Note:
Before using this information and the product it supports, be sure to read the general information under “Notices” on page 355.

Sixth Edition (September 2009)
This edition applies to Version 1 Release 11 of z/OS (5694-A01) and to all subsequent releases and modifications until otherwise indicated in new editions.

IBM welcomes your comments. You may send your comments to the following address.
International Business Machines Corporation
Attn: z/OS Communications Server Information Development
Department AKCA, Building 501
P.O. Box 12195, 3039 Cornwallis Road
Research Triangle Park, North Carolina 27709-2195

You can send us comments electronically by using one of the following methods:

Fax (USA and Canada):
1+919-254-1258
Send the fax to “Attn: z/OS Communications Server Information Development”

Internet e-mail:
comsvrcf@us.ibm.com

World Wide Web:

If you would like a reply, be sure to include your name, address, telephone number, or FAX number. Make sure to include the following in your comment or note:

• Title and order number of this document
• Page number or topic related to your comment

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

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
Chapter 3. Principles of operation of the Listener and the Assist module
Overview of the Listener and the Assist module
The role of the IMS Listener
The role of the IMS Assist module
Client/server logic flow
How the connection is established
How the server exchanges data with the client
How the IMS Listener manages multiple connection requests
Use of the IMS message queue
Call sequence for the IMS Listener
Application design considerations
Restrictions for operation of the Listener and the Assist module

Chapter 4. How to write an IMS TCP/IP client program
General client program logic flow
Explicit-mode client program logic flow
Explicit-mode client call sequence
Explicit-mode application data
Implicit-mode client logic flow
Implicit-mode client call sequence
Implicit-mode application data stream
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/1 coding

Chapter 5. How to write an IMS TCP/IP server program
General server program logic flow
Explicit-mode server program logic flow
Explicit-mode call sequence
Explicit-mode application data
Transaction-initiation message segment
Program design considerations
I/O PCB explicit-mode server
Explicit-mode server PL/1 programming considerations
Implicit-mode server program logic flow
Implicit-mode server call sequence
Implicit-mode application data
Programming to the Assist module interface
Implicit-mode server PL/1 programming considerations
Implicit-mode server C language programming considerations
I/O PCB implicit-mode server

Chapter 6. How to customize and operate the IMS Listener
How to start the IMS Listener
How to stop the IMS Listener
The IMS Listener configuration file
TCPIP statement
LISTENER statement
TRANSACTION statement
The IMS Listener security exit
TCP/IP services definitions
The hlq.PROFILE.TCPIP data set
The hlq.TCPIP.DATA data set
Figures

1. The use of TCP/IP with IMS .................................. 5
2. TCP/IP protocols when compared to the OSI Model and SNA ......................... 7
3. A typical client/server session .................................. 11
4. An iterative server ............................................. 12
5. A concurrent server ............................................. 12
6. The SELECT call .............................................. 17
7. How user applications access TCP/IP networks with IMS TCP/IP ................. 22
8. IMS TCP/IP message flow for transaction initiation .................................. 25
9. IMS TCP/IP message flow for explicit-mode input/output ......................... 27
10. IMS TCP/IP message flow for implicit mode input/output .................... 29
11. JCL: Sample run Listener procedure .................................. 51
12. Definition of the TCP/IP profile .................................. 56
13. The TCPIPJOBNAME Parameter in the DATA data set .......................... 56
14. Storage definition statement examples .................................. 60
15. ACCEPT call instructions example .................................. 62
16. BIND call instruction example .................................. 65
17. CLOSE call instruction example .................................. 67
18. CONNECT call instruction example .................................. 69
19. FCNTL call instruction example .................................. 71
20. FREEADDRINFO call instruction example .................................. 73
21. GETADDRINFO call instruction example .................................. 75
22. GETCLIENTID call instruction example .................................. 83
23. GETHOSTBYADDR call instruction example .................................. 84
24. HOSTENT structure that is returned by the GETHOSTBYADDR call ................. 85
25. GETHOSTBYNAME call instruction example .................................. 87
26. HOSTENT structure returned by the GETHOSTBYNAME call ................. 88
27. GETHOSTID call instruction example .................................. 89
28. GETHOSTNAME call instruction example .................................. 90
29. GETIBMOPT call instruction example .................................. 91
30. Example of name field ............................................. 93
31. GETNAMEINFO call instruction example .................................. 94
32. GETPEERNAME call instruction example .................................. 98
33. GETSOCKNAME call instruction example .................................. 100
34. GETSOCKOPT call instruction example .................................. 102
35. GIVESOCKET call instruction example .................................. 116
36. INITAPI call instruction example .................................. 118
37. IOCTL call instruction example .................................. 120
38. COBOL language example for SIOCGHOMEIF6 .................................. 122
39. COBOL language example for SIOCGIFNAMEINDEX .......................... 124
40. COBOL II example for SIOCGIFCONF .................................. 129
41. LISTEN call instruction example .................................. 130
42. NTOP call instruction example .................................. 132
43. PTON call instruction example .................................. 134
44. READ call instruction example .................................. 136
45. READV call instruction example .................................. 137
46. RECV call instruction example .................................. 139
47. RECVFROM call instruction example .................................. 142
48. RECVMSG call instruction example .................................. 145
49. SELECT call instruction example .................................. 151
50. SELECTEX call instruction example .................................. 156
51. SEND call instruction example .................................. 159
52. SENDMSG call instruction example .................................. 161
53. SENDTO call instruction example .................................. 166
54. SETSOCKOPT call instruction example .................................. 168
55. SHUTDOWN call instruction example .................................. 183

© Copyright IBM Corp. 1994, 2009 vii
56. SOCKET call instruction example ...................................... 184
57. TAKESOCKET call instruction example .............................. 186
58. TERMAPI call instruction example .................................. 188
59. WRITE call instruction example ...................................... 189
60. WRITEV call instruction example .................................... 190
61. EZACIC04 EBCDIC-to-ASCII table .................................. 192
62. EZACIC04 call instruction example ................................. 192
63. EZACIC05 ASCII-to-EBCDIC table .................................. 194
64. EZACIC05 call instruction example ................................. 194
65. EZACIC06 call instruction example ................................. 196
66. EZACIC08 call instruction example ................................. 199
67. EZACIC09 call instruction example ................................. 202
68. EZACIC14 EBCDIC-to-ASCII table .................................. 206
69. EZACIC14 call instruction example ................................. 206
70. EZACIC15 ASCII-to-EBCDIC table .................................. 208
71. EZACIC15 call instruction example ................................. 208
72. EZASOKPS PL/1 sample server program for IPv4 .................. 211
73. EZASOKPC PL/1 sample client program for IPv4 .................. 215
74. EZASO6PS PL/1 sample server program for IPv6 .................. 219
75. EZASO6PC PL/1 sample client program for IPv6 .................. 225
76. CBLOCK PL/1 common variables .................................... 229
77. EZACOBOL COBOL common variables ............................. 241
78. EZASO6CS COBOL call interface sample IPv6 server program ................ 249
79. EZASO6CC COBOL call interface sample IPv6 client program ................ 262
80. Cross zone Lnk IMS application interface ......................... 273
81. Sample C client to drive IMS Listener ............................. 276
82. Sample assembler IMS server ....................................... 279
83. Sample C client to drive IMS Listener ............................. 286
84. Sample assembler IMS server ....................................... 291
85. Sample of IMS program as a client ................................ 295
86. Sample of IMS program as a server ................................ 305
## Tables

1. First fullword passed in a bit string in select ........................................... 18
2. Second fullword passed in a bit string in select ........................................... 18
3. OPTNAME options for GETSOCKOPT and SETSOCKOPT ................................. 103
4. IOCTL call arguments .................................................................................... 127
5. OPTNAME options for GETSOCKOPT and SETSOCKOPT ................................. 169
6. Sockets ERRNOs ......................................................................................... 317
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 Chapter 3, “Principles of operation of the Listener and the Assist module,” on page 23.
- 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 Chapter 4, “How to write an IMS TCP/IP client program,” on page 35 and Chapter 7, “CALL instruction application programming interface,” on page 57.
- 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 57.

- 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 51.

**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.
- "Appendices" provides additional information for this document.
- “Notices” contains notices and trademarks used in this information.
- "Bibliography" 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.

**Determining whether a publication is current**

As needed, IBM updates its publications with new and changed information. For a given publication, updates to the hardcopy and associated BookManager softcopy are usually available at the same time. Sometimes, however, the updates to hardcopy and softcopy are available at different times. The following information describes how to determine if you are looking at the most current copy of a publication:

- At the end of a publication’s order number there is a dash followed by two digits, often referred to as the dash level. A publication with a higher dash level is more current than one with a lower dash level. For example, in the publication order number GC28-1747-07, the dash level 07 means that the publication is more current than previous levels, such as 05 or 04.
- If a hardcopy publication and a softcopy publication have the same dash level, it is possible that the softcopy publication is more current than the hardcopy publication. Check the dates shown in the Summary of Changes. The softcopy publication might have a more recently dated Summary of Changes than the hardcopy publication.
- To compare softcopy publications, you can check the last two characters of the publication’s file name (also called the book name). The higher the number, the more recent the publication. Also, next to the publication titles in the CD-ROM booklet and the readme files, there is an asterisk (*) that indicates whether a publication is new or changed.
How to contact IBM service

For immediate assistance, visit this Web site: http://www.software.ibm.com/network/commserver/support/

Most problems can be resolved at this Web site, where you can submit questions and problem reports electronically, as well as 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 “Communicating Your Comments to IBM” on page 375.

