-canonical-name.
set output-name to nulls.
         Set output-next-addrinfo to nulls.
Call 'EZACIC09' using input-addrinfo-ptr
                       output-name-len
output-canonical-name
                       output-name
                       output-next-addrinfo
                        retcode.
         retcode.

Move 'EZACIC09 call failed' to ezaerror-text.

display 'EZACIC09 output:'

display 'canonical name = ' output-canonical-name.

display 'name length = ' output-name-len.

display 'name = ' output-name.

display 'next addrinfo = ' output-next-addrinfo.

If retcode < 0 move 24 to failure.

Perform Return-Code-Check thru Return-Code-Exit.

display 'Formatting result address in address'.

set address of output-in-name to output-name.
         display formatting result address ip address
set address of output-ip-name to output-name.
move results-ai-family to ntop-family.
display 'ntop-family = ' ntop-family.
if ntop-family = AF-INET then
    display 'Formatting ipv4 addres'
    move output-ipv4-ipaddr to numeric-addr
                  move output-ipv4-ipaddr to numeric-addr
                  move 16 to presentable-addr-len
          else
                 display 'Formatting ipv6 addres'
move output-ipv6-ipaddr to numeric-addr
                  move 45 to presentable-addr-len.
```

```
move spaces to presentable-addr.
       Perform Numeric-To-Presentation thru
                                                Numeric-To-Presentation-Exit.
 Format-Result-AI-Exit.
       Exit.
                                         * Release resolver storage
 Free-Address-Info.
       move soket-freeaddrinfo to ezaerror-function.
Call 'EZASOKET' using soket-freeaddrinfo
results-addrinfo-ptr
            errno retcode.
       Move 'FREEADDRINFO call failed' to ezaerror-text.
       If retcode < 0 move 24 to failure.
       Perform Return-Code-Check thru Return-Code-Exit.
 Free-Address-Info-Exit.
       Exit.
* Bind socket to our server port number
 Bind-Socket.
       Move soket-bind to ezaerror-function.
Call 'EZASOKET' using soket-bind socket-descriptor
       server-socket-address errno retcode.
Display 'Port = ' server-port
' Address = ' presentable-addr.
       Note 'Bind call failed' to exaerror-text
If retcode < 0 move 24 to failure.
Perform Return-Code-Check thru Return-Code-Exit.
 Bind-Socket-Exit.
        Exit.
  Listen to the socket
 Listen-To-Socket.
       Move soket-listen to ezaerror-function.
Call 'EZASOKET' using soket-listen socket-descriptor
       backlog errno retcode.
Display 'Backlog=' backlog.
Move 'Listen call failed' to ezaerror-text.
If retcode < 0 move 24 to failure.
        Perform Return-Code-Check thru Return-Code-Exit.
 Listen-To-Socket-Exit.
        Exit.
  Accept a connection request
 Accept-Connection.
       Nove soket-accept to ezaerror-function.
Call 'EZASOKET' using soket-accept socket-descriptor
server-socket-address errno retcode.
       Move retcode to socket-descriptor-new.
Display 'New socket=' retcode.
Move 'Accept call failed' to ezaerror-text .
If retcode < 0 move 24 to failure.
Perform Return-Code-Check thru Return-Code-Exit.
 Accept-Connection-Exit.
         Exit
* Use NTOP to display the IP address.
 Numeric-To-Presentation.
       move soket-ntop to ezaerror-function.
Call 'EZASOKET' using soket-ntop ntop-family
            numeric-addr
            presentable-addr presentable-addr-len
       error retcode.
Display 'Presentable address = ' presentable-addr.
Move 'NTOP call failed' to ezaerror-text.
If retcode < 0 move 24 to failure.
Perform Return-Code-Check thru Return-Code-Exit.
 Numeric-TO-Presentation-Exit.
       Exit.
* Read a message from the client.
 Read-Message.
       move soket-read to ezaerror-function.
       move spaces to buf.
display 'New socket desciptor = ' socket-descriptor-new.
Call 'EZASOKET' using soket-read socket-descriptor-new
           nbyte buf
       errno retcode.
display 'Message received = ' buf.
Move 'Read call failed' to ezaerror-text.
         If retcode < 0 move 24 to failure.
```

```
Perform Return-Code-Check thru Return-Code-Exit.
 Read-Message-Exit.
      Exit.
*-----
                                  * Write a message to the client.
 Write-Message.
     move soket-write to ezaerror-function.
move 'Message from EZAS06SC' to buf.
Call 'EZAS0KET' using soket-write socket-descriptor-new
         nbyte buf
errno retcode.
      Nove 'Write call failed' to ezaerror-text
If retcode < 0 move 24 to failure.
      Perform Return-Code-Check thru Return-Code-Exit.
 Write-Message-Exit.
     Exit.
*-----
* Close connected socket
 Close-Socket.
     move soket-close to ezaerror-function
Call 'EZASOKET' using soket-close socket-descriptor-new
     Accept cur-time from time.
Display cur-time ' EZASO6CS : CLOSE RETCODE=' RETCODE
' ERRNO= ' ERRNO.
If retcode < 0 move 24 to 5 to 5
      If retcode < 0 move 24 to failure
move 'Close call Failed' to ezaerror-text
perform write-ezaerror-msg thru write-ezaerror-msg-exit.
 Close-Socket-Exit.
     Exit.
                          4-
* Terminate socket API
 exit-term-api
      Call 'EZASOKET' using soket-termapi.
        -----
* Terminate program
 exit-now.
      move failure to return-code.
      Goback.
            * Subroutine
+ -
* Write out an error message
                                 write-ezaerror-msg.
move errno to ezaerror-errno.
move retcode to ezaerror-retcode.
      display ezaerror-msg.
 write-ezaerror-msg-exit.
      exit.
* Check Return Code after each Socket Call
                                                 Return-Code-Check
       Accept Cur-Time from TIME.
       move errno to ezaerror-errno.
       move retcode to ezaerror-retcode.
       Display Cur-Time ' EZASO6CS: ' ezaerror-function
' RETCODE= ' ezaerror-retcode
' ERRNO= ' ezaerror-errno.
       IF RETCODE < 0
          Perform Write-ezaerror-msg thru write-ezaerror-msg-exit
Move zeros to errno retcode
IF Opened-Socket Go to Close-Socket
ELSE IF Opened-API Go to exit-term-api
                ELSE Go to exit-now.
       Move zeros to errno retcode.
 Return-Code-Exit.
      Exit.
```

Figure 79. EZASO6CS COBOL call interface sample IPv6 server program

COBOL call interface sample IPv6 client program

The EZASO6CC program is a client module that shows you how to use the following calls provided by the call socket interface:

- CLOSE
- CONNECT
- GETCLIENTID
- GETNAMEINFO
- INITAPI
- NTOP
- PTON
- READ
- SHUTDOWN
- SOCKET
- TERMAPI
- WRITE

```
MODULE NAME: EZASO6CC - THIS IS A VERY SIMPLE IPV6 CLIENT *
* Copyright: Licensed Materials - Property of IBM
            "Restricted Materials of IBM"
            5694-A01
            Copyright IBM Corp. 2002, 2008
            US Government Users Restricted Rights -
            Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
* Note:
            COBOL variable names can contain a maximum of
            30 characters.
* Status:
            CSV1R10
  LANGUAGE: COBOL
Identification Division.
*=============================
Program-id. EZASO6CC.
*===============================
Environment Division.
*===============================
*===============
Data Division.
*================
Working-storage Section.
                     -----
* Socket interface function codes
```

02 soket-getnameinfo pic x(16) value 'GETNAMEINFO pic x(16) value 'GETPEERNAME pic x(16) value 'GETPEERNAME pic x(16) value 'GETSOCKNAME pic x(16) value 'GETSOCKOPT pic x(16) value 'GIVESOCKET pic x(16) value 'INITAPI pic x(16) value 'TNITAPI 02 soket-getpeername 02 soket-getsockname 02 soket-getsockopt 02 soket-givesocket 02 soket-initapi value 'IOCTL value 'LISTEN 02 soket-ioctl pic x(16) 02 soket-listen pic x(16) 02 soket-ntop pic x(16) value 'NTOP value 'PTON 02 soket-pton pic x(16) 02 soket-read pic x(16) value 'READ 02 soket-recv 02 soket-recvfrom pic x(16) value 'RECV 'RECVFROM pic x(16) value 02 soket-select pic x(16) 'SELECT value pic x(16) value 'SEND pic x(16) value 'SENDTO 02 soket-send 02 soket-sendto pic x(16) value 'SETSOCKOPT pic x(16) value 'SHUTDOWN 02 soket-setsockopt 02 soket-shutdown 02 soket-socket 02 soket-snuluown 02 soket-socket 02 soket-takesocket 02 soket-termapi pic x(16) value 'SOCKET pic x(16) value 'TAKESOCKET pic x(16) value 'TERMAPI pic x(16) value 'TERMAPI pic x(16) value 'WRITE 02 soket-write * Work variables 01 errno pic 9(8) binary value zero. pic s9(8) binary value zero. pic 9(8) binary value zero. 01 retcode 01 index-counter 01 buffer-element. 05 buffer-element-nbr 05 filler pic 9(5). pic x(3) value space. pic x(15) value space. pic x(15) value space. pic x(15) value space. pic 9(8) Binary value zero. 05 filler server-ipaddr-dotted client-ipaddr-dotted 01 01 01 close-server 88 close-server-down value 1. Connect-Flag 88 CONNECTED pic x value space. value 'Y'. 01 pic x value space. value 'C'. value 'S'. Client-Server-Flag 01 88 CLIENTS 88 SERVERS Terminate-Options 88 Opened-API pic x value space. value 'A'. value 'S'. 01 88 Opened-Socket pic 9(8) Binary value zero. pic 9(8) Binary value 2000. 01 timer-accum 01 timer-interval 01 Cur-time. 02 Hour 02 Minute pic 9(2). pic 9(2). pic 9(2). 02 Second 02 Hund-Sec pic 9(2) 77 Failure Pic S9(8) comp. * Variables used for the INITAPI call *-----01 maxsoc-fwd pic 9(8) Binary. maxsoc-rdf redefines maxsoc-fwd. 01 pic x(2). 02 filler pic 9(4) Binary. 02 maxsoc 01 initapi-ident. pic x(8) Value 'TCPCS '.
pic x(8) Value space.
pic x(8) value 'EZSO6CC'. 05 tcpname 05 asname 01 subtask 01 maxsno pic 9(8) Binary Value 1. * Variables used by the SHUTDOWN Call 01 how pic 9(8) Binary. * Variables returned by the GETCLIENTID Call clientid. 05 clientid-domain pic 9(8) Binary value 19. 05 clientid-name pic x(8) value space. 05 clientid-task pic x(8) value space. 05 filler pic x(20) value low-value. 01 clientid. * Variables returned by the GETNAMEINFO Call 01 name-len pic 9(8) binary. pic x(255). pic 9(8) binary. 01 host-name 01 host-name-len 01 service-name pic x(32) pic X(32). pic 9(8) binary. pic 9(8) binary value 0. pic 9(8) binary value 1. pic 9(8) binary value 2. pic 9(8) binary value 4. pic 9(8) binary value 8. service-name-len 01 01 name-info-flags 01 ni-nofqdn ni-numerichost 01 ni-namereqd 01 01 ni-numericserver 01 ni-dgram pic 9(8) binary value 16. *--* Variables used for the SOCKET call _____ 01 AF-INET pic 9(8) Binary Value 2.

```
01 AF-INET6
                                         pic 9(8) Binary Value 19.
                                        pic 9(8) Binary Value 1.
pic 9(8) Binary Value 2.
pic 9(8) Binary Value 3.
 01
     SOCK-STREAM
     SOCK-DATAGRAM
 01
 01
     SOCK-RAW
                                        pic 9(8) Binary Value J.
pic 9(8) Binary Value Zero.
pic 9(8) Binary Value 6.
pic 9(8) Binary Value 17.
pic 9(8) Binary Value 41.
     IPPROTO-IP
 01
     IPPROTO-TCP
 01
 01 IPPROTO-UDP
 01
     IPPROTO-IPV6
                                        pic 9(4) Binary Value zero.
 01
     socket-descriptor
                       * Server socket address structure
*-
 01 server-socket-address.
     05 server-afinet
05 server-port
                                        pic 9(4) Binary Value 19.
pic 9(4) Binary Value 1031.
pic 9(8) Binary Value zero.
                            pic 9(16) Binary Value Zero.
pic 9(16) Binary Value 0.
pic 9(16) Binary Value 0.
pic 9(8) Binary Value zero.
PIC 9(8) COMP value 80.
PIC X(80).
     05
        server-flowinfo
     05 server-ipaddr.
10 filler
          10 filler
     05 server-scopeid
 01
     NBYTE
01 BUF
*--
* Variables used by the BIND Call
 01
     client-socket-address.
     05 client-family
05 client-port
05 client-flowinfo
05 client-ipaddr.
                                      pic 9(4) Binary Value 19.
pic 9(4) Binary Value 1032.
pic 9(8) Binary Value 0.
                                        pic 9(16) Binary Value 0.
pic 9(16) Binary Value 0.
pic 9(8) Binary Value 0.
          10 filler
          10 filler
     05 client-scopeid
                                _ _ _ _ _ _ _ _ _
                                                   -----
* Buffer and length fields for send operation
 01 send-request-length
                                      pic 9(8) Binary value zero.
 01
     send-buffer.
     send-buffer.
05 send-buffer-total pic x(4000) value s
05 closedown-message redefines send-buffer-total.
                                        pic x(4000) value space.
          10closedown-idpic x(8)10fillerpic x(39)
                                        pic x(3992)
     05 send-buffer-seq redefines send-buffer-total
         pic x(8) occurs 500 times.
* Variables used for the NTOP/PTON call
                 ----*
 01 IN6ADDR-ANY
                                        pic x(45)
                           value '::'.
                           pic x(45) value '::1'.
 01 IN6ADDR-LOOPBACK
                                        pic x(45) value spaces.
 01 presentable-addr
                                        pic 9(4) Binary value 45.
 01 presentable-addr-len
01 numeric-addr.
05 filler
                                        pic 9(16) Binary Value 0.
                                        pic 9(16) Binary Value 0.
     05 filler
                       . . . . . . . . . . . . . . . .
* Buffer and length fields for recv operation
*---
                      ----*
                                        pic 9(8) Binary value zero.
pic x(4000) value space.
01 read-request-length
 01 read-buffer
                   * Other fields for send and reccfrom operation
*-
                  01 send-flag
                                        pic 9(8) Binary value zero.
 01 recv-flag
                                        pic 9(8) Binary value zero.
* Error message for socket interface errors
     ezaerror-msg.
 01
     05 filler
                                        pic x(9) Value 'Function='.
        ezaerror-function
                                        pic x(16) Value space.
     05
         filler
                                        pic x value '
     05
                                        pic x(8) Value 'Retcode='.
pic ---99.
     05
         filler
         ezaerror-retcode
     05
                                        pic x value ' '.
     05
         filler
                                        pic x(9) Value 'Errorno='.
     05
         filler
                                        pic zzz99.
     05
          ezaerror-errno
                                        pic x value ' '.
     05
          filler
                                        pic x(50) value ' '.
     05
         ezaerror-text
 Linkage Section.
Procedure Division.
==================================
         PROCEDURE CONTROLS
                                                           ~~~~~*
      Perform Initialize-API thru Initialize-API-Exit.
```

```
thru Get-Client-ID-Exit.
         Perform Get-Client-ID
         Perform Sockets-Descriptor thru
                                                              Sockets-Descriptor-Exit.
         Perform Presentation-To-Numeric thru
                                                Presentation-To-Numeric-Exit.
         Perform CONNECT-Socket thru CONNECT-Socket-Exit.
Perform Numeric-TO-Presentation thru
                                                Numeric-To-Presentation-Exit.
         Perform Get-Name-Information thru
                                                Get-Name-Information-Exit
         Perform Write-Message
                                                   thru Write-Message-Exit.
         Perform Shutdown-Send
                                                   thru
                                                              Shutdown-Send-Exit.
         Perform Read-Message
Perform Shutdown-Receive
                                                              Read-Message-Exit.
Shutdown-Receive-Exit.
                                                    thru
                                                   thru
         Perform Close-Socket
                                                   thru
                                                              Exit-Now.
* Initialize socket API
 Initialize-API.
       Move soket-initapi to ezaerror-function.
Call 'EZASOKET' using soket-initapi maxsoc initapi-ident
Move 'Initapi failed' to ezaerror-text.
If retcode < 0 move 12 to failure.
Perform Return-Code-Check thru Return-Code-Exit.
Initialize-API-Exit.
Exit.
* Let us see the client-id
 Get-Client-ID.
         Nove soket-getclientid to ezaerror-function.
Call 'EZASOKET' using soket-getclientid clientid errno
                                            retcode
         Perform Return-Code-Check thru Return-Code-Exit.

Move 'Getclientid failed' to ezaerror-text.

If retcode < 0 move 24 to failure.

Perform Return-Code-Check thru Return-Code-Exit.

Move 'C' to client-server-flag.
 Get-Client-ID-Exit.
       Fxit.
* Get us a stream socket descriptor
 Sockets-Descriptor.
         Move soket-socket to ezaerror-function.
Call 'EZASOKET' using soket-socket AF-INET6 SOCK-STREAM
IPPROTO-IP errno retcode.
         Move 'Socket call failed' to ezaerror-text.
         If retcode < 0 move 60 to failure.
Perform Return-Code-Check thru Return-Code-Exit.
Move 'S' to Terminate-Options.
         Move retcode to socket-descriptor.
 Sockets-Descriptor-Exit.
       Exit.
* Use PTON to create an IP address to bind to.
 Presentation-To-Numeric.
         move soket-pton to ezaerror-function.
move IN6ADDR-LOOPBACK to presentable-addr.
Call 'EZASOKET' using soket-pton AF-INET6
             presentable-addr presentable-addr-len
              numeric-addr
         error retcode.
Move 'PTON call failed' to ezaerror-text.
If retcode < 0 move 24 to failure.
Perform Return-Code-Check thru Return-Code-Exit.
         move numeric-addr to server-ipaddr.
 Presentation-To-Numeric-Exit.
       Exit.
* CONNECT
 Connect-Socket.
         Move space to Connect-Flag.
         Move zeros to erno retcode.
move soket-connect to ezaerror-function.
CALL 'EZASOKET' USING SOKET-CONNECT socket-descriptor
         Move 'Connection call failed' to ezaerror-text.

If retcode < 0 move 24 to failure.

Perform Return-Code-Check thru Return-Code-Exit.

If retcode = 0 Move 'Y' to Connect-Flag.
 Connect-Socket-Exit.
       Fxit.
```

```
* Use NTOP to display the IP address.
 Numeric-To-Presentation.
      move soket-ntop to ezaerror-function.
      move server-ipaddr to numeric-addr.
      move soket-ntop to ezaerror-function.
Call 'EZASOKET' using soket-ntop AF-INET6
          numeric-addr
          presentable-addr presentable-addr-len
      errno retcode.
Display 'Presentable address = ' presentable-addr.
      Move 'NTOP call failed' to exaerror-text.
If retcode < 0 move 24 to failure.
      Perform Return-Code-Check thru Return-Code-Exit.
 Numeric-TO-Presentation-Exit.
      Exit.
* Use GETNAMEINFO to get the host and service names
 Get-Name-Information
      move 28 to name-len.
move 255 to host-name-len.
      move 32 to service-name-len.
      move ni-namereqd to name-info-flags.
      move soket-getnameinfo to ezaerror-function.
Call 'EZASOKET' using soket-getnameinfo
server-socket-address name-len
          host-name host-name-len
          service-name service-name-len
          name-info-flags
      errno retcode.
Display 'Host name = ' host-name.
Display 'Service = ' service-name.
Move 'Getaddrinfo call failed' to ezaerror-text.
      If retcode < 0 move 24 to failure.
Perform Return-Code-Check thru Return-Code-Exit.
 Get-Name-Information-Exit.
      Exit.
* Write a message to the server
 Write-Message.
       Move soket-write to ezaerror-function.
       Nove 'Message from EZAS06CC' to buf.
Call 'EZAS0KET' using soket-write socket-descriptor
            nbyte buf
       errno retcode.
Move 'Write call failed' to ezaerror-text.
If retcode < 0 move 84 to failure.
       Perform Return-Code-Check thru Return-Code-Exit.
 Write-Message-Exit.
      Fxit.
* Shutdown to pipe
 Shutdown-Send.
       Move soket-shutdown to ezaerror-function.
move 1 to how.
        Call 'EZASOKET' using soket-shutdown socket-descriptor
            how
       errno retcode.
Move 'Shutdown call failed' to ezaerror-text.
If retcode < 0 move 99 to failure.
       Perform Return-Code-Check thru Return-Code-Exit.
 Shutdown-Send-Exit.
      Exit.
* Read a message from the server.
 Read-Message
        Move soket-read to ezaerror-function.
       Move spaces to buf.
Call 'EZASOKET' using soket-read socket-descriptor
nbyte buf
       erno retcode.

If retcode < 0

Move 'Read call failed' to ezaerror-text

move 120 to failure
            Perform Return-Code-Check thru Return-Code-Exit.
 Read-Message-Exit.
      Exit.
* Shutdown receive pipe
 Shutdown-Receive
       Move soket-shutdown to ezaerror-function. move \boldsymbol{0} to how.
       Call 'EZASOKET' using soket-shutdown socket-descriptor
```

```
how
            errno retcode.
       Move 'Shutdown call failed' to ezaerror-text.
If retcode < 0 move 99 to failure.
Perform Return-Code-Check thru Return-Code-Exit.
 Shutdown-Receive-Exit.
      Exit.
*-----
* Close socket
*--
 Close-Socket.
        Move soket-close to ezaerror-function.
Call 'EZASOKET' using soket-close socket-descriptor
        Move 'Close call failed' to ezaerror-text.
If retcode < 0 move 132 to failure
perform write-ezaerror-msg thru
                      write-ezaerror-msg-exit.
        Accept Cur-Time from TIME.
Display Cur-Time ' EZASO6CC: ' ezaerror-function
' RETCODE=' RETCODE ' ERRNO= ' ERRNO.
 Close-Socket-Exit.
      Exit.
* Terminate socket API
 exit-term-api.
      ACCEPT cur-time from TIME.
Display cur-time ' EZASO6CC: TERMAPI '
' RETCODE= ' RETCODE ' ERRNO= ' ERRNO.
      Call 'EZASOKET' using soket-termapi.
* Terminate program
                    *---
       -----
 exit-now.
      Move failure to return-code.
      Goback.
*-----
* Subroutine.
                                                                                *
*
* Write out an error message
 write-ezaerror-msg.
      Move errno to ezaerror-errno.
      Move retcode to ezaerror-retcode.
Display ezaerror-msg.
 write-ezaerror-msg-exit.
     Exit.
*-
* Check Return Code after each Socket Call
 Return-Code-Check.
       Accept Cur-Time from TIME.
       Display Cur-Time ' EZASO6CC: ' ezaerror-function
' RETCODE=' RETCODE ' ERRNO= ' ERRNO.
       IF RETCODE < 0
           Perform Write-ezaerror-msg thru write-ezaerror-msg-exit
           Move zeros to errno retcode
IF Opened-Socket Go to Close-Socket
ELSE IF Opened-API Go to exit-term-api
ELSE Go to exit-now.
       Move zeros to errno retcode.
 Return-Code-Exit.
      Exit.
```

Figure 80. EZASO6CC COBOL call interface sample IPv6 client program

Chapter 8. IMS Listener samples

This topic includes sample programs using the IMS Listener. The following samples are included:

- "IMS TCP/IP control statements" on page 237
- "Sample program explicit-mode" on page 240
- "Sample program implicit-mode" on page 248
- "Sample program IMS MPP client" on page 253

IMS TCP/IP control statements

This topic contains examples of the control statements required to define and initiate the various IMS TCP/IP components.

JCL for starting a message processing region

This topics shows an example of the JCL that is required to start an IMS message processing region in which TCP/IP servers can operate. Note the STEPLIB statements that point to TCP/IP and the C run-time library. A C run-time library is required when you use the GETHOSTBYADDR or GETHOSTBYNAME call. For more information, see *z/OS Program Directory* or the topic on C compilers and run-time libraries in the *z/OS* Communications Server: IP Sockets Application Programming Interface Guide and Reference.

This sample is based on the IMS procedure (DFSMPR). You might have to modify the language run-time libraries to match your programming language requirements.

```
PROC SOUT=A, RGN=2M, SYS2=,
CL1=001, CL2=000, CL3=000, CL4=000,
//
||
||
||
                   OPT=N,OVLA=0,SPIE=0,VALCK=0,TLIM=00,
                   PCB=000, PRLD=, STIMER=, SOD=, DBLDL=,
                   NBA=, OBA=, IMSID=IMS1, AGN=, VSFX=, VFREE=,
||
||
                   SSM=, PREINIT=, ALTID=, PWFI=N,
                   APARM=
//*
//*
//REGION EXEC
//
//
//
//
//
                   PGM=DFSRRC00, REGION=&RGN,;
                   TIME=1440, DPRTY=(12,0),
PARM=(MSG,&CL1&CL2&CL3&CL4
                   &OPT&OVLA&SPIE&VALCK&TLIM&PCB,;
                   &PRLD,&STIMER,&SOD,&DBLDL,&NBA,;
                   &OBA,&IMSID,&AGN,&VSFX,&VFREE,;
                   &SSM,&PREINIT,&ALTID,&PWFI,;
//
//&*;
                    '&APARM')
//STEPLIB DD DSN=IMS31.&SYS2;RESLIB,DISP=SHR
              DD DSN=IMS31.&SYS2;PGMLIB,DISP=SHR
DD DSN=PLI.LL.V2R3M0.SIBMLINK,DISP=SHR
||
||
||
||
              DD DSN=PLI.LL.V2R3M0.PLILINK, DISP=SHR
              DD DSN=C370.LL.V2R2M0.SEDCLINK,DISP=SHR
              Use the following for LE/370 C run-time libraries:
DD DSN=CEE.V1R3M0.SCEERUN,DISP=SHR
DD DSN=TCPIP.SEZATCP,DISP=SHR
//*
//*
//PROCLIB DD DSN=IMS31.&SYS2;PROCLIB,DISP=SHR
//SYSUDUMP DD
                   SYSOUT=&SOUT, DCB=(LRECL=121, BLKSIZE=3129, RECFM=VBA),;
11
                   SPACE=(125, (2500, 100), RLSE, , ROUND)
```

JCL for linking the IMS Listener

The following examples are JCL that can be used to link the IMS Listener.

```
//EZAIMSCZ JOB (accounting, information), programmer.name,
11
            MSGLEVEL=(1,1), MSGCLASS=A, CLASS=A
//*NOTE: ANY ZONE UPDATED WITH THE LINK COMMAND OR CROSS-ZONE
1/*
        INFORMATION CANNOT BE PROCESSED BY SMP/E R6 OR EARLIER.*
//*
//*
     5694-A01 Copyright IBM Corp. 1997, 2007
//*
     Licensed Materials - Property of IBM
     This product contains "Restricted Materials of IBM"
//*
//*
     All rights reserved.
//*
     US Government Users Restricted Rights -
//*
     Use, duplication or disclosure restricted by
//*
     GSA ADP Schedule Contract with IBM Corp.
//*
     See IBM Copyright Instructions.
//*
//*
//*
      Function: Perform SMP/E LINK for IMS module
//*
//*
      Instructions:
//*
          Change all lower case characters to values
//*
          suitable for your installation.
//*
//*
      targetzone:
                   z/OS Target Zone
//*
        imszone :
                  IMS Target Zone
//*
//*
//*
     Change the high-level qualifier 'imshlq' to match the
//*
     high-level qualifier for your installation's IMS target
//*
     data set.
//*
//*
     Beginning with IMS V1R7 the target lib has changed from
//*
     RESLIB to SDFSRESL. If you are running IMS V1R7 or higher,
//*
     you must comment or delete the RESLIB DD card and uncomment
//*
     the SDFSRESL DD card.
//*
//EZAIMSCZ EXEC PGM=GIMSMP,REGION=4096K
DD DISP=SHR, DSN=imshlq.RESLIB
//RESLIB
//*SDFSRESL DD DISP=SHR,DSN=imshlq.SDFSRESL
//*
//SMPCSI
           dd
              dsn=zos.global.csi,disp=old
//*
//SYSUT1
          DD UNIT=SYSDA, SPACE=(1700, (900, 200))
//SYSUT2
          DD UNIT=SYSDA, SPACE=(1700, (600, 100))
//SYSUT3
          DD UNIT=SYSDA, SPACE=(1700, (600, 100))
          DD UNIT=SYSDA, SPACE=(1700, (600, 100))
//SYSUT4
          DD UNIT=SYSDA, SPACE=(8800, (75,0,216)),
//SMPWRK1
          DCB=(BLKSIZE=8800,LRECL=80)
//
//SMPWRK2
         DD UNIT=SYSDA, SPACE=(8800, (75,0,216)),
          DCB=(BLKSIZE=8800,LRECL=80)
//
//SMPWRK3
         DD UNIT=SYSDA, SPACE=(3200, (75,0,216)),
          DCB=(BLKSIZE=3200,LRECL=80)
//
//SMPWRK4
         DD UNIT=SYSDA, SPACE=(3200, (75,0,216)),
          DCB=(BLKSIZE=3200,LRECL=80)
//
```

```
//SMPWRK6
         DD UNIT=SYSDA, SPACE=(3200, (75,0,216))
//*
//SMPLIST
           DD
               SYSOUT=*
//SMPOUT
              SYSOUT=*
           DD
//SMPRPT
           DD
               SYSOUT=*
//SMPSNAP
           DD SYSOUT=*
//SMPHOLD
           DD DUMMY
//SYSPRINT
           DD SYSOUT=*
//*
//*
//SMPCNTL
          DD *
                     /* z/OS target zone */
SET BDY(targetzone).
LINK MODULE(DFSLI000)
FROMZONE(imszone)
                      /* IMS target zone
                                         */
INTOLMOD(EZAIMSLN)
RC(LINK=00).
```

Figure 81. Cross zone Lnk IMS application interface

EZAIMSPL JCLIN

```
//LINKIMS JOB (accounting, information), programmer.name,
            MSGLEVEL=(1,1), MSGCLASS=A, CLASS=A
//*
//*
     THIS JOB SERVES AS AN ALTERNATIVE TO THE CROSS ZONE LINK *
//*
     PERFORMED BY RUNNING EZAIMSCZ.
//*
//*
     UPDATE THE JOB, SYSLMOD AND RESLIB DD CARDS TO SUIT YOUR \star
//*
     INSTALLATION .
//*
//LNKIMS EXEC PGM=IEWL,PARM='XREF,LIST,REUS'
//SYSPRINT DD SYSOUT=*
//SYSUT1
          DD UNIT=SYSDA, SPACE=(CYL, (1,1))
//SYSLMOD DD DSN=tcpip.v3r1.SEZALINK,DISP=SHR
//RESLIB DD DSN=ims.RESLIB,DISP=SHR
//SYSLIN
          DD *
 ORDER CMCOPYR
 INCLUDE RESLIB(DFSLI000)
 INCLUDE SYSLMOD(EZAIMSLN)
ENTRY EZAIMSLN
 MODE RMODE(24) AMODE(31)
 NAME EZAIMSLN(R)
/*
```

Listener IMS definitions

The following statements define the Listener as an IMS BMP application and the PSB that it uses. Note that the name ALTPCB is required.

PSB definition

```
ALTPCB PCB TYPE=TP,MODIFY=YES
PSBGEN PSBNAME=EZAIMSLN,IOASIZE=1000
SSASIZE=1000,LANG=ASSEM
```

TRANSACT MODE=SNGL

Application definition

APPLCTN PSB=EZAIMSLN, PGMTYPE=BATCH

Sample program explicit-mode

This topic shows an example of an explicit-mode client server program pair. The client program name is EZAIMSC2; you can find it in SEZAINST(EZAIMSC2). The server program name is EZASVAS2; its IMS trancode is DLSI102. You can find the sample in SEZAINST(EZASVAS2).

Sample explicit-mode program flow

The client begins execution and obtains the host name and port number from startup parameters. It then issues SOCKET and CONNECT calls to establish connectivity to the specified host and port. Upon successful completion of the connect, the client sends the TRM, which tells the Listener to schedule the specified transaction (DLSI102). The Listener schedules that transaction and places a TIM on the IMS message queue. Finally, it issues a GIVESOCKET call and waits for the server to take the socket.

When the requested server (EZASVAS2) begins execution, it issues a GU call to obtain the TIM. Using addressability information from the TIM, it issues INITAPI and TAKESOCKET calls. The server then sends SERVER MSG #1 to the client.

When the client receives the message, it displays SERVER MSG #1 on stdout and then sends END CLIENT MSG #2 to the server, and displays a success message on stdout. It then blocks on another receive() until the server responds.

The server, upon receipt of a message with the characters END as the first 3 characters, sends SERVER MSG #2 back to the client and closes the socket.

When the client receives this message, it prints SERVER MSG #2 on stdout, closes the socket, and ends.

Sample explicit-mode client program (C language)

```
* Include Files.
/* #define RESOLVE_VIA_LOOKUP */
#pragma runopts(NOSPIE NOSTAE)
#define lim 50

#include <manifest.h>
#include <bsdtypes.h>
#include <in.h>
#include <socket.h>
#include <netdb.h>
#include <stdio.h>
′* Client Main.
 */
main(argc, argv)
int argc;
char **argv;
£
                                        /* port client will connect to
/* sned receive buffers 0 -3
     unsigned short port;
                                                                                                 */
*/
     char buf ??(lim??);
     char buf1 ??(lim??);
char buf2 ??(lim??);
char buf3 ??(lim??);
     struct hostent *hostnm; /* server host name information
struct sockaddr_in server; /* server address
                                                                                                 */
*/
     int s:
                                           /* client socket
      * Check Arguments Passed. Should be hostname and port.
     if (argc != 3)
     /* fprintf(stderr, "Usage: %s hostname port\n", argv[0]); */
    printf("Usage: %s hostname port\n", argv [0]);
           exit(1):
     }
            printf("Usage: %s hostname port\n", argv
                                                                               [0]);
     /*
      * The host name is the first argument. Get the server address.
       */
     hostnm = gethostbyname(argv[1]);
     if (hostnm == (struct hostent *) 0)
     ÷
```

```
/* fprintf(stderr, "Gethostbyname failed\n"); */
    printf("Gethostbyname failed\n");
    exit(2);
3
/*
 * The port is the second argument.
port = (unsigned short) atoi(argv[2]);
 * Put a message into the buffer.
strcpy(buf,"2000*TRNREQ*DLSI102 ");
/*
 * Put the server information into the server structure.
 * The port must be put into network byte order.
 */
server.sin_family = AF_INET;
server.sin_port = htons(port);
server.sin_addr.s_addr = *((unsigned long *)hostnm->h_addr);
/*
* Get a stream socket.
if ((s = socket(AF_INET, SOCK_STREAM, 0)) < 0)
£
    tcperror("Socket()");
    exit(3);
}
/*
   Connect to the server.
 *
if (connect(s, (struct sockaddr *)&server, sizeof(server)) < 0)
£
    tcperror("Connect()");
    exit(4);
3
if (send(s, buf, sizeof(buf), 0) < 0)
£
    tcperror("Send()");
    exit(5);
}
printf("send one complete\n");
 * The server sends message #1. Receive it into buffer1
 */
if (recv(s, bufl, sizeof(bufl), 0) < 0)
£
    tcperror("Recv()");
    exit(6);
7
printf("receive one complete\n");
printf(buf1,"\n");
/* fprintf(stdout,buf1,"\n"); */
/*
 * Put end message into the buffer.
 */
 strcpy(buf2, "END CLIENT MESSAGE #2 ");
if (send(s, buf2, sizeof(buf2), 0) < 0)
£
    tcperror("Send()");
    exit(7);
ł
printf("send two complete\n");
/* 
 * The server sends back message #2. Receive it into buffer 2.
if (recv(s, buf3, sizeof(buf3), 0) < 0)
£
    tcperror("Recv()");
    exit(8);
}
printf("receive two complete\n");
/* fprintf(stdout,buf3,"\n"); */
printf(buf3,"\n");
```

```
/*
 * Close the socket.
 */
close(s);
printf("Client Ended Successfully\n");
exit(0);
}
```

Figure 82. Sample C client to drive IMS Listener

Sample explicit-mode server program (Assembly language)

EZASVAS2 CSECT ENTRY POINT ADDRESSABILITY USING EZASVAS2, BASE SAVE (14, 12)SAVE DL/I REGS LR BASE,15 SAVE AREA CHAINING ST R13,SAVEAREA+4 R13,SAVEAREA LA NEW SAVE AREA MVC PSBS(L'PSBS*3),0(1) SAVE PCB LIST * * REG 1 CONTAINS PTR TO PCB ADDR LIST * REG 13 CONTAINS PTR TO DL/I SAVE AREA * REG 14 CONTAINS PTR DL/I RETURN ADDRESS * REG 15 CONTAINS PROGRAMS ENTRY POINT * LOAD ADDR OF I/O PCB L R2,0(R0,R1) * USING IOPCB,R2 ADDRESSABILITY * R3, 4(R0, R1)LOAD ADDR OF ALT PCB L * USING ALTPCB1,R3 ADDRESSABILITY * LOAD ADDR OF ALT PCB R4,8(R0,R1) L LA R4,0(R0,R4) REMOVE HIGH ORDER BIT * USING ALTPCB2,R4 ADDRESSABILITY * LA R5,IOAREAIN POINT TO OUTPUT AREA FOR TCPIP LA R7, IOAREAOT GUCALL DS GET UNIQUE CALL 0H * Get Transaction-initiation message containing Sockets data * CALL ASMTDLI, (GUFUNCT, (2), (5)), VL GET TIM CLC STATUS(L'STATUS),=CL2'QC' IF NO MESSAGES RETURN TO IMS ELSE NEXT INSTR ΒE GOBACK * STATUS(L'STATUS),=CL2' т. CLC IF BLANK OK BNE ERRRTN SOME WRONG HERE ELSE NEXT INSTR * * XR R6,R6 CLEAR REG BAL R6, INITAPI GO INSERT SEGMENT В GUCALL SET RETURN ADDRESS * *

```
INITAPI DS OH
* Set up for INITAPI
       MVC TCPNAME(L'TCPNAME), TIMTCPAS
                                   TCP Address space
          ASDNAME(L'ASDNAME),TIMSAS
       MVC
                                   Server address space
       MVC SUBTASK(L'SUBTASK),TIMSTD
                                   Server task id
* Set up for takeSOCKET
       MVC NAME(L'NAME), TIMLAS
                                   Listener address space
       MVC TASK(L'TASK),TIMLTD
                                   Listener task id
       MVC S(L'S),TIMSD
                                   Socket descriptor
*
       XC
           ERRNO(L'ERRNO), ERRNO
       XC
           RETCODE(L'RETCODE), RETCODE
       EX
           0,*
*
*
   Issue INITAPI
*
                                                   *
*
       CALL EZASOKET,
(INITFUNC, MAXSOC, IDENT, SUBTASK,
                                   Х
           MAXSNO, ERRNO, RETCODE), VL
           R9, RETCODE
       L
       LTR R9,R9
       BNM TAKESOC
*
INITERR
       DC
           CL21'INITAPI COMMAND ERROR'
*
TAKESOC
       DS
           0H
*
*
   Issue takeSOCKET
*
       CALL EZASOKET, (TAKEFUNC, S, CLIENT, ERRNO, RETCODE), VL
*
           R9, RETCODE
       L
       LTR
           R9,R9
       BNM SENDTEXT
*
TAKERR
       DC
           CL16'TAKESOCKET ERROR'
*Set up to send "SERVER MSG #1"
SENDTEXT DS
           0H
*
       MVC
           S(L'S), RETCODE+2
       XC
           BUF(LENG), BUF
       MVC BUF(13),=CL13'SERVER MSG #1'
*Translate to ASCII, if necessary
       CALL EZACIC04, (BUF, LENGTH), VL
Send "SERVER MSG #1"
*
                                                   *
CALL EZASOKET,
(SENDFUNC, S, FLAGS, NBYTE, BUF, ERRNO, RETCODE),
                                   Х
           VL
           R9, RETCODE
       L
       LTR R9,R9
       BNM RECVTEXT
*
SENDERR1 DC CL16'SEND ERROR'
                                  Abend on error
```

```
RECVTEXT DS
         0H
*
   Receive client message #2
*
*
*
      CALL EZASOKET,
(RECVFUNC, S, FLAGS, NBYTE, BUF, ERRNO, RETCODE),
                              Х
         VL
* Translate to EBCDIC if necessary
      CALL EZACIC05, (BUF, LENGTH), VL
*
*
         R9, RETCODE
      L
      LTR R9,R9
      BNM CHECKTXT
*
      DC
         CL16'RECEIVE ERROR'
                             Abend on error
*
CHECKTXT
      DS
         0H
*
      CLC BUF(3),=CL3'END'
                             Test for end of message
      BNE RECVTEXT
                             If not eom, read again
*
*
   Set up to send shutdown message
SENDEND
      DS
         0H
*
      XC
         BUF(LENG), BUF
      MVC BUF(13),=CL13'SERVER MSG #2'
   Translate to ASCII if necessary
*
      CALL EZACIC04, (BUF, LENGTH), VL
*
Send "SERVER MSG #2" to indicate shutdown
*
CALL EZASOKET,
(SENDFUNC, S, FLAGS, NBYTE, BUF, ERRNO, RETCODE),
                              Х
         VL
         R9, RETCODE
      L
      LTR R9,R9
      BNM SOCKCLOS
         CL16'SEND ERROR'
SENDERR2 DC
                             Abend on error
SOCKCLOS DS
         0H
Close the socket
*
CALL EZASOKET, (CLOSFUNC, S, ERRNO, RETCODE), VL
*
         R9, RETCODE
      LTR R9,R9
      BNM TERMAPI
*
CLOSERR
      DC
         CL16'CLOSE ERROR'
*
TERMAPI
      DS
         0H
Terminate the API
*
                                          *
```

	CALL	EZASOKEI, (IERMFUNC), VL	
* PROCTCP * *	DS XR LA STH XC MVC MVC UNPK OI UNPK OI VNPK OI UNPK OI UNPK OI WVC MVC	0H R9,R9 R9,OTLEN R9,OTLTH OTRSV(L'OTRSV),OTRSV OTMSG(L'OTMSG),DCINMSG OTLITDT(L'OTLITDT),DCD OTLITIME(L'OTLITIME),D OTDATE,CDATE OTDATE+7,X'F0' OTTIME+7,X'F0' R9,R9 R9,INPUTMSN R9,DLBWORK OTINPUTN,DLBWORK OTINPUTN,T,X'F0' OTFILL(L'OTFILL),=28X' OTLTERM(L'OTLTERM),LTE	Talk to TCPIP Client AND ALTERNATE SUCESSFUL MSG CLEAR REG LOAD LENGTH STORE LEN THERE CLEAR RESERVE DATA MOVE IN MSG ATE MOVE IN DATE CTIME MOVE IN TIME MAKE TIME & DATE EBCDIC GET READY INPUT COUNT INPUT COUNT INPUT COUNT FIX SIGN 40' FILL CHAR RMN ADD TERMINAL
*	nvc	OTETERN(E OTETERN), ETE	
*	CALL	ASMTDLI,(ISRTFUNCT,(3)	,(7),,USER1),VL
*	XC BR	IOAREAOT(L'IOAREAOT),IO R6	AREAOT
* ERRRTN *	DS	өн	SOME WRONG HERE
*	L RETUR	R13,4(R13) RN (14,12),RC=8	RELOAD DL/I REGS & RETURN ERROR
GOBACK *	DS	өн	RETURN TO IMS
*	L RETUR	R13,4(R13) RN (14,12),RC=0	RELOAD DL/I REGS & RETURN
PSBS BASE RC R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 P13	DS DS SPACE EQU EQU EQU EQU EQU EQU EQU EQU EQU EQ	0D 3F 1 12 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13	

R14 R15	EQU EQU SPACE	14 15 1
*	50	
SAVEAREA	DS DC	0F 18F'0'
GUFUNCT GNFUNCT PURGFUNCT ISRTFUNCT BADCALL ERROPT	DC DC DC DC DC DC DC	CL4'GUGET UNIQUE CALLCL4'GNGET NEXTCL4'PURG'PURGE CALLCL4'ISRT'INSERT CALLCL8'BAD CALL'BAD LITF'0'1=nodump 0=dump
DCINMSG DCDATE DCTIME USER1 USER2 WTOR	DC DC DC DC DC DC	CL26' INPUT MESSAGE SUCESSFUL ' CL6' DATE ' CL6' TIME ' CL8'USER1 ' CL8'USER2 ' CL8'WTOR '
INITFUNC TAKEFUNC SENDFUNC RECVFUNC CLOSFUNC TERMFUNC SELEFUNC	DC DC DC DC DC DC DC DC	CL16'INITAPI' CL16'TAKESOCKET' CL16'SEND' CL16'RECV' CL16'CLOSE' CL16'TERMAPI' CL16'SELECT'
WORKTCPIF APITYPE MAXSOC MAX MAXSNO	DC DC DC EQU DS	CL27'TCPIP WORK DATA BEGINS HERE' AL2(2) AL2(MAX) 50 F'00'
* IDENT TCPNAME ASDNAME	DS DS DS	OCL16 CL8 CL8
CLIENT DOMAIN NAME TASK RESERVED	DS DC DS DS DS	0CL38 F'2' CL8 CL8 20B'0'
* SUBTASK ERRNO RETCODE FLAGS NBYTE BUF LENG LENGTH TIMEOUT SECONDS MILLISEC RSNDMASK WSNDMASK ESNDMASK	DS DS DC DC DS EQU DC DS DS DS DS DS DS DS	CL8 F F'0' F'50' CL(LENG) 50 AL4(LENG) 0D F F CL(MAX) CL(MAX) CL(MAX)

WRETMASK WRETMASK ERETMASK S	DS DS DS DS	CL(MAX) CL(MAX) CL(MAX) H	
* DLBWORK	DS DS	0D D	
IOAREAIN TIMLEN TIMRSV TIMID TIMLAS TIMLTD TIMSAS TIMSTD TIMSD in	DS DS DS DS DS DS DS DS DS DS DS	0CL56 H H CL8 CL8 CL8 CL8 CL8 CL8 CL8 CL8 H	I/O AREA INPUT Length of trans init msg reserved set to zeros LISTENER ID set to LISTNR LISTENER addr space name LISTENER taskid for takesocket SERVER addr space name SERVER TASK ID user in initapi socket given in LISTENER used
* TIMTCPAS TIMDT *	DS DS	CL8 H	tasksocket TCPIP addr space name Data type of client ASCII(0) or EBCDIC(1)
IOAREAOT OTLTH OTRSV OTLTERM OTINPUTN OTMSG OTLITDT OTDATE OTLITIME OTLITIME OTFILL OTFILL	DS DS DS DS DS DS DS DS DS DS DS DS EQU	OF OCL119 BL2 BL2 CL8 CL8 CL25 CL6 CL8 CL6 CL8 CL6 CL8 CL6 CL8 CL28 (*-IOAREAOT)	I/O AREA OUTPUT
IOPCB LTERMN STATUS CDATE CTIME INPUTMSN MSGOUTDN USERID	DSECT DS DS DS DS DS DS DS DS DS DS	CL8 CL2 CL2 PL4 PL4 BL4 CL8 CL8	I/O AREA LOGICAL TERMINAL NAME RESERVED FOR IMS STATUS CODE CURRENT DATE YYDDD CURRENT TIME HHMMSST SEQUENCE NUMBER MESSAGE OUT DESC NAME USER ID OF SOURCE
* ALTPCB1 ALTERM1 ALSTAT1	DSECT DS DS DS	CL8 CL2 CL2	ALTERNATE PCB DESTINATION NAME RESERVED FOR IMS STATUS CODE
* ALTPCB2 ALTERM2 ALSTAT2	DSECT DS DS DS	CL8 CL2 CL2	ALTERNATE PCB DESTINATION NAME RESERVED FOR IMS STATUS CODE
*	END		

Figure 83. Sample assembler IMS server

Sample program implicit-mode

The topic shows an example of an implicit-mode client server program pair. The client program name is EZAIMSC1; you can find it in *hlq*.SEZAINST(EZAIMSC1). The server program name is EZASVAS1; its IMS trancode is DLSI101. The sample program is located in *hlq*.SEZAINST(EZASVAS1). When link editing the sample program, module EZAIMSAS should be included from the SEZALOAD target library.

Sample implicit-mode program flow

The client begins execution and obtains the host name and port number from the startup parameters. It then issues SOCKET and CONNECT calls to establish connectivity to the specified host and port. Upon successful completion of the CONNECT, the client sends the TRM, which tells the Listener to schedule the specified transaction (DLSI101). Because implicit-mode protocol requires that all input data segments be transmitted before the server application is scheduled, the client follows the TRM with 2 segments of application data and an end-of-message (EOM) segment. The Listener schedules DLSI101 and places a TIM on the IMS message queue, followed by the 2 segments of application data. Finally, the Listener issues a GIVESOCKET call and waits for the server to take the socket.

When the requested server (EZASVAS1) begins execution, it issues a GU call to ASMADLI. Behind the scenes, the Assist module issues its own GU and retrieves the TIM from the IMS message queue. Using addressability information from the TIM, it issues INITAPI and takeSOCKET calls, which establish connectivity with the client.

Once connectivity is established, the Assist module issues a GN to the IMS message queue, which returns the first segment of application data sent by the client. This data is returned to the server mainline. (Thus, to the server mainline, the first segment of application data is returned in response to its GU.) In the sample program, the first segment of application data is the data record: THIS IS FIRST TEXT MESSAGE SEND TO SERVER. This record is echoed back to the client by means of an IMS ISRT call to ASMADLI. The IMS Assist module intercepts the ISRT and issues a TCP/IP write() to echo the segment back to the client. The server mainline then issues a GN ASMADLI (which the Assist module intercepts and executes another GN ASMTDLI) to receive the second segment of user data. This segment is also echoed back to the client, using an IMS ISRT call, which the Assist module intercepts and replaces with a TCP/IP write() to the client.

After the second client data segment, the message queue contains an EOM segment, denoting the client's end-of-message. When the server has echoed the second input segment to the client, it issues another GN to ASMADLI. ASMADLI receives an end-of-message indication from the message queue and passes a QD status code back to the server mainline.

At this point, the server mainline has completed processing that message and issues a GU to see whether another message has arrived for that trancode. This GU triggers the Assist module to send a final CSMOKY message to the client, indicating successful completion. It then issues another GU to the IMS message queue to determine whether another message for that trancode has been queued. If so, the server program repeats itself; if not, the server issues a GOBACK and ends.

Sample implicit-mode client program (C language)

```
/*
 * Include Files.
 */
 * #define RESOLVE_VIA_LOOKUP */
#pragma runopts(NOSPIE NOSTAE)
#define lim 119
#include <manifest.h>
#include <bdtypes.h>
#include <in.h>
#include <in.h>
#include <socket.h>
#include <socket.h>
#include <stdio.h>
/*
 * Client Main.
 */
main(argc, argv)
int argc;
char **argv;
```

```
£
    unsigned short port;
struct sktmsg
                                  /* port client will connect to
                                                                               */
            £
               short msglen;
               short msgrsv;
               char msgtrn??(8??);
char msgdat??(lim??);
               msgbuff;
    struct datmsg
            £
               short datlen;
short datrsv;
char datdat??(lim??);
            } datbuff;
    char buf ??(lim??);
                                  /* send receive buffer
                                                                               */
    struct hostent *hostnm;
                                  /* server host name information
                                                                               */
    struct sockaddr_in server; /* server address
int s; /* client socket
int len; /* length for send
                                                                               */
*/
*/
    /*
     * Check Arguments Passed. Should be hostname and port.
    if (argc != 3)
        printf("Invalid parameter count\n");
        exit(1);
    3
    printf("Usage: %s program name\n",argv??(0??));
     * The host name is the first argument. Get the server address.
     */
    printf("Usage: %s host name\n",argv??(1??));
    hostnm = gethostbyname(argv[1]);
if (hostnm == (struct hostent *) 0)
    £
          printf("Gethostbyname failed\n");
        exit(2);
    }
    /*
     * The port is the second argument.
    printf("Usage: %s port name\n",argv??(2??));
    port = (unsigned short) atoi(argv[2]);
    /*
     \star Put the server information into the server structure.
     * The port must be put into network byte order.
     */
    server.sin_family = AF_INET;
server.sin_port = htons(port);
    server.sin_addr.s_addr = *((unsigned long *)hostnm->h_addr);
    /*
* Get a stream socket.
    if ((s = socket(AF_INET, SOCK_STREAM, 0)) < 0)</pre>
    £
         tcperror("Socket()");
        exit(3);
    3
    /*
     * Connect to the server.
    if (connect(s, (struct sockaddr *)&server, sizeof(server)) < 0)</pre>
    £
         tcperror("Connect()");
        exit(4);
    }
/*
     * Put a message into the buffer.
     */
   msgbuff.msgdat??(0??)='\0';
   lim-strlen(msgbuff.msgdat)-1);
   len=20;
```

```
if (send(s, (char *)\&msgbuff, len, 0) < 0)
 ÷
      tcperror("Send()");
      exit(5);
 }
 printf("\n");
printf(msgbuff.msgdat);
printf("send one complete\n");
 /*
  * Put a text message into the buffer.
datbuff.datdat??(0??)='\0';
datbuff.datlen = 46;
datbuff.datrsv = 0;
len=46;
 if (send(s, (char *)&datbuff, len, 0) < 0)
 £
      tcperror("Send()");
      exit(6);
 }
 printf("\n");
printf(datbuff.datdat);
printf("\n");
printf("send for first text message complete\n");
 /* * \ {\rm Put} \ {\rm a} \ {\rm text} \ {\rm message} into the buffer.
  */
datbuff.datdat??(0??)='\0';
datbuff.datlen = 47;
strncat(datbuff.datdat,"THIS IS 2ND TEXT MESSAGE SENDING TO SERVER",
         lim-strlen(datbuff.datdat)-1);
len=47;
 if (send(s, (char *)\&datbuff, len, 0) < 0)
 £
      tcperror("Send()");
     exit(7);
 z
 printf("\n");
printf(datbuff.datdat);
printf("\n");
printf("send for 2nd test message complete\n");
 /*
 * Put a end message into the buffer.
 */
datbuff.datdat??(0??)='0';
datbuff.datlen = 4;
strncpy(datbuff.datdat," ",lim);
 len=4:
 if (send(s, (char *)\&datbuff, len, 0) < 0)
 £
      tcperror("Send()");
      exit(8);
 3
 printf("\n");
 printf(datbuff.datdat);
printf("\n");
printf("send for end message complete\n");
 /*
  * The server sends back the same message. Receive it into the
  * buffer.
 strncpy(datbuff.datdat," ",lim);
 if (recv(s, (char *)\&datbuff, lim, 0) < 0)
 £
      tcperror("Recv()");
     exit(9);
 3
 printf("receive one text complete\n");
printf(datbuff.datdat);
printf("\n");
  * The server sends back the same message. Receive it into the
  * buffer.
  */
```