Conventions and terminology that are used in this document

Commands in this book 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 of the exit routines described in this document are installation-wide exit routines. You will see the installation-wide exit routines also called installation-wide exits, exit routines, and exits throughout this document.

The TPF logon manager, although shipped with VTAM®, is an application program. Therefore, the logon manager is documented separately from VTAM.

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

For definitions of the terms and abbreviations that are used in this document, you can view the latest IBM terminology at the IBM Terminology Web site.

Clarification of notes

Information traditionally qualified as Notes is further qualified as follows:

Note Supplemental detail
Tip Offers shortcuts or alternative ways of performing an action; a hint
Guideline Customary way to perform a procedure
Rule Something you must do; limitations on your actions
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:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Marks the beginning of the command syntax.</td>
</tr>
<tr>
<td></td>
<td>Indicates that the command syntax is continued.</td>
</tr>
<tr>
<td>Marks the beginning and end of a fragment or part of the command syntax.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Marks the end of the command syntax.</td>
</tr>
</tbody>
</table>

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 USER command is a keyword. The required variable parameter is `user_id`, and the optional variable parameter is `password`. Replace the variable parameters with your own values.

```
USER user_id
 password
```

**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.

```
OPERAND
```

**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).

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).

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.

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:

```plaintext
1ST_OPERAND , 2ND_OPERAND , 3RD_OPERAND
```
The following table lists documents that might be helpful to readers.

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNA Formats</td>
<td>GA27-3136</td>
</tr>
<tr>
<td>TCP/IP Tutorial and Technical Overview</td>
<td>GG24-3376</td>
</tr>
<tr>
<td>Understanding LDAP</td>
<td>SG24-4986</td>
</tr>
<tr>
<td>z/OS Cryptographic Services System SSL Programming</td>
<td>SC24-5901</td>
</tr>
<tr>
<td>z/OS Integrated Security Services LDAP Server Administration and Use</td>
<td>SC24-5923</td>
</tr>
<tr>
<td>z/OS JES2 Initialization and Tuning Guide</td>
<td>SA22-7532</td>
</tr>
<tr>
<td>z/OS Problem Management</td>
<td>G325-2564</td>
</tr>
<tr>
<td>z/OS MVS Diagnosis: Reference</td>
<td>GA22-7588</td>
</tr>
<tr>
<td>z/OS MVS Diagnosis: Tools and Service Aids</td>
<td>GA22-7589</td>
</tr>
<tr>
<td>z/OS MVS Using the Subsystem Interface</td>
<td>SA22-7642</td>
</tr>
<tr>
<td>z/OS Program Directory</td>
<td>GI10-0670</td>
</tr>
<tr>
<td>z/OS UNIX System Services Command Reference</td>
<td>SA22-7802</td>
</tr>
<tr>
<td>z/OS UNIX System Services Planning</td>
<td>GA22-7800</td>
</tr>
<tr>
<td>z/OS UNIX System Services Programming: Assembler Callable Services Reference</td>
<td>SA22-7803</td>
</tr>
<tr>
<td>z/OS UNIX System Services User’s Guide</td>
<td>SA22-7801</td>
</tr>
<tr>
<td>z/OS XL C/C++ Run-Time Library Reference</td>
<td>SA22-7821</td>
</tr>
<tr>
<td>System z10, System z9 and zSeries OSA-Express Customer’s Guide and Reference</td>
<td>SA22-7935</td>
</tr>
</tbody>
</table>

**Redbooks**

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

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communications Server for z/OS V1R10 TCP/IP Implementation, Volume 1: Base Functions, Connectivity, and Routing</td>
<td>SG24-7696</td>
</tr>
<tr>
<td>Communications Server for z/OS V1R10 TCP/IP Implementation, Volume 2: Standard Applications</td>
<td>SG24-7697</td>
</tr>
<tr>
<td>Communications Server for z/OS V1R10 TCP/IP Implementation, Volume 3: High Availability, Scalability, and Performance</td>
<td>SG24-7698</td>
</tr>
<tr>
<td>Communications Server for z/OS V1R10 TCP/IP Implementation, Volume 4: Security and Policy-Based Networking</td>
<td>SG24-7699</td>
</tr>
<tr>
<td>IBM Communication Controller Migration Guide</td>
<td>SG24-6298</td>
</tr>
<tr>
<td>IP Network Design Guide</td>
<td>SG24-2580</td>
</tr>
<tr>
<td>Managing OS/390® TCP/IP with SNMP</td>
<td>SG24-5866</td>
</tr>
<tr>
<td>Title</td>
<td>Number</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Migrating Subarea Networks to an IP Infrastructure Using Enterprise Extender</td>
<td>SG24-5957</td>
</tr>
<tr>
<td>SecureWay™ Communications Server for OS/390 V2R8 TCP/IP: Guide to Enhancements</td>
<td>SG24-5631</td>
</tr>
<tr>
<td>SNA and TCP/IP Integration</td>
<td>SG24-5291</td>
</tr>
<tr>
<td>TCP/IP in a Sysplex</td>
<td>SG24-5235</td>
</tr>
<tr>
<td>TCP/IP Tutorial and Technical Overview</td>
<td>GG24-3376</td>
</tr>
<tr>
<td>Threadsafe Considerations for CICS</td>
<td>SG24-6351</td>
</tr>
</tbody>
</table>

**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


**z/OS Internet Library**

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


**IBM Communications Server product**

The primary home page for information about z/OS 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


**IBM Communications Server performance information**

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


**IBM Systems Center publications**

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


**IBM Systems Center flashes**

Search the Technical Sales Library for Techdocs (including Flashes, presentations, Technotes, FAQs, white papers, Customer Support Plans, and Skills Transfer information)


**RFCs**
Search for and view Request for Comments documents in this section of the Internet Engineering Task Force Web site, 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 Web site

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

Information about Web addresses can also be found in information APAR III1334.

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

DNS Web sites

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

USENET news groups
comp.protocols.dns.bind

BIND mailing lists

http://www.isc.org/ml-archives/

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 system 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 Web site, which is available to all users (no login required):

http://publib.boulder.ibm.com/infocenter/zosinfctr/v1r7/index.jsp
How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this document or any other z/OS Communications Server documentation, do one of the following:

- Go to the z/OS contact page at [http://www.ibm.com/systems/z/os/zos/webqs.html](http://www.ibm.com/systems/z/os/zos/webqs.html). You can enter and submit your comments in the form provided at this Web site.

- Send your comments by e-mail to comsvrcf@us.ibm.com. Be sure to include the name of the document, the part number of the document, the version of z/OS Communications Server, and, if applicable, the specific location of the text that you are commenting on (for example, a section number, a page number or a table number).
Summary of changes

Summary of changes
for SC31-8830-05
z/OS Version 1 Release 11

This material contains information previously presented in SC31-8830-04, which supports z/OS V1R10.

New information
• New API to obtain IPv4 network interface MTU, see "IOCTL" on page 119.
• AT-TLS enhancements, see Appendix A, “Return codes,” on page 317.

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

You might notice changes in the style and structure—for example, headings that use uppercase for the first letter of initial words only, and procedures that have a different look and format. The changes are ongoing improvements to the consistency and retrievability of information in our information.

Summary of changes
for SC31-8830-04
z/OS Version 1 Release 10

This material contains information previously presented in SC31-8830-03, which supports z/OS V1R9.

New information
• Socket API timeout support, see Chapter 7, “CALL instruction application
programming interface,” on page 57.

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

You might notice changes in the style and structure—for example, headings that use uppercase for the first letter of initial words only, and procedures that have a different look and format. The changes are ongoing improvements to the consistency and retrievability of information in our information.

Summary of changes
for SC31-8830-03
z/OS Version 1 Release 9

This material contains information previously presented in SC31-8830-02, which supports z/OS V1R7.

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.

**New information**

- Enable application identifier in MI, SMF, and Netstat, see Chapter 7, “CALL instruction application programming interface,” on page 57.
- New sample for IMS ASCII/EBCDIC translation, see Chapter 7, “CALL instruction application programming interface,” on page 57.
- IPv6 scoped address architecture API, see Chapter 7, “CALL instruction application programming interface,” on page 57.
- MLDv2 and IGMPv3 support, see Chapter 7, “CALL instruction application programming interface,” on page 57.

**Deleted information**

- The APPC Application Suite is removed from the z/OS V1R9 Communications Server product and therefore information describing APPC Application Suite support has been deleted.

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

You might notice changes in the style and structure—for example, headings that use uppercase for the first letter of initial words only, and procedures that have a different look and format. The changes are ongoing improvements to the consistency and retrievability of information in our information.
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, it allows client/server applications to be developed 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. Once control of the connection is passed, data transfer between the server and the remote client is performed 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 of 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 via 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 need to use 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. (Refer to the z/OS Communications Server: IP User's Guide and Commands for more information).

- When a TCP/IP host needs to support a client/server application, it should use the IMS TCP/IP feature of TCP/IP Services. This feature is specifically designed to support 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* refers to 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* refers to 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 is capable of maintaining 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 the z/OS 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. Its purpose is to allow the use of conventional IMS calls for TCP/IP communication between client and server. In use, 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, as well as 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 assembler 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

The IMS TCP/IP feature allows remote users to access IMS client/server applications over TCP/IP internets. It is a feature of TCP/IP Services. Figure 1 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.

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, assembler language, or C.

Using IMS with SNA or TCP/IP

IMS is an online transaction processing system. This means that application programs using 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 7 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 shows the SNA layers, which correspond very closely to the OSI layers.
The protocols implemented by TCP/IP Services and used by IMS TCP/IP, are highlighted in Figure 2:

**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* assembler 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

---

**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.

**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.
Communications Server: 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, Assembler Language)**. The Sockets Extended API (Sockets Extended) is for those who want to write in COBOL, PL/I, or assembler language, or who have COBOL, PL/I, or assembler 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 assembler 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 57 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 would represent the connection to the programmer, yet shield the program (as much as possible) from the details of communication programming. A **socket is an end-point 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 discussed here for background purposes.