```
strncpy(datbuff.datdat," ",lim);
if (recv(s, (char *)\&datbuff, lim, 0) < 0)
£
     tcperror("Recv()");
     exit(10);
}
printf("receive two text complete\n");
printf(datbuff.datdat);
printf("\n");
/*
 * The server sends eof message. Receive it into the
 * buffer.
 */
strncpy(datbuff.datdat," ",lim);
if (recv(s, (char *)\&datbuff, 4, 0) < 0)
£
     tcperror("Recv()");
     exit(11);
}
printf("receive eof complete\n");
printf("\n");
printf(datbuff.datdat);
printf("\n");
strncpy(datbuff.datdat," ",lim);
if (recv(s, (char *)\&datbuff, 12, 0) < 0)
£
     tcperror("Recv()");
     exit(12);
}
printf("receive CSMOKY complete\n");
printf("\n");
printf(datbuff.datdat);
printf("\n");
/* Close the socket.
close(s);
printf("Client Ended Successfully\n");
exit(0);
```

Figure 84. Sample C client to drive IMS Listener

}

Sample implicit-mode server program (Assembly language)

EZASVAS1	CSECT USING SAVE LR ST LA MVC	EZASVAS1,BASE (14,12) BASE,15 R13,SAVEAREA+4 R13,SAVEAREA PSBS(L'PSBS*3),0(1)	ENTRY POINT ADDRESSABILITY SAVE DL/I REGS SAVE AREA CHAINING NEW SAVE AREA SAVE PCB LIST
* REG 1 C * REG 13 * REG 14 * REG 14	ONTAINS CONTAIS CONTAIS CONTAIS	S PTR TO PCB ADDR LIST NS PTR TO DL/I SAVE ARE/ NS PTR DL/I RETURN ADDRI NS PROGRAMS ENTRY POINT	A ESS
*	L	R2,0(R0,R1)	LOAD ADDR OF I/O PCB
*	USING	IOPCB,R2	ADDRESSABILITY
*	L	R3,4(R0,R1)	LOAD ADDR OF ALT PCB
*	USING	ALTPCB1,R3	ADDRESSABILITY
*	L LA	R4,8(R0,R1) R4,0(R0,R4)	LOAD ADDR OF ALT PCB REMOVE HIGH ORDER BIT
*	USING	ALTPCB2,R4	ADDRESSABILITY
*	LA LA	R5,IOAREAIN R7,IOAREAOT	POINT TO OUTPUT AREA
GUCALL	DS	ΘH	GET UNIQUE CALL

CALL ASMADLI, (GUFUNCT, (2), (5)), VL STATUS(L'STATUS),=CL2'QC' CLC IF NO MESSAGES RETURN TO IMS ELSE NEXT INSTR BF GOBACK CLC STATUS(L'STATUS),=CL2' ' IF BLANK OK SOME WRONG HERE BNE ERRRTN ELSE NEXT INSTR XR R6,R6 CLEAR REG R6, GNCALL SET RETURN ADDRESS LA GO INSERT SEGMENT BAL R6, ISRTCALL GET NEXT CALL GNCALL DS ΘH * * CALL ASMADLI, (GNFUNCT, (2), (5)), VL * CLC STATUS(L'STATUS),=CL2'QD' IF NO MORE SEGMENTS RETURN TO IMS IF NO MORE SEGMENTS SOME WRONG HERE BE GUCALL STATUS(L'STATUS),=CL2' ' CLC BNE ERRRTN * XR R6,R6 CLEAR REG LA R6,GNLOOP SET RETURN ADDRESS BAL R6, ISRTCALL GO INSERT SEGMENT GNLOOP В GNCALL ISRTCALL DS ΘH INSERT - WRITE TO TERMINAL AND ALTERNATE SUCESSFUL MSG * XR R9,R9 CLEAR REG R9, OTLEN LOAD LENGTH LA STORE LEN THERE CLEAR RESERVE DATA MOVE IN MSG " " DATE " " TIME STH R9,OTLTH R9, OILIH OTRSV(L'OTRSV), OTRSV OTMSG(L'OTMSG), DCINMSG OTLITDT(L'OTLITDT), DCDATE OTLITIME(L'OTLITIME), DCTIME XC MVC MVC MVC OTDATE, CDATE OTDATE+7,X'F0' OTTIME,CTIME UNPK MAKE TIME & DATE 0T FBCDTC UNPK OTTIME+7,X'F0' ΟI XR R9,R9 GET READY GET READY INPUT COUNT INPUT COUNT INPUT COUNT FIX SIGN FILL CHAR R9, INPUTMSN R9, DLBWORK CVD UNPK OTINPUTN,DLBWORK OI OTINPUTN+7,X'F0' MVC OTFILL(L'OTFILL),=28X'40' MVC OTLTERM(L'OTLTERM),LTERMN ADD TERMINAL * For LTERM USER1.... * CALL ASMADLI, (ISRTFUNCT, (2), (7)), VL * For LTERM USER2.... * XC IOAREAOT(L'IOAREAOT),IOAREAOT BR R6 ERRRTN DS ΘH SOME WRONG HERE L R13,4(R13) RETURN (14,12),RC=8 RELOAD DL/I REGS & RETURN * ERROR GOBACK DS ΘH RETURN TO IMS R13,4(R13) 1 RETURN (14,12), RC=0 RELOAD DL/I REGS & RETURN * DS 0D DS 3F SPACE 1 PSBS 3F BASE EQU 12 ĒŲŪ 15 RC R0 EŲŪ 0 EÕU EÕU R1 1 2 R2 R3 EQU 3 EQU 4 R4 R5 EŲŪ 5 R6 R7 EQU 6 7 EQU R8 EQU 8 R9 EQU 9 R10 EQU 10 R11 EŲU 11 R12 EQU EQU 12 13 R13 ĒQU R14 14 R15 EŲŪ 15

*

	JFACE .	L	
SAVEAREA GUFUNCT PURGFUNCT ISRTFUNCT BADCALL ERROPT DCINMSG DCDATE DCTIME USER1 USER1 USER2 WTOR	DS DC DC DC DC DC DC DC DC DC DC DC DC DC	OF 18F'0' CL4'GU ' CL4'GN ' CL4'PURG' CL4'ISRT' CL3'BAD CALL' F'1' CL26' INPUT MESSAGE CL6' DATE ' CL6' TIME ' CL6'TIME ' CL8'USER1 ' CL8'USER2 ' CL8'WTOR '	GET UNIQUE CALL GET NEXT PURGE CALL INSERT CALL BAD LIT 1=NODUMP 2=DUMP SUCESSFUL '
* DLBWORK IOAREAIN IOAREAOT	DS DS DS DS DS DS	0D D OF CL119 OF OCL119	I/O AREA INPUT I/O AREA OUTPUT
OTLTH OTRSV OTLTERM OTINPUTN OTMSG OTLITDT OTDATE OTLITIME OTTIME OTTIME OTTLL OTLEN	DS DS DS DS DS DS DS DS DS EQU	BL2 BL2 CL8 CL25 CL6 CL8 CL6 CL8 CL6 CL8 CL46 (*-IOAREAOT)	
ÎOPCB LTERMN STATUS CDATE CTIME INPUTMSN MSGOUTDN USERID	DSECT DS DS DS DS DS DS DS DS DS	CL8 CL2 CL2 PL4 PL4 BL4 CL8 CL8	I/O AREA LOGICAL TERMINAL NAME RESERVED FOR IMS STATUS CODE CURRENT DATE YYDDD CURRENT TIME HHMMSST SEQUENCE NUMBER MESSAGE OUT DESC NAME USER ID OF SOURCE
ALTPCB1 ALTERM1 ALSTAT1	DSECT DS DS DS	CL8 CL2 CL2	ALTERNATE PCB DESTINATION NAME RESERVED FOR IMS STATUS CODE
* ALTPCB2 ALTERM2 ALSTAT2	DSECT DS DS DS	CL8 CL2 CL2	ALTERNATE PCB DESTINATION NAME RESERVED FOR IMS STATUS CODE
*	END		

Figure 85. Sample assembler IMS server

Sample program - IMS MPP client

This information assumes that the IMS system is the server; however, some applications require that the server be a TCP/IP host. The following information shows an example of a program in which the *client* is an IMS MPP, and the *server* is a TCP/IP host.

For simplicity, we have coded both client and server to execute on an MVS host. The client (EZAIMSC3) is initiated by a 3270-driven IMS MPP; the server (EZASVAS3) is a TSO job which is already running when the client starts.

The samples are located in *hlq*.SEZAINST(EZAIMSC3) and *hlq*.SEZAINST(EZASVAS3).

Sample IMS MPP client program flow

A TSO Submit command is used to start the server. Once started, it executes the TCP/IP connection sequence for an iterative server (INITAPI, SOCKET, BIND, LISTEN, SELECT, and ACCEPT) and then waits for the client to request connection.

Note that the BIND call returns a socket descriptor which is then used to listen for a connection request. The ACCEPT call also returns a socket descriptor, which is used for the application data connection. Meanwhile, the original listener socket is available to receive additional connection requests.

The client is started by calling an IMS transaction which, in turn, executes the TCP/IP connection sequence for a client (INITAPI, SOCKET, and CONNECT).

Upon receiving the connection request from the client, the server issues a READ and waits for the client to WRITE the initial message. The server contains a READ/WRITE loop which echoes client transmissions until an "END" message is received. When this message is received, it sets a 'last record' switch, echoes the end message to the client, and terminates.

Note that in order for the server to terminate, it must close two sockets: one -- the socket on which it listens for connection requests; the other -- the socket on which the data transfers took place.

The client and server both include Write To Operator macros, which allow you to monitor progress through the application logic flow. At the end of this appendix you will find a sample of the WTO output from the client and the server.

Sample client program for non-IMS server

EZAIMSC3 CSECT EZAIMSC3 AMODE ANY EZAIMSC3 RMODE ANY GBLB &TRACE ASSEMBLER VARIABLE TO CONTROL TRACE GENERATION
 GBLB
 &SETB
 1
 1=TRACE
 ON
 0=TRACE
 OFF

 GBLB
 &SUBTR
 ASSEMBLER
 VARIABLE
 TO
 CONTROL
 SUBTRACE

 SETB
 0
 1=SUBTRACE
 ON
 0=SUBTRACE
 OFF
 &TRACE &SUBTR * MODULE NAME: EZAIMSC3 Licensed Materials - Property of IBM Copyright: "Restricted Materials of IBM" 5694-A01 Copyright IBM Corp. 2009 US Government Users Restricted Rights -Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp. CSV1R11 Status: MODULE FUNCTION: Sample program of an IMS MPP TCP client. This module connects with a TCP/IP server and exchanges msgs with it. The number of msgs exchanged is determined by a constant and the length of the messages is also determined by a constant. Note: If an error occurs during processing, this module will send an error message to the system console and then Abends0c1. LANGUAGE: Assembler ATTRIBUTES: Reusable INPUT: None Change History: Flag Reason Release Date Origin Description \$Q1= D316.15 CSV1R5 020604 BKELSEY : Support 64K sockets \$F1= RBBASE CSV1R11 080612 Herr : Cleaned up >72 lines SOC0000 DS B SOC00100 DC CL16/TMC ΘH Tell assembler to use reg 15 Branch to startup address CL16'IMSTCPCLEYECATCH' EQU 1000 BUELEN Set length of I/O buffers A(SOC0000+4096) R4BASE DC Control Variables for this program

 *
 Control variables for this program

 *
 *

 SOCMSGN DC
 F'005'
 Number of messages to be exchanged

 SOCMSGL DC
 F'200'
 Length of messages to be exchanged

 SOCMSGL DC
 F'200'
 Port Address of Server

 SOCTASK DC
 F'0'
 Task number for this client

 SERVLEN DC
 H'0'
 Length of server's name

 SERVNAME DC
 CL24'
 Internet name of server

 SENDINT DC
 CL8'00000010'
 Delay interval between sends

 * Constants used for call functions

INITAPI GETHSTID SOCKET GHBN CONNECT READ WRITE CLOSE TERMAPI	DC DC DC DC DC DC DC DC DC DC DC	CL16'INITAPI' CL16'GETHOSTD' CL16'GCHOSTD' CL16'GETHOSTBYNAME' CL16'GETHOSTBYNAME' CL16'CONNECT' CL16'READ' CL16'WRITE' CL16'LOSE' CL16'TERMAPI'	,	*
*	Beginr	ning of program exec	cution statements	* *
* SOC00100	DS STM LR L DROP USING	0H R14,R12,12(R13) R3,R15 R4,R4BASE R15 S0C0000,R3,R4	Beginning of program Save callers registers Move base reg to R3 Add R4 as second base reg Tell assembler to drop R15 as base Tell assembler to use R3 and R4 as base registers	* X
S0C00200	LR LA SR SR MVCL LA USING ST ST DS	R7,R13 R12,S0CSTG R13,S0CSTGL R14,R14 R15,R15 R12,R14 R13,S0CSTG S0CSTG,R13 R7,S0CSAVEL R13,8(R7) OH	Save address of previous save area Move address of program stg to R12 Move length of program stge to R13 Clear R14 Clear R15 Clear program storage Move address of program stg to R13 Tell Assembler about storage Save address of lower save area Complete save area chain	
* Build	d messa	age for console		
	MVC L CVD UNPK OI MVC LA MVC WVC WTO	MSG1D,MSG1C R0,SOCTASK R0,DWORK MSGTD,DWORK+5(3) MSGTD+4,X'F0' MSG2D,MSG2CS R6,MSG MSGLEN,=AL2(MSGTL) WTOLIST,WTOPROT TEXT=(R6), MF=(E,WTOLIST)	Initialize first part of message Get task number Convert task number to decimal Convert decimal to character Clear sign Move 'Started' to message Put text address in R6 Put length of text in msg hdr. Move prototype WTO to list form Write message to operator	x
*	Issue	INITAPI Call to cor	nnect to interface	
*	MVC MVC MVC MVC MVC MVC	SOCTASKC(3),=CL3'SC SOCTASKC+3(5),MSGTD MSG2D,MSG2C1 MAXSOC,=AL2(50) ASTCPNAM,=CL8'TCPV3 ASCLNAME,=CL8'TCPCI	DC' Build Task Identifier D Move 'INITAPI'to message Initialize MAXSOC field 3 ' Initialize TCP Name INT' Initialize AS Name	
*	CALL	EZASOKET, (INITAPI,MAXSOC,AS] RETCODE), VL	IDENT,SOCTASKC,HISOC,ERRNO, Specify variable parameter list	X X X
*	L C BL AIF	R6,RETCODE R6,=F'0' SOCERR (NOT_&TRACE).TRACE	Check for sucessful call Is it less than zero Yes, go display error and terminat	
* TRACE	ENTRY LA MVC WTO	FOR INITAPI TRACE R6,MSG MSGLEN,=AL2(MSGTL) TEXT=(R6), MF=(E,WTOLIST)	TYPE = 1 Put text address in R6 Put length of text in msg hdr. Write message to operator	х
.TRACE01 *	ANOP			
*	Issue	GETHOSTID Call to	obtain internet address of host	
*	MVC	MSG2D,MSG2C8	Move 'GTHSTID'to message	
*	CALL	EZASOKET, (GETHSTID,SERVIADD) VL	Issue GETHOSTID Call), Specify Variable parameter list	X X
* TRACE	AIF ENTRY LA MVC WTO	(NOT &TRACE).TRACEG FOR GETHOSTID TRAC R6,MSG MSGLEN,=AL2(MSGTL) TEXT=(R6), MF=(E,WTOLIST)	08 CE TYPE = 8 Put text address in R6 Put length of text in msg hdr. Write message to operator	x
.TRACE08	ANOP			
*	Issue	SOCKET Call to obta	ain a socket descriptor	
	MVC MVC MVC XC	MSG2D,MSG2C2 AF,=F'2' SOCTYPE,=F'1' PROTO,PROTO	Move 'SOCKET' to message Address Family = Internet Type = Stream Sockets Clear protocol field	

CALL EZASOKET. EZASOKET, Issue SOCKET Call (SOCKET,AF,SOCTYPE,PROTO,ERRNO,RETCODE), Х Х Specify variable parameter list Check for sucessful call Is it less than zero Yes, go display error and terminat L R6, RETCODE R6,=F'0' SOCERR С BL AIF (NOT &TRACE).TRACE02 TRACE ENTRY FOR SOCKET TRACE TYPE = 2 LA R6,MSG Put text address in R6 MSGLEN,=AL2(MSGTL) Put length of text in msg hdr. TEXT=(R6), Write message to operator MVC WTO Х MF=(E,WTOLIST) .TRACE02 ANOP * Get socket descriptor number * R6, RETCODE Descriptor number returned STH R6, SOCDESC Save it Issue CONNECT Command to Connect to Server MVC SSOCAF,=H'2' SSOCPORT,SERVPORT Move Port Number SSOCINET,SERVIADD Move Internet Address of Server MSG2D.MSG2C4 Move 'CONNECT' to message SSOCAF,=H'2' Set AF=INET MVC MVC MVC EZASOKET, Issue CONNECT Call (CONNECT,SOCDESC,SERVSOC,ERRNO,RETCODE), CALL EZASOKET, Х Х **VL** Specify variable parameter list R6, RETCODE Check for sucessful call Is it less than zero L R6,=F'0' SOCERR С Yes, go display error and terminat BL AIF (NOT &TRACE).TRACE04 TRACE ENTRY FOR CONNECT TRACE TYPE = 4 AIF R6,MSG Put text address in R6 MSGLEN,=AL2(MSGTL) Put length of text in msg hdr. TEXT=(R6), Write message to operator LA MVC Х WTO MF=(E,WTOLIST) .TRACE04 ANOP Send initial message to server * * BUFFER(L'MSG1),MSG1 Move Message to Buffer R6,L'MSG1 Get length of message R6,DATALEN Put length in data field MSG2D,MSG2C5 Move 'WRITE' to message MVC LA ST MVC MSG2D,MSG2C5 Issue WRITE Call CALL EZASOKET (WRITE, SOCDESC, DATALEN, BUFFER, ERRNO, RETCODE), Х VI R6,RETCODE R6,=F'0' Check for sucessful call Is it less than zero ΒL SOCERR Yes, go display error and terminat AIF (NOT &TRACE).TRACE05 TRACE ENTRY FOR WRITE TRACE TYPE = 5 MVC MSGLEN,=AL2(MSGTL+18) Put length of text in msg hdr. MVC MSG3D,ERR3C 'RETCODE= ' Move sign MVI MSG3S,C'+' R6, RETCODE Get return code value CVD R6, DWORK MSG4D, DWORK+4(4) Convert it to decimal Unpack it UNPK Correct the sign 0I MSG4D+6,X'F0' Put text address in R6 LA R6,MSG WTO TEXT=(R6) Write message to operator Х MF=(E,WTOLIST) .TRACE05 ANOP * Read response to initial message * Move 'READ' to message Get length of buffer Put length in data field MSG2D,MSG2C6 R6,L'BUFFER MVC. LA R6, DATALEN ST EZASOKET EZASOKET, Issue READ Call (READ,SOCDESC,DATALEN,BUFFER,ERRNO,RETCODE), CALL X X VI Specify variable parameter list R6, RETCODE Check for sucessful call C NO.F. BL SOCERR AIF (NOT &TRACE).TRACE06 TRACE ENTRY FOR READ TRACE TYPE = 6 MVC MSGLEN,=AL2(MSGTL+18) Put length of text in msg hdr. MCC3D FRR3C ' RETCODE= ' Is it less than zero R6,=F'0' Yes, go display error and terminat R6, RETCODE Get return code value R6,DWORK MSG4D,DWORK+4(4) CVD Convert it to decimal Unpack it UNPK OI MSG4D+6,X'F0' Correct the sign

	LA WTO	R6,MSG TEXT=(R6),	Put text address in R6 Write message to operator	Х
.TRACE06	ANOP	MF=(E,WTOLIST)		
* *	Send	second message to server		
*	MVC LA ST MVC	BUFFER(L'MSG2),MSG2 R6,L'MSG2 R6,DATALEN MSG2D,MSG2C5	2 Move Message to Buffer Get length of message Put length in data field Move 'WRITE' to message	
*	CALL	EZASOKET, (WRITE,SOCDESC,DATA VL	Issue WRITE Call LLEN,BUFFER,ERRNO,RETCODE),	X X
* * TRACE	L C BL AIF ENTRY MVC MVC MVI L CVD UNPK OI LA	R6,RETCODE R6,=F'0' SOCERR (NOT &TRACE).TRACE1 FOR WRITE TRACE TY MSGLEN,=AL2(MSGTL+1 MSG3D,ERR3C MSG3S,C'+' R6,RETCODE R6,DWORK MSG4D,DWORK+4(4) MSG4D+6,X'F0' R6,MSG TEVT-(BC)	Check for sucessful call Is it less than zero Yes, go display error and terminat IS (PE = 5 18) Put length of text in msg hdr. 'RETCODE= ' Move sign Get return code value Convert it to decimal Unpack it Correct the sign Put text address in R6	
.TRACE15	ANOP L C BI	MF=(E,WTOLIST) R6,RETCODE R6,=F'0' SOCEBR	Check for sucessful call Is it less than zero Yes go display error and terminat	~
*	Read	response to second m	nessage	
*	MVC	MSG2D,MSG2C6	Move 'READ' to message	
*	CALL	EZASOKET, (READ,SOCDESC,SOCMS VL	Issue READ Call GL,BUFFER,ERRNO,RETCODE), Specify variable parameter list	X X
*	L C BL	R6,RETCODE R6,=F'0' SOCERR	Check for sucessful call Is it less than zero Yes, go display error and terminat	
* TRACE	AIF ENTRY MVC MVC L CVD UNPK OI LA WTO	<pre>(NOT &TRACE).TRACE1 FOR READ TRACE TYF MSGLEN,=AL2(MSGTL+1 MSG3D,ERR3C MSG3S,C'+' R6,RETCODE R6,DWORK MSG4D,DWORK+4(4) MSG4D+6,X'F0' R6,MSG TEXT=(R6), ME=(E WTOLTST)</pre>	L6 PE = 6 L8) Put length of text in msg hdr. ' RETCODE= ' Move sign Get return code value Convert it to decimal Unpack it Correct the sign Put text address in R6 Write message to operator	x
.TRACE16	ANOP			
*	Send	End message to serve	er	
ч	MVC LA ST MVC	BUFFER(L'ENDMSG),EM R6,L'ENDMSG R6,SOCMSGL MSG2D,MSG2C5	NDMSG Move end message to buffer Get length of message Put length in length field Move 'WRITE' to message	
^	CALL	EZASOKET, (WRITE,SOCDESC,SOCM VL	Issue WRITE Call ISGL,BUFFER,ERRNO,RETCODE),	X X
* TRACE	L BL AIF ENTRY MVC MVC MVI L CVD UNPK OI	R6,RETCODE R6,=F'0' SOCERR (NOT &TRACE).TRACEZ FOR WRITE TRACE TY MSGLEN,=AL2(MSGTL+2 MSG3S,C'+' R6,RETCODE R6,DWORK MSG4D,DWORK+4(4) MSG4D+6,X'F0'	Check for sucessful call Is it less than zero Yes, go display error and terminat 25 (PE = 5 (8) Put length of text in msg hdr. ' RETCODE= ' Move sign Get return code value Convert it to decimal Unpack it Correct the sign	
.TRACE25	LA WTO ANOP	R6,MSG TEXT=(R6), MF=(E,WTOLIST)	Put text address in R6 Write message to operator	х
* *	Read	response to end mess	sage	
*	MVC	MSG2D, MSG2C6	Move 'READ' to message	

CALL EZASOKET, Issue READ Call Х (READ, SOCDESC, SOCMSGL, BUFFER, ERRNO, RETCODE), Х Specify variable parameter list * Check for sucessful call Is it less than zero Yes, go display error and terminat Т R6, RETCODE R6,=F'0' SOCERR С BL AIF (NOT &TRACE).TRACE26 ENTRY FOR READ TRACE TYPE = 6 MVC MSGLEN,=AL2(MSGTL+18) Put length of text in msg hdr. TRACE MSG3D, ERR3C MVC RETCODE= MSG3S,C'+ Move sign MVI R6, RETCODE Get return code value CVD R6, DWORK Convert it to decimal MSG4D, DWORK+4(4) UNPK Unpack it 0T MSG4D+6,X'F0' Correct the sign Put text address in R6 LA R6,MSG TEXT=(R6) Write message to operator WTO Х MF=(E,WTOLIST) .TRACE26 ANOP Close socket * * MVC MSG2D, MSG2C7 Move 'CLOSE' to message EZASOKET, Issue CLOSE Call (CLOSE,SOCDESC,ERRNO,RETCODE), CALL EZASOKET X X Specify variable parameter list VL R6, RETCODE Check for sucessful call C BL R6,=F'0' SOCERR Is it less than zero Yes, go display error and terminat AIF (NOT &TRACE).TRACE07 TRACE ENTRY FOR CLOSE TRACE TYPE = 7 R6,MSG Put text address in R6 MSGLEN,=AL2(MSGTL) Put length of text in msg hdr. TEXT=(R6), Write message to constant LA MVC Х WTO MF=(E,WTOLIST) .TRACE07 ANOP * Terminate Connection to API * CALL EZASOKET, Issue TERMAPI Call Х (TERMAPI), х ÝL Specify variable parameter list Issue console message for task termination * Move 'Ended' to message MVC MSG2D, MSG2CE Put text address in R6 LA R6,MSĠ MSGLEN,=AL2(MSGTL) Put length of text in msg hdr. MVC. WTO TEXT=(R6), Write message to operator Х MF=(E,WTOLIST) Return to Caller L R13, SOCSAVEL LM R14,R12,12(R13) BR R14 * Write error message to operator and ABENDS0C1 SOCERR Write error message to operator 'IMSTCPCL, TASK #' DS ΘH MVC ERR1D, MSG1D Move task number to message Call Type ERRTD, MSGTD MVC ERR2D, MSG2D MVC MVC ERR3D, ERR3C RETCODE= ERR3S,C'-' ERR5D,ERR5C Move sign which is always minus ' ERRNO= ' MVT MVC R6, RETCODE Get return code value CVD R6, DWORK Convert it to decimal UNPK ERR4D, DWORK+4(4) Unpack it ΟI ERR4D+6,X'F0' Correct the sign R6,ERRNO Get errno value L CVD Convert it to decimal R6, DWORK ERR6D, DWORK+4(4) Unpack it UNPK ERR60+6,X'FO' Correct the sign R6,ERR Put text address in R6 ERRLEN,=AL2(ERRTL) Put length of text in msg hdr. ΟI I A MVC $TEXT = (\dot{R}6)$, WTO Write message to operator Х MF=(E,WTOLIST) ABEND DS ΘH H'0' DC Force ABEND WTOPROT TEXT=, List form of WTO Macro WTO Х MF=L Length of WTO Prototype WTOPROTL EQU *-WTOPROT CL17'IMSTCPCL, TASK # CL8' STARTED' MSG1C DČ MSG2CS DC CL8' ENDED ' CL10' RETCODE= MSG2CE DC. FRR3C DC CL8' ERRNO= ERR5C DC MSG2C1 DC CL8' INITAPI

NGECC DC CLB GTGSTLD NSG2205 DC CLB'GTGSTLD NSG2205 DC CL16'CLIENT MESSAGE 1' First msg to server NSG1 DC CL16'CLIENT MESSAGE 2' 2nd msg to server ENDNSG DC CL3'CLDY MESSAGE 2' 2nd msg to server DC CL3'END' End indicator for SRV1 DC CL3''ND' End for high-level languages SOCSAVE DS F Word for high-level languages SOCSAVED S F Reg 14 SOCSAV15 DS F Reg 15 SOCSAV15 DS F Reg 15 SOCSAV10 DS F Reg 1 SOCSAV1 DS F Reg 1 SOCSAV2 DS F Reg 3 SOCSAV3 DS F Reg 3 SOCSAV3 DS F Reg 3 SOCSAV3 DS F Reg 3 SOCSAV3 DS F Reg 4 SOCSAV3 DS F Reg 1 SOCSAV3 DS F Reg 10 SOCSAV1 DS F Reg 10 SOCSAV1 DS F Reg 13 MAXSOC DS H Maximum number of sockets for this X application SOCTASKC DS CL8 Character task identifier SOCSAV1 DS F Reg 13 MAXSOC DS H Maximum number of sockets for this X application SOCTASKC DS CL8 Character task identifier SOCSAV1 DS F Neg 13 MAXSOC DS H Maximum number of socket call SOCTYFE DS F New Socket Address family for socket call SOCTYFE DS F New Socket Address for Server SSOCPAF DS H Address family of Server = 2 SSOCPAF DS H Address family of Server = 2 SSOCPAF DS H Port number for Server SSOCPAF DS H End of message MSGL DS CL17 '' MSTCPCL, TASK #' MSG3D DS CL10 '' RETCODE = ' MSG3D DS CL10 '' RETCODE = ' MSG3D DS CL17 '' RSTCPCL. TASK #' SG3D DS CL17 '' INSTCPCL, TASK #' SG3D DS CL17 '' INSTCPCL. TASK #' SG3D DS CL17 '' INSTCPCL. TASK #'
MSG1DCCl16'CLIENT MESSAGE 1'First msg to serverMSG2DCCl16'CLIENT MESSAGE 2'2nd msg to serverDCCL3'END'End Message for ServerDCCL45''Pad with blanksDSOFPROGRAM STORAGESOCSTG DSOFPROGRAM STORAGESOCSAVE DSFWord for high-level languagesSOCSAVE DSFAddress of previous save areaSOCSAVED DSFReg 14SOCSAVID DSFReg 14SOCSAVID DSFReg 1SOCSAVID DSFReg 1SOCSAVID DSFReg 1SOCSAVID DSFReg 1SOCSAVID DSFReg 1SOCSAVID DSFReg 1SOCSAVID DSFReg 3SOCSAVID DSFReg 4SOCSAVID DSFReg 7SOCSAVID DSFReg 7SOCSAVID DSFReg 10SOCSAVID DSFReg 11SOCSAVID DSFReg 11SOCSAVID DSFReg 11SOCSAVID DSFReg 11SOCSAVID DSFReg 12SOCSAVID DSFReg 13MAXSOCDSFReg 13MAXSOCSFNew socket number of sockets for this X applicationSOCTAKKC DSCLBCharacter task identifierSOCSAVID SFReg 13MAXSOCSFReg 14SOCTAKKC DSCLBCharacter task identifierSOCSAV
ENDMSGUS0C(L48End Message for ServerDCCL3'END'End indicator for SRV1DCCL45''Pad with blanksS090PROGRAM STORAGESOCSAVE DS0FSave AreaSOCSAVE DSFWord for high-level languagesSOCSAVED SFAddress of previous save areaSOCSAVED SFReg 14SOCSAVIA DSFReg 14SOCSAVIA DSFReg 1SOCSAVIA DSFReg 1SOCSAVIA DSFReg 1SOCSAVIA DSFReg 1SOCSAVIA DSFReg 1SOCSAVIA DSFReg 1SOCSAVIA DSFReg 3SOCSAVA DSFReg 4SOCSAVA DSFReg 6SOCSAVA DSFReg 10SOCSAVA DSFReg 11SOCSAVA DSFReg 12SOCSAV1 DSFReg 12SOCSAV1 DSFReg 13MAXSOCDSFReg 13MAXSOC DSHSocket Descriptor NumberHISOCSFHighest socket descriptor availableAFDSFAddress family for socket callSOCTASKC DSGHSocket Address for ServerSSOCAPORT DSFType of socketSSOCTAVE DSFTermet address for ServerSSOCAVE DSFReservedMSSOLOUS DSFReservedMSGDSFReservedMSGDSCL
DSDOPROGRAM STORAGESOCSAVE DSOFSave AreaSOCSAVED DSFWord for high-level languagesSOCSAVED DSFAddress of previous save areaSOCSAVED DSFReg 14SOCSAVID DSFReg 15SOCSAVID DSFReg 1SOCSAVID DSFReg 1SOCSAVID DSFReg 1SOCSAVID DSFReg 1SOCSAVID DSFReg 3SOCSAVID DSFReg 3SOCSAVID DSFReg 4SOCSAVID DSFReg 5SOCSAV3 DSFReg 6SOCSAV1D DSFReg 7SOCSAV1D DSFReg 10SOCSAV1D DSFReg 10SOCSAV1D DSFReg 11SOCSAV1D DSFReg 12SOCSAV1D DSFReg 13MAXSOCDSHMAXSOC DSHMaximum number of sockets for this X applicationSOCTASKC DSCL8Character task identifierSOCTASKC DSFNew socket numberMSDSFNew socket callSERVAL DS12FAlias array for serverSSOCAPORT DSHPort number for socket callSERVAL DS12FAlias array for serverSSOCTAPEDSFReservedMSDSCL17'IMSTCPCL, TASK #'MSG1DDSCL17'IMSTCPCL, TASK #'SG2DSCL10'Return codeRSG2DS
SUCSAVE DSOFSave AreaSOCSAVEL DSFWord for high-level languagesSOCSAVEL DSFAddress of previous save areaSOCSAVEH DSFReg 14SOCSAVD DSFReg 14SOCSAVD DSFReg 1SOCSAVD DSFReg 1SOCSAVD DSFReg 1SOCSAVD DSFReg 1SOCSAVD DSFReg 1SOCSAVD DSFReg 3SOCSAVD DSFReg 4SOCSAVD DSFReg 6SOCSAVD DSFReg 7SOCSAVD DSFReg 10SOCSAV1 DSFReg 10SOCSAV1D DSFReg 11SOCSAV1D DSFReg 11SOCSAV1D DSFReg 13MXSOC DSHMaximum number of sockets for this X applicationSOCTASKC DSCL8Character task identifierSOCDESC DSHSocket Descriptor NumberHISOC DSFType of socket callSOCTASKC DSCL8Character task identifierSOCTASKC DSFNew socket number for socket callSOCTASKC DSFNew socket number for socket callSOCTASKC DSFNew socket number for ServerSOCAF DSFNew socket number for ServerSOCAF DSHAddress family of Server = 2SOCAF DSHAddress family of Server = 2SOCAF DSHLength of messageMSG1DDSCL17Task NumberMSG2D
SOCSAVEH DSFAddress of next save areaSOCSAV14 DSFReg 14SOCSAV15 DSFReg 1SOCSAV2 DSFReg 1SOCSAV2 DSFReg 1SOCSAV3 DSFReg 3SOCSAV4 DSFReg 4SOCSAV4 DSFReg 4SOCSAV4 DSFReg 5SOCSAV4 DSFReg 6SOCSAV4 DSFReg 7SOCSAV4 DSFReg 7SOCSAV4 DSFReg 10SOCSAV7 DSFReg 10SOCSAV10 DSFReg 11SOCSAV12 DSFReg 12SOCSAV10 DSFReg 12SOCSAV12 DSFReg 13MAXSOC DSHMaximum number of sockets for this X applicationSOCTASKC DSCL8Character task identifierSOCTASKC DSFAddress family for socket callSOCTYPE DSFNew socket number for socket callSOCTYPE DSFNew socket number for socket callSCTYPE DSFInternet address for ServerSSOCPART DSHAddress family of Server = 2SSOCPART DSHAddress for ServerSSOCPART DSHLength of messageMSG1DDSCL17'IMSTCPCL, TASK #'MSG3DDSCL10'RETCODE = 'MSG4DDSCL17'REtron message areaERRLEN DSHLength of messageERRLEN DSHLength of messageERRLEN DSH<
SOCSAV15 DSFReg 15SOCSAV0 DSFReg 0SOCSAV1 DSFReg 1SOCSAV3 DSFReg 1SOCSAV3 DSFReg 3SOCSAV4 DSFReg 4SOCSAV5 DSFReg 6SOCSAV6 DSFReg 7SOCSAV8 DSFReg 10SOCSAV9 DSFReg 11SOCSAV9 DSFReg 11SOCSAV1 DSFReg 12SOCSAV1 DSFReg 13MAXSOC DSHMaximum number of sockets for this X applicationSOCTASKC DSCL8Character task identifierSOCDESC DSHSocket Descriptor NumberHISOCFNew socket number for socket callSOCTYPE DSFNew socket number for socket callSOCTYPE DSFNew socket number for socket callSOCTYPE DSFNew socket number for socket callSSOCAF DSHAddress Family of ServerSSOCAF DSHAddress for ServerSSOCAF DSHAddress for ServerSSOCAF DSHLength of messageMSGLEN DSHLength of messageMSG1D DSCL17'INSTCPCL, TASK #'MSG2D DSCL10'RetrondMSG3D DSCL17'RetrondMSG3D DSCL17'RetrondMSG3D DSCL17'INSTCPCL, TASK #'MSG3D DSCL17'INSTCPCL, TASK #'MSG4D DSCL17'INSTCPCL, TASK #'MSG4D DSCL17'I
SOCSAV1DSFReg 1SOCSAV3DSFReg 3SOCSAV4DSFReg 4SOCSAV4DSFReg 5SOCSAV7DSFReg 6SOCSAV7DSFReg 7SOCSAV10DSFReg 10SOCSAV10DSFReg 11SOCSAV10DSFReg 11SOCSAV11DSFReg 12SOCSAV12DSFReg 13MAXSOCDSHMaximum number of sockets for this X applicationSOCTASKC DSCL8Character task identifierSOCESC DSHSocket Descriptor NumberHISOCDSFHighest socket descriptor availableAFDSFNew socket number for socket callSOCTYPEDSFNew socket number for socket callSOCTAFKDS12FAlias array for serverSSOCAFDSHPort number for ServerSSOCPORTDSHPort number for ServerSSOCPORTDSHPort number for ServerSSOCINETDSFInternet address for ServerSSOCINETDSHLength of messageMSGLDDSCL17'IMSTCPCL, TASK #'MSG2DDSCL10'RetrodeMSG3DDSCL10'RetrodeRRDSCL7Return codeERRDSHLength of messageRSG3DSCSign which is always -
SOCSAV4DSFReg 4SOCSAV5DSFReg 5SOCSAV6DSFReg 7SOCSAV7DSFReg 7SOCSAV9DSFReg 9SOCSAV10DSFReg 10SOCSAV10DSFReg 11SOCSAV12DSFReg 12SOCSAV13DSFReg 13MAXSOCDSHMaximum number of sockets for this XapplicationSOCTASKC DSCL8Character task identifierSOCTASKC DSCL8Character task identifierSOCTASKC DSFHighest socket descriptor availableAFDSFAddress family for socket callSOCTYPE DSFNew socket number for socket callSERVALDS12FAlias array for serverSSOCAFDSHAddress Family of ServerSSOCAFDSHAddress for ServerSSOCAFDSHAddress for ServerSSOCAFDSHAddress for ServerSSOCAFDSHAddress for ServerSSOCAFDSHLength of messageMSG1DDSCL17'IMSTCPCL, TASK #'MSG3DDSCL10'RETCODE = 'MSG3DDSCL10'RETCODE = 'MSG3DDSCL10'RETCODE = 'MSG3DDSCL17Tressage areaERR DSOFError message areaERRDDSHLength of message