**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 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.

---

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

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

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 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 internet address, and an eight-byte reserved field. In COBOL, a socket address looks like this:

```
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).
```

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 IMS, FAMILY is always set to a value of 2, which specifies the AF_INET internet address family. The PORT field identifies the application port number; it must be specified in network byte order. The IP_ADDRESS field is the internet 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.

Internet addresses
An internet addresses (otherwise known as 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 internet address within an administered AF_INET domain must be unique. A common misunderstanding is that a host must have only one internet address. In fact, a single host may have several internet addresses — one for each network interface.

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 IMS as port 2000, and Telnet as port 23. In this example, a client desiring connection to 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.

---

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 IMS only supports the internet address family.
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, internet 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 11 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.
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 12, once Client A starts a transaction with the server, Client B cannot make a call until A has finished.
So, for lengthy transactions, a different sort of server is needed — the concurrent server, as shown in Figure 5. 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 Chapter 6, “How to customize and operate the IMS Listener,” on page 51.

The following is 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 Chapter 7, “CALL instruction application programming interface,” on page 57.

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; “Client TCP/IP calls” on page 15 discusses the typical call sequence for a client. This is the logical presentation sequence 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 11.

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 wishes 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 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 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, 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. The ACCEPT call temporarily blocks further progress.

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.

---

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.
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 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 “GIVESOCKET and TAKESOCKET calls” on page 19 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 “Client Read/Write calls — the conversation” on page 16 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 only has to support 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 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

Once 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.

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 and RECV 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 signalling 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 17.
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 only waits 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 xSNDMASK sets) 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 only test 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
\[ \text{INT(highest socket descriptor / 32)} + 1 \]

Look at the first fullword you pass in a bit string in Table 1

<table>
<thead>
<tr>
<th>Socket descriptor numbers represented by byte</th>
<th>Bit 0</th>
<th>Bit 1</th>
<th>Bit 2</th>
<th>Bit 3</th>
<th>Bit 4</th>
<th>Bit 5</th>
<th>Bit 6</th>
<th>Bit 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte 0</td>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
</tr>
<tr>
<td>Byte 1</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
</tr>
<tr>
<td>Byte 2</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>Byte 3</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

In these examples, we use standard assembler numbering notation; the left-most
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'00000020'). 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'00000030').

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'0000000200000001'.

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

<table>
<thead>
<tr>
<th>Socket descriptor numbers represented by byte</th>
<th>Bit 0</th>
<th>Bit 1</th>
<th>Bit 2</th>
<th>Bit 3</th>
<th>Bit 4</th>
<th>Bit 5</th>
<th>Bit 6</th>
<th>Bit 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte 4</td>
<td>63</td>
<td>62</td>
<td>61</td>
<td>60</td>
<td>59</td>
<td>58</td>
<td>57</td>
<td>56</td>
</tr>
</tbody>
</table>
Table 2. Second fullword passed in a bit string in select (continued)

<table>
<thead>
<tr>
<th>Socket descriptor numbers represented by byte</th>
<th>Bit 0</th>
<th>Bit 1</th>
<th>Bit 2</th>
<th>Bit 3</th>
<th>Bit 4</th>
<th>Bit 5</th>
<th>Bit 6</th>
<th>Bit 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte 5</td>
<td>55</td>
<td>54</td>
<td>53</td>
<td>52</td>
<td>51</td>
<td>50</td>
<td>49</td>
<td>48</td>
</tr>
<tr>
<td>Byte 6</td>
<td>47</td>
<td>46</td>
<td>45</td>
<td>44</td>
<td>43</td>
<td>42</td>
<td>41</td>
<td>40</td>
</tr>
<tr>
<td>Byte 7</td>
<td>39</td>
<td>38</td>
<td>37</td>
<td>36</td>
<td>35</td>
<td>34</td>
<td>33</td>
<td>32</td>
</tr>
</tbody>
</table>

If you develop your program in COBOL or PL/I, you may find that the EZACIC06 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
- The name of the task to which it wants to give 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.
If the server does not know the name of the subtask that will receive the socket, it blanks out the name of the subtask. The first subtask calling TAKESOCKET with the server’s unique name receives 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.

---

6. This is the case in IMS because the Listener has no way of knowing which Message Processing Region will inherit the socket.
An example of a concurrent server is the IMS Listener. It is described in Chapter 6, “How to customize and operate the IMS Listener,” on page 51. Figure 5 on page 12 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 22 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 22)

The socket API is available both in the C language and in COBOL, PL/I, or assembler language. It includes the following socket calls:

**Basic calls:**
- socket, bind, connect, listen, accept, shutdown, close

**Read/write calls:**
- send, sendto, recvfrom, read, write

**Advanced calls:**
- gethostname, gethostbyaddr, gethostbyname, getpeermname, getsockname, getsockopt,
  setsockopt, fcntl, ioctl, select

**IBM-specific calls:**
- initapi, getclientid, givesocket, takesocket
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** (3) in Figure 7

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** (4) in Figure 7

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 SELECT 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 35 and Chapter 5, “How to write an IMS TCP/IP server program,” on page 43.

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. 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 “When the client is an IMS MPP” on page 33 and the sample program on “Sample program - IMS MPP client” on page 294.

The role of the IMS Listener

Since 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.

7. Shipped with the TCP/IP Services base product.
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.
- 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. Since 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 explicit- and 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.
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

---

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

1. Connection Request
2. listen()
3. accept()
4. read TRM
5. verify transaction
6. ISRT TIM
7. read() client data
8. givesocket()
9. SYNC
9* implicit-mode only

---

Chapter 3. Principles of operation of the Listener and the Assist module
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 executed 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 prior to allowing the transaction to be scheduled.
   - The Listener issues an IMS CHNG call to a modifiable alternate PCB, specifying the TRANCODE of the desired 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 that 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.

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. 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 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.
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
   Once 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
   Once 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 2 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 "LISTENER statement" on page 53.

2. The active sockets queue contains the sockets that are held by the Listener while they wait for assignment to a server program. Once 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
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 only use 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 only used 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 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

It is expected that, in most cases, IMS server applications will be 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 S0Cx 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 Chapter 5, “How to write an IMS TCP/IP server program,” on page 43.

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 “Request-status message segment” on page 40 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:

<table>
<thead>
<tr>
<th>Call</th>
<th>Explanation of Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>INITAPI</td>
<td>Open the interface. (Only required for client programs that use MVS TCP/IP socket calls).</td>
</tr>
<tr>
<td>SOCKET</td>
<td>Obtain a socket descriptor.</td>
</tr>
<tr>
<td>CONNECT</td>
<td>Request connection to the IMS Listener port.</td>
</tr>
<tr>
<td>WRITE</td>
<td>Send a transaction-request message (TRM)</td>
</tr>
<tr>
<td>READ</td>
<td>Test for successful transaction initiation <strong>10</strong></td>
</tr>
<tr>
<td>WRITE/READ</td>
<td>Explicit-mode transactions can issue any number of READ or WRITE socket call sequences.</td>
</tr>
<tr>
<td>READ</td>
<td>Ensure that the server ended normally and that the database changes are committed.</td>
</tr>
<tr>
<td>CLOSE</td>
<td>Terminate the connection and release socket resources.</td>
</tr>
</tbody>
</table>

**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.

---
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.
**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.

The client requests an IMS MPP by sending the transaction-request message (TRM). (See “Transaction-request message segment (client to Listener)” on page 39 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 “Request-status message segment” on page 40 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 may have completed normally and database changes committed, or the transaction may 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:

<table>
<thead>
<tr>
<th>Call</th>
<th>Explanation of Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>INITAPI</td>
<td>Open the interface. (Only required for client programs that use MVS TCP/IP Sockets calls).</td>
</tr>
<tr>
<td>SOCKET</td>
<td>Obtain a socket descriptor.</td>
</tr>
<tr>
<td>CONNECT</td>
<td>Request connection to the IMS Listener port.</td>
</tr>
<tr>
<td>WRITE</td>
<td>Send a transaction-request message (TRM).</td>
</tr>
<tr>
<td>WRITE</td>
<td>Send server input data formatted as IMS segments</td>
</tr>
<tr>
<td>READ</td>
<td>Receive response.</td>
</tr>
</tbody>
</table>
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** is the length in bytes of this data segment in binary.

**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:

<table>
<thead>
<tr>
<th>Field</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>H</td>
<td>Length of the data segment (including this field)</td>
</tr>
<tr>
<td>Reserved (zz)</td>
<td>CL2</td>
<td>Reserved field</td>
</tr>
<tr>
<td>Data</td>
<td>CLnn</td>
<td>Client-supplied data</td>
</tr>
</tbody>
</table>

---

11. This example uses Assembler language notation. See Chapter 7, “CALL instruction application programming interface,” on page 57 for COBOL and PL/I equivalents.
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*-mode in the TRANSACTION configuration statement, the Listener translates the ASCII application data into EBCDIC. 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 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 41).