SUCSAVODSFReg 0SUCSAV7DSFReg 7SUCSAV7DSFReg 10SUCSAV10DSFReg 10SUCSAV10DSFReg 11SUCSAV12DSFReg 12SUCSAV13DSFReg 13MAXSOCDSHMaximum number of sockets for this X applicationSUCTASKCDSCL8Character task identifierSUCTASKCDSFHighest socket descriptor availableAFDSFAddress family for socket callSUCTYPEDSFNew socket number for serverSERVSOCDSOFSocket Address for ServerSSOCAFDSHAddress Family of Server = 2SSOCPORTDSHPort number for ServerSSOCINETDSFInternet address for ServerSOCINETDSHLength of messageMSGDSOFMessage areaMSG1DDSCL17'IMSTCPCL, TASK #'MSG2DDSCL10'RETCODE = 'MSG3DDSCL10'RETCODE = 'MSG3DDSCL17Return codeERRDSHLength of messageRERLENDSHLength of messageRERLENDS<
SOCSAV9DSFReg9SOCSAV10DSFReg10SOCSAV11DSFReg11SOCSAV12DSFReg12SOCSAV13DSFReg13MAXSOCDSHMaximum number of sockets for thisXapplicationSOCTASKCDSCL8Character task identifierSOCDESCDSHSocket Descriptor NumberHISOCDSFHighest socket descriptor availableAFDSFAddress family for socket callSOCTYPEDSFNew socket number for socket callSOCTYPEDSFNew socket number for socket callSERVALDS12FAlias array for serverSERVSOCDS0FSocket Address of ServerSSOCINETDSHPort number for ServerSSOCINETDSFInternet address for ServerSSOCINETDSFInternet address for ServerSSOCINETDSHLength of messageMSGDSCL17'IMSTCPCL, TASK #'MSG1DDSCL17'RETCODE = 'MSG2DDSCL10'RETCODE = 'MSG3SDSCL7Return codeERRDS0FError message areaERRLENDSHLength of messageERRLENDSHLength of messageERRLENDSHLength of messageERRLENDS
SUCSAVIL DSFReg11SUCSAVI2 DSFReg12SUCSAV12 DSFReg13MAXSOCDSHMaximum number of sockets for thisXapplicationapplicationSUCTASKC DSCL8Character task identifierSUCDESCDSHSocket Descriptor NumberHISOCDSFHighest socket descriptor availableAFDSFAddress family for socket callSUCTYPEDSFType of socketNSDSFNew socket number for socket callSERVALDS12FAlias array for serverSERVALDSHAddress Family of Server = 2SSOCAFDSHPort number for ServerSSOCINETDSFInternet address for ServerSSOCINETDSFInternet address for ServerSSOCINETDSFInternet address for ServerSSOCINETDSHLength of messageMSGDSCL17'IMSTCPCL, TASK #'MSG2DDSCL8Last part of messageMSG3DDSCL10'RETCODE = 'MSG3DDSCL17Return codeERRDS0FError message areaERRLENDSHLength of messageERRLENDSHLength of messageERRLENDSCL17'IMSTCPCL, TASK #'
MAXSOCDSHMaximum number of sockets for thisX applicationSOCTASKC DSCL8Character task identifierSOCDESCDSHSocket Descriptor NumberHISOCDSFHighest socket descriptor availableAFDSFAddress family for socket callSOCTYPEDSFNew socket number for socket callSOCTYPEDSFNew socket number for socket callSERVALDS12FAlias array for serverSERVSOCDSOFSocket Address of ServerSSOCINETDSHAddress Family of Server = 2SSOCINETDSFInternet address for ServerDCD'0'ReservedMSG1DDSCL17'IMSTCPCL, TASK #'MSG2DDSCL10'RETCODE = 'MSG3DDSCL10'RETCODE = 'MSG4DDSCL7Return codeERRDS0FError messageERRLENDSHLength of messageERRLENDSCL17'IMSTCPCL, TASK #'
SOCHASKC DSCLOCharacter task IdentifierSOCDESC DSHSocket Descriptor NumberHISOC DSFHighest socket descriptor availableAFDSFAddress family for socket callSOCTYPE DSFType of socketNSDSFNew socket number for socket callSERVALDS12FAlias array for serverSERVSOC DSOFSocket Address of ServerSSOCPORT DSHPort number for ServerSSOCINET DSFInternet address for ServerDCD'0'ReservedMSG1DDSCL17MSG1DDSCL17MSG2DDSCL17MSG2DDSCL18MSG2DDSCL10MSG3DDSCL10MSG4DDSCL7MSG4DDSCL7RERDSOFHSG4DDSCL7Heurn codeERRDSHLength of messageERRLENDSHLength of messageERRLENDSHLength of messageERRLENDSHLength of messageERRLENDSHLength of messageERRLDDSHLength of messageERRLENSHLength of messageERRLDSHLength of messageERRLENSHLength of messageERRLENS <t< td=""></t<>
AFDSFAddress family for socket callSOCTYPEDSFType of socketNSDSFNew socket number for socket callSERVALDS12FAlias array for serverSERVSOCDS0FSocket Address of ServerSSOCAFDSHAddress Family of Server = 2SSOCPORTDSHPort number for ServerSSOCINETDSFInternet address for ServerDCD'0'ReservedMSGDSOFMSG1DDSCL17MSG2DDSCL5MSG2DDSCL8MSG3DDSCL10MSG3DDSCL10MSG3DDSCL10MSG4DDSCL7MSG4DDSCL7RERDSHLength of messageRERDSHLength of messageRERDSCLYMSG4DDSCL7Return codeERRDSHLength of messageERRLENDSHLength of messageERRLENDSHLength of messageERRLENDSHLength of messageERRLDDSCL17'IMSTCPCL, TASK #'
NS DS F New Socket Hinder Tot Socket Call SERVAL DS 12F Alias array for server SERVSOC DS 0F Socket Address of Server SERVSOC DS 0F Socket Address of Server SSOCAF DS H Address Family of Server = 2 SSOCINET DS H Port number for Server SSOCINET DS F Internet address for Server DC D'0' Reserved MSG DS 0F Message area MSG1D DS CL17 'IMSTCPCL, TASK #' MSG2D DS CL8 Last part of message MSG2D DS CL8 Last part of message MSG3D DS CL10 ' RETCODE = ' MSG3D DS CL10 ' Return code ERR DS 0F Error message area ERRLEN DS H Length of message ERRLEN DS CL7 Return code ERR DS 0F Error message area ERRLEN DS H
SSOCAFDSHAddress Family of Server = 2SSOCPORT DSHPort number for ServerSSOCINET DSFInternet address for ServerDCD'0'ReservedMSGDSOFMessage areaMSGLENDSHLength of messageMSGTDDSCL17'IMSTCPCL, TASK #'MSGTDDSCL5Task NumberMSG2DDSCL8Last part of messageMSGTLEQU*End of messageMSG3DDSCL10'RETCODE = 'MSG3SDSCL7Return codeERRDSOFError message areaERRLENDSHLength of messageERRLDDSCL17'RETCODE = 'MSG4DDSCL17Return codeERRDSOFError message areaERRLENDSHLength of messageERRLENDSHLength of messageERRLDDSCL17'IMSTCPCL, TASK #'
Social Post DC D'o' Reserved MSG DS OF Message area MSGLEN DS H Length of message MSG1D DS CL17 'IMSTCPCL, TASK #' MSG2D DS CL5 Task Number MSG2D DS CL8 Last part of message MSG1L EQU * End of message MSG3D DS CL10 ' RETCODE = ' MSG4D DS CL7 Return code ERR DS OF Error message area ERRLEN DS H Length of message ERRLEN DS H Length of message ERRLD DS CL7 Return code ERRLEN DS H Length of message ERRLEN DS H Length of message ERRLIN DS H Length of message
MSGLEN DS H Length of message MSG1D DS CL17 'IMSTCPCL, TASK #' MSG7D DS CL5 Task Number MSG2D DS CL8 Last part of message MSG1L EQU * End of message MSG3D DS CL10 Length of message text MSG3D DS CL10 'RETCODE = ' MSG4D DS CL7 Return code ERR DS OF Error message area ERRLEN DS H Length of message ERRLD DS CL17 'IMSTCPCL, TASK #'
MSG2D DS CL8 Last part of message MSGE EQU * End of message MSGTL EQU MSGE-MSG1D Length of message MSG3D DS CL10 ' RETCODE = ' MSG3S DS C Sign which is always - MSG4D DS CL7 Return code ERR DS 0F Error message area ERRLEN DS H Length of message ERR1D DS CL17 ' IMSTCPCL.
MSGTL EQU MSGE-MSG1D Length of message text MSG3D DS CL10 'RETCODE = ' MSG3S DS C Sign which is always - MSG4D DS CL7 Return code ERR DS OF Error message area ERRLEN DS H Length of message ERR1D DS CL17 'IMSTCPCL.
MSG4D DS CL7 Refurn code ERR DS 0F Error message area ERRLEN DS H Length of message ERR1D DS CL17 'IMSTCPCL. TASK #'
ERRLEN DS H Length of message ERR1D DS CL17 'IMSTCPCL. TASK #'
FRRTD DS CL5 Task Number
ERR2D DS CL8 Last part of message ERR3D DS CL10 'RETCODE = '
ERR35 DS C Sign which is always - ERR4D DS CL7 Return code FRR5D DS CL8 ' FRRNO ='
ERR6D DS CL7 Error number ERRE EQU * End of message
ERFIL EQU ERRE-ERRID Length of message text BUFFER DS CL(BUFLEN) Socket I/O Buffer DATALEN DS F Length of buffer data
DWORK DS D Double word work area RECNO DS PL4 Record Number
RETCODE DS F Return code from call PROTO DS F Protocol field for socket
ASIDENT DS OF Address space identifier for initapi ASTCPNAM DS CL8 Name of TCP/IP Address Space
ASCLNAME DS CL8 Our name as known to TCP/IP WTOLIST DS CL(WTOPROTL) List form of WTO Macro
SOCSTGE EQU * End of Program Storage SOCSTGL EQU SOCSTGE-SOCSTG Length of Program Storage
RO EQU O R1 EQU 1
R2 EQU 2 R3 EQU 3 R4 F0U 4
R5 EQU 5 R6 EQU 6

R8	EQU	8
R9	EQU	9
R10	EQU	10
R11	EQU	11
R12	EQU	12
R13	EQU	13
R14	EQU	14
R15	EQU	15
GWABAR	EQU	13
	END	

Figure 86. Sample of IMS program as a client

Sample server program for IMS MPP client

EZASVAS3 CSECT EZASVAS3 AMODE ANY EZASVAS3 RMODE ANY GBLB &TRACE ASSEMBLER VARIABLE TO CONTROL TRACE GENERATION SETB 1 1=TRACE ON 0=TRACE OFF GBLB &SUBTR ASSEMBLER VARIABLE TO CONTROL SUBTRACE &TRACE &SUBTR SETB 0 1=SUBTRACE ON 0=SUBTRACE OFF * MODULE NAME: EZASVAS3 Licensed Materials - Property of IBM * Copyright: "Restricted Materials of IBM" 5694-A01 * Copyright IBM Corp. 2009 US Government Users Restricted Rights -Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp. * Status: CSV1R11 MODULE FUNCTION: Test module for Extended Sockets. This module accepts connection request from IMS client program named EZAIMSC3. * LANGUAGE: Assembler * ATTRIBUTES: Non-reusable * Change History: Flag Reason Release Date Origin Description \$Q1= D316.15 CSV1R5 020604 BKELSEY : Support 64K sockets \$F1= RBBASE CSV1R11 080612 Herr : Cleaned up >72 lines * SOC0000 DS ΘH USING *,R15 B SOC00100 Tell assembler to use reg 15 Branch to startup address CL14 'SERVEREYECATCH' DC Address Space Identifier for initapi Name of TCP/IP Address Space Our name as known to TCP/IP 0F ASIDENT DS CL8'TCPV3 ASTCPNAM DC CL8'CALLSRVER' ASCLNAME DC TIMEOUT DS TIMESEC DC 0F F'180' Timeout value for select Timeout value in seconds Timeout value in milliseconds TIMEMSEC DC F'0' BUFLEN EQU 1000 Set length of I/O buffers A(SOC0000+4096) R4BASE DČ Beginning of program Save callers registers Move base reg to R3 S0C00100 DS ΘH STM R14,R12,12(R13) LR R3, R15 R4, R4BASE Add R4 as second base reg Tell assembler to drop R15 as base DROP R15 USING SOC0000,R3,R4 Tell assembler to use R3 and R4 as X base registers R6,SOCSTG Clear program storage I A R7,SOCSTGL LA SR R14,R14 SR R15,R15 MVCL R6, R14 R13, SOCSAVEH Save address of higher save area ST R7, SOCSAVE Complete save area chain Tell caller where our save area is Point R13 at our save area LA R7,8(R13) R13,SOCSAVE ST I A MVT ENDSW, X'00' Clear end-of-transmission switch * Build message for console MVC MSG1D, MSG1C Initialize first part of message
MSGTD,=CL5'00000'Move subtask number from clientidMSG2D,MSG2CSMove 'Started' to messageR6,MSGPut text address in R6 MVC MVC LA MSGLER,=AL2(MSGTL) Put length of text in msg hdr. WTOLIST,WTOPROT Move prototype WTO to list for MVC MVC Move prototype WTO to list form WTO TEXT = (R6), Write message to operator Х MF=(E,WTOLIST) Issue INITAPI Call to connect to interface SOCTASKC,=CL8'TAS00000' Give subtask a name MSG2D,MSG2C00 Move 'INITAPI'to message MAXSOC,=AL2(50) Initialize MAXSOC parameter MVC MVC MVC CALL EZASOKET, Х (INITAPI, MAXSOC, ASIDENT, SOCTASKC, HISOC, ERRNO, Х RETCODE), χ VL R6, RETCODE Check for sucessful call R6,=F'0' С Is it less than zero SOCERR RI Yes, go display error and terminat AIF (NOT &TRACE).TRACE00 TRACE ENTRY FOR INITAPI TRACE TYPE = 0 R6,MSG Put text address in R6 LA MVC. MSGLEN,=AL2(MSGTL) Put length of text in msg hdr. TEXT=(R6), MF=(E,WTOLIST) WTO Write message to operator Х .TRACE00 ANOP Issue SOCKET Call to obtain socket to listen on Move 'SOCKET'to message MVC MSG2D, MSG2C25 AF,=F'2' SOCTYPE,=F'1' MVC Initialize AF to '2' (INET) Specify stream sockets MVC MVC PROTO,=F'0' Protocol is ignored for stream EZASOKET, Issue SOCKET CALL (SOCKET,AF,SOCTYPE,PROTO,ERRNO,RETCODE), CALL EZASOKET X X Т R6,RETCODE Check for sucessful call C R6 = F'0'Is it less than zero BL SOCERR Yes, go display error and terminate AIF (NOT &TRACE).TRACE25 TRACE ENTRY FOR SOCKET TRACE TYPE = 25 R6,MSG Put text address in R6 MSGLEN,=AL2(MSGTL) Put length of text in msg hdr. LA MVC TEXT=(R6), Write message to operator Х WTO MF=(E,WTOLIST) .TRACE25 ANOP R0,RETCODE Get descriptor number of socket STH R0,LISTSOC Save it * Issue GETHOSTID call to determine our internet address * * MVC MSG2D, MSG2C07 Move 'GETHSTID'to message CALL EZASOKET, Issue GETHOSTID Call Х (GETHSTID, RETCODE), VL AIF (NOT &TRACE).TRACE07 TRACE ENTRY FOR SOCKET TRACE TYPE = 07 R6,MSG Put text address in R6 MSGLEN,=AL2(MSGTL) Put length of text in msg hdr. LA MVC Write message to operator WTO TEXT = (R6), Х MF=(E,WTOLIST) .TRACE07 ANOP R0, RETCODE Get internet address of host ST R0,SINETADR Save it * Issue BIND call to establish port MSG2D,MSG2C02 SPORT,=H'5000' Move 'BIND' to message Move port number to structure Move AF (INET) to structure MVC. MVC SAF,=H'2' MVC EZASOKET, Issue BIND Call (BIND,LISTSOC,SOCKNAME,ERRNO,RETCODE), CALL EZASOKET X X VI R6,RETCODE Check for sucessful call Is it less than zero L C R6,=F'0' BL SOCERR Yes, go display error and terminat (NOT &TRACE).TRACE02 AIF TRACE ENTRY FOR BIND TRACE TYPE = 02 R6,MSG Put text address in R6 MSGLEN,=AL2(MSGTL) Put length of text in msg hdr. TEXT=(R6), Write message to operator LA MVC WTO Х MF=(E,WTOLIST) .TRACE02 ANOP *

Issue LISTEN call to establish backlog of connection requests * * MVC MSG2D, MSG2C13 Move 'LISTEN' to message Set backlog to 5 MVC BACKLOG, =F'5' EZASOKET, ISSUE LISTEN Call (LISTEN,LISTSOC,BACKLOG,ERRNO,RETCODE),VL R6.RETCODE Check for successful call CALL Х R6, RETCODE R6, =F'0' С Is it less than zero BL SOCERR Yes, go display error and terminate AIF (NOT &TRACE).TRACE13 TRACE ENTRY FOR LISTEN TRACE TYPE = 13 R6,MSG Put text address in R6 LA MSGLEN,=AL2(MSGTL) Put length of text in msg hdr. MVC. WTO TEXT = (R6), Write message to operator Х MF=(E,WTOLIST) .TRACE13 ANOP Issue SELECT call to wait on connection request MSG2D,MSG2C19 SELSOC,=F'31' Move 'SELECT' to message Maximum number of sockets MVC MVC Not checking for writes Not checking for exceptions WSNDMASK, =F'0' MVC ESNDMASK, =F'0' MVC LA R0,1 Put 1 in rightmost position of R0 Put listener socket number in R1 Create mask for read R1,LISTSOC R0,0(R1) I H SLL R0, RSNDMASK Put value in mask field ST EZASOKET, Issue SELECT Call (SELECT,SELSOC,TIMEOUT,RSNDMASK,WSNDMASK,ESNDMASK, RRETMASK,WRETMASK,ERETMASK,ERRNO,RETCODE), CALL X X VI L R6, RETCODE Check for sucessful call Is it less than zero С R6,=F'0' SOCERR BL Yes, go display error and terminat (NOT &TRACE).TRACE19 AIF TRACE ENTRY FOR SELECT TRACE TYPE = 19 LA R6,MSG Put text address in R6 MSGLEN,=AL2(MSGTL) Put length of text in msg hdr. MVC. TEXT=(R6). Х WTO Write message to operator MF=(E,WTOLIST) .TRACE19 ANOP Issue ACCEPT call to accept a new connection * MSG2D,MSG2C01 NS,=F'4' Move 'ACCEPT' to message MVC Use socket 4 for connection socket MVC CALL EZASOKET Issue ACCEPT Call Х (ACCEPT, LISTSOC, SOCKNAME, ERRNO, RETCODE), Х VI R6, RETCODE Check for sucessful call Is it less than zero L С R6,=F'0' ΒL SOCERR Yes, go display error and terminat AIF (NOT &TRACE).TRACE01 ENTRY FOR ACCEPT TRACE TYPE = 01 TRACE R6,MSG Put text address in R6 LA MVC. MSGLEN,=AL2(MSGTL) Put length of text in msg hdr. WTO TEXT=(R6), Write message to operator Х MF=(E,WTOLIST) .TRACE01 ANOP R0, RETCODE Get descriptor number of new socket STH R0, CONNSOC Save it for future use Issue READ call to get first message from client R6,L'BUFFER Get length of buffer LA Put length in data field Move 'READ' to message R6, DATALEN ST MVC MSG2D, MSG2C14 Clear the FLAGS field XC FLAGS, FLAGS CALL EZASOKET, Issue READ Call Х (READ, CONNSOC, DATALEN, BUFFER, ERRNO, RETCODE), VL Check for sucessful call Is it less than zero R6, RETCODE R6, =F'0' C BL SOCERR Yes, go display error and terminat ATF (NOT &TRACE).TRAC14A TRACE ENTRY FOR READ TRACE TYPE = 14 R6,MSG Put text address in R6 MSGLEN,=AL2(MSGTL) Put length of text in msg hdr. LA MVC TEXT = (R6), Write message to operator Х WTO MF=(E,WTOLIST) .TRAC14A ANOP * Send Initial Message to client to continue transaction * MVC. BUFFER(L'RESPMSG), RESPMSG Move Message to Buffer LA R6,L'RESPMSG Get length of message

Put length in data field Clear FLAGS field Move 'WRITE' to message ST R6,DATALEN FLAGS, FLAGS XC MVC MSG2D, MSG2C26 CALL EZASOKET Issue WRITE call Х (WRITE, CONNSOC, DATALEN, BUFFER, ERRNO, RETCODE), VL Check for sucessful call Is it less than zero R6, RETCODE R6,=F'0' SOCERR С BL Yes, go display error and terminat (NOT &TRACE).TRAC26A ATE TRACE ENTRY FOR WRITE TRACE TYPE = 22 MSG Put text address in R6 MSGLEN,=AL2(MSGTL) Put length of text in msg hdr. LA MVC TEXT=(R6), Write message to operator Х WTO MF=(E,WTOLIST) .TRAC26A ANOP S0C0300 DS ΘH Read Message from Client * Move 'READ' to message Get length of buffer Use it for data length MVC MSG2D,MSG2C14 R0,L'BUFFER LA ST R0, DATALEN FLAGS, FLAGS Clear FLAGS field XC CALL EZASOKET Х (READ, CONNSOC, DATALEN, BUFFER, ERRNO, RETCODE), VL Check for sucessful call Is it less than zero R6, RETCODE R6,=F'0' C SOCERR **BNH** Yes, go display error and terminat (NOT &TRACE).TRAC14B AIF TRACE ENTRY FOR RECV TRACE TYPE = 14 R6,MSG Put text address in R6 LA MVC MSGLEN,=AL2(MSGTL) Put length of text in msg hdr. TEXT=(R6), WTO Write message to operator Х MF=(E,WTOLIST) .TRAC14B ANOP BUFFER(3),=CL3'END' Was this last record CLC SOC0350 ENDSW,C'E' BNE No MVT Yes, set end-of-transmission switch S0C0350 DS ΘH Send Response to Client * * Move 'WRITE' to message Get message length from previous call Clear FLAGS field MVC MSG2D,MSG2C26 MVC DATALEN, RETCODE FLAGS, FLAGS XC CALL EZASOKET Х (WRITE, CONNSOC, DATALEN, BUFFER, ERRNO, RETCODE), VL R6,RETCODE L Check for sucessful call Is it less than zero R6,=F'0' С BNH SOCERR Yes, go display error and terminat AIF (NOT &TRACE).TRAC26B TRACE ENTRY FOR SEND TRACE TYPE = 26 R6,MSG Put text address in R6 MSGLEN,=AL2(MSGTL) Put length of text in msg hdr. TEXT=(R6), Write message to operator R6,MSG LA MVC WTO Х MF=(E,WTOLIST) .TRAC26B ANOP CLI ENDSW,C'E' Have we received last record BNE S0C0300 No, so go back and do another Close sockets * Move 'CLOSE1' to message MVC MSG2D, MSG2C03 * Issue CLOSE call for connection skt X CALL EZASOKET (CLOSE, CONNSOC, ERRNO, RETCODE), VL Check for sucessful call Is it less than zero R6, RETCODE L R6,=F'0' SOCERR С Yes, go display error and terminat ΒL AIF (NOT &TRACE).TRACE03 TRACE ENTRY FOR CLOSE TRACE TYPE = 3 LA R6,MSG Put te R6,MSG Put text address in R6 MSGLEN,=AL2(MSGTL) Put length of text in msg hdr. TEXT=(R6), Write message to operator MVC WTO Х MF=(E,WTOLIST) .TRACE03 ANOP * MVC MSG2D.MSG2C03A Move 'CLOSE2' to message EZASOKET, Issue CLOSE call for listen socket X (CLOSE,LISTSOC,ERRNO,RETCODE),VL CALL EZASOKET R6, RETCODE Check for sucessful call Is it less than zero Т С R6,=F'0' SOCERR BL Yes, go display error and terminat

(NOT &TRACE).TRAC103 AIF TRACE ENTRY FOR CLOSE TRACE TYPE = 3 R6,MSG Put text address in R6 MSGLEN,=AL2(MSGTL) Put length of text in msg hdr. LA MVC WTO TEXT = (R6), Write message to operator Х MF=(E,WTOLIST) .TRAC103 ANOP * Terminate Connection to API * CALL EZASOKET Х (TERMAPI), VL Issue console message for task termination * MSG2D,MSG2CEMove 'Ended' to messageR6,MSGPut text address in R6MSGLEN,=AL2(MSGTL)Put length of text in msg hdr. MVC IA MVC Write message to operator WTO TEXT = (R6), Х MF=(E,WTOLIST) * Return to Caller R13, SOCSAVEH IM R14, R12, 12(R13) BR R14 * Write error message to operator * SOCERR Write error message to operator DS ΘH MVC ERR1D,MSG1D 'SERVER, TASK #' Move task number to message Call Type ' RETCODE= ' ERRTD, MSGTD ERR2D, MSG2D MVC MVC ERR3D, ERR3C MVC ERR35,C'-' MVI Move sign which is always minus MVC ERR5D, ERR5C ' ERRNO= R6, RETCODE Get return code value CVD R6, DWORK Convert it to decimal UNPK ERR4D, DWORK+4(4) OI ERR4D+6, X'F0' Unpack it ΟI Correct the sign R6,ERRNÓ Get errno value Convert it to decimal CVD R6, DWORK ERR6D, DWORK+4(4) UNPK Unpack it ΟI ERR6D+6,X'F0' Correct the sign Put text address in R6 I A R6,ERR MVC ERRLEN,=AL2(ERRTL) Put length of text in msg hdr. WTO TEXT=(R6), Write message to operator Х MF=(E,WTOLIST) * * Return to Caller * * L. R13, SOCSAVEH * IM R14,R12,12(R13) BR R14 ABEND DS ΘH H'0' DC Force ABEND _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ Constants *----TEXT=, WTOPROT WTO List form of WTO Macro MF=L *-WTOPROT WTOPROTL EQU Length of WTO Prototype CL17'SERVER, CL8' STARTED' CL8' ENDED ' DČ DC TASK # MSG1C MSG2CS MSG2CE DC CL10' RETCODE= ERR3C DC CL8' ERRNO= CL8' INITAPI' CL8' ACCEPT' CL8' BIND ERR5C DC MSG2C00 DC MSG2C01 MSG2C02 DC DC MSG2C03 DC CL8' CLOSE CL8' CLOSE2 MSG2C03A DC CL8' GTHSTID MSG2C07 DC CL8' LISTEN MSG2C13 MSG2C14 DC CL8' READ DC MSG2C19 DC CL8' SELECT DC CL8' SOCKET MSG2C25 CL8' WRITE ' CL8' TAKESKT' MSG2C26 DC DC MSG2C32 CL50'FIRST RESPONSE FROM SERVER ' RESPMSG DC *----Constants used for call types * CL16'INITAPI' CL16'BIND' INITAPI DC DC BIND CL16 LISTEN CL16 ACCEPT LISTEN DC ACCEPT DC CL16 'READ' CL16 'SELECT' READ DC SELECT DC CL16'WRITE WRTTE DC SOCKET DC CL16'SOCKET' CL16'CLOSE' CLOSE DC GETHSTID DC CL16'GETHOSTID

TERMAPI	DC	CL16'TERMAPI'	
* *	Progra	am Storage Area	*
* SOCSTG SOCSAVE SOCSAVE SOCSAVE SOCSAVE SOCSAV14 SOCSAV15 SOCSAV15 SOCSAV15 SOCSAV15 SOCSAV2 SOCSAV2 SOCSAV2 SOCSAV2 SOCSAV3 SOCSAV4 SOCSAV5 SOCSAV5 SOCSAV6 SOCSAV10 SOCSAV10 SOCSAV11 S	DS DS DS DS DS DS DS DS DS DS DS DS DS D	0F 0F F F F F F F F F F F F F F F F F F	PROGRAM STORAGE Save Area Word for high-level languages Address of previous save area Address of next save area Reg 14 Reg 15 Reg 0 Reg 1 Reg 2 Reg 3 Reg 4 Reg 5 Reg 6 Reg 7 Reg 8 Reg 9 Reg 10 Reg 11 Reg 12 Reg 13 Address of parameter list Address of Global Work Area Address of Task Information Element Socket number used for listen Socket number used for listen Socket number created by accept Number of messages to be exchanged Length of messages to be exchanged Character task identifier Highest socket descriptor available
SERVLEN SERVLEN SERVAF SERVAF SERVIADD ENDSW MSG MSGLEN MSG1D MSG1D MSG1D MSG2	DS DS DS DS DS DS DS DS DS DS DS DS DS D	H OF H F C OF H CL17 CL5 CL8 * MSGE-MSG1D OF H CL17 CL5 CL8 CL17 CL5 CL8 CL17 CL5 CL8 CL17 CL5 CL8 CL17 CL5 CL8 * MSGE-MSG1D OF H CL17 CL5 CL8 * MSGE-MSG1D OF H CL17 CL5 CL8 * MSGE-MSG1D OF H CL17 CL5 CL8 * MSGE-MSG1D OF H CL17 CL5 CL17 CL5 CL8 * MSGE-MSG1D OF H CL17 CL5 CL17 CL5 CL17 CL5 CL17 CL5 CL17 CL5 CL17 CL5 CL17 CL5 CL17 CL5 CL17 CL5 CL17 CL5 CL17 CL5 CL17 CL5 CL17 CL5 CL17 CL5 CL17 CL5 CL17 CL5 CL17 CL17 CL5 CL17 CL5 CL17 CL5 CL17 CL5 CL17 CL5 CL17 CL5 CL17 CL5 CL17 CL5 CL17 CL5 CL17 CL5 CL17 CL5 CL17 CL5 CL10 CL10 CL7 CL5 CL10 CL7 CL5 CL10 CL7 CL5 CL17 CL5 CL10 CL7 CL5 CL7 CL5 CL17 CL5 CL10 CL7 CL5 CL7 CL5 CL7 CL5 CL7 CL5 CL7 CL5 CL7 CL7 CL5 CL7 CL5 CL7 CL7 CL5 CL7 CL7 CL7 CL5 CL7 CL7 CL7 CL7 CL7 CL7 CL7 CL7	Socket Address of Server Address Family of Server = 2 Port Address of Server Internet Address of Server End of transmission switch Message area Length of message 'SERVER, TASK #' Task Number Last part of message End of message Length of message text Error message area Length of message 'SERVER, TASK #' Task Number Last part of message 'SERVER, TASK #' Task Number Last part of message 'RETCODE = ' Sign which is always - Return code 'ERRNO =' Error number End of message text
* Nan	ne stru	ucture used by bind	*
SOCKNAME SAF SPORT SINETADR SOCKNAML CLIENTID CDOMAIN CNAME CSUBTASK	DS DS DS DS DS EQU DS DS DS DS	OF H H F D V *-SOCKNAME OF F CL8 CL8 CL8	Socket Name structure The address family of the socket The port number of this socket The internet address of this socket Reserved Length of SOCKNAME Structure Client Id structure The domain of this client (2) The major name of this client The minor (subtask) name of this X client
CLIENTL BUFFER DATALEN DWORK SENDINT RECNO AF NS SOCTYPE PROTO ERRNO RETCODE CINADDR CPORT MAXSOC SELSOC BACKLOG FLAGS RSNDMASK	US EQU DS DS DS DS DS DS DS DS DS DS DS DS DS	U *-CLIENTID CL(BUFLEN) F D PL4 F F F F F F F F F F F F F F F F F F F	Reserved Socket I/O Buffer Length of buffer data Double word work area Time interval for send Record Number Address family for socket call New socket number for socket call Protocol for socket call Protocol for socket call Protocol for socket call Error number returned from call Return code from call Internet address of client Port number of client Maximum # sockets for INITAPI Maximum # sockets for SELECT Backlog value for LISTEN FLAGS field for RECV and RECVFROM Read send mask for select

WSNDMASK ESNDMASK RRETMASK WRETMASK ERETMASK WTOLIST EZASMTI	DS DS DS DS DS EZASM	F F F CL(WTOPROTL) I TYPE=TASK,	Write send mask for select Exception send mask for select Read return mask for select Write return mask for select Exception return mask for select List form of WTO Macro	x
EZASMGW	EZASM	STORAGE=CSECT I TYPE=GLOBAL, STORAGE=CSECT	Generate task storage for interface Storage definition for GWA	Х
SOCSTGE SOCSTGL	EQU EQU	* SOCSTGE-SOCSTG	End of Program Storage Length of Program Storage	
R0 R1 R2 R3 R4 R5 R6 R7 R7 R7 R8 R9 R10 R11 R12 R13 R14 R15 GWABAR	EQU EQU EQU EQU EQU EQU EQU EQU EQU EQU	0 1 2 3 4 5 6 6 7 8 9 10 11 12 13 14 15 13		

Figure 87. Sample of IMS program as a server

WTO output from sample program

Client Output

$\begin{array}{ccccccc} 13.29.18 & JOB00084 \\ 13.29.18 & JOB00084 \\ 13.29.19 & JOB00084 \\ 13.29.19 & JOB00084 \\ 13.29.19 & JOB00084 \\ 13.29.19 & JOB00084 \\ 13.29.20 & JOB00084 \\ 13.29.41 & JOB00084 \\ 13.29.42 & JOB00084 \\ JAUVERPRPRPRPPRPPRPPPPPPPPPPPPPPPPPPPPPPPP$	IEF403I SO +SERVER, +SERVER, +SERVER, +SERVER, +SERVER, +SERVER, +SERVER, +SERVER, +SERVER, +SERVER, +SERVER, +SERVER, +SERVER, +SERVER, +SERVER,	DCCALLS - STARTED - TIME=13.29.18 TASK # 00000 STARTED TASK # 00000 INITAPI TASK # 00000 GOCKET TASK # 00000 GINSTID TASK # 00000 BIND TASK # 00000 SELECT TASK # 00000 ACCEPT TASK # 00000 READ TASK # 00000 CLOSE TASK # 00000 CLOSE	
13.29.42 J0B00084	+SERVER,	TASK # 00000 CLOSE	
13.29.42 J0B00084	+SERVER,	TASK # 00000 CLOSE2	
13.29.42 J0B00084	+SERVER,	TASK # 00000 ENDED	

Server Output

13.27.45	J0B00082	IEF403I MES	SSAGE	-	STARTE	ED - TIME	=13.27.45	5
13.29.40	J0B00082	+IMSTCPCL,	TASK	#	00000	STARTED		
13.29.41	J0B00082	+IMSTCPCL,	TASK	‡⊧	00000	INITAPI		
13.29.41	J0B00082	+IMSTCPCL,	TASK	#	00000	GTHSTID		
13.29.41	J0B00082	+IMSTCPCL,	TASK	#	00000	SOCKET		
13.29.41	J0B00082	+IMSTCPCL,	TASK	#	00000	CONNECT		
13.29.41	J0B00082	+IMSTCPCL,	TASK	#	00000	WRITE	RETCODE=	+0000016
13.29.41	J0B00082	+IMSTCPCL,	TASK	#	00000	READ	RETCODE=	+0000050
13.29.41	J0B00082	+IMSTCPCL,	TASK	‡⊧	00000	WRITE	RETCODE=	+0000016
13.29.41	J0B00082	+IMSTCPCL,	TASK	#	00000	READ	RETCODE=	+0000016
13.29.41	J0B00082	+IMSTCPCL,	TASK	#	00000	WRITE	RETCODE=	+0000048
13.29.42	J0B00082	+IMSTCPCL,	TASK	#	00000	READ	RETCODE=	+0000048
13.29.42	J0B00082	+IMSTCPCL,	TASK	‡⊧	00000	CLOSE		
13.29.42	J0B00082	+IMSTCPCL,	TASK	#	00000	ENDED		

Appendix A. Return codes

This appendix covers the following return codes and error messages

- Error numbers from MVS TCP/IP
- · Error codes from the Sockets Extended interface

Sockets return codes (ERRNOs)

This section provides the system-wide message numbers and codes set by the system calls. These message numbers and codes are in the TCPERRNO.H include file supplied with TCP/IP Services.

Table 49	Table 49. Sockets ERRNOs					
Error numbe r	Message name	Socket API type	Error description	Programmer's response		
1	EAI_NONAME	GETADDRINFO GETNAMEINFO	NODE or HOST cannot be found.	Ensure the NODE or HOST name can be resolved.		
1	EDOM	All	Argument too large.	Check parameter values of the function call.		
1	EPERM	All	Permission is denied. No owner exists.	Check that TCP/IP is still active; check protocol value of socket () call.		
1	EPERM	IOCTL (SIOCGPARTNERINFO)	Both endpoints do not reside in the same security domain.	Check and modify the security domain name for the endpoints. After you correct the security domain name, the application might need to close the connection if the IOCTL is needed.		
1	EPERM	IOCTL (SIOCGPARTNERINFO, SIOCSPARTNERINFO)	The security domain name is not defined.	Define the security domain name on both endpoints. After you define the security domain name, the application might need to close the connection if the IOCTL is needed.		
1	EPERM	IOCTL (SIOCTTLSCTL)	 Denotes one of the following error conditions: The TTLS_INIT_CONNECTION option was requested with either TTLS_RESET_SESSION, TTLS_RESET_CIPHER, TTLS_RESET_WRITE_CIPHER, TTLS_SEND_SESSION_TICKET, or TTLS_STOP_CONNECTION The TTLS_STOP_CONNECTION The TTLS_STOP_CONNECTION option was requested along with TTLS_RESET_CIPHER, or TTLS_RESET_WRITE_CIPHER, or TTLS_SEND_SESSION_TICKET The TTLS_ALLOW_HSTIMEOUT option was requested without TTLS_RESET_WRITE_CIPHER option was requested along with TTLS_RESET_SESSION_TICKET The TTLS_SEND_SESSION_TICKET The TTLS_RESET_WRITE_CIPHER option was requested along with TTLS_RESET_SESSION or TTLS_RESET_CIPHER The TTLS_RESET_SESSION_TICKET option was requested along with TTLS_RESET_CIPHER The TTLS_RESET_SESSION_TICKET option was requested along with TTLS_RESET_CIPHER The TTLS_RESET_CIPHER or TTLS_RESET_CIPHER, or TTLS_RESET_CIPHER optiON, TTLS_RESET_SESSION, TTLS_RESET_CIPHER, or TTLS_RESET_CIPHER, or TTLS_RESET_CIPHER, or TTLS_RESET_CIPHER, or TTLS_RESET_CIPHER, or TTLS_RESET_WRITE_CIPHER 	Request TTLS_RESET_SESSION, TTLS_RESET_CIPHER, TTLS_RESET_WRITE_CIPHER, or TTLS_SEND_SESSION_TICKET only when TTLS_INIT_CONNECTION and TTLS_STOP_CONNECTION are not requested. Always request TTLS_INIT_ CONNECTION when TTLS_ALLOW_ HSTIMEOUT is requested. Use seperate SIOCTTLSCTL ioctls to request TTLS_INIT_CONNECTION and TTLS_STOP_CONNECTION. Use separate SIOCTTLSCTL ioctls to request TTLS_RESET_WRITE_CIPHER or TTLS_SEND_SESSION_TICKET.		
2	EAI_AGAIN	FREEADDRINFO GETADDRINFO GETNAMEINFO	For GETADDRINFO, NODE could not be resolved within the configured time interval. For GETNAMEINFO, HOST could not be resolved within the configured time interval. The Resolver address space has not been started. The request can be retried later.	Ensure the Resolver is active, then retry the request.		
2	ENOENT	All	The data set or directory was not found.	Check files used by the function call.		
2	ERANGE	All	The result is too large.	Check parameter values of the function call.		

Table 4	Table 49. Sockets ERRNOs (continued)					
Error numbe r	Message name	Socket API type	Error description	Programmer's response		
3	EAI_FAIL	FREEADDRINFO GETADDRINFO GETNAMEINFO	This is an unrecoverable error. NODELEN, HOSTLEN, or SERVLEN is incorrect. For FREEADDRINFO, the resolver storage does not exist.	Correct the NODELEN, HOSTLEN, or SERVLEN. Otherwise, call your system administrator.		
3	ESRCH	All	The process was not found. A table entry was not located.	Check parameter values and structures pointed to by the function parameters.		
4	EAI_OVERFLOW	GETNAMEINFO	The output buffer for the host name or service name was too small.	Increase the size of the buffer to 255 characters, which is the maximum size permitted.		
4	EINTR	All	A system call was interrupted.	Check that the socket connection and TCP/IP are still active.		
5	EAI_FAMILY	GETADDRINFO GETNAMEINFO	The AF or the FAMILY is incorrect.	Correct the AF or the FAMILY.		
5	EIO	All	An I/O error occurred.	Check status and contents of source database if this occurred during a file access.		
6	EAI_MEMORY	GETADDRINFO GETNAMEINFO	The resolver cannot obtain storage to process the host name.	Contact your system administrator.		
6	ENXIO	All	The device or driver was not found.	Check status of the device attempting to access.		
7	E2BIG	All	The argument list is too long.	Check the number of function parameters.		
7	EAI_BADFLAGS	GETADDRINFO GETNAMEINFO	FLAGS has an incorrect value.	Correct the FLAGS.		
8	EAI_SERVICE	GETADDRINFO	The SERVICE was not recognized for the specified socket type.	Correct the SERVICE.		
8	ENOEXEC	All	An EXEC format error occurred.	Check that the target module on an exec call is a valid executable module.		
9	EAI_SOCKTYPE	GETADDRINFO	The SOCTYPE was not recognized.	Correct the SOCTYPE.		
9	EBADF	All	An incorrect socket descriptor was specified.	Check socket descriptor value. It might be currently not in use or incorrect.		
9	EBADF	Givesocket	The socket has already been given. The socket domain is not AF_INET or AF_INET6.	Check the validity of function parameters.		
9	EBADF	Select	One of the specified descriptor sets is an incorrect socket descriptor.	Check the validity of function parameters.		
9	EBADF	Takesocket	The socket has already been taken.	Check the validity of function parameters.		
9	EAI_SOCKTYPE	GETADDRINFO	The SOCTYPE was not recognized.	Correct the SOCTYPE.		
10	ECHILD	All	There are no children.	Check if created subtasks still exist.		
11	EAGAIN	All	There are no more processes.	Retry the operation. Data or condition might not be available at this time.		
11	EAGAIN	All	TCP/IP is not active at the time of the request.	Start TCP/IP, and retry the request.		
11	EAGAIN	IOCTL (SIOCGPARTNERINFO)	The IOCTL was issued in no-suspend mode and the SIOCSPARTNERINFO IOCTL has not been issued.	Reissue the IOCTL with a timeout value to set the amount of time to wait while the partner security credentials are being retrieved. Restriction: You cannot use a select mask to determine when an IOCTL is complete, because an IOCTL is		
				not affected by whether the socket is running in blocking or nonblocking mode. If the IOCTL times out, reissue the IOCTL to retrieve the partner security credentials.		
12	ENOMEM	All	There is not enough storage.	Check the validity of function parameters.		
13	EACCES	All	Permission denied, caller not authorized.	Check access authority of file.		
13	EACCES	IOCTL (SIOCGPARTNERINFO)	The application is not running in supervisor state, is not APF authorized, or is not permitted to the appropriate SERVAUTH profile.	Allow the application to issue this IOCTL, or provide the user ID with the proper SERVAUTH permission.		

Table 4	Table 49. Sockets ERRNOs (continued)					
Error		Socket				
r	Message name	API type	Error description	Programmer's response		
13	EACCES	IOCTL (SIOCTTLSCTL)	The IOCTL is requesting a function that requires that the socket be mapped to policy that specifies ApplicationControlled On.	Check policy and add ApplicationControlled On if the application should be permitted to issue the controlled SIOCTTLSCTL functions.		
13	EACCES	Takesocket	The other application (listener) did not give the socket to your application. Permission denied, caller not authorized.	Check access authority of file.		
14	EFAULT	All	An incorrect storage address or length was specified.	Check the validity of function parameters.		
14	EFAULT	All EZASMI macros when using an asynchronous exit routine.	The exit routine has abnormally ended (ABEND condition).	Correct the error in the routine's code. Add an ESTAE routine to the exit.		
14	EFAULT	IOCTL (SIOCSAPPLDATA)	An abend occurred while attempting to copy the SetADcontainer structure from the address provided in the SetAD_ptr field.	Check the validity of function parameters.		
15	ENOTBLK	All	A block device is required.	Check device status and characteristics.		
16	EBUSY	All	Listen has already been called for this socket. Device or file to be accessed is busy.	Check if the device or file is in use.		
17	EEXIST	All	The data set exists.	Remove or rename existing file.		
18	EXDEV	All	This is a cross-device link. A link to a file on another file system was attempted.	Check file permissions.		
19	ENODEV	All	The specified device does not exist.	Check file name and if it exists.		
20	ENOTDIR	All	The specified directory is not a directory.	Use a valid file that is a directory.		
21	EISDIR	All	The specified directory is a directory.	Use a valid file that is not a directory.		
22	EINVAL	All types	An incorrect argument was specified.	Check the validity of function parameters.		
22	EINVAL	Multicast Source filter APIs	Mix of any-source, source-specific or full-state APIs	Specify the correct type of APIs.		
22	EINVAL	MCAST_JOIN_GROUP, MCAST_JOIN_SOURCE_ GROUP, MCAST_BLOCK_SOURCE, MCAST_LEAVE_GROUP, MCAST_LEAVE_SOURCE_ GROUP, MCAST_UNBLOCK_SOURCE, SIOCGMSFILTER, SIOCSMSFILTER	The socket address family or the socket length of the input multicast group or the source IP address is not correct.	Specify the correct value.		
22	EINVAL	SIOCSMSFILTER, SIOCSIPMSFILTER	The specified filter mode is not correct.	Specify the correct value.		
23	ENFILE	All	Data set table overflow occurred.	Reduce the number of open files.		
24	EMFILE	All	The socket descriptor table is full.	Check the maximum sockets specified in MAXDESC().		
25	ENOTTY	All	An incorrect device call was specified.	Check specified IOCTL() values.		
26	ETXTBSY	All	A text data set is busy.	Check the current use of the file.		
27	EFBIG	All	The specified data set is too large.	Check size of accessed dataset.		
28	ENOSPC	All	There is no space left on the device.	Increase the size of accessed file.		
29	ESPIPE	All	An incorrect seek was attempted.	Check the offset parameter for seek operation.		
30	EROFS	All	The data set system is Read only.	Access data set for read only operation.		
31	EMLINK	All	There are too many links.	Reduce the number of links to the accessed file.		
32	EPIPE	All	The connection is broken. For socket write/send, peer has shut down one or both directions.	Reconnect with the peer.		
32	EPIPE	IOCTL (SIOCTTLSCTL requesting TTLS_INIT_ CONNECTION, TTLS_RESET_CIPHER, TTLS_RESET_WRITE_CIPHER, TTLS_SEND_SESSION_TICKET or TTLS_STOP_ CONNECTION)	The TCP connection is not in the established state.	Issue the SIOCTTLSCTL IOCTL when the socket is connected.		
33	EDOM	All	The specified argument is too large.	Check and correct function parameters.		
34	ERANGE	All	The result is too large.	Check function parameter values.		

Table 49	Table 49. Sockets ERRNOs (continued)					
Error numbe r	Message name	Socket API type	Error description	Programmer's response		
35	EWOULDBLOCK	Accept	The socket is in nonblocking mode and connections are not queued. This is not an error condition.	Reissue Accept().		
35	EWOULDBLOCK	IOCTL (SIOCTTLSCTL)	The handshake is in progress and the socket is a nonblocking socket.	For a nonblocking socket, you can wait for the handshake to complete by issuing Select or Poll for Socket Writable.		
35	EWOULDBLOCK	Read Recvfrom	The socket is in nonblocking mode and read data is not available. This is not an error condition.	Issue a select on the socket to determine when data is available to be read or reissue the Read()/Recvfrom().		
35	EWOULDBLOCK	All receive calls (RECV, RECVMSG, RECVFROM, READV, READ), when the socket is set with the SO_RCVTIMEO socket option	The socket is in blocking mode and the receive call has blocked for the time period that was specified in the SO_RCVTIMEO option. No data was received.	The application should reissue the receive call.		
35	EWOULDBLOCK	Send Sendto Write	The socket is in nonblocking mode and buffers are not available.	Issue a select on the socket to determine when data is available to be written or reissue the Send(), Sendto(), or Write().		
35	EWOULDBLOCK	All send calls (SEND, SENDMSG, SENDTO, WRITEV, WRITE), when the socket is set with the SO_SNDTIMEO socket option	The socket is in blocking mode and the send call has blocked for the time period that was specified in the SO_SNDTIMEO option. No data was sent.	The application should reissue the send call.		
36	EINPROGRESS	Connect	The socket is marked nonblocking and the connection cannot be completed immediately. This is not an error condition.	See the Connect() description for possible responses.		
36	EINPROGRESS	IOCTL (SIOCGPARTNERINFO)	The IOCTL was issued in no-suspend mode after the SIOCSPARTNERINFO IOCTL was issued, but the partner security credentials are not currently available.	Retry the IOCTL, or issue the IOCTL with a timeout value to set the amount of time to wait while the partner security credentials are being retrieved. Restriction: You cannot use a select mask to determine when an IOCTL is complete, because an IOCTL is not affected by whether the socket is running in blocking or nonblocking mode. If the IOCTL times out, reissue the IOCTL to retrieve the partner security credentials.		
36	EINPROGRESS	IOCTL (SIOCTTLSCTL requesting TTLS_INIT_ CONNECTION or TTLS_STOP_ CONNECTION)	The handshake is already in progress and the socket is a nonblocking socket.	For a nonblocking socket, you can wait for the handshake to complete by issuing Select or Poll for Socket Writable.		
37	EALREADY	Connect	The socket is marked nonblocking and the previous connection has not been completed.	Reissue Connect().		
37	EALREADY	IOCTL (SIOCGPARTNERINFO)	The request is already in progress. Only one IOCTL can be outstanding.	Check and modify the socket descriptor, if specified; otherwise, no action is needed.		
37	EALREADY	IOCTL (SIOCTTLSCTL requesting TTLS_INIT_ CONNECTION or TTLS_STOP_ CONNECTION)	For TTLS_INIT_ CONNECTION, the socket is already secure. For TTLS_STOP_ CONNECTION, the socket is not secure.	Modify the application so that it issues the SIOCTTLSCTL IOCTL that requests TTLS_INIT_CONNECTION only when the socket is not already secure and that requests TTLS_STOP_CONNECTION only when the socket is secure.		
37	EALREADY	Maxdesc	A socket has already been created calling Maxdesc() or multiple calls to Maxdesc().	Issue Getablesize() to query it.		
37	EALREADY	Setibmopt	A connection already exists to a TCP/IP image. A call to SETIBMOPT (IBMTCP_IMAGE), has already been made.	Call Setibmopt() only once.		
38	ENOTSOCK	All	A socket operation was requested on a nonsocket connection. The value for socket descriptor was not valid.	Correct the socket descriptor value and reissue the function call.		
39	EDESTADDRREQ	All	A destination address is required.	Fill in the destination field in the correct parameter and reissue the function call.		
40	EMSGSIZE	Sendto Sendmsg Send Write for Datagram (UDP) or RAW sockets	The message is too long. It exceeds the IP limit of 64K or the limit set by the setsockopt() call.	Either correct the length parameter, or send the message in smaller pieces.		

Table 49	Table 49. Sockets ERRNOs (continued)					
Error numbe r	Message name	Socket API type	Error description	Programmer's response		
41	EPROTOTYPE	All	The specified protocol type is incorrect for this socket.	Correct the protocol type parameter.		
41	EPROTOTYPE	bind2addrsel	The referenced socket is not a stream (TCP) or datagram (UDP) socket.	Issue bind2addrsel() on TCP or UDP sockets only.		
41	EPROTOTYPE	IOCTL (SIOCGPARTNERINFO, SIOCSAPPLDATA, SIOCSPARTNERINFO, SIOCTTLSCTL)	Socket is not a TCP socket.	Issue the IOCTL on TCP sockets only.		
42	ENOPROTOOPT	Getsockopt Setsockopt	The socket option specified is incorrect or the level is not SOL_SOCKET. Either the level or the specified optname is not supported.	Correct the level or optname.		
42	ENOPROTOOPT	Getibmsockopt Setibmsockopt	Either the level or the specified optname is not supported.	Correct the level or optname.		
43	EPROTONOSUPPORT	Socket	The specified protocol is not supported.	Correct the protocol parameter.		
44	ESOCKTNOSUPPORT	All	The specified socket type is not supported.	Correct the socket type parameter.		
45	EOPNOTSUPP	Accept Givesocket	The selected socket is not a stream socket.	Use a valid socket.		
45	EOPNOTSUPP	bind2addrsel	The referenced socket is not a type that supports the requested function	Use a socket of the correct type.		
45	EOPNOTSUPP	Getibmopt Setibmopt	The socket does not support this function call. This command is not supported for this function.	Correct the command parameter. See Getibmopt() for valid commands. Correct by ensuring a Listen() was not issued before the Connect().		
45	EOPNOTSUPP	GETSOCKOPT	The specified GETSOCKOPT OPTNAME option is not supported by this socket API.	Correct the GETSOCKOPT OPTNAME option.		
45	EOPNOTSUPP	IOCTL	The specified IOCTL command is not supported by this socket API.	Correct the IOCTL COMMAND.		
45	EOPNOTSUPP	IOCTL (SIOCSPARTNERINFO)	The request must be issued before the listen call or the connect call.	Check and modify the socket descriptor, or close the connection and reissue the call.		
45	EOPNOTSUPP	IOCTL (SIOCTTLSCTL requesting TTLS_INIT_CONNECTION, TTLS_RESET_SESSION, TTLS_RESET_ CIPHER, TTLS_RESET_WRITE_CIPHER, TTLS_SEND_SESSION_TICKET, or TTLS_STOP_ CONNECTION)	Mapped policy indicates that AT-TLS is not enabled for the connection.	Modify the policy to enable AT-TLS for the connection.		
45	EOPNOTSUPP	Listen	The socket does not support the Listen call.	Change the type on the Socket() call when the socket was created. Listen() supports only a socket type of SOCK_STREAM.		
45	EOPNOTSUPP	RECV, RECVFROM, RECVMSG, SEND, SENDTO, SENDMSG	The specified flags are not supported on this socket type or protocol.	Correct the FLAG.		
46	EPFNOSUPPORT	All	The specified protocol family is not supported or the specified domain for the client identifier is not AF_INET=2.	Correct the protocol family.		
47	EAFNOSUPPORT	bind2addrsel inet6_is_srcaddr	You specified an IP address that is not an AF_INET6 IP address	Correct the IP address. If the IP address is an IPv4 address, you must specify it as an IPv4-mapped IPv6 address.		
47	EAFNOSUPPORT	bind2addrsel inet6_is_srcaddr	You attempted an IPv6-only API for a stack that does not support the AF_INET6 domain.	Activate the AF_INET6 stack, and retry the request.		
47	EAFNOSUPPORT	Bind Connect Socket	The specified address family is not supported by this protocol family.	For Socket(), set the domain parameter to AF_INET. For Bind() and Connect(), set Sin_Family in the socket address structure to AF_INET.		
47	EAFNOSUPPORT	Getclient Givesocket	The socket specified by the socket descriptor parameter was not created in the AF_INET domain.	The Socket() call used to create the socket should be changed to use AF_INET for the domain parameter.		
47	EAFNOSUPPORT	IOCTL	You attempted to use an IPv4-only ioctl on an AF_INET6 socket.	Use the correct socket type for the ioctl or use an ioctl that supports AF_INET6 sockets.		

Table 49	Table 49. Sockets ERRNOs (continued)					
Error numbe r	Message name	Socket API type	Error description	Programmer's response		
48	EADDRINUSE	Bind, Connect	The address is in a timed wait because a LINGER delay from a previous close or another process is using the address. This error can also occur if the port specified in the bind call has been configured as RESERVED on a port reservation statement in the TCP/IP profile.	To reuse the same address, use Setsockopt() with SO_REUSEADDR. See the section about Setsockopt() in z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference for more information. Otherwise, use a different address or port in the socket address structure.		
48	EADDRINUSE	IP_ADD_MEMBERSHIP, IP_ADD_SOURCE_ MEMBERSHIP, IPV6_JOIN_GROUP, MCAST_JOIN_GROUP, MCAST_JOIN_SOURCE_ GROUP	The specified multicast address and interface address (or interface index) pair is already in use.	Correct the specified multicast address, interface address, or interface index.		
49	EADDRNOTAVAIL	Bind	The specified address is incorrect for this host.	Correct the function address parameter.		
49	EADDRNOTAVAIL	Connect	The calling host cannot reach the specified destination.	Correct the function address parameter.		
49	EADDRNOTAVAIL	bind2addrsel	 For the specified destination address, there is no source address that the application can bind to. Possible reasons can be one of the following situations: The socket is a stream socket, but the specified destination address is a multicast address. No ephemeral ports are available to assign to the socket. 	Correct the function address parameter or issue the request when ephemeral ports are available.		
49	EADDRNOTAVAIL	inet6_is_srcaddr	 The address specified is not correct for one of these reasons: The address is not an address on this node. The address was not active at the time of the request. The scope ID specified for a linklocal IPV6 address is incorrect. 	Correct or activate the address		
49	EADDRNOTAVAIL	IP_BLOCK_SOURCE, IP_ADD_SOURCE_ MEMBERSHIP, MCAST_BLOCK_SOURCE, MCAST_JOIN_SOURCE_ GROUP	A duplicate source IP address is specified on the multicast group and interface pair.	Correct the specified source IP address.		
49	EADDRNOTAVAIL	IP_UNBLOCK_SOURCE, IP_DROP_SOURCE_ MEMBERSHIP, MCAST_UNBLOCK_SOURCE, MCAST_LEAVE_SOURCE_ GROUP	A previously blocked source multicast group cannot be found.	Correct the specified address.		
49	EADDRNOTAVAIL	Multicast APIs	The specified multicast address, interface address, or interface index is not correct.	Correct the specified address.		
50	ENETDOWN	All	The network is down.	Retry when the connection path is up.		
51	ENETUNREACH	Connect	The network cannot be reached.	Ensure that the target application is active.		
52	ENETRESET	All	The network dropped a connection on a reset.	Reestablish the connection between the applications.		
53	ECONNABORTED	All	The software caused a connection abend.	Reestablish the connection between the applications.		
54	ECONNRESET	All	The connection to the destination host is not available.	N/A		
54	ECONNRESET	Send Write	The connection to the destination host is not available.	The socket is closing. Issue Send() or Write() before closing the socket.		
55	ENOBUFS	All	No buffer space is available.	Check the application for massive storage allocation call.		
55	ENOBUFS	Accept	Not enough buffer space is available to create the new socket.	Call your system administrator.		
55	ENOBUFS	IOCTL (SIOCGPARTNERINFO)	The buffer size provided is too small.	Create a larger input buffer based on the value returned in the PI_Buflen field.		
55	ENOBUFS	IOCTL (SIOCSAPPLDATA)	There is no storage available to store the associated data.	Call your system administrator.		
55	ENOBUFS	IOCTL (SIOCTTLSCTL TTLS_Version1 requesting TTLS_RETURN_ CERTIFICATE or TTLS_Version2 query)	The buffer size provided is too small.	For TTLS_Version1 use the returned certificate length to allocate a larger buffer and reissue IOCTL with the larger buffer.		

Table 4	Table 49. Sockets ERRNOs (continued)					
Error		Socket				
r	Message name	API type	Error description	Programmer's response		
55	ENOBUFS	IP_BLOCK_SOURCE, IP_ADD_SOURCE_ MEMBERSHIP, MCAST_BLOCK_SOURCE, MCAST_JOIN_SOURCE_GROUP, SIOCSIPMSFILTER, SIOCSMSFILTER, setipv4sourcefilter, setsourcefilter	A maximum of 64 source filters can be specified per multicast address, interface address pair.	Remove unneeded source IP addresses and reissue the command.		
55	ENOBUFS	Send Sendto Write	Not enough buffer space is available to send the new message.	Call your system administrator.		
55	ENOBUFS	Takesocket	Not enough buffer space is available to create the new socket.	Call your system administrator.		
56	EISCONN	Connect	The socket is already connected.	Correct the socket descriptor on Connect() or do not issue a Connect() twice for the socket.		
57	ENOTCONN	All	The socket is not connected.	Connect the socket before communicating.		
57	ENOTCONN	IOCTL (SIOCGPARTNERINFO)	The requested socket is not connected.	Check and modify the socket descriptor, or reissue the IOCTL after the connect call from the client side or after the accept call from the server side.		
57	ENOTCONN	IOCTL (SIOCTTLSCTL)	The socket is not connected.	Issue the SIOCTTLSCTL IOCTL only after the socket is connected.		
58	ESHUTDOWN	All	A Send cannot be processed after socket shutdown.	Issue read/receive before shutting down the read side of the socket.		
59	ETOOMANYREFS	All	There are too many references. A splice cannot be completed.	Call your system administrator.		
59	ETOOMANYREFS	IP_ADD_MEMBERSHIP, IP_ADD_SOURCE_ MEMBERSHIP, MCAST_JOIN_GROUP, MCAST_JOIN_SOURCE_ GROUP, IPV6_JOIN_GROUP	A maximum of 20 multicast groups per single UDP socket or a maximum of 256 multicast groups per single RAW socket can be specified.	Remove unneeded multicast groups and reissue the command.		
60	ETIMEDOUT	Connect	The connection timed out before it was completed.	Ensure the server application is available.		
61	ECONNREFUSED	Connect	The requested connection was refused.	Ensure server application is available and at specified port.		
62	ELOOP	All	There are too many symbolic loop levels.	Reduce symbolic links to specified file.		
63	ENAMETOOLONG	All	The file name is too long.	Reduce size of specified file name.		
64	EHOSTDOWN	All	The host is down.	Restart specified host.		
65	EHOSTUNREACH	All	There is no route to the host.	Set up network path to specified host and verify that host name is valid.		
66	ENOTEMPTY	All	The directory is not empty.	Clear out specified directory and reissue call.		
67	EPROCLIM	All	There are too many processes in the system.	Decrease the number of processes or increase the process limit.		
68	EUSERS	All	There are too many users on the system.	Decrease the number of users or increase the user limit.		
69	EDQUOT	All	The disk quota has been exceeded.	Call your system administrator.		
70	ESTALE	All	An old NFS ^{**} data set handle was found.	Call your system administrator.		
71	EREMOTE	All	There are too many levels of remote in the path.	Call your system administrator.		
72	ENOSTR	All	The device is not a stream device.	Call your system administrator.		
73	ETIME	All	The timer has expired.	Increase timer values or reissue function.		
73	ЕТІМЕ	IOCTL (SIOCGPARTNERINFO)	The wait time for the request has expired, possibly as the result of network problems.	Retry the request. Restriction: You cannot use a select mask to determine when an IOCTL is complete, because an IOCTL is not affected by whether the socket is running in blocking or nonblocking mode. If the IOCTL times out, reissue the IOCTL to retrieve the partner security credentials.		
74	ENOSR	All	There are no more stream resources.	Call your system administrator.		

Table	Table 49. Sockets ERRNOs (continued)						
Error numb r	e Message name	Socket API type	Error description	Programmer's response			
75	ENOMSG	All	There is no message of the desired type.	Call your system administrator.			
76	EBADMSG	All	The system cannot read the message.	Verify that z/OS Communications Server installation was successful and that message files were properly loaded.			
77	EIDRM	All	The identifier has been removed.	Call your system administrator.			