- 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 intercepts 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:

<table>
<thead>
<tr>
<th>Field</th>
<th>Format</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRMLen</td>
<td>H</td>
<td>Length of the segment (in binary) including this field. This field is sent in network byte order.</td>
</tr>
<tr>
<td>TRMRsv</td>
<td>CL2</td>
<td>Reserved</td>
</tr>
<tr>
<td>TRMId</td>
<td>CL8</td>
<td>Identifying string. Always <em>TRNREQ</em>. 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.</td>
</tr>
<tr>
<td>TRMTrnCod</td>
<td>CL8</td>
<td>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.</td>
</tr>
<tr>
<td>TRMUsrDat</td>
<td>XLn</td>
<td>This variable-length field contains client data that is passed directly to the security exit without translation.</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Field</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSMLen</td>
<td>H</td>
<td>Length of message (in binary), including this field.</td>
</tr>
<tr>
<td>RSMRsv</td>
<td>CL2</td>
<td>Reserved</td>
</tr>
<tr>
<td>RSMId</td>
<td>CL8</td>
<td>Identifying string. Always <em>REQSTS</em>. This field is translated to ASCII if the Listener has determined that the client is transmitting in ASCII.</td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>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).</td>
</tr>
<tr>
<td>RSMRsnCod</td>
<td>F</td>
<td>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.</td>
</tr>
</tbody>
</table>
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.
8  The application data buffer for the Client-to-Server Data Stream contains an invalid value for the data segment length.
9  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:

<table>
<thead>
<tr>
<th>Field</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>H</td>
<td>Length of the data segment (in binary) including this field</td>
</tr>
<tr>
<td>CSMRsv</td>
<td>H</td>
<td>Reserved field; must be set to zero</td>
</tr>
<tr>
<td>CSMId</td>
<td>CL8</td>
<td><em>CSMOKY</em> This field is translated to ASCII if the client is transmitting in ASCII.</td>
</tr>
</tbody>
</table>

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 implicit-mode 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.
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 Chapter 4, “How to write an IMS TCP/IP client program,” on page 35 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 Chapter 7, “CALL instruction application programming interface,” on page 57 for the call syntax of the socket calls.

<table>
<thead>
<tr>
<th>Server call</th>
<th>Explanation of Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL CBLTDLI (GU) I/O PCB</td>
<td>Obtain transaction-initiation message (TIM) from IMS message queue.</td>
</tr>
</tbody>
</table>
### INITAPI

Initialize the connection with TCP/IP.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADSNAME</td>
<td>Server address space (TIMSrvAddrSp from the TIM)</td>
</tr>
<tr>
<td>SUBTASK</td>
<td>Server task ID (TIMSrvTaskID from the TIM)</td>
</tr>
<tr>
<td>TCPNAME</td>
<td>TCP address space (TIMTCPAddrSpc from the TIM)</td>
</tr>
</tbody>
</table>

### TAKE_SOCKET

Accept the socket from the Listener.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLIENT.name</td>
<td>Listener address space (TIMLstAddrSp from the TIM)</td>
</tr>
<tr>
<td>CLIENT.task</td>
<td>Listener task ID (TIMLstTaskID from the TIM)</td>
</tr>
<tr>
<td>SOCRECV</td>
<td>Socket descriptor (TIMSktDesc from the TIM)</td>
</tr>
</tbody>
</table>

Note that the TAKE_SOCKET 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.

### TERMIN

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 57 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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Format</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMLen 12</td>
<td>H</td>
<td>The length of the transaction-initiation message segment (in binary), including the length of this field. (X'0038').</td>
</tr>
<tr>
<td>TIMRsv</td>
<td>H</td>
<td>Reserved field set to zero. (X'0000').</td>
</tr>
<tr>
<td>TIMId</td>
<td>CL8</td>
<td>Identifies the message as having been created by the IMS Listener. Always contains the characters &quot;LISTNR&quot;.</td>
</tr>
<tr>
<td>TMLstAddrSpc</td>
<td>CL8</td>
<td>Listener address space name. Used in server TAKESOCKET.</td>
</tr>
<tr>
<td>TMLstTaskId</td>
<td>CL8</td>
<td>Listener task ID. Used in server TAKESOCKET.</td>
</tr>
<tr>
<td>TMSrvAddrSpc</td>
<td>CL8</td>
<td>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.</td>
</tr>
<tr>
<td>TMSrvTaskID</td>
<td>CL8</td>
<td>Server task ID. Used in server INITAPI.</td>
</tr>
<tr>
<td>TIMSktdesc</td>
<td>H</td>
<td>Contains the descriptor of the socket given by Listener. Used in server TAKESOCKET.</td>
</tr>
<tr>
<td>TIMTCPAddrSpc</td>
<td>CL8</td>
<td>The TCP/IP address space name of TCP/IP. Used in INITAPI.</td>
</tr>
<tr>
<td>TIMDataType</td>
<td>H</td>
<td>Indicates the data type of the client messages: ASCII(0) or EBCDIC(1).</td>
</tr>
</tbody>
</table>

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

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" (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. Since 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 "Explicit-mode server program logic flow" on page 43). However, the IMS Assist module issues the TCP/IP calls on behalf of the server program; consequently, the implicit-mode application programmer need only issue 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.  

**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:

<table>
<thead>
<tr>
<th>Field</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>H</td>
<td>Length of the data segment (in binary) including this field.</td>
</tr>
<tr>
<td>Reserved</td>
<td>H</td>
<td>Reserved field; must be set to zero.</td>
</tr>
<tr>
<td>Data</td>
<td>CLn</td>
<td>Application data.</td>
</tr>
</tbody>
</table>

#### 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.)

---

13. For database I/O, you can use either CBLTDLI or CBLADLI. The Assist module simply converts database calls from CBLADLI to CBLTDLI.

14. Database PCB and I/O PCB calls can be intermixed.
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 41).

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

```c
#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:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LTERM NAME</td>
<td>Blanks (8 bytes)</td>
</tr>
<tr>
<td>RESERVED</td>
<td>See &quot;Status codes&quot; (2 bytes)</td>
</tr>
<tr>
<td>STATUS CODE</td>
<td>See &quot;Status codes&quot; (2 bytes)</td>
</tr>
<tr>
<td>DATE/TIME</td>
<td>Undefined (8 bytes)</td>
</tr>
<tr>
<td>INPUT MSG. SEQ. #</td>
<td>Undefined (4 bytes)</td>
</tr>
<tr>
<td>MESSAGE OUTPUT DESC. NAME</td>
<td>Blanks (8 bytes)</td>
</tr>
<tr>
<td>USERID</td>
<td>PSBname of Listener (8 bytes)</td>
</tr>
</tbody>
</table>

**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 317).

- 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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EA</td>
<td>A call that used the AIB interface to determine the I/O PCB address failed.</td>
</tr>
<tr>
<td>EB</td>
<td>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.</td>
</tr>
<tr>
<td>EC</td>
<td>A QD status code was received in response to a GU or ROLB call when...</td>
</tr>
</tbody>
</table>
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.
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. Below is 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 52.

```
//EZAIMSJL  PROC MBR=EZAIMSLN,PSB=EZAIMSLN,IMSID=IMS,CFG=TCPIMS,SOUT=A
//*
//LISTENER EXEC PROC=IMSBATCH,MBR=&MBR.,SOUT=&SOUT.,IMSID=&IMSID.,
// PSB=&PSB.,CPUTIME=1440
//G.STEPLIB DD DSN=IMSVS31.&SYS2.RESLIB,DISP=SHR
// DD DSN=IMSVS31.&SYS2.PGMLIB,DISP=SHR
// DD DSN=TCPIP.SEZALOAD,DISP=SHR
// 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 given below.

```
----START---procname----.identifier----
```

**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 of this command and a description of the parameters is given below.

```plaintext
MODIFY procname,identifier,STOP
```

**procname**

The name of the cataloged procedure that was used to start the Listener. This is only required 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 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.

```plaintext
TCPIP ADDRSPC=name
```

**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.

\[
\begin{align*}
\text{LISTENER} & \quad \text{PORT}= \text{port} \quad \text{MAXTRANS}= \text{maxtrans} \quad \text{MAXACTSKT}= \text{maxskt} \\
\text{ADDRSPCPFX}= \text{prefix} & \quad \text{BACKLOG}= \text{10} \quad \text{BACKLOG}= \text{backlog}
\end{align*}
\]

**PORT** = *port*
Port number that the Listener binds to for connection requests. Use an integer between 0 and 65 535, inclusive.

**MAXTRANS** = *maxtrans*
The maximum number of TRANSACTION statements to be processed in the configuration file. Use an integer between 1 and 32 767, 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 32 767 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 (EZAIBMSLN) 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 (described below). 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.)

The format of the data passed to the security exit is:

<table>
<thead>
<tr>
<th>Field</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IpAddr</td>
<td>F</td>
<td>The address of a fullword containing the client’s IP address.</td>
</tr>
<tr>
<td>Port</td>
<td>H</td>
<td>The address of a halfword containing the client’s port number.</td>
</tr>
<tr>
<td>TransNam</td>
<td>CL8</td>
<td>The address of an 8-character string defining the name of the requested transaction.</td>
</tr>
<tr>
<td>DataType</td>
<td>H</td>
<td>The address of a halfword containing the data type (0 if ASCII or 1 if EBCDIC).</td>
</tr>
<tr>
<td>DataLen</td>
<td>F</td>
<td>The address of a fullword containing the length of the user data.</td>
</tr>
<tr>
<td>Userdata</td>
<td>XLn</td>
<td>The address of the user-supplied data.</td>
</tr>
<tr>
<td>RetnCode</td>
<td>F</td>
<td>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.</td>
</tr>
</tbody>
</table>
Field | Format | Description
-----|-------|------------------
ReasonCode | 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 `tcip.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 IMS socket Listener to TCP/IP on MVS in 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 56](#).

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 56](#) 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.
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 "TCPIP statement" on page 52. In the example below, TCPIPJOBNAME is set to TCPV3. The default name is TCPIP.

```
Name of Data Set: hlq.TCPIP.DATA

This data, TCPIP.DATA, is used to specify configuration information required by TCP/IP client programs.

TCPIPJOBNAME TCPIP
```

Figure 13. The TCPIPJOBNAME Parameter in the DATA data set
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 58
- “Understanding COBOL, Assembler, and PL/I call formats” on page 59
- “Converting parameter descriptions” on page 60
- “Diagnosing problems in applications using the CALL instruction API” on page 60
- “CALL instruction API error messages and return codes” on page 61
- “Code CALL instructions” on page 61
- “Using data translation programs for socket call interface” on page 191
- “Call interface sample programs” on page 210

CALL instruction API environmental restrictions and programming requirements

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRB mode</td>
<td>These APIs can only be invoked in TCB mode (task mode).</td>
</tr>
<tr>
<td>Cross-memory mode</td>
<td>These APIs can only be invoked in a non-cross-memory environment (PASN=SASN=HASN).</td>
</tr>
<tr>
<td>Functional Recovery Routine (FRR)</td>
<td>Do not invoke these APIs with an FRR set. This causes system recovery routines to be bypassed and severely damage the system.</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks should be held when issuing these calls.</td>
</tr>
<tr>
<td>INITAPI and TERMAPI socket commands</td>
<td>The INITAPI and TERMAPI socket commands must be issued under the same task.</td>
</tr>
<tr>
<td>Storage</td>
<td>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.</td>
</tr>
<tr>
<td>Nested socket API calls</td>
<td>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.</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Contains the entry point address EZASOKET</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system.</td>
</tr>
</tbody>
</table>

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 [z/OS 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 Assembler language. The format and parameters are described for each socket call.
Notes:
1. Unless your program is running in a CICS environment, reentrant code and multithread applications are not supported by this interface.
2. For a PL/I program, include the following statement before your first call instruction.
    
    DCL EZASOKET ENTRY OPTIONS(ASM,INTER) EXT;

3. If you use the CALL instruction from code that will run as a part of a CICS transaction, see the z/OS Communications Server: IP CICS Sockets Guide for additional considerations.
4. 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 is 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-justified 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**

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.

**Assembler language call format**

The following is the EZASOKET call format for assembler language programs.

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

**PL/I language call format**

The following is 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-justified and padded on the right with blanks. Set to the name of the call.

parm
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 shows examples of storage definition statements for COBOL, PL/I, and assembler 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 may be the cause of an error. For more information about the SOCKAPI trace option, see \textit{z/OS Communications Server: IP Diagnosis Guide}.

\section*{CALL instruction API error messages and return codes}

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

For information about error codes that are returned by TCP/IP, see Appendix A. Return codes on page 317.

\section*{Code CALL instructions}

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

\textbf{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 \texttt{s}, and returns its descriptor in \texttt{RETCODE}. The original sockets remain available to the calling program to accept more connection requests.
3. The address of the client is returned in \texttt{NAME} for use by subsequent server calls.

\textbf{Notes:}

1. 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.
2. 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.
3. 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.
4. 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.

The following requirements apply to this call:

| Authorization: | Supervisor state or problem state, any PSW key. |
| Dispatchable unit mode: | Task. |
Cross memory mode: \(\text{PASN} = \text{HASN}\).

Amode: 31-bit or 24-bit.

Note: See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.

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 15 shows an example of ACCEPT call instructions.

WORKING-STORAGE SECTION.
  01 SOC-FUNCTION PIC X(16) VALUE IS 'ACCEPT'.
  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.
    03 FAMILY PIC 9(4) BINARY.
    03 PORT PIC 9(4) BINARY.
    03 FLOWINFO PIC 9(8) BINARY.
    03 IP-ADDRESS.
    10 FILLER PIC 9(16) BINARY.
    10 FILLER PIC 9(16) BINARY.
    03 SCOPE-ID PIC X(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 15. ACCEPT call instructions example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 60.

Parameter values set by the application

SOC-FUNCTION
  A 16-byte character field containing ACCEPT. Left-justify 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 Internet 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 Internet 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 Appendix A. Return codes on page 317 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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 0</td>
<td>Successful call.</td>
</tr>
<tr>
<td>−1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

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 only interested in receiving 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.

**Note:** Even if an application specifies a value of 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. This has a similar effect to the application specifying an explicit IP address on the BIND CALL. For more information, see [z/OS Communications Server: IP Configuration Reference](z/OS V1R11.0 Comm Svr: IP IMS Sockets Guide).

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization:</td>
<td>Supervisor state or problem state, any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Amode:</td>
<td>31-bit or 24-bit.</td>
</tr>
<tr>
<td></td>
<td>[Note: See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57](z/OS V1R11.0 Comm Svr: IP IMS Sockets Guide)</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary address space control (ASC) mode.</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>Unlocked.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>All parameters must be addressable by the caller and in the primary address space.</td>
</tr>
</tbody>
</table>

[Figure 16 on page 65](z/OS V1R11.0 Comm Svr: IP IMS Sockets Guide) shows an example of BIND call instructions.
WORKING-STORAGE SECTION.
   01 SOC-FUNCTION PIC X(16) VALUE IS 'BIND'.
   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.
      03 FAMILY PIC 9(4) BINARY.
      03 PORT PIC 9(4) BINARY.
      03 FLOWINFO PIC 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 assembler language declarations, see "Converting
parameter descriptions" on page 60.

Parameter values set by the application

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

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

NAME
   See z/OS 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 Internet
   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 Internet 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 may be specified for any address types and scopes. For a link scope IPv6-ADDRESS, SCOPE-ID may 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 [Appendix A. Return codes on page 317](#) for information about ERRNO return codes.

**RETCODE**
A fullword binary field that returns one of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful call.</td>
</tr>
<tr>
<td>−1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

**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 182 for more information.
Notes:
1. If a stream socket is closed while input or output data is queued, the TCP connection is reset and data transmission may 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 period of time after the CLOSE call is issued. See SO-LINGER in the description of "SETSOCKOPT" on page 168.
2. 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.
3. 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.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization</td>
<td>Supervisor state or problem state, any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Amode</td>
<td>31-bit or 24-bit.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> See “Addressability mode (Amode) considerations” under &quot;CALL instruction API environmental restrictions and programming requirements&quot; on page 57.</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary address space control (ASC) mode.</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for interrupts.</td>
</tr>
<tr>
<td>Locks</td>
<td>Unlocked.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>All parameters must be addressable by the caller and in the primary address space.</td>
</tr>
</tbody>
</table>

Figure 17 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 9(8) BINARY.

PROCEDURE DIVISION.
  CALL 'EZASOKET' USING SOC-FUNCTION $ ERRNO RETCODE.
```

Figure 17. CLOSE call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 60.

**Parameter values set by the application**

**SOC-FUNCTION**

A 16-byte field containing CLOSE. Left-justify 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 [Appendix A. Return codes on page 317](#) for information about ERRNO return codes.

**RETCODE**
A fullword binary field that returns one of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful call.</td>
</tr>
<tr>
<td>−1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

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

**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.

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, but since 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 "SELECT" on page 148.
The following requirements apply to this call:

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 “Addressability mode (Amode) considerations” under "CALL instruction API environmental restrictions and programming requirements" on page 57.

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 18 shows an example of CONNECT call instructions.

```
WORKING-STORAGE SECTION.
  01 SOC-FUNCTION PIC X(16) VALUE IS 'CONNECT'.
  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.
    03 FAMILY PIC 9(4) BINARY.
    03 PORT PIC 9(4) BINARY.
    03 IP-ADDRESS PIC 9(8) BINARY.
    03 FLOWINFO PIC 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.
```

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 60.

**Parameter values set by the application**

**SOC-FUNCTION**

A 16-byte field containing CONNECT. Left-justify 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 Internet address of the server’s host machine in network byte order. For example, if the Internet 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 Internet address of the server’s host machine in network byte order. For example, if the IPv6 Internet address is 12ab:0:0:cd30:123:4567:89ab:cedf in colon hex notation, it is set to X’12AB00000000CD30123456789ABCDEF’.

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 may be specified for any address types and scopes. For a link scope IPv6-ADDRESS, SCOPE-ID may 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 Appendix A. Return codes on page 317 for information about ERRNO return codes.
RETCODE
A fullword binary field that returns one of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful call.</td>
</tr>
<tr>
<td>−1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

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 119 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 z/OS UNIX System Services Assembler Callable Services Reference for more information.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Authorization:</th>
<th>Supervisor state or problem state, any PSW key.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Amode:</td>
<td>31-bit or 24-bit.</td>
</tr>
<tr>
<td>Note:</td>
<td>See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary address space control (ASC) mode.</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>Unlocked.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>All parameters must be addressable by the caller and in the primary address space.</td>
</tr>
</tbody>
</table>

Figure 19 shows an example of FCNTL call instructions.

```
WORKING-STORAGE SECTION
  01 SOC-FUNCTION PIC X(16) VALUE IS 'FCNTL'.
  01 S PIC 9(4) BINARY.
  01 COMMAND PIC 9(8) BINARY.
  01 REQARG PIC 9(8) BINARY.
  01 ERRNO PIC 9(8) BINARY.
  01 RETCODE PIC S9(8) BINARY.