78	EDEADLK	All	A deadlock condition has occurred.	Call your system administrator.			
78	EDEADLK	Select Selectex	None of the sockets in the socket descriptor sets are either AF_INET or AF_IUCV sockets and there is no timeout value or no ECB specified. The select/selectex would never complete.	Correct the socket descriptor sets so that an AF_INET or AF_IUCV socket is specified. A timeout or ECB value can also be added to avoid the select/ selectex from waiting indefinitely.			
79	ENOLCK	All	No record locks are available.	Call your system administrator.			
80	ENONET	All	The requested machine is not on the network.	Call your system administrator.			
81	ERREMOTE	All	The object is remote.	Call your system administrator.			
82	ENOLINK	All	The link has been severed.	Release the sockets and reinitialize the client-server connection.			
83	EADV	All	An ADVERTISE error has occurred.	Call your system administrator.			
84	ESRMNT	All	An SRMOUNT error has occurred.	Call your system administrator.			
85	ECOMM	All	A communication error has occurred on a Send call.	Call your system administrator.			
86	EPROTO	All	A protocol error has occurred.	Call your system administrator.			
86	EPROTO	IOCTL (SIOCTTLSCTL requesting TTLS_RESET_SESSION, TTLS_RESET_CIPHER, TTLS_SEND_SESSION_TICKET, TTLS_STOP_CONNECTION, or TTLS_ALLOW_HSTIMEOUT)	 One of the following errors occurred: A TTLS_INIT_CONNECTION request was not received for the connection. TTLS_STOP_CONNECTION was requested on a connection that has outstanding application data, the errno junior is JrTTLSStopReadDataPending. For unwritten application data, the errno junior is JrTLSStopWriteDataPending. TTLS_STOP_CIPHER or TTLS_STOP_CIPHER or TTLS_STOP_CIPHER was requested on a connection that is secured using SSL version 2. TTLS_RESET_WRITE_CIPHER or TTLS_SEND_SESSION_TICKET was requested on a connection that is secured using a protocol version less than TLS version 1.3. TTLS_SEND_SESSION_TICKET was requested but the policy has the HandshakeRole value client or the HandshakeRole value client or GSK_SESSION_TICKET_SERVER_EN ABLE value Off or the GSK_SESSION_TICKET_SERVER_CO UNT value is not 0. 	 Request TILS_INIT_CONNECTION before requesting TTLS_RESET_SESSION, TTLS_RESET_CIPHER, TTLS_RESET_WRITE_CIPHER, or TTLS_SEND_SESSION_TICKET. Request TTLS_STOP_CONNECTION after all application data is cleared from the connection. For JrTTLSStopReadDataPending, read all available application data. For JrTTLSStopReadDataPending, wait for all the outstanding application data to be written. Request TTLS_RESET_CIPHER or TTLS_STOP_CONNECTION only on connections secured using SSL version 3 or TLS version 1.0 or higher. Request TTLS_SEND_SESSION_TICKET only on connections secured using TLS version 1.3 or higher. Request TTLS_SEND_SESSION_TICKET only only when the security type is TTLS_SEC_SERVER or higher and the HandshakeTimeout value is not 0. Request TTLS_SESION_TICKET_only when the security type is TTLS_SEC_SERVER or higher and when GSK_SESSION_TICKET_SERVER_EN ABLE is On and GSK_SESSION_TICKET_SERVER_CO UNT is 0. 			
87	EMULTIHOP	All 	A multi-hop address link was attempted. A cross-mount point was detected. This	Call your system administrator.			
			is not an error.	,,			
89	EREMCHG	All	The remote address has changed.	Call your system administrator.			
90	ECONNCLOSED	All	The connection was closed by a peer.	Check that the peer is running.			

Table 49	Table 49. Sockets ERRNOs (continued)						
Error numbe r	Message name	Socket API type	Error description	Programmer's response			
113	EBADF	All	Socket descriptor is not in correct range. The maximum number of socket descriptors is set by MAXDESC(). The default range is 0–49.	Reissue function with corrected socket descriptor.			
113	EBADF	Bind socket	The socket descriptor is already being used.	Correct the socket descriptor.			
113	EBADF	Givesocket	The socket has already been given. The socket domain is not AF_INET.	Correct the socket descriptor.			
113	EBADF	Select	One of the specified descriptor sets is an incorrect socket descriptor.	Correct the socket descriptor. Set on Select() or Selectex().			
113	EBADF	Takesocket	The socket has already been taken.	Correct the socket descriptor.			
113	EBADF	Accept	A Listen() has not been issued before the Accept().	Issue Listen() before Accept().			
121	EINVAL	All	An incorrect argument was specified.	Check and correct all function parameters.			
121	EINVAL	IOCTL (SIOCSAPPLDATA)	The input parameter is not a correctly formatted SetApplData structure. • The SetAD_eye1 value is not valid.	Check and correct all function parameters.			
			• The SetAD_ver value is not valid.				
			The storage pointed to by SetAD_ptr does not contain a correctly formatted SetADcontainer structure.				
			• The SetAD_eye2 value is not valid.				
			 The SetAD_len value contains an incorrect length for the SetAD_ver version of the SetADcontainer structure. 				
121	EINVAL	inet6_is_srcaddr	One or more invalid IPV6_ADDR_ PREFERENCES flags were specified	Correct the function parameters			
			A scope ID was omitted for a link local IP address				
			 A scope ID was specified for an IP address that is not link-local 				
			 The socket address length was not valid 				
122	ECLOSED						
126	ENMELONG						
134	ENOSYS	IOCTL	The function is not implemented	Either configure the system to support the ioctl command or remove the ioctl command from your program.			
134	ENOSYS	IOCTL - siocgifnameindex	The TCP/IP stack processing the siocgifnameindex IOCTL is configured as a pure IPv4 TCP/IP stack. Additionally, UNIX System Services is configured to process as INET.	Either configure the system to support the ioctl command or remove the ioctl command from your program.			
136	ENOTEMPT						
145	E2BIG	All	The argument list is too long.	Eliminate excessive number of arguments.			
156	EMVSINITIAL	All	Process initialization error. This indicates an z/OS UNIX process initialization failure. This is usually an indication that a proper OMVS RACF [®] segment is not defined for the user ID associated with application. The RACF OMVS segment might not be defined or might contain errors such as an improper HOME() directory specification.	Attempt to initialize again. After ensuring that an OMVS Segment is defined, if the errno is still returned, call your MVS system programmer to have IBM service contacted.			
157	EMISSED						
157	EMVSERR		An MVS environmental or internal error occurred.				

Table 4	Table 49. Sockets ERRNOs (continued)					
Error numbe r	Message name	Socket API type	Error description	Programmer's response		
1002	EIBMSOCKOUTOFRANGE	Socket, Accept, Takesocket	A new socket cannot be created because the MAXSOC value, which is specified on the INITAPI call, has been reached.	 Take either one of the following actions: Verify whether all open sockets are intended to be in use. Increase the MAXSOC value to a value that is appropriate for the current workload. If the default value is currently being used, you might be required to add the INITAPI call. 		
1003	EIBMSOCKINUSE	Socket	A socket number assigned by the client interface code is already in use.	Use a different socket descriptor.		
1004	EIBMIUCVERR	All	The request failed because of an IUCV error. This error is generated by the client stub code.	Ensure IUCV/VMCF is functional.		
1008	EIBMCONFLICT	All	This request conflicts with a request already queued on the same socket.	Cancel the existing call or wait for its completion before reissuing this call.		
1009	EIBMCANCELLED	All	The request was canceled by the CANCEL call.	Informational, no action needed.		
1011	EIBMBADTCPNAME	All	A TCP/IP name that is not valid was detected.	Correct the name specified in the IBM_TCPIMAGE structure.		
1011	EIBMBADTCPNAME	Setibmopt	A TCP/IP name that is not valid was detected.	Correct the name specified in the IBM_TCPIMAGE structure.		
1011	EIBMBADTCPNAME	INITAPI	A TCP/IP name that is not valid was detected.	Correct the name specified on the IDENT option TCPNAME field.		
1012	EIBMBADREQUESTCODE	All	A request code that is not valid was detected.	Contact your system administrator.		
1013	EIBMBADCONNECTIONSTATE	All	A connection token that is not valid was detected; bad state.	Verify TCP/IP is active.		
1014	EIBMUNAUTHORIZEDCALLER	All	An unauthorized caller specified an authorized keyword.	Ensure user ID has authority for the specified operation.		
1015	EIBMBADCONNECTIONMATCH	All	A connection token that is not valid was detected. There is no such connection.	Verify TCP/IP is active.		
1016	EIBMTCPABEND	All	An abend occurred when TCP/IP was processing this request.	Verify that TCP/IP has restarted.		
1023	EIBMTERMERROR	All	Encountered a terminating error while processing.	Call your system administrator.		
1026	EIBMINVDELETE	All	Delete requestor did not create the connection.	Delete the request from the process that created it.		
1027	EIBMINVSOCKET	All	A connection token that is not valid was detected. No such socket exists.	Call your system programmer.		
1028	EIBMINVTCPCONNECTION	All	Connection terminated by TCP/IP. The token was invalidated by TCP/IP.	Reestablish the connection to TCP/IP.		
1032	EIBMCALLINPROGRESS	All	Another call was already in progress.	Reissue after previous call has completed.		
1036	EIBMNOACTIVETCP	All	TCP/IP is not installed or not active.	Correct TCP/IP name used.		
1036	EIBMNOACTIVETCP	Select	EIBMNOACTIVETCP	Ensure TCP/IP is active.		
1036	EIBMNOACTIVETCP	Getibmopt	No TCP/IP image was found.	Ensure TCP/IP is active.		
1037	EIBMINVTSRBUSERDATA	All	The request control block contained data that is not valid.	Call your system programmer.		
1038	EIBMINVUSERDATA	All	The request control block contained user data that is not valid.	Check your function parameters and call your system programmer.		
1040	EIBMSELECTEXPOST	SELECTEX	SELECTEX passed an ECB that was already posted.	Check whether the user's ECB was already posted.		
1112	ECANCEL					
1162	ENOPARTNERINFO	IOCTL (SIOCGPARTNERINFO)	The partner resides in a TCP/IP stack running a release that is earlier than V1R12, or the partner is not in the same sysplex.	Ensure that both endpoints reside in TCP/IP stacks that are running V1R12 or any later release, or check and modify the socket descriptor. If the partner is not in the same sysplex, security credentials will not be returned.		
2001	EINVALIDRXSOCKETCALL	REXX	A syntax error occurred in the RXSOCKET parameter list.	Correct the parameter list passed to the REXX socket call.		

Table 49	able 49. Sockets ERRNOs (continued)						
Error numbe r	Message name	Socket API type	Error description	Programmer's response			
2002	ECONSOLEINTERRUPT	REXX	A console interrupt occurred.	Retry the task.			
2003	ESUBTASKINVALID	REXX	The subtask ID is incorrect.	Correct the subtask ID on the INITIALIZE call.			
2004	ESUBTASKALREADYACTIVE	REXX	The subtask is already active.	Issue the INITIALIZE call only once in your program.			
2005	ESUBTASKNOTACTIVE	REXX	The subtask is not active.	Issue the INITIALIZE call before any other socket call.			
2006	ESOCKETNOTALLOCATED	REXX	The specified socket or needed control block could not be allocated.	Increase the user storage allocation for this job.			
2007	EMAXSOCKETSREACHED	REXX	The maximum number of sockets has been reached.	Increase the number of allocate sockets, or decrease the number of sockets used by your program.			
2009	ESOCKETNOTDEFINED	REXX	The socket is not defined.	Issue the SOCKET call before the call that fails.			
2011	EDOMAINSERVERFAILURE	REXX	A Domain Name Server failure occurred.	Call your MVS system programmer.			
2012	EINVALIDNAME	REXX	An incorrect <i>name</i> was received from the TCP/IP server.	Call your MVS system programmer.			
2013	EINVALIDCLIENTID	REXX	An incorrect <i>clientid</i> was received from the TCP/IP server.	Call your MVS system programmer.			
2014	ENIVALIDFILENAME	REXX	An error occurred during NUCEXT processing.	Specify the correct translation table file name, or verify that the translation table is valid.			
2016	EHOSTNOTFOUND	REXX	The host is not found.	Call your MVS system programmer.			
2017	EIPADDRNOTFOUND	REXX	Address not found.	Call your MVS system programmer.			
2019	ENORECOVERY	REXX	A non-recoverable failure occurred during the Resolver's processing of the GETHOSTBYADDR or GETHOSTBYNAME call.	Contact the IBM support center.			
2020	EINVALIDCOMBINATION	REXX	An invalid combination of IPV6_ADDR_ PREFERENCES flags was received from the caller.	Correct the specified flags			
2021	EOPTNAMEMISMATCH	REXX	The caller specified an OPTNAME that is invalid for the LEVEL that it specified.	Correct either the OPTNAME or the LEVEL.			
2022	EFLAGSMISMATCH	REXX	The caller issued a GETADDRINFO with conflicting FLAGS and EFLAGS parameters: either AI_EXT_FLAGS was specified with a null EFLAGS, or AI_EXT_FLAGS was not specified but EFLAGS was not null.	Correct either the FLAGS parameter or the EFLAGS parameter. A non-null EFLAGS should be specified if and only if AI_EXT_FLAGS is specified in the FLAGS.			
2051	EFORMATERROR	REXX	The name server was unable to interpret the query	Contact the IBM support center.			
3412	ENODATA		Message does not exist.				
3416	ELINKED		Stream is linked.				
3419	ERECURSE		Recursive attempt rejected.				
3420	EASYNC		Asynchronous I/O scheduled. This is a normal, internal event that is NOT returned to the user.				
3448	EUNATCH		The protocol required to support the specified address family is not available.				
3464	ETERM		Operation terminated.				
3474	EUNKNOWN		Unknown system state.				
3495	EBADOBJ		You attempted to reference an object that does not exist.				
3513	EOUTOFSTATE		Protocol engine has received a command that is not acceptable in its current state.				

278 z/OS Communications Server: z/OS V2R5.0 Communications Server: IP IMS Sockets Guide

Appendix B. Related protocol specifications

This appendix lists the related protocol specifications (RFCs) for TCP/IP. The Internet Protocol suite is still evolving through requests for comments (RFC). New protocols are being designed and implemented by researchers and are brought to the attention of the Internet community in the form of RFCs. Some of these protocols are so useful that they become recommended protocols. That is, all future implementations for TCP/IP are recommended to implement these particular functions or protocols. These become the *de facto* standards, on which the TCP/IP protocol suite is built.

RFCs are available at http://www.rfc-editor.org/rfc.html.

Draft RFCs that have been implemented in this and previous Communications Server releases are listed at the end of this topic.

Many features of TCP/IP Services are based on the following RFCs:

RFC

Title and Author

RFC 652

Telnet output carriage-return disposition option D. Crocker

RFC 653

Telnet output horizontal tabstops option D. Crocker

RFC 654

Telnet output horizontal tab disposition option D. Crocker

RFC 655

Telnet output formfeed disposition option D. Crocker

RFC 657

Telnet output vertical tab disposition option D. Crocker

RFC 658

Telnet output linefeed disposition D. Crocker

RFC 698

Telnet extended ASCII option T. Mock

RFC 726

Remote Controlled Transmission and Echoing Telnet option J. Postel, D. Crocker

RFC 727

Telnet logout option M.R. Crispin

RFC 732

Telnet Data Entry Terminal option J.D. Day

RFC 733

Standard for the format of ARPA network text messages D. Crocker, J. Vittal, K.T. Pogran, D.A. Henderson

RFC 734

SUPDUP Protocol M.R. Crispin

RFC 735

Revised Telnet byte macro option D. Crocker, R.H. Gumpertz

RFC 736

Telnet SUPDUP option M.R. Crispin

RFC 749

Telnet SUPDUP-Output option B. Greenberg

RFC 765

File Transfer Protocol specification J. Postel

User Datagram Protocol J. Postel

RFC 779

Telnet send-location option E. Killian

RFC 791

Internet Protocol J. Postel

RFC 792

Internet Control Message Protocol J. Postel

RFC 793

Transmission Control Protocol J. Postel

RFC 820

Assigned numbers J. Postel

RFC 823

DARPA Internet gateway R. Hinden, A. Sheltzer

RFC 826

Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware D. Plummer

RFC 854

Telnet Protocol Specification J. Postel, J. Reynolds

RFC 855

Telnet Option Specification J. Postel, J. Reynolds

RFC 856

Telnet Binary Transmission J. Postel, J. Reynolds

RFC 857

Telnet Echo Option J. Postel, J. Reynolds

RFC 858

Telnet Suppress Go Ahead Option J. Postel, J. Reynolds

RFC 859

Telnet Status Option J. Postel, J. Reynolds

RFC 860

Telnet Timing Mark Option J. Postel, J. Reynolds

RFC 861

Telnet Extended Options: List Option J. Postel, J. Reynolds

RFC 862

Echo Protocol J. Postel

RFC 863

Discard Protocol J. Postel

RFC 864

Character Generator Protocol J. Postel

RFC 865

Quote of the Day Protocol J. Postel

RFC 868

Time Protocol J. Postel, K. Harrenstien

RFC 877

Standard for the transmission of IP datagrams over public data networks J.T. Korb

RFC 883

Domain names: Implementation specification P.V. Mockapetris

RFC 884

Telnet terminal type option M. Solomon, E. Wimmers

Telnet end of record option J. Postel

RFC 894

Standard for the transmission of IP datagrams over Ethernet networks C. Hornig

RFC 896

Congestion control in IP/TCP internetworks J. Nagle

RFC 903

Reverse Address Resolution Protocol R. Finlayson, T. Mann, J. Mogul, M. Theimer

RFC 904

Exterior Gateway Protocol formal specification D. Mills

RFC 919

Broadcasting Internet Datagrams J. Mogul

RFC 922

Broadcasting Internet datagrams in the presence of subnets J. Mogul

RFC 927

TACACS user identification Telnet option B.A. Anderson

RFC 933

Output marking Telnet option S. Silverman

RFC 946

Telnet terminal location number option R. Nedved

RFC 950

Internet Standard Subnetting Procedure J. Mogul, J. Postel

RFC 952

DoD Internet host table specification K. Harrenstien, M. Stahl, E. Feinler

RFC 959

File Transfer Protocol J. Postel, J.K. Reynolds

RFC 961

Official ARPA-Internet protocols J.K. Reynolds, J. Postel

RFC 974

Mail routing and the domain system C. Partridge

RFC 1001

Protocol standard for a NetBIOS service on a TCP/UDP transport: Concepts and methods NetBios Working Group in the Defense Advanced Research Projects Agency, Internet Activities Board, End-to-End Services Task Force

RFC 1002

Protocol Standard for a NetBIOS service on a TCP/UDP transport: Detailed specifications NetBios Working Group in the Defense Advanced Research Projects Agency, Internet Activities Board, End-to-End Services Task Force

RFC 1006

ISO transport services on top of the TCP: Version 3 M.T. Rose, D.E. Cass

RFC 1009

Requirements for Internet gateways R. Braden, J. Postel

RFC 1011

Official Internet protocols J. Reynolds, J. Postel

RFC 1013

X Window System Protocol, version 11: Alpha update April 1987 R. Scheifler

RFC 1014

XDR: External Data Representation standard Sun Microsystems

RFC 1027

Using ARP to implement transparent subnet gateways S. Carl-Mitchell, J. Quarterman

Domain administrators guide M. Stahl

RFC 1033

Domain administrators operations guide M. Lottor

RFC 1034

Domain names-concepts and facilities P.V. Mockapetris

RFC 1035

Domain names-implementation and specification P.V. Mockapetris

RFC 1038

Draft revised IP security option M. St. Johns

RFC 1041

Telnet 3270 regime option Y. Rekhter

RFC 1042

Standard for the transmission of IP datagrams over IEEE 802 networks J. Postel, J. Reynolds

RFC 1043

Telnet Data Entry Terminal option: DODIIS implementation A. Yasuda, T. Thompson

RFC 1044

Internet Protocol on Network System's HYPERchannel: Protocol specification K. Hardwick, J. Lekashman

RFC 1053

Telnet X.3 PAD option S. Levy, T. Jacobson

RFC 1055

Nonstandard for transmission of IP datagrams over serial lines: SLIP J. Romkey

RFC 1057

RPC: Remote Procedure Call Protocol Specification: Version 2 Sun Microsystems

RFC 1058

Routing Information Protocol C. Hedrick

RFC 1060

Assigned numbers J. Reynolds, J. Postel

RFC 1067

Simple Network Management Protocol J.D. Case, M. Fedor, M.L. Schoffstall, J. Davin

RFC 1071

Computing the Internet checksum R.T. Braden, D.A. Borman, C. Partridge

RFC 1072

TCP extensions for long-delay paths V. Jacobson, R.T. Braden

RFC 1073

Telnet window size option D. Waitzman

RFC 1079

Telnet terminal speed option C. Hedrick

RFC 1085

ISO presentation services on top of TCP/IP based internets M.T. Rose

RFC 1091

Telnet terminal-type option J. VanBokkelen

RFC 1094

NFS: Network File System Protocol specification Sun Microsystems

RFC 1096

Telnet X display location option G. Marcy

RFC 1101

DNS encoding of network names and other types P. Mockapetris

Host extensions for IP multicasting S.E. Deering

RFC 1113

Privacy enhancement for Internet electronic mail: Part I – message encipherment and authentication procedures J. Linn

RFC 1118

Hitchhikers Guide to the Internet E. Krol

RFC 1122

Requirements for Internet Hosts-Communication Layers R. Braden, Ed.

RFC 1123

Requirements for Internet Hosts-Application and Support R. Braden, Ed.

RFC 1146

TCP alternate checksum options J. Zweig, C. Partridge

RFC 1155

Structure and identification of management information for TCP/IP-based internets M. Rose, K. McCloghrie

RFC 1156

Management Information Base for network management of TCP/IP-based internets K. McCloghrie, M. Rose

RFC 1157

Simple Network Management Protocol (SNMP) J. Case, M. Fedor, M. Schoffstall, J. Davin

RFC 1158

Management Information Base for network management of TCP/IP-based internets: MIB-II M. Rose

RFC 1166

Internet numbers S. Kirkpatrick, M.K. Stahl, M. Recker

RFC 1179

Line printer daemon protocol L. McLaughlin

RFC 1180

TCP/IP tutorial T. Socolofsky, C. Kale

RFC 1183

New DNS RR Definitions C.F. Everhart, L.A. Mamakos, R. Ullmann, P.V. Mockapetris

RFC 1184

Telnet Linemode Option D. Borman

RFC 1186

MD4 Message Digest Algorithm R.L. Rivest

RFC 1187

Bulk Table Retrieval with the SNMP M. Rose, K. McCloghrie, J. Davin

RFC 1188

Proposed Standard for the Transmission of IP Datagrams over FDDI Networks D. Katz

RFC 1190

Experimental Internet Stream Protocol: Version 2 (ST-II) C. Topolcic

RFC 1191

Path MTU discovery J. Mogul, S. Deering

RFC 1198

FYI on the X window system R. Scheifler

RFC 1207

FYI on Questions and Answers: Answers to commonly asked "experienced Internet user" questions G. Malkin, A. Marine, J. Reynolds

RFC 1208

Glossary of networking terms O. Jacobsen, D. Lynch

Management Information Base for Network Management of TCP/IP-based internets: MIB-II K. McCloghrie, M.T. Rose

RFC 1215

Convention for defining traps for use with the SNMP M. Rose

RFC 1227

SNMP MUX protocol and MIB M.T. Rose

RFC 1228

SNMP-DPI: Simple Network Management Protocol Distributed Program Interface G. Carpenter, B. Wijnen

RFC 1229

Extensions to the generic-interface MIB K. McCloghrie

RFC 1230

IEEE 802.4 Token Bus MIB K. McCloghrie, R. Fox

RFC 1231

IEEE 802.5 Token Ring MIB K. McCloghrie, R. Fox, E. Decker

RFC 1236

IP to X.121 address mapping for DDN L. Morales, P. Hasse

RFC 1256

ICMP Router Discovery Messages S. Deering, Ed.

RFC 1267

Border Gateway Protocol 3 (BGP-3) K. Lougheed, Y. Rekhter

RFC 1268

Application of the Border Gateway Protocol in the Internet Y. Rekhter, P. Gross

RFC 1269

Definitions of Managed Objects for the Border Gateway Protocol: Version 3 S. Willis, J. Burruss

RFC 1270

SNMP Communications Services F. Kastenholz, ed.

RFC 1285

FDDI Management Information Base J. Case

RFC 1315

Management Information Base for Frame Relay DTEs C. Brown, F. Baker, C. Carvalho

RFC 1321

The MD5 Message-Digest Algorithm R. Rivest

RFC 1323

TCP Extensions for High Performance V. Jacobson, R. Braden, D. Borman

RFC 1325

FYI on Questions and Answers: Answers to Commonly Asked "New Internet User" Questions G. Malkin, A. Marine

RFC 1327

Mapping between X.400 (1988)/ISO 10021 and RFC 822 S. Hardcastle-Kille

RFC 1340

Assigned Numbers J. Reynolds, J. Postel

RFC 1344

Implications of MIME for Internet Mail Gateways N. Bornstein

RFC 1349

Type of Service in the Internet Protocol Suite P. Almquist

RFC 1351

SNMP Administrative Model J. Davin, J. Galvin, K. McCloghrie

SNMP Security Protocols J. Galvin, K. McCloghrie, J. Davin

RFC 1353

Definitions of Managed Objects for Administration of SNMP Parties K. McCloghrie, J. Davin, J. Galvin

RFC 1354

IP Forwarding Table MIB F. Baker

RFC 1356

Multiprotocol Interconnect® on X.25 and ISDN in the Packet Mode A. Malis, D. Robinson, R. Ullmann

RFC 1358

Charter of the Internet Architecture Board (IAB) L. Chapin

RFC 1363

A Proposed Flow Specification C. Partridge

RFC 1368

Definition of Managed Objects for IEEE 802.3 Repeater Devices D. McMaster, K. McCloghrie

RFC 1372

Telnet Remote Flow Control Option C. L. Hedrick, D. Borman

RFC 1374

IP and ARP on HIPPI J. Renwick, A. Nicholson

RFC 1381

SNMP MIB Extension for X.25 LAPB D. Throop, F. Baker

RFC 1382

SNMP MIB Extension for the X.25 Packet Layer D. Throop

RFC 1387

RIP Version 2 Protocol Analysis G. Malkin

RFC 1388

RIP Version 2 Carrying Additional Information G. Malkin

RFC 1389

RIP Version 2 MIB Extensions G. Malkin, F. Baker

RFC 1390

Transmission of IP and ARP over FDDI Networks D. Katz

RFC 1393

Traceroute Using an IP Option G. Malkin

RFC 1398

Definitions of Managed Objects for the Ethernet-Like Interface Types F. Kastenholz

RFC 1408

Telnet Environment Option D. Borman, Ed.

RFC 1413

Identification Protocol M. St. Johns

RFC 1416

Telnet Authentication Option D. Borman, ed.

RFC 1420

SNMP over IPX S. Bostock

RFC 1428

Transition of Internet Mail from Just-Send-8 to 8bit-SMTP/MIME G. Vaudreuil

RFC 1442

Structure of Management Information for version 2 of the Simple Network Management Protocol (SNMPv2) J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1443

Textual Conventions for version 2 of the Simple Network Management Protocol (SNMPv2) J. Case, K. McCloghrie, M. Rose, S. Waldbusser

Administrative Model for version 2 of the Simple Network Management Protocol (SNMPv2) J. Galvin, K. McCloghrie

RFC 1447

Party MIB for version 2 of the Simple Network Management Protocol (SNMPv2) K. McCloghrie, J. Galvin

RFC 1448

Protocol Operations for version 2 of the Simple Network Management Protocol (SNMPv2) J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1464

Using the Domain Name System to Store Arbitrary String Attributes R. Rosenbaum

RFC 1469

IP Multicast over Token-Ring Local Area Networks T. Pusateri

RFC 1483

Multiprotocol Encapsulation over ATM Adaptation Layer 5 Juha Heinanen

RFC 1514

Host Resources MIB P. Grillo, S. Waldbusser

RFC 1516

Definitions of Managed Objects for IEEE 802.3 Repeater Devices D. McMaster, K. McCloghrie

RFC 1521

MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies N. Borenstein, N. Freed

RFC 1535

A Security Problem and Proposed Correction With Widely Deployed DNS Software E. Gavron