PROCEDURE DIVISION
  CALL 'EZASOKET' USING SOC-FUNCTION S COMMAND REQARG ERRNO RETCODE.
```

Figure 19. FCNTL call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 60.
Parameter values set by the application

SOC-FUNCTION
A 16-byte character field containing FCNTL. The field is left-justified 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Query the blocking mode of the socket.</td>
</tr>
<tr>
<td>4</td>
<td>Set the mode to blocking or nonblocking for the socket.</td>
</tr>
</tbody>
</table>

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 Appendix A. Return codes on page 317 for information about ERRNO return codes.

RETCODE
A fullword binary field that returns one of the following.
- 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'00000000', 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.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Authorization:</th>
<th>Supervisor state or problem state, any PSW key.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>SOC-FUNCTION</td>
<td>A 16-byte character field containing FREEADDRINFO. The field is left-justified and padded on the right with blanks.</td>
</tr>
<tr>
<td>ADDRINFO</td>
<td>Input parameter. The address of a set of address information structures returned by the GETADDRINFO RES argument.</td>
</tr>
<tr>
<td>ERRNO</td>
<td>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 317 for information about ERRNO return codes.</td>
</tr>
<tr>
<td>RETCODE</td>
<td>Output parameter. A fullword binary field that returns one of the following:</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>0</td>
<td>Successful call.</td>
</tr>
<tr>
<td>-1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

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.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Requirement Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization</td>
<td>Supervisor state or problem state, any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Amode:</td>
<td>31-bit or 24-bit.</td>
</tr>
<tr>
<td>Note:</td>
<td>See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary address space control (ASC) mode.</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>Unlocked.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>All parameters must be addressable by the caller and in the primary address space.</td>
</tr>
</tbody>
</table>

Figure 21 on page 75 shows an example of GETADDRINFO call instructions.
Figure 21. GETADDRINFO call instruction example

Parameter values set by the application

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
</table>

SOC-FUNCTION
A 16-byte character field containing GETADDRINFO. The field is left-justified 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 queried host’s IP address 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 z/OS Communications 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 may 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:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
</table>

76  z/OS V1R11.0 Comm Svr: IP IMS Sockets Guide
FLAGS

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

AI-PASSIVE (X'00000001') or a decimal value of 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 (127.0.0.0) or the default loopback address for an IPv6 address (::1).
- This flag is ignored if the NODE argument is specified.

AI-CANONNAMEOK (X'00000002') or a decimal value of 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 a decimal value of 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 a decimal value of 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 a decimal value of 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 will accept IPv4-mapped IPv6 addresses. When the AI-ALL flag is not also specified, if no IPv6
addresses are found, a query is made for IPv4 addresses. If IPv4 addresses are found, they are returned as IPv4-mapped IPv6 addresses.

- If the AF field does not have the value of AF_INET6 or the AF field contains AF_UNSPEC but IPv6 is not supported on the system, this flag is ignored.

AI-ALL (X'00000020') or a decimal value of 32.

- When 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). When the AF field has a value of AF_UNSPEC when the system supports IPv6 and AI-ALL is set, the caller accepts IPv6 addresses and either IPv4 address (if AI-V4MAPPED is not set), or IPv4-mapped IPv6 addresses (if AI-V4MAPPED is set). A query is first made for IPv6 addresses and if successful, the IPv6 addresses are returned. Another query is then made for IPv4 addresses, and any IPv4 addresses found are returned as either IPv4-mapped IPv6 addresses (if AI-V4MAPPED is also specified), or as IPv4 addresses (if AI-V4MAPPED is not specified).

- If the AF field does not have the value of AF_INET6 or does not have the value of AF_UNSPEC when the system supports IPv6, this flag is ignored.

AI-ADDRCONFIG (X'00000040') or a decimal value of 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 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.

Note: 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 a 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-ADDRCONFIG 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 are the acceptable socket types:

<table>
<thead>
<tr>
<th>Type name</th>
<th>Decimal value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOCK_STREAM</td>
<td>1</td>
<td>for stream socket</td>
</tr>
<tr>
<td>SOCK_DGRAM</td>
<td>2</td>
<td>for datagram socket</td>
</tr>
<tr>
<td>SOCK_RAW</td>
<td>3</td>
<td>for raw-protocol interface</td>
</tr>
</tbody>
</table>

Anything else will fail with return code EAI_SOCTYPE.
Note that although SOCK_RAW will be accepted, it will
only be valid 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 both SOCTYPE and PROTO are 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:
• 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 are the acceptable protocols:

<table>
<thead>
<tr>
<th>Protocol name</th>
<th>Decimal value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPPROTO_TCP</td>
<td>6</td>
<td>TCP</td>
</tr>
<tr>
<td>IPPROTO_UDP</td>
<td>17</td>
<td>user datagram</td>
</tr>
</tbody>
</table>

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:
• 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. On input, this field must be 0.

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

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

NEXT
A fullword binary field. On input, this field must be 0.
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. Do not use or reference these structures between MVS
tasks. When you are finished using the structures, explicitly free
their storage by specifying the returned pointer on a
FREEADDRINFO call; storage that is not explicitly freed is released
when the task is ended. 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
may 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
may 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
may 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. The length of the NAME socket
address structure. The value returned in this field may be
used as the arguments for the CONNECT or BIND call
with such a socket, according to the AI-PASSIVE flag.

**CANONNAME**
A fullword binary field. 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. The address of the returned socket address structure. The value returned in this field may be used as the arguments for the CONNECT or BIND 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.

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.

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

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 assembler language, this structure can be processed in a relatively straight-forward manner. If you are coding in COBOL, this structure may 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 115 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.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization:</td>
<td>Supervisor state or problem state, any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Amode:</td>
<td>31-bit or 24-bit.</td>
</tr>
</tbody>
</table>

**Note:** See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.
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 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).
  01 ERRNO PIC 9(8) BINARY.
  01 RETCODE PIC S9(8) BINARY.

PROCEDURE DIVISION.
  CALL 'EZASOKET' USING SOC-FUNCTION CLIENT ERRNO RETCODE.
```

Figure 22. GETCLIENTID call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 60.

**Parameter values set by the application**

**SOC-FUNCTION**

A 16-byte character field containing GETCLIENTID. The field is left-justified 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 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 Appendix A. Return codes on page 317 for information about ERRNO return codes.

**RETCODE**

A fullword binary field that returns one of the following:
<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful call.</td>
</tr>
<tr>
<td>-1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

**GETHOSTBYADDR**

The GETHOSTBYADDR call returns the domain name and alias name of a host whose IPv4 Internet address is specified in the call. A given TCP/IP host can have multiple alias names and multiple host IPv4 Internet addresses. The address resolution attempted depends on how the resolver is configured and if any local host tables exist. See z/OS Communications Server: IP Configuration Guide for information about configuring the resolver and how local host tables can be used.

The following requirements apply to this call:

- **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 “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.

- **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 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.
```

**Figure 23. GETHOSTBYADDR call instruction example**

For equivalent PL/1 and assembler language declarations, see “Converting parameter descriptions” on page 60.

**Parameter values set by the application**

- **SOC-FUNCTION**
  A 16-byte character field containing GETHOSTBYADDR. The field is left-justified and padded on the right with blanks.

- **HOSTADDR**
  A fullword binary field set to the Internet address (specified in network byte order) of the host whose name is being sought. See Appendix A. Return codes on page 317 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful call.</td>
</tr>
<tr>
<td>-1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

GETHOSTBYADDR returns the HOSTENT structure shown in Figure 24. The HOSTENT structure is a task’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.

Figure 24. HOSTENT structure that is returned by the GETHOSTBYADDR call
• The length of the host Internet 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 Internet 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 Internet addresses. If you are coding in PL/I or assembler language, this structure can be processed in a relatively straight-forward manner. If you are coding in COBOL, this structure may 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 “EZACIC08” on page 198.

GETHOSTBYNAME

The GETHOSTBYNAME call returns the alias name and the IPv4 Internet 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 Internet addresses.

The name resolution attempted depends on how the resolver is configured and if any local host tables exist. See z/OS Communications Server: IP Configuration Guide for information about configuring the resolver and how local host tables can be used.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Requirement Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization:</td>
<td>Supervisor state or problem state. The PSW key must match the key in which the MVS application task was attached.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Amode:</td>
<td>31-bit or 24-bit.</td>
</tr>
<tr>
<td></td>
<td>Note: See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary address space control (ASC) mode.</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>Unlocked.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>All parameters must be addressable by the caller and in the primary address space.</td>
</tr>
</tbody>
</table>

Figure 25 on page 87 shows an example of GETHOSTBYNAME call instructions.
WORKING-STORAGE SECTION.
  01 SOC-FUNCTION 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 25. GETHOSTBYNAME call instruction example

For equivalent PL/I and assembler language declarations, see "Converting
parameter descriptions" on page 60.

Parameter values set by the application

SOC-FUNCTION
  A 16-byte character field containing GETHOSTBYNAME. The field is
  left-justified 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful call.</td>
</tr>
<tr>
<td>-1</td>
<td>An error occurred.</td>
</tr>
</tbody>
</table>
GETHOSTBYNAME returns the HOSTENT structure shown in Figure 26. The HOSTENT structure is a task'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 Internet 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 Internet 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 Internet addresses. If you are coding in PL/I or assembler language, this structure can be processed in a relatively straightforward manner. If you are coding in COBOL, this structure may 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 "EZACIC08" on page 198.
GETHOSTID

The GETHOSTID call returns the 32-bit Internet address for the current host.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Authorization</th>
<th>Supervisor state or problem state, any PSW key.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatchable unit mode</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Amode</td>
<td>31-bit or 24-bit.</td>
</tr>
</tbody>
</table>

**Note:** See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.

<table>
<thead>
<tr>
<th>ASC mode</th>
<th>Primary address space control (ASC) mode.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt status</td>
<td>Enabled for interrupts.</td>
</tr>
<tr>
<td>Locks</td>
<td>Unlocked.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>All parameters must be addressable by the caller and in the primary address space.</td>
</tr>
</tbody>
</table>

Figure 27 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 27. GETHOSTID call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 60.

**Parameter values set by the application**

**SOC-FUNCTION**

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

**RETCODE**

Returns a fullword binary field containing the 32-bit Internet 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 TCPIPDATA file that was found.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Authorization</th>
<th>Supervisor state or problem state, any PSW key.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatchable unit mode</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN = HASN.</td>
</tr>
</tbody>
</table>
Amode: 31-bit or 24-bit.

Note: See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.

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 shows an example of GETHOSTNAME call instructions.

```assembler
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 28. GETHOSTNAME call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 60.

Parameter values set by the application

SOC-FUNCTION
A 16-byte character field containing GETHOSTNAME. The field is left-justified and padded on the right with blanks.

NAMELEN
A fullword binary field set to the length of the NAME field.

Parameter values returned to the application

NAME
Indicates the receiving field for the host name. TCP/IP Services allows a maximum length of 24 characters. The Internet standard is a maximum name length of 255 characters. The actual length of the NAME field is found in NAMELEN.

ERRNO
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix A. Return codes on page 317 for information about ERRNO return codes.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful call.</td>
</tr>
<tr>
<td>-1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>
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.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Authorization</th>
<th>Supervisor state or problem state, any PSW key.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatchable unit mode</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Amode</td>
<td>31-bit or 24-bit.</td>
</tr>
<tr>
<td>Note:</td>
<td>See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary address space control (ASC) mode.</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for interrupts.</td>
</tr>
<tr>
<td>Locks</td>
<td>Unlocked.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>All parameters must be addressable by the caller and in the primary address space.</td>
</tr>
</tbody>
</table>

Figure 29 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)
  01 ERRNO PIC 9(8) BINARY.
  01 RETCODE PIC S9(8) BINARY.

PROCEDURE DIVISION.

  CALL 'EZASOKET' USING SOC-FUNCTION COMMAND BUF ERRNO RETCODE.
```

*Figure 29. GETIBMOPT call instruction example*

**Parameter values set by the application**

**SOC-FUNCTION**

A 16-byte character field containing GETIBMOPT. The field is left-justified 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:

<table>
<thead>
<tr>
<th>Status field</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'8xxx'</td>
<td>Active</td>
</tr>
<tr>
<td>X'4xxx'</td>
<td>Terminating</td>
</tr>
<tr>
<td>X'2xxx'</td>
<td>Down</td>
</tr>
<tr>
<td>X'1xxx'</td>
<td>Stopped or stopping</td>
</tr>
</tbody>
</table>

Note: In the above 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:

<table>
<thead>
<tr>
<th>Version</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP/IP z/OS Communications Server V1R4</td>
<td>X'0614'</td>
</tr>
<tr>
<td>TCP/IP z/OS Communications Server V1R5</td>
<td>X'0615'</td>
</tr>
<tr>
<td>TCP/IP z/OS Communications Server V1R6</td>
<td>X'0616'</td>
</tr>
<tr>
<td>TCP/IP z/OS Communications Server V1R7</td>
<td>X'0617'</td>
</tr>
<tr>
<td>TCP/IP z/OS Communications Server V1R8</td>
<td>X'0618'</td>
</tr>
<tr>
<td>TCP/IP z/OS Communications Server V1R9</td>
<td>X'0619'</td>
</tr>
<tr>
<td>TCP/IP z/OS Communications Server V1R10</td>
<td>X'061A'</td>
</tr>
<tr>
<td>TCP/IP z/OS Communications Server V1R11</td>
<td>X'061B'</td>
</tr>
</tbody>
</table>

The name field is the PROC name, left-justified, and padded with blanks.
ERRNO
A fullword binary field. If RETCODE is negative, this contains an error number. See Appendix A. Return codes on page 317 for information about ERRNO return codes.

RETCODE
A fullword binary field with the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Call returned error. See ERRNO field.</td>
</tr>
<tr>
<td>0</td>
<td>Successful call.</td>
</tr>
</tbody>
</table>

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.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization:</td>
<td>Supervisor state or problem state, any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Amode:</td>
<td>31-bit or 24-bit.</td>
</tr>
</tbody>
</table>

**Note:** See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.  

| ASC mode:                     | Primary address space control (ASC) mode. |
| Interrupt status:            | Enabled for interrupts.                   |
Parameter values set by the application

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOC-FUNCTION</td>
<td>A 16-byte character field containing GETNAMEINFO. The field is left-justified and padded on the right with blanks.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAMILY</td>
<td>A halfword binary number specifying the IPv4 addressing family. For TCP/IP the value is a decimal 2, indicating AF_INET.</td>
</tr>
<tr>
<td>PORT</td>
<td>A halfword binary number specifying the port number.</td>
</tr>
<tr>
<td>IP-ADDRESS</td>
<td>A fullword binary number specifying the 32-bit IPv4 Internet address.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>An 8-byte reserved field. This field is required, but is not used.</td>
</tr>
</tbody>
</table>

The IPv6 socket address structure specifies the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAMILY</td>
<td>A halfword binary field specifying the IPv6 addressing family. For TCP/IP the value is a decimal 19, indicating AF_INET6.</td>
</tr>
<tr>
<td>PORT</td>
<td>A halfword binary number specifying the port number.</td>
</tr>
<tr>
<td>FLOWINFO</td>
<td>A fullword binary field specifying the traffic class and flow label. This value of this field is undefined.</td>
</tr>
<tr>
<td>IP-ADDRESS</td>
<td>A 16-byte binary field specifying the 128-bit IPv6 Internet address, in network byte order.</td>
</tr>
<tr>
<td>SCOPE-ID</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 may 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 may occur. If the host’s name cannot be located, the numeric form of the host’s 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. This 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 using the format \textit{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
\texttt{GETNAMEINFO} processing, see \textit{z/OS Communications Server: IPv6
Network and Application Design Guide}.

\textbf{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 \texttt{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. This 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:
\begin{itemize}
  \item The HOST and HOSTLEN parameters
  \item The SERVICE and SERVLEN parameters
\end{itemize}
Otherwise, an error occurs.

\textbf{SERVICE} On input, storage capable of holding the returned resolved service
name, which may 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 may 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. This 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:
\begin{itemize}
  \item The HOST and HOSTLEN parameters
  \item The SERVICE and SERVLEN parameters
\end{itemize}
Otherwise, an error occurs.

\textbf{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. \texttt{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. This 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:
\begin{itemize}
  \item The HOST and HOSTLEN parameters
The SERVICE and SERVLEN parameters
Otherwise, an error occurs.

FLAGS
An input parameter. A fullword binary field. This is an optional field. The FLAGS field must contain either a binary or decimal value, depending on the programming language used:

<table>
<thead>
<tr>
<th>Flag name</th>
<th>Binary value</th>
<th>Decimal value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'NI_NOFQDN'</td>
<td>X'00000001'</td>
<td>1</td>
<td>Return the NAME portion of the fully qualified domain name.</td>
</tr>
<tr>
<td>'NI_NUMERICHOST'</td>
<td>X'00000002'</td>
<td>2</td>
<td>Only return the numeric form of host’s address.</td>
</tr>
<tr>
<td>'NI_NAMEREQD'</td>
<td>X'00000004'</td>
<td>4</td>
<td>Return an error if the host’s name cannot be located.</td>
</tr>
<tr>
<td>'NI_NUMERICSERV'</td>
<td>X'00000008'</td>
<td>8</td>
<td>Only return the numeric form of the service address.</td>
</tr>
<tr>
<td>'NI_DGRAM'</td>
<td>X'00000010'</td>
<td>16</td>
<td>Indicates that the service is a datagram service. The default behavior is to assume that the service is a stream service.</td>
</tr>
<tr>
<td>'NI_NUMERICSCOPE'</td>
<td>X'00000020'</td>
<td>32</td>
<td>Only return the numeric form of the scope information, when applicable</td>
</tr>
</tbody>
</table>

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 317 for information about ERRNO return codes.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful call.</td>
</tr>
<tr>
<td>-1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

GETPEERNAME
The GETPEERNAME call returns the name of the remote socket to which the local socket is connected.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization:</td>
<td>Supervisor state or problem state, any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Amode:</td>
<td>31-bit or 24-bit.</td>
</tr>
<tr>
<td>Note:</td>
<td>See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary address space control (ASC) mode.</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>Unlocked.</td>
</tr>
</tbody>
</table>
Control parameters: All parameters must be addressable by the caller and in the primary address space.

Figure 32 shows an example of GETPEERNAME call instructions.

```
WORKING-STORAGE SECTION.
   01 SOC-FUNCTION PIC X(16) VALUE IS 'GETPEERNAME'.
   01 S PIC 9(4) BINARY.
   * 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.
      03 FAMILY PIC 9(4) BINARY.
      03 PORT PIC 9(4) BINARY.
      03 FLOWINFO PIC 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 32. GETPEERNAME call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 60.

Parameter values set by the application

SOC-FUNCTION
A 16-byte character field containing GETPEERNAME. The field is left-justified 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 Internet 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 Internet 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 317] for information
   about ERRNO return codes.

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

   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.