RFC 1536

Common DNS Implementation Errors and Suggested Fixes A. Kumar, J. Postel, C. Neuman, P. Danzig, S. Miller

RFC 1537

Common DNS Data File Configuration Errors P. Beertema

RFC 1540

Internet Official Protocol Standards J. Postel

RFC 1571

Telnet Environment Option Interoperability Issues D. Borman

RFC 1572

Telnet Environment Option S. Alexander

RFC 1573

Evolution of the Interfaces Group of MIB-II K. McCloghrie, F. Kastenholz

RFC 1577

Classical IP and ARP over ATM M. Laubach

RFC 1583

OSPF Version 2 J. Moy

RFC 1591

Domain Name System Structure and Delegation J. Postel

RFC 1592

Simple Network Management Protocol Distributed Protocol Interface Version 2.0 B. Wijnen, G. Carpenter, K. Curran, A. Sehgal, G. Waters

RFC 1594

FYI on Questions and Answers— Answers to Commonly Asked "New Internet User" Questions A. Marine, J. Reynolds, G. Malkin

RFC 1644

T/TCP – TCP Extensions for Transactions Functional Specification R. Braden

TN3270 Extensions for LUname and Printer Selection C. Graves, T. Butts, M. Angel

RFC 1647

TN3270 Enhancements B. Kelly

RFC 1652

SMTP Service Extension for 8bit-MIMEtransport J. Klensin, N. Freed, M. Rose, E. Stefferud, D. Crocker

RFC 1664

Using the Internet DNS to Distribute RFC1327 Mail Address Mapping Tables C. Allochio, A. Bonito, B. Cole, S. Giordano, R. Hagens

RFC 1693

An Extension to TCP: Partial Order Service T. Connolly, P. Amer, P. Conrad

RFC 1695

Definitions of Managed Objects for ATM Management Version 8.0 using SMIv2 M. Ahmed, K. Tesink

RFC 1701

Generic Routing Encapsulation (GRE) S. Hanks, T. Li, D. Farinacci, P. Traina

RFC 1702

Generic Routing Encapsulation over IPv4 networks S. Hanks, T. Li, D. Farinacci, P. Traina

RFC 1706

DNS NSAP Resource Records B. Manning, R. Colella

RFC 1712

DNS Encoding of Geographical Location C. Farrell, M. Schulze, S. Pleitner D. Baldoni

RFC 1713

Tools for DNS debugging A. Romao

RFC 1723

RIP Version 2-Carrying Additional Information G. Malkin

RFC 1752

The Recommendation for the IP Next Generation Protocol S. Bradner, A. Mankin

RFC 1766

Tags for the Identification of Languages H. Alvestrand

RFC 1771

A Border Gateway Protocol 4 (BGP-4) Y. Rekhter, T. Li

RFC 1794

DNS Support for Load Balancing T. Brisco

RFC 1819

Internet Stream Protocol Version 2 (ST2) Protocol Specification—Version ST2+ L. Delgrossi, L. Berger Eds.

RFC 1826

IP Authentication Header R. Atkinson

RFC 1828

IP Authentication using Keyed MD5 P. Metzger, W. Simpson

RFC 1829

The ESP DES-CBC Transform P. Karn, P. Metzger, W. Simpson

RFC 1830

SMTP Service Extensions for Transmission of Large and Binary MIME Messages G. Vaudreuil

RFC 1831

RPC: Remote Procedure Call Protocol Specification Version 2 R. Srinivasan

RFC 1832

XDR: External Data Representation Standard R. Srinivasan

RFC 1833

Binding Protocols for ONC RPC Version 2 R. Srinivasan

OSPF Version 2 Management Information Base F. Baker, R. Coltun

RFC 1854

SMTP Service Extension for Command Pipelining N. Freed

RFC 1869

SMTP Service Extensions J. Klensin, N. Freed, M. Rose, E. Stefferud, D. Crocker

RFC 1870

SMTP Service Extension for Message Size Declaration J. Klensin, N. Freed, K. Moore

RFC 1876

A Means for Expressing Location Information in the Domain Name System C. Davis, P. Vixie, T. Goodwin, I. Dickinson

RFC 1883

Internet Protocol, Version 6 (IPv6) Specification S. Deering, R. Hinden

RFC 1884

IP Version 6 Addressing Architecture R. Hinden, S. Deering, Eds.

RFC 1886

DNS Extensions to support IP version 6 S. Thomson, C. Huitema

RFC 1888

OSI NSAPs and IPv6 J. Bound, B. Carpenter, D. Harrington, J. Houldsworth, A. Lloyd

RFC 1891

SMTP Service Extension for Delivery Status Notifications K. Moore

RFC 1892

The Multipart/Report Content Type for the Reporting of Mail System Administrative Messages G. Vaudreuil

RFC 1894

An Extensible Message Format for Delivery Status NotificationsK. Moore, G. Vaudreuil

RFC 1901

Introduction to Community-based SNMPv2 J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1902

Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2) J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1903

Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2) J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1904

Conformance Statements for Version 2 of the Simple Network Management Protocol (SNMPv2) J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1905

Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2) J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1906

Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2) J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1907

Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2) J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1908

Coexistence between Version 1 and Version 2 of the Internet-standard Network Management Framework J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1912

Common DNS Operational and Configuration Errors D. Barr

Address Allocation for Private Internets Y. Rekhter, B. Moskowitz, D. Karrenberg, G.J. de Groot, E. Lear

RFC 1928

SOCKS Protocol Version 5 M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, L. Jones

RFC 1930

Guidelines for creation, selection, and registration of an Autonomous System (AS) J. Hawkinson, T. Bates

RFC 1939

Post Office Protocol-Version 3 J. Myers, M. Rose

RFC 1981

Path MTU Discovery for IP version 6 J. McCann, S. Deering, J. Mogul

RFC 1982

Serial Number Arithmetic R. Elz, R. Bush

RFC 1985

SMTP Service Extension for Remote Message Queue Starting J. De Winter

RFC 1995

Incremental Zone Transfer in DNS M. Ohta

RFC 1996

A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY) P. Vixie

RFC 2010

Operational Criteria for Root Name Servers B. Manning, P. Vixie

RFC 2011

SNMPv2 Management Information Base for the Internet Protocol using SMIv2 K. McCloghrie, Ed.

RFC 2012

SNMPv2 Management Information Base for the Transmission Control Protocol using SMIv2 K. McCloghrie, Ed.

RFC 2013

SNMPv2 Management Information Base for the User Datagram Protocol using SMIv2 K. McCloghrie, Ed.

RFC 2018

TCP Selective Acknowledgement Options M. Mathis, J. Mahdavi, S. Floyd, A. Romanow

RFC 2026

The Internet Standards Process – Revision 3 S. Bradner

RFC 2030

Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI D. Mills

RFC 2033

Local Mail Transfer Protocol J. Myers

RFC 2034

SMTP Service Extension for Returning Enhanced Error CodesN. Freed

RFC 2040

The RC5, RC5-CBC, RC-5-CBC-Pad, and RC5-CTS AlgorithmsR. Baldwin, R. Rivest

RFC 2045

Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies N. Freed, N. Borenstein

RFC 2052

A DNS RR for specifying the location of services (DNS SRV) A. Gulbrandsen, P. Vixie

RFC 2065

Domain Name System Security Extensions D. Eastlake 3rd, C. Kaufman

RFC 2066

TELNET CHARSET Option R. Gellens

RIPng for IPv6 G. Malkin, R. Minnear

RFC 2096

IP Forwarding Table MIB F. Baker

RFC 2104

HMAC: Keyed-Hashing for Message Authentication H. Krawczyk, M. Bellare, R. Canetti

RFC 2119

Keywords for use in RFCs to Indicate Requirement Levels S. Bradner

RFC 2133

Basic Socket Interface Extensions for IPv6 R. Gilligan, S. Thomson, J. Bound, W. Stevens

RFC 2136

Dynamic Updates in the Domain Name System (DNS UPDATE) P. Vixie, Ed., S. Thomson, Y. Rekhter, J. Bound

RFC 2137

Secure Domain Name System Dynamic Update D. Eastlake 3rd

RFC 2163

Using the Internet DNS to Distribute MIXER Conformant Global Address Mapping (MCGAM) C. Allocchio

RFC 2168

Resolution of Uniform Resource Identifiers using the Domain Name System R. Daniel, M. Mealling

RFC 2178

OSPF Version 2 J. Moy

RFC 2181

Clarifications to the DNS Specification R. Elz, R. Bush

RFC 2205

Resource ReSerVation Protocol (RSVP)–Version 1 Functional Specification R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, S. Jamin

RFC 2210

The Use of RSVP with IETF Integrated Services J. Wroclawski

RFC 2211

Specification of the Controlled-Load Network Element Service J. Wroclawski

RFC 2212

Specification of Guaranteed Quality of Service S. Shenker, C. Partridge, R. Guerin

RFC 2215

General Characterization Parameters for Integrated Service Network Elements S. Shenker, J. Wroclawski

RFC 2217

Telnet Com Port Control Option G. Clarke

RFC 2219

Use of DNS Aliases for Network Services M. Hamilton, R. Wright

RFC 2228

FTP Security Extensions M. Horowitz, S. Lunt

RFC 2230

Key Exchange Delegation Record for the DNS R. Atkinson

RFC 2233

The Interfaces Group MIB using SMIv2 K. McCloghrie, F. Kastenholz

RFC 2240

A Legal Basis for Domain Name Allocation O. Vaughn

RFC 2246

The TLS Protocol Version 1.0 T. Dierks, C. Allen

Lightweight Directory Access Protocol (v3) M. Wahl, T. Howes, S. Kille

RFC 2253

Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names M. Wahl, S. Kille, T. Howes

RFC 2254

The String Representation of LDAP Search Filters T. Howes

RFC 2261

An Architecture for Describing SNMP Management Frameworks D. Harrington, R. Presuhn, B. Wijnen

RFC 2262

Message Processing and Dispatching for the Simple Network Management Protocol (SNMP) J. Case, D. Harrington, R. Presuhn, B. Wijnen

RFC 2271

An Architecture for Describing SNMP Management Frameworks D. Harrington, R. Presuhn, B. Wijnen

RFC 2273

SNMPv3 Applications D. Levi, P. Meyer, B. Stewartz

RFC 2274

User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3) U. Blumenthal, B. Wijnen

RFC 2275

View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP) B. Wijnen, R. Presuhn, K. McCloghrie

RFC 2279

UTF-8, a transformation format of ISO 10646 F. Yergeau

RFC 2292

Advanced Sockets API for IPv6 W. Stevens, M. Thomas

RFC 2308

Negative Caching of DNS Queries (DNS NCACHE) M. Andrews

RFC 2317

Classless IN-ADDR.ARPA delegation H. Eidnes, G. de Groot, P. Vixie

RFC 2320

Definitions of Managed Objects for Classical IP and ARP Over ATM Using SMIv2 (IPOA-MIB) M. Greene, J. Luciani, K. White, T. Kuo

RFC 2328

OSPF Version 2 J. Moy

RFC 2345

Domain Names and Company Name Retrieval J. Klensin, T. Wolf, G. Oglesby

RFC 2352

A Convention for Using Legal Names as Domain Names O. Vaughn

RFC 2355

TN3270 Enhancements B. Kelly

RFC 2358

Definitions of Managed Objects for the Ethernet-like Interface Types J. Flick, J. Johnson

RFC 2373

IP Version 6 Addressing Architecture R. Hinden, S. Deering

RFC 2374

An IPv6 Aggregatable Global Unicast Address Format R. Hinden, M. O'Dell, S. Deering

RFC 2375

IPv6 Multicast Address Assignments R. Hinden, S. Deering

Protection of BGP Sessions via the TCP MD5 Signature Option A. Hefferman

RFC 2389

Feature negotiation mechanism for the File Transfer Protocol P. Hethmon, R. Elz

RFC 2401

Security Architecture for Internet Protocol S. Kent, R. Atkinson

RFC 2402

IP Authentication Header S. Kent, R. Atkinson

RFC 2403

The Use of HMAC-MD5-96 within ESP and AH C. Madson, R. Glenn

RFC 2404

The Use of HMAC-SHA-1-96 within ESP and AH C. Madson, R. Glenn

RFC 2405

The ESP DES-CBC Cipher Algorithm With Explicit IV C. Madson, N. Doraswamy

RFC 2406

IP Encapsulating Security Payload (ESP) S. Kent, R. Atkinson

RFC 2407

The Internet IP Security Domain of Interpretation for ISAKMPD. Piper

RFC 2408

Internet Security Association and Key Management Protocol (ISAKMP) D. Maughan, M. Schertler, M. Schneider, J. Turner

RFC 2409

The Internet Key Exchange (IKE) D. Harkins, D. Carrel

RFC 2410

The NULL Encryption Algorithm and Its Use With IPsec R. Glenn, S. Kent,

RFC 2428

FTP Extensions for IPv6 and NATs M. Allman, S. Ostermann, C. Metz

RFC 2445

Internet Calendaring and Scheduling Core Object Specification (iCalendar) F. Dawson, D. Stenerson

RFC 2459

Internet X.509 Public Key Infrastructure Certificate and CRL Profile R. Housley, W. Ford, W. Polk, D. Solo

RFC 2460

Internet Protocol, Version 6 (IPv6) Specification S. Deering, R. Hinden

RFC 2461

Neighbor Discovery for IP Version 6 (IPv6) T. Narten, E. Nordmark, W. Simpson

RFC 2462

IPv6 Stateless Address Autoconfiguration S. Thomson, T. Narten

RFC 2463

Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification A. Conta, S. Deering

RFC 2464

Transmission of IPv6 Packets over Ethernet Networks M. Crawford

RFC 2466

Management Information Base for IP Version 6: ICMPv6 Group D. Haskin, S. Onishi

RFC 2476

Message Submission R. Gellens, J. Klensin

RFC 2487

SMTP Service Extension for Secure SMTP over TLS P. Hoffman

RFC 2505

Anti-Spam Recommendations for SMTP MTAs G. Lindberg

Photuris: Extended Schemes and Attributes P. Karn, W. Simpson

RFC 2535

Domain Name System Security Extensions D. Eastlake 3rd

RFC 2538

Storing Certificates in the Domain Name System (DNS) D. Eastlake 3rd, O. Gudmundsson

RFC 2539

Storage of Diffie-Hellman Keys in the Domain Name System (DNS) D. Eastlake 3rd

RFC 2540

Detached Domain Name System (DNS) Information D. Eastlake 3rd

RFC 2554

SMTP Service Extension for Authentication J. Myers

RFC 2570

Introduction to Version 3 of the Internet-standard Network Management Framework J. Case, R. Mundy, D. Partain, B. Stewart

RFC 2571

An Architecture for Describing SNMP Management Frameworks B. Wijnen, D. Harrington, R. Presuhn

RFC 2572

Message Processing and Dispatching for the Simple Network Management Protocol (SNMP) J. Case, D. Harrington, R. Presuhn, B. Wijnen

RFC 2573

SNMP Applications D. Levi, P. Meyer, B. Stewart

RFC 2574

User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3) U. Blumenthal, B. Wijnen

RFC 2575

View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP) B. Wijnen, R. Presuhn, K. McCloghrie

RFC 2576

Co-Existence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework R. Frye, D. Levi, S. Routhier, B. Wijnen

RFC 2578

Structure of Management Information Version 2 (SMIv2) K. McCloghrie, D. Perkins, J. Schoenwaelder

RFC 2579

Textual Conventions for SMIv2 K. McCloghrie, D. Perkins, J. Schoenwaelder

RFC 2580

Conformance Statements for SMIv2 K. McCloghrie, D. Perkins, J. Schoenwaelder

RFC 2581

TCP Congestion Control M. Allman, V. Paxson, W. Stevens

RFC 2583

Guidelines for Next Hop Client (NHC) Developers R. Carlson, L. Winkler

RFC 2591

Definitions of Managed Objects for Scheduling Management Operations D. Levi, J. Schoenwaelder

RFC 2625

IP and ARP over Fibre Channel M. Rajagopal, R. Bhagwat, W. Rickard

RFC 2635

Don't SPEW A Set of Guidelines for Mass Unsolicited Mailings and Postings (spam*) S. Hambridge, A. Lunde

RFC 2637

Point-to-Point Tunneling Protocol K. Hamzeh, G. Pall, W. Verthein, J. Taarud, W. Little, G. Zorn

Internationalization of the File Transfer Protocol B. Curtin

RFC 2665

Definitions of Managed Objects for the Ethernet-like Interface Types J. Flick, J. Johnson

RFC 2671

Extension Mechanisms for DNS (EDNSO) P. Vixie

RFC 2672

Non-Terminal DNS Name Redirection M. Crawford

RFC 2675

IPv6 Jumbograms D. Borman, S. Deering, R. Hinden

RFC 2710

Multicast Listener Discovery (MLD) for IPv6 S. Deering, W. Fenner, B. Haberman

RFC 2711

IPv6 Router Alert Option C. Partridge, A. Jackson

RFC 2740

OSPF for IPv6 R. Coltun, D. Ferguson, J. Moy

RFC 2753

A Framework for Policy-based Admission Control R. Yavatkar, D. Pendarakis, R. Guerin

RFC 2782

A DNS RR for specifying the location of services (DNS SRV) A. Gubrandsen, P. Vixix, L. Esibov

RFC 2821

Simple Mail Transfer Protocol J. Klensin, Ed.

RFC 2822

Internet Message Format P. Resnick, Ed.

RFC 2840

TELNET KERMIT OPTION J. Altman, F. da Cruz

RFC 2845

Secret Key Transaction Authentication for DNS (TSIG) P. Vixie, O. Gudmundsson, D. Eastlake 3rd, B. Wellington

RFC 2851

Textual Conventions for Internet Network Addresses M. Daniele, B. Haberman, S. Routhier, J. Schoenwaelder

RFC 2852

Deliver By SMTP Service Extension D. Newman

RFC 2874

DNS Extensions to Support IPv6 Address Aggregation and Renumbering M. Crawford, C. Huitema

RFC 2915

The Naming Authority Pointer (NAPTR) DNS Resource Record M. Mealling, R. Daniel

RFC 2920

SMTP Service Extension for Command Pipelining N. Freed

RFC 2930

Secret Key Establishment for DNS (TKEY RR) D. Eastlake, 3rd

RFC 2941

Telnet Authentication Option T. Ts'o, ed., J. Altman

RFC 2942

Telnet Authentication: Kerberos Version 5 T. Ts'o

RFC 2946

Telnet Data Encryption Option T. Ts'o

RFC 2952

Telnet Encryption: DES 64 bit Cipher Feedback T. Ts'o

Telnet Encryption: DES 64 bit Output Feedback T. Ts'o

RFC 2992

Analysis of an Equal-Cost Multi-Path Algorithm C. Hopps

RFC 3019

IP Version 6 Management Information Base for The Multicast Listener Discovery Protocol B. Haberman, R. Worzella

RFC 3060

Policy Core Information Model—Version 1 Specification B. Moore, E. Ellesson, J. Strassner, A. Westerinen

RFC 3152

Delegation of IP6.ARPA R. Bush

RFC 3164

The BSD Syslog Protocol C. Lonvick

RFC 3207

SMTP Service Extension for Secure SMTP over Transport Layer Security P. Hoffman

RFC 3226

DNSSEC and IPv6 A6 aware server/resolver message size requirements O. Gudmundsson

RFC 3291

Textual Conventions for Internet Network Addresses M. Daniele, B. Haberman, S. Routhier, J. Schoenwaelder

RFC 3363

Representing Internet Protocol version 6 (IPv6) Addresses in the Domain Name System R. Bush, A. Durand, B. Fink, O. Gudmundsson, T. Hain

RFC 3376

Internet Group Management Protocol, Version 3 B. Cain, S. Deering, I. Kouvelas, B. Fenner, A. Thyagarajan

RFC 3390

Increasing TCP's Initial Window M. Allman, S. Floyd, C. Partridge

RFC 3410

Introduction and Applicability Statements for Internet-Standard Management Framework J. Case, R. Mundy, D. Partain, B. Stewart

RFC 3411

An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks D. Harrington, R. Presuhn, B. Wijnen

RFC 3412

Message Processing and Dispatching for the Simple Network Management Protocol (SNMP) J. Case, D. Harrington, R. Presuhn, B. Wijnen

RFC 3413

Simple Network Management Protocol (SNMP) Applications D. Levi, P. Meyer, B. Stewart

RFC 3414

User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3) U. Blumenthal, B. Wijnen

RFC 3415

View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP) B. Wijnen, R. Presuhn, K. McCloghrie

RFC 3416

Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP) R. Presuhn, J. Case, K. McCloghrie, M. Rose, S. Waldbusser

Transport Mappings for the Simple Network Management Protocol (SNMP) R. Presuhn, J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 3418

Management Information Base (MIB) for the Simple Network Management Protocol (SNMP) R. Presuhn, J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 3419

Textual Conventions for Transport Addresses M. Daniele, J. Schoenwaelder

RFC 3484

Default Address Selection for Internet Protocol version 6 (IPv6) R. Draves

RFC 3493

Basic Socket Interface Extensions for IPv6 R. Gilligan, S. Thomson, J. Bound, J. McCann, W. Stevens

RFC 3513

Internet Protocol Version 6 (IPv6) Addressing Architecture R. Hinden, S. Deering

RFC 3526

More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE) T. Kivinen, M. Kojo

RFC 3542

Advanced Sockets Application Programming Interface (API) for IPv6 W. Richard Stevens, M. Thomas, E. Nordmark, T. Jinmei

RFC 3566

The AES-XCBC-MAC-96 Algorithm and Its Use With IPsec S. Frankel, H. Herbert

RFC 3569

An Overview of Source-Specific Multicast (SSM) S. Bhattacharyya, Ed.

RFC 3584

Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework R. Frye, D. Levi, S. Routhier, B. Wijnen

RFC 3602

The AES-CBC Cipher Algorithm and Its Use with IPsec S. Frankel, R. Glenn, S. Kelly

RFC 3629

UTF-8, a transformation format of ISO 10646 R. Kermode, C. Vicisano

RFC 3658

Delegation Signer (DS) Resource Record (RR) O. Gudmundsson

RFC 3678

Socket Interface Extensions for Multicast Source Filters D. Thaler, B. Fenner, B. Quinn

RFC 3715

IPsec-Network Address Translation (NAT) Compatibility Requirements B. Aboba, W. Dixon

RFC 3810

Multicast Listener Discovery Version 2 (MLDv2) for IPv6 R. Vida, Ed., L. Costa, Ed.

RFC 3826

The Advanced Encryption Standard (AES) Cipher Algorithm in the SNMP User-based Security Model U. Blumenthal, F. Maino, K McCloghrie.

RFC 3947

Negotiation of NAT-Traversal in the IKE T. Kivinen, B. Swander, A. Huttunen, V. Volpe

RFC 3948

UDP Encapsulation of IPsec ESP Packets A. Huttunen, B. Swander, V. Volpe, L. DiBurro, M. Stenberg

RFC 4001

Textual Conventions for Internet Network Addresses M. Daniele, B. Haberman, S. Routhier, J. Schoenwaelder

RFC 4007

IPv6 Scoped Address Architecture S. Deering, B. Haberman, T. Jinmei, E. Nordmark, B. Zill
RFC 4022

Management Information Base for the Transmission Control Protocol (TCP) R. Raghunarayan

RFC 4106

The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP) J. Viega, D. McGrew

RFC 4109

Algorithms for Internet Key Exchange version 1 (IKEv1) P. Hoffman

RFC 4113

Management Information Base for the User Datagram Protocol (UDP) B. Fenner, J. Flick

RFC 4191

Default Router Preferences and More-Specific Routes R. Draves, D. Thaler

RFC 4217

Securing FTP with TLS P. Ford-Hutchinson

RFC 4292

IP Forwarding Table MIB B. Haberman

RFC 4293

Management Information Base for the Internet Protocol (IP) S. Routhier

RFC 4301

Security Architecture for the Internet Protocol S. Kent, K. Seo

RFC 4302

IP Authentication Header S. Kent

RFC 4303

IP Encapsulating Security Payload (ESP) S. Kent

RFC 4304

Extended Sequence Number (ESN) Addendum to IPsec Domain of Interpretation (DOI) for Internet Security Association and Key Management Protocol (ISAKMP) S. Kent

RFC 4307

Cryptographic Algorithms for Use in the Internet Key Exchange Version 2 (IKEv2) J. Schiller

RFC 4308

Cryptographic Suites for IPsec P. Hoffman

RFC 4434

The AES-XCBC-PRF-128 Algorithm for the Internet Key Exchange Protocol P. Hoffman

RFC 4443

Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification A. Conta, S. Deering

RFC 4552

Authentication/Confidentiality for OSPFv3 M. Gupta, N. Melam

RFC 4678

Server/Application State Protocol v1 A. Bivens

RFC 4753

ECP Groups for IKE and IKEv2 D. Fu, J. Solinas

RFC 4754

IKE and IKEv2 Authentication Using the Elliptic Curve Digital Signature Algorithm (ECDSA) D. Fu, J. Solinas

RFC 4809

Requirements for an IPsec Certificate Management Profile C. Bonatti, Ed., S. Turner, Ed., G. Lebovitz, Ed.

RFC 4835

Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH) V. Manral

RFC 4862

IPv6 Stateless Address Autoconfiguration S. Thomson, T. Narten, T. Jinmei

RFC 4868

Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec S. Kelly, S. Frankel

RFC 4869

Suite B Cryptographic Suites for IPsec L. Law, J. Solinas

RFC 4941

Privacy Extensions for Stateless Address Autoconfiguration in IPv6 T. Narten, R. Draves, S. Krishnan

RFC 4945

The Internet IP Security PKI Profile of IKEv1/ISAKMP, IKEv2, and PKIX B. Korver

RFC 5014

IPv6 Socket API for Source Address Selection E. Nordmark, S. Chakrabarti, J. Laganier

RFC 5095

Deprecation of Type 0 Routing Headers in IPv6 J. Abley, P. Savola, G. Neville-Neil

RFC 5175

IPv6 Router Advertisement Flags Option B. Haberman, Ed., R. Hinden

RFC 5282

Using Authenticated Encryption Algorithms with the Encrypted Payload of the Internet Key Exchange version 2 (IKEv2) Protocol D. Black, D. McGrew

RFC 5996

Internet Key Exchange Protocol Version 2 (IKEv2) C. Kaufman, P. Hoffman, Y. Nir, P. Eronen

RFC 7627

Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension K. Bhargavan, A. Delignat-Lavaud, A. Pironti, Inria Paris-Rocquencourt, A. Langley, M. Ray

RFC 8446

The Transport Layer Security (TLS) Protocol Version 1.3 E. Rescorla

Internet drafts

Internet drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Other groups can also distribute working documents as Internet drafts. You can see Internet drafts at http://www.ietf.org/ID.html.

Appendix C. Accessibility

Publications for this product are offered in Adobe Portable Document Format (PDF) and should be compliant with accessibility standards. If you experience difficulties when using PDF files, you can view the information through the z/OS Internet Library website <u>http://www.ibm.com/systems/z/os/zos/library/bkserv/</u> or IBM Documentation <u>https://www.ibm.com/docs/en</u>. If you continue to experience problems, send a message to Contact z/OS web page(www.ibm.com/systems/z/os/zos/webqs.html) or write to:

IBM Corporation Attention: MHVRCFS Reader Comments Department H6MA, Building 707 2455 South Road Poughkeepsie, NY 12601-5400 USA

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. See <u>z/OS TSO/E Primer</u>, <u>z/OS TSO/E User's</u> <u>Guide</u>, and <u>z/OS ISPF User's Guide Vol I</u> for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

z/OS information

One exception is command syntax that is published in railroad track format, which is accessible using screen readers with IBM Documentation, as described in <u>"Dotted decimal syntax diagrams" on page 299</u>.

Dotted decimal syntax diagrams

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

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

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

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

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

If a syntax element is preceded by the % symbol, this indicates a reference that is defined elsewhere. The string following the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you should see separate syntax fragment OP1.

The following words and symbols are used next to the dotted decimal numbers:

- A question mark (?) means an optional syntax element. A dotted decimal number followed by the ? symbol indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that syntax elements NOTIFY and UPDATE are optional; that is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.
- An exclamation mark (!) means a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicate that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the same dotted decimal number can specify a ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In this example, if you include the FILE keyword but do not specify an option, default option KEEP will be applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1.1 (DELETE), the default option KEEP applies only to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.
- An asterisk (*) means a syntax element that can be repeated 0 or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3*, 3 HOST, and 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

Notes:

- 1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
- 2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you could write HOST STATE, but you could not write HOST HOST.
- 3. The * symbol is equivalent to a loop-back line in a railroad syntax diagram.

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

302 z/OS Communications Server: z/OS V2R5.0 Communications Server: IP IMS Sockets Guide

Notices

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

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

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

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

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

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

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

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

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

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

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

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

IBM Corporation Site Counsel 2455 South Road Poughkeepsie, NY 12601-5400 USA

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

Terms and conditions for product documentation

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

Applicability

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

Personal use

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

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

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

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

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

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

Policy for unsupported hardware

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

Minimum supported hardware

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

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

Policy for unsupported hardware

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

Trademarks

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

Bibliography

This bibliography contains descriptions of the documents in the z/OS Communications Server library.

z/OS Communications Server documentation is available online at the z/OS Internet Library web page at http://www.ibm.com/systems/z/os/zos/library/bkserv/.

z/OS Communications Server library updates

Updates to documents are also available on RETAIN and in information APARs (info APARs). Go to <u>https://</u>www.ibm.com/mysupport to view information APARs.

- z/OS Communications Server V2R1 New Function APAR Summary
- z/OS Communications Server V2R2 New Function APAR Summary
- z/OS Communications Server V2R3 New Function APAR Summary
- z/OS Communications Server V2R4 New Function APAR Summary

z/OS Communications Server information

z/OS Communications Server product information is grouped by task in the following tables.

Planning

Title	Number	Description
z/OS Communications Server: New Function Summary	GC27-3664	This document is intended to help you plan for new IP or SNA functions, whether you are migrating from a previous version or installing z/OS for the first time. It summarizes what is new in the release and identifies the suggested and required modifications needed to use the enhanced functions.
z/OS Communications Server: IPv6 Network and Application Design Guide	SC27-3663	This document is a high-level introduction to IPv6. It describes concepts of z/OS Communications Server's support of IPv6, coexistence with IPv4, and migration issues.

Resource definition, configuration, and tuning

Title	Number	Description
z/OS Communications Server: IP Configuration Guide	SC27-3650	This document describes the major concepts involved in understanding and configuring an IP network. Familiarity with the z/OS operating system, IP protocols, z/OS UNIX System Services, and IBM Time Sharing Option (TSO) is recommended. Use this document with the <u>z/OS</u> Communications Server: IP Configuration Reference.

Title	Number	Description
z/OS Communications Server: IP Configuration Reference	SC27-3651	This document presents information for people who want to administer and maintain IP. Use this document with the z/OS Communications Server: IP Configuration Guide. The information in this document includes:
		 TCP/IP configuration data sets
		 Configuration statements
		Translation tables
		 Protocol number and port assignments
z/OS Communications Server: SNA Network Implementation Guide	SC27-3672	This document presents the major concepts involved in implementing an SNA network. Use this document with the z/OS Communications Server: SNA Resource Definition Reference.
z/OS Communications Server: SNA Resource Definition Reference	SC27-3675	This document describes each SNA definition statement, start option, and macroinstruction for user tables. It also describes NCP definition statements that affect SNA. Use this document with the <u>z/OS Communications Server: SNA</u> Network Implementation Guide.
z/OS Communications Server: SNA Resource Definition Samples	SC27-3676	This document contains sample definitions to help you implement SNA functions in your networks, and includes sample major node definitions.
z/OS Communications Server: IP Network Print Facility	SC27-3658	This document is for systems programmers and network administrators who need to prepare their network to route SNA, JES2, or JES3 printer output to remote printers using TCP/IP Services.

Operation

Title	Number	Description
z/OS Communications Server: IP User's Guide and Commands	SC27-3662	This document describes how to use TCP/IP applications. It contains requests with which a user can log on to a remote host using Telnet, transfer data sets using FTP, send electronic mail, print on remote printers, and authenticate network users.
z/OS Communications Server: IP System Administrator's Commands	SC27-3661	This document describes the functions and commands helpful in configuring or monitoring your system. It contains system administrator's commands, such as TSO NETSTAT, PING, TRACERTE and their UNIX counterparts. It also includes TSO and MVS commands commonly used during the IP configuration process.
z/OS Communications Server: SNA Operation	SC27-3673	This document serves as a reference for programmers and operators requiring detailed information about specific operator commands.
z/OS Communications Server: Quick Reference	SC27-3665	This document contains essential information about SNA and IP commands.

Customization

Title	Number	Description
z/OS Communications Server: SNA Customization	SC27-3666	This document enables you to customize SNA, and includes the following information:
		Communication network management (CNM) routing table
		 Logon-interpret routine requirements
		 Logon manager installation-wide exit routine for the CLU search exit
		 TSO/SNA installation-wide exit routines
		 SNA installation-wide exit routines

Writing application programs

Title	Number	Description
z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference	SC27-3660	This document describes the syntax and semantics of program source code necessary to write your own application programming interface (API) into TCP/IP. You can use this interface as the communication base for writing your own client or server application. You can also use this document to adapt your existing applications to communicate with each other using sockets over TCP/IP.
z/OS Communications Server: IP CICS Sockets Guide	SC27-3649	This document is for programmers who want to set up, write application programs for, and diagnose problems with the socket interface for CICS using z/OS TCP/IP.
z/OS Communications Server: IP IMS Sockets Guide	SC27-3653	This document is for programmers who want application programs that use the IMS TCP/IP application development services provided by the TCP/IP Services of IBM.
z/OS Communications Server: IP Programmer's Guide and Reference	SC27-3659	This document describes the syntax and semantics of a set of high-level application functions that you can use to program your own applications in a TCP/IP environment. These functions provide support for application facilities, such as user authentication, distributed databases, distributed processing, network management, and device sharing. Familiarity with the z/OS operating system, TCP/IP protocols, and IBM Time Sharing Option (TSO) is recommended.
z/OS Communications Server: SNA Programming	SC27-3674	This document describes how to use SNA macroinstructions to send data to and receive data from (1) a terminal in either the same or a different domain, or (2) another application program in either the same or a different domain.
z/OS Communications Server: SNA Programmer's LU 6.2 Guide	SC27-3669	This document describes how to use the SNA LU 6.2 application programming interface for host application programs. This document applies to programs that use only LU 6.2 sessions or that use LU 6.2 sessions along with other session types. (Only LU 6.2 sessions are covered in this document.)
z/OS Communications Server: SNA Programmer's LU 6.2 Reference	SC27-3670	This document provides reference material for the SNA LU 6.2 programming interface for host application programs.

Title	Number	Description
z/OS Communications Server: CSM Guide	SC27-3647	This document describes how applications use the communications storage manager.

Diagnosis

Title	Number	Description
z/OS Communications Server: IP Diagnosis Guide	GC27-3652	This document explains how to diagnose TCP/IP problems and how to determine whether a specific problem is in the TCP/IP product code. It explains how to gather information for and describe problems to the IBM Software Support Center.
z/OS Communications Server: ACF/TAP Trace Analysis Handbook	GC27-3645	This document explains how to gather the trace data that is collected and stored in the host processor. It also explains how to use the Advanced Communications Function/Trace Analysis Program (ACF/TAP) service aid to produce reports for analyzing the trace data information.
z/OS Communications Server: SNA Diagnosis Vol 1, Techniques and Procedures and z/OS Communications Server: SNA Diagnosis Vol 2, FFST Dumps and the VIT	GC27-3667 GC27-3668	These documents help you identify an SNA problem, classify it, and collect information about it before you call the IBM Support Center. The information collected includes traces, dumps, and other problem documentation.
z/OS Communications Server: SNA Data Areas Volume 1 and z/OS Communications Server: SNA Data Areas Volume 2	GC31-6852 GC31-6853	These documents describe SNA data areas and can be used to read an SNA dump. They are intended for IBM programming service representatives and customer personnel who are diagnosing problems with SNA.

Messages and codes

Title	Number	Description
z/OS Communications Server: SNA Messages	SC27-3671	This document describes the ELM, IKT, IST, IUT, IVT, and USS messages. Other information in this document includes:
		 Command and RU types in SNA messages
		 Node and ID types in SNA messages
		 Supplemental message-related information
z/OS Communications Server: IP Messages Volume 1 (EZA)	SC27-3654	This volume contains TCP/IP messages beginning with EZA.
z/OS Communications Server: IP Messages Volume 2 (EZB, EZD)	SC27-3655	This volume contains TCP/IP messages beginning with EZB or EZD.
z/OS Communications Server: IP Messages Volume 3 (EZY)	SC27-3656	This volume contains TCP/IP messages beginning with EZY.
z/OS Communications Server: IP Messages Volume 4 (EZZ, SNM)	SC27-3657	This volume contains TCP/IP messages beginning with EZZ and SNM.
z/OS Communications Server: IP and SNA Codes	SC27-3648	This document describes codes and other information that appear in z/OS Communications Server messages.

Index

A

accept 20 ACCEPT (call) 55 accessibility 299 active sockets 46 active sockets queue 25 ADDRSPC parameter 46 ADDRSPCPFX parameter 46 AF parameter on call interface, on SOCKET 180 alternate PCB 22 APPC 1 application data 22, 25 application data, explicit mode data translation 30 end-of-message indicator 30 format 30 network byte order 30 application data, explicit-mode format 37, 38 protocol 37, 38 translation 37, 38 application data, implicit-mode data translation 32, 40 end-of-message 40 end-of-message indicator 32 format 32, 40 Application types 32701 client-server 1 ASCII to EBCDIC translation 30 ASMADLI 42 Assist module role of 19 trade-offs 19 use of IMS message queue 19

В

BACKLOG parameter 46 BACKLOG parameter on call interface, LISTEN call 128 backlog queue 25 backlog queue, length 46 bb status code 40, 42 **Berkeley Sockets** BSD 4.3 2 big-endian 30 BIND 20 BIND (call) 57 BIND2ADDRSEL (call) 59 bit-mask on call interface, on EZACIC06 call 191 bit-mask-length on call interface, on EZACIC06 call 191 BMP 46 BUF parameter on call socket interface on GETIBMOPT 86 on READ 134 on RECV 138

BUF parameter on call socket interface *(continued)* on RECVFROM <u>140</u> on SEND <u>155</u> on SENDTO <u>160</u> on WRITE <u>185</u> buffer full <u>34</u>

С

C language list of calls 17 CADLI 42 CALL Instruction Interface for Assembler, PL/I, and COBOL 51 Call Instructions for Assembler, PL/1, and COBOL Programs ACCEPT 55 BIND 57 **BIND2ADDRSEL 59** CLOSE 62 CONNECT 63 EZACIC04 188 **EZACIC05 189** EZACIC06 190 **EZACIC08 192** FCNTL 66 **GETCLIENTID 76 GETHOSTBYADDR 77 GETHOSTBYNAME 80 GETHOSTID 83 GETHOSTNAME 83 GETIBMOPT 85 GETPEERNAME 91 GETSOCKNAME 93 GETSOCKOPT 95 GIVESOCKET 111** INET6_IS_SRCADDR 113 **INITAPI 116** IOCTL 118 LISTEN 127 **READ 133** READV 134 **RECV 136 RECVFROM 138 RECVMSG 141** SELECT 145 SELECTEX 149 SENDMSG 155 SENDTO 159 SETSOCKOPT 161 SHUTDOWN 178 SOCKET 179 **TAKESOCKET 182 TERMAPI 183** WRITE 184 WRITEV 185 Call Instructions for Assembler, PL/I, and COBOL Programs **EZACIC14 197**

Call Instructions for Assembler, PL/I, and COBOL Programs (continued and parameter on call socket interface **EZACIC15 198** call interface sample PL/I programs 200 call sequence, explicit-mode client 30 CBLADLI 42 CHAR-MASK parameter on call interface, on EZACIC06 191 child server 9 CHNG 22 client defined 29 explicit-mode 29 logic flow 29 client call sequence, implicit-mode 30 CLIENT parameter on call socket interface on GETCLIENTID 77 on GIVESOCKET 112 on TAKESOCKET 183 client-server 1 client/server processing 4 COBOL language list of calls 17 codes, RSM reason 34 COMMAND parameter on call interface, IOCTL call 119 COMMAND parameter on call socket interface on EZACIC06 191 on FCNTL 67 on GETIBMOPT 86 COMMIT 37, 38 commit database updates 22 commit, explicit-mode 29 Communications Server for z/OS, online information xxiv complete-status message 35 concurrent server defined 8 illustrated 8.9 configuration file 46 configuring IMS TCP/IP 49 connection, how established 20 conversation, TCP/IP 20 CSMOKY 33, 35 CSMOKY message 30

D

data translation explicit-mode 30 data translation, socket interface ASCII to EBCDIC 189 bit-mask to character 190 character to bit-mask 190 EBCDIC to ASCII 188, 197 data, application 22, 25 database calls 22 database updates, commit 22 DataLen 47 DataType 47 disability 299 DNS, online information \underline{xxv}

E

EBCDIC to ASCII translation 30 ERETMSK parameter on call interface, on SELECT 149

on ACCEPT 57 on BIND 59 on BIND2ADDRSEL 62 on CLOSE 63 on CONNECT 66 on FCNTL 67 on GETCLIENTID 77 on GETHOSTNMAE 84 on GETIBMOPT 87 on GETPEERNAME 93 on GETSOCKNAME 95 on GETSOCKOPT 110 on GIVESOCKET 113 on INET6_IS_SRCADDR 115 on INITAPI 118 on IOCTL 127 on LISTEN 129 on READ 134 on READV 136 on RECV 138 on RECVFROM 141 on RECVMSG 145 on SELECT 149 on SELECTEX 153 on SEND 155 on SENDMSG 159 on SENDTO 161 on SETSOCKOPT 177 on SHUTDOWN 179 on SOCKET 181 on TAKESOCKET 183 on WRITE 185 on WRITEV 187 ERRNO parameter on macro socket interface on FCNTL 68, 76 ESDNMASK parameter on call interface, on SELECT 148 EWOULDBLOCK error return, call interface calls **RECV 136** RECVFROM 139 explicit-mode 2 explicit-mode client application data format 30 call sequence 30 data format 30 data translation 30 network byte order 30 explicit-mode server application data 37 call sequence 37 I/O PCB 37 PL/I programming 37 TIM 37 transaction-initiation message 37 EZACIC04, call interface, EBCDIC to ASCII translation 188 EZACIC05, call interface, ASCII to EBCDIC translation 189 EZACICO6 15 EZACIC06, call interface, bit-mask translation 190 EZACIC08, HOSTENT structure interpreter utility 192 EZACIC09, RES structure interpreter utility 194 EZACIC14, call interface, EBCDIC to ASCII translation 197 EZACIC15, call interface, ASCII to EBCDIC translation 198 EZASOKET

EZASOKET (continued) COBOL language call format <u>53</u> PL/I language call format <u>53</u>

F

FCNTL (call) <u>66</u> FLAGS parameter on call socket interface on RECV <u>137</u> on RECVFROM <u>140</u> on RECVMSG <u>144</u> on SEND <u>154</u> on SENDMSG <u>158</u> on SENDTO <u>160</u> FNDELAY flag on call interface, on FCNTL <u>67</u>

G

GETCLIENTID (call) 76 GETHOSTBYADDR (call) 77 GETHOSTBYNAME (call) 80 GETHOSTID (call) 83 GETHOSTNAME (call) 83 GETIBMOPT (call) 85 GETPEERNAME (call) 91 GETSOCKNAME (call) 93 GETSOCKOPT (call) 95 GIVESOCKET 22 GIVESOCKET (call) 111

Н

hlq.PROFILE.TCPIP data set <u>48</u> hlq.TCPIP.DATA data set <u>49</u> HOSTADDR parameter on call interface, on GETHOSTBYADDR <u>78</u> HOSTENT parameter on call socket interface on GETHOSTBYADDR <u>78</u> on GETHOSTBYADDR <u>78</u> HOSTENT structure interpreter parameters, on EZACIC08 <u>193</u> HOW parameter on call interface, on SHUTDOWN 179

I

I/O Area size 42 I/O PCB in explicit-mode server 38 IBM Software Support Center, contacting xviii IDENT parameter on call interface, INITAPI call 117 implicit mode 2 implicit-mode client 30 client call sequence 30 client logic flow 30 complete status message 30 CSM 30 data stream 30 transaction-request message 30 **TRM 30** implicit-mode client application data stream 33 application data, format 33 call sequence 33

implicit-mode client (continued) data format 33 data translation 33 end-of-message indicator 33 logic flow 33 implicit-mode server application data 40 Assist module 40 call sequence 40 I/O PCB 40 PL/I programming 40 programming 40 IMS Assist Module 2 IMS error 34 **IMS** Listener role of 19 use of IMS message queue 19 IMSLSECX, Listener security exit name 47 IN-BUFFER parameter on call interface, EZACIC05 call 190 INET6_IS_SRCADDR (call) 113 Information APARs xxii initapi 37, 38 INITAPI(call) 116 INOY 22 Internet, finding z/OS information online xxiv internets, TCP/IP 3 IOCTL (call) 118 IOV parameter on call socket interface on READV 135 on WRITEV 186 IOVCNT parameter on call socket interface on READV 136 on RECVMSG 144 on SENDMSG 158 on WRITEV 187 IP protocol 5 IpAddr 47 ISRT 40 iterative server defined 8 illustrated 9

Κ

keyboard 299

L

length of backlog queue <u>46</u> LENGTH parameter on call socket interface on EZACIC04 <u>188</u> on EZACIC05 <u>190</u> on EZACIC15 <u>190</u> license, patent, and copyright information <u>303</u> LISTEN <u>20</u> LISTEN (call) <u>127</u> Listener call sequence <u>26</u> Listener configuration file LISTENER statement <u>46</u> TCPIP statement <u>46</u> TRANSACTION statement <u>46</u> Listener ReasnCode <u>47</u> Listener RetnCode <u>47</u> Listener startup parameters <u>46</u> Listener statement <u>46</u> LISTNR <u>38</u> little-endian <u>30</u> LTERM name <u>43</u> LU 6.2 1

Μ

mainframe education xxii MAXACTSKT 25 MAXACTSKT parameter 46 MAXSNO parameter on call interface, INITAPI call 117 MAXSOC parameter on call socket interface on INITAPI 117 on SELECT 148 on SELECTEX 152 MAXTRANS parameter 46 Message Format Services 1 Message format services (MFS) 25 message queue 19, 20, 22 message queue, use of 25 messages complete-status message 35 MFS 1 MODE=SNGL 37 MSG parameter on call socket interface on RECVMSG 143 on SENDMSG 157 multiple connection requests 25

Ν

NAME parameter on call socket interface on ACCEPT 56 on BIND 58 on BIND2ADDRSEL 61 on CONNECT 65 on GETHOSTBYNAME 81 on GETHOSTNAME 84 on GETPEERNAME 92 on GETSOCKNAME 94 on INET6_IS_SRCADDR 114 on RECVFROM 140 NAMELEN parameter on call socket interface on GETHOSTBYNAME 81 on GETHOSTNAME 84 NBYTE parameter on call socket interface on READ 134 on RECV 138 on RECVFROM 140 on SEND 155 on SENDTO 160 on WRITE 185 network byte order 30

0

OSI <u>4</u> OUT-BUFFER parameter on call interface, on EZACICO4 <u>188</u> OUT-BUFFER parameter on call interface, on EZACIC14 <u>198</u> OUT-BUFFER parameter on call interface, on EZACIC15 $\underline{199}$ output area size $\underline{42}$ Overview 1

Ρ

pending activity 14 pending exception 15 pending read 15 PL/I coding 35 PLIADLI 42 Port 47 port numbers reserving port numbers 48 PORT parameter 46 ports compared with sockets 7 reserving port numbers 48 prerequisite information xxii program variable definitions, call interface assembler definition 54 COBOL PIC 54 PL/I declare 54 VS COBOL II PIC 54 PROTO parameter on call interface, on SOCKET 181 PURG call 42

Q

QC status code $\frac{40}{40}$, $\frac{42}{42}$ QD status code $\frac{40}{40}$, $\frac{42}{42}$

R

READ 22 READ (call) 133 READV (call) 134 ReasnCode, Listener 47 reason codes 34 RECV (call) 136 RECVFROM (call) 138 RECVMSG (call) 141 REQARG and RETARG parameter on call socket interface on FCNTL 67 on IOCTL 125 REQSTS 33 request-status message 33 Request-status message 29 requirements for IMS TCP/IP 16 RETARG parameter on call interface, on IOCTL 127 **RETCODE** parameter on call socket interface on ACCEPT 57 on BIND 59 on BIND2ADDRSEL 62 on CLOSE 63 on CONNECT 66 on EZACIC06 191 on FCNTL 67 on GETCLIENTID 77 on GETHOSTBYADDR 78 on GETHOSTBYNAME 81 on GETHOSTID 83 on GETHOSTNAME 84

RETCODE parameter on call socket interface (continued) on GETIBMOPT 87 on GETPEERNAME 93 on GETSOCKNAME 95 on GETSOCKOPT 111 on GIVESOCKET 113 on INET6 IS SRCADDR 115 on INITAPI 118 on IOCTL 127 on LISTEN 129 on READ 134 on READV 136 on RECV 138 on RECVFROM 141 on RECVMSG 145 on SELECT 149 on SELECTEX 153 on SEND 155 on SENDMSG 159 on SENDTO 161 on SETSOCKOPT 178 on SHUTDOWN 179 on SOCKET 181 on TAKESOCKET 183 on WRITE 185 on WRITEV 187 RETCODE parameter on macro socket interface on FCNTL 69, 76 RetnCode, Listener 47 return codes call interface 54 return codes, I/O PCB bb 43 EA 43 EB 43 EC 43 QC 43 OD 43 ZZ 43 RFC (request for comments) accessing online xxiv ROLB call 43 RRETMSK parameter on call interface, on SELECT 148 **RSM 29** RSM reason codes 34 RSMId 33 RSMLen 33 RSMRetCod 33 RSMRsnCod 33 RSMRsv 33 RSNDMSK parameter on call interface, on SELECT 148

S

S, defines socket descriptor on macro interface on FCNTL <u>68</u>, <u>70</u>, <u>71</u>, <u>89</u> S, defines socket descriptor on socket interface on ACCEPT <u>56</u> on BIND <u>58</u> on BIND2ADDRSEL <u>61</u> on CLOSE <u>63</u> on CONNECT <u>65</u> on FCNTL <u>67</u> on GETPEERNAME 92

S, defines socket descriptor on socket interface (continued) on GETSOCKNAME 94 on GETSOCKOPT 110 on GIVESOCKET 112 on IOCTL 119 on LISTEN 128 on READ 134 on READV 135 on RECV 137 on RECVFROM 140 on RECVMSG 143 on SEND 154 on SENDMSG 157 on SENDTO 160 on SETSOCKOPT 177 on SHUTDOWN 179 on WRITE 185 on WRITEV 186 sample programs call interface CBLOCK, PL/I 211 client, PL/I 203 server, PL/I 200 security exit 20 security exit reason codes 34 security exit, data passed by Listener 47 security exit, Listener 47 security exit, return codes 47 SELECT (call) 145 select mask 14 SELECTEX (call) 149 SEND (call) 153 SENDMSG (call) 155 SENDTO (call) 159 server call sequence, explicit-mode 37 server programming, logic flow 37 server, defined 29 server, explicit mode see explicit mode server 37 SETSOCKOPT (call) 161 shortcut keys 299 SHUTDOWN (call) 178 SNA 1 SNA protocols compared with SNA 3 compared with TCP/IP 3 SOCKET (call) 179 Socket interface 2 sockets compared with ports 7 introduction 5 Sockets 1 Sockets Extended API 5 SOCRECV parameter on call interface, TAKESOCKET call 182 SOCTYPE parameter on call interface, on SOCKET 180 softcopy information xxii SUBTASK parameter on call interface, INITAPI call 117 summary of changes xxvii SYNC 22 syntax diagram, how to read xix

Т

takesocket 22, 37, 38

TAKESOCKET (call) 182 TCP protocol 4 TCP/IP online information xxiv protocol specifications 279 TCP/IP for MVS, modifying data sets modifying data sets 48 TCP/IP protocols 4 TCP/IP Services 16 **TCPIP statement 46** Technotes xxii **TELNET 1** TERMAPI (call) 183 TIM 22, 38 TIMDataType 38 TIMEOUT parameter on call interface, on SELECT 148 TIMEOUT parameter on call socket interface on SELECTEX 152 TIMId 38 TIMLen 38 TIMListTaskID 37 TIMLstAddrSpc 37, 38 TIMLstTaskID 38 TIMRsv 38 TIMSktDesc 37, 38 TIMSrvAddrSpc 37, 38 TIMSrvTaskID 37, 38 TIMTCPAddrSpc 37, 38 TN32701 trademark information 306 TRANCODE 19, 20 Transaction code 19 transaction name, IMS 47 transaction not defined 34 transaction request message 20 **TRANSACTION statement 47** transaction unavailable 34 transaction verification 47 Transaction-initiation message 38 transaction-request message 33 Transaction-request message 29 TransNam 47 TRM 20, 29, 33 TRM bad format 34 TRMId 33 TRMlen 33 TRMRsv 33 TRMTrnCod 33 TRMUsrDat 33

U

UDP protocol <u>4</u> updates, database commit <u>22</u> use of HOSTENT structure interpreter, EZACIC08 <u>192</u> Userdata <u>47</u> utility programs EZACIC04 <u>188</u> EZACIC05 <u>189</u> EZACIC06 <u>190</u> EZACIC08 <u>192</u> EZACIC14 <u>197</u> EZACIC15 <u>198</u>

V

verification, transaction <u>47</u> VTAM <u>1</u> VTAM, online information <u>xxiv</u>

W

WRETMSK parameter on call interface, on SELECT <u>148</u> WRITE (call) <u>184</u> write() <u>22</u>, <u>25</u> WRITEV (call) <u>185</u> WSNDMSK parameter on call interface, on SELECT <u>148</u>

Ζ

z/OS Basic Skills Information Center xxii z/OS, documentation library listing 307 ZZ status code 42

Communicating your comments to IBM

Important: If your comment regards a technical question or problem, see instead <u>"If you have a technical</u> problem" on page 317.

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

Feedback on z/OS function

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

Feedback on IBM Documentation function

If your comment or question is about the IBMDocumentation functionality, for example search capabilities or how to arrange the browser view, send a detailed email to IBM Documentation Support at ibmdocs@us.ibm.com.

Feedback on the z/OS product documentation and content

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

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

- · Your name, company/university/institution name, and email address
- The title and order name of the document, and the version of z/OS Communications Server
- · The section title of the specific information to which your comment relates
- The text of your comment.

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

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

If you have a technical problem

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

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

z/OS Communications Server: z/OS V2R5.0 Communications Server: IP IMS Sockets Guide



Product Number: 5650-ZOS

SC27-3653-50