Since 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.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Authorization:</th>
<th>Supervisor state or problem state, any PSW key.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Amode:</td>
<td>31-bit or 24-bit.</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Note:</td>
<td>See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary address space control (ASC) mode.</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>Unlocked.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>All parameters must be addressable by the caller and in the primary address space.</td>
</tr>
</tbody>
</table>

Figure 33 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.
      03 FAMILY PIC 9(4) BINARY.
      03 PORT PIC 9(4) BINARY.
      03 FLOWINFO PIC 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.
```

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 60.

**Parameter values set by the application**

**SOC-FUNCTION**
A 16-byte character field containing GETSOCKNAME. The field is left-justified 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 Internet address of the local host machine.

RESERVED  
Specifies 8 bytes of binary 0s. 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 Internet 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 Appendix A. Return codes on page 317 for information about ERRNO return codes.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful call.</td>
</tr>
<tr>
<td>-1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

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 below. You must specify the option to be queried when you issue the GETSOCKOPT call.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Authorization:</th>
<th>Supervisor state or problem state, any PSW key.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN.</td>
</tr>
</tbody>
</table>

Chapter 7. CALL instruction application programming interface 101
### Amode:
31-bit or 24-bit.

**Note:** See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.

### 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** shows an example of GETSOCKOPT call instructions.

```plaintext
WORKING-STORAGE SECTION.
  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-LINGER 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' USING SOC-FUNCTION S OPTNAME OPTVAL OPTLEN ERRNO RETCODE.
```

**Figure 34. GETSOCKOPT call instruction example**

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 60.

### Parameter values set by the application

**SOC-FUNCTION**
A 16-byte character field containing GETSOCKOPT. The field is left-justified 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 *z/OS 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.

**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 Appendix A. Return codes on page 317 for information about ERRNO return codes.

**RETCODE**
A fullword binary field that returns one of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful call.</td>
</tr>
<tr>
<td>−1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

### Table 3. OPTNAME options for GETSOCKOPT and SETSOCKOPT

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IP_ADD_MEMBERSHIP</strong></td>
<td>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.</td>
<td>N/A</td>
</tr>
<tr>
<td>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.</td>
<td>See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ.</td>
<td></td>
</tr>
<tr>
<td>This is an IPv4-only socket option.</td>
<td>See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ.</td>
<td></td>
</tr>
<tr>
<td><strong>IP_ADD_SOURCE_MEMBERSHIP</strong></td>
<td>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.</td>
<td>N/A</td>
</tr>
<tr>
<td>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.</td>
<td>See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE.</td>
<td></td>
</tr>
<tr>
<td>This is an IPv4-only socket option.</td>
<td>See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ_SOURCE.</td>
<td></td>
</tr>
</tbody>
</table>
### Table 3. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IP_BLOCK_SOURCE</strong></td>
<td>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.</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ-SOURCE.</td>
<td></td>
</tr>
<tr>
<td><strong>IP_DROP_MEMBERSHIP</strong></td>
<td>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.</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ.</td>
<td></td>
</tr>
<tr>
<td><strong>IP_DROP_SOURCE_MEMBERSHIP</strong></td>
<td>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.</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ-SOURCE.</td>
<td></td>
</tr>
<tr>
<td><strong>IP_MULTICAST_IF</strong></td>
<td>A 4-byte binary field containing an IPv4 interface address.</td>
<td>A 4-byte binary field containing an IPv4 interface address.</td>
</tr>
<tr>
<td></td>
<td>This is an IPv4-only socket option.</td>
<td></td>
</tr>
<tr>
<td><strong>Note:</strong> Multicast datagrams can be transmitted only on one interface at a time.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 3. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use this option to control or determine 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.</td>
<td>To enable, set to 1.</td>
<td>If enabled, will contain a 1.</td>
</tr>
<tr>
<td>This is an IPv4-only socket option.</td>
<td>To disable, set to 0.</td>
<td>If disabled, will contain a 0.</td>
</tr>
</tbody>
</table>

| **IP_MULTICAST_TTL**    | A 1-byte binary field containing the value of '00'x to 'FF'x. | A 1-byte binary field containing the value of '00'x to 'FF'x. |
| 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. | 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. | 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. |
| 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. | N/A |
| 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. | See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE. | See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ-SOURCE. |
| This is an IPv4-only socket option. | | |

| **IPv6_JOIN_GROUP**     | 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. | N/A |
| Use this option to control the reception of multicast packets and specify that the socket join a multicast group. | 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. |
| This is an IPv6-only socket option. | See SEZAINST(EZACOBOL) for the COBOL example of IPv6-MREQ. | |
Table 3. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IPV6_LEAVE_GROUP</strong></td>
<td>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/1 example of IPV6_MREQ. See SEZAINST(EZACOBOL) for the COBOL example of IPV6-MREQ.</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>IPV6_MULTICAST_HOPS</strong></td>
<td>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. <strong>Note:</strong> 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.</td>
<td>Contains a 4-byte binary value in the range 0 – 255 indicating the number of multicast hops.</td>
</tr>
<tr>
<td><strong>IPV6_MULTICAST_IF</strong></td>
<td>Contains a 4-byte binary field containing an IPv6 interface index number.</td>
<td>Contains a 4-byte binary field containing an IPv6 interface index number.</td>
</tr>
<tr>
<td><strong>IPV6_MULTICAST_LOOP</strong></td>
<td>A 4-byte binary field. To enable, set to 1. To disable, set to 0.</td>
<td>A 4-byte binary field. If enabled, contains a 1. If disabled, contains a 0.</td>
</tr>
<tr>
<td>OPTNAME options (input)</td>
<td>SETSOCKOPT, OPTVAL (input)</td>
<td>GETSOCKOPT, OPTVAL (output)</td>
</tr>
<tr>
<td>----------------------------</td>
<td>--------------------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| **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.  
-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. | Contains a 4-byte binary value in the range 0 – 255 indicating the number of unicast hops. |

This is an IPv6-only socket option.

| **IPV6_V6ONLY**            | 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. | A 4-byte binary field.  
To enable, set to 1.  
To disable, set to 0. | A 4-byte binary field.  
If enabled, contains a 1.  
If disabled, contains a 0. |

This is an IPv6-only socket option.

| **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.  
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_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.  
See SEZAINST(CBLOCK) for the PL/I example of GROUP_REQ.  
See SEZAINST(EZACOBOL) for the COBOL example of GROUP-REQ. | N/A |
<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MCAST_JOIN_SOURCE_GROUP</strong></td>
<td>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.</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>MCAST_LEAVE_GROUP</strong></td>
<td>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. See SEZAINST(CBLOCK) for the PL/I example of GROUP_REQ. See SEZAINST(EZACOBOL) for the COBOL example of GROUP-REQ.</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>MCAST_LEAVE_SOURCE_GROUP</strong></td>
<td>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.</td>
<td>N/A</td>
</tr>
</tbody>
</table>
### Table 3. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MCAST_UNBLOCK_SOURCE</strong></td>
<td>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.</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>See SEZAINST(CBLOCK) for the PL/I example of GROUP_SOURCE_REQ.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>See SEZAINST(EZACOBOL) for the COBOL example of GROUP-SOURCE-REQ.</td>
<td></td>
</tr>
<tr>
<td><strong>SO_ASCII</strong></td>
<td>To enable, set to ON. To disable, set to OFF. <strong>Note:</strong> The optvalue is returned and is optionally followed by the name of the translation table that is used if translation is applied to the data.</td>
<td>If enabled, contains ON. If disabled, contains OFF. <strong>Note:</strong> The optvalue is returned and is optionally followed by the name of the translation table that is used if translation is applied to the data.</td>
</tr>
<tr>
<td><strong>SO_BROADCAST</strong></td>
<td>A 4-byte binary field. To enable, set to 1 or a positive value. To disable, set to 0.</td>
<td>A 4-byte field. If enabled, contains a 1. If disabled, contains a 0.</td>
</tr>
<tr>
<td><strong>SO_DEBUG</strong></td>
<td>To enable, set to ON. To disable, set to OFF.</td>
<td>If enabled, contains ON. If disabled, contains OFF.</td>
</tr>
</tbody>
</table>

**Note:** This option has no meaning for stream sockets.

**Notes:**
1. This is a REXX-only socket option.
2. This option has meaning only for stream sockets.
<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SO_EBCDIC</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use this option to set or determine the translation to EBCDIC data option. When 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.</td>
<td>To enable, set to ON. To disable, set to OFF. <strong>Note:</strong> The optvalue is returned and is optionally followed by the name of the translation table that is used if translation is applied to the data.</td>
<td>If enabled, contains ON. If disabled, contains OFF. <strong>Note:</strong> The optvalue is returned and is optionally followed by the name of the translation table that is used if translation is applied to the data.</td>
</tr>
<tr>
<td><strong>SO_ERROR</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>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.</td>
<td>N/A</td>
<td>A 4-byte binary field containing the most recent ERRNO for the socket.</td>
</tr>
<tr>
<td><strong>SO_KEEPALIVE</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>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. 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.</td>
<td>A 4-byte binary field. To enable, set to 1 or a positive value. To disable, set to 0.</td>
<td>A 4-byte binary field. If enabled, contains a 1. If disabled, contains a 0.</td>
</tr>
</tbody>
</table>
### Table 3. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SO_LINGER</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use this option to control or determine how TCP/IP processes data that has not been transmitted when a CLOSE is issued for the socket. The default is disabled.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Notes:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. This option has meaning only for stream sockets.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use of the SO_LINGER option does not guarantee successful completion because TCP/IP only waits the amount of time specified in OPTVAL for SO_LINGER.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SO_OOBINLINE</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use this option to control or determine whether out-of-band data is received.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>This option has meaning only for stream sockets.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>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.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A 4-byte binary field.  
To enable, set to 1 or a positive value.  
To disable, set to 0.  
A 4-byte binary field.  
If enabled, contains a 1.  
If disabled, contains a 0.  
A 4-byte binary field.  
Contains an 8-byte field containing two 4-byte binary fields.  
Assembler coding:
```
ONOFF   DS F
LINGER  DS F
```
COBOL coding:
```
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 after the CLOSE is issued.
### Table 3. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SO_RCVBUF</strong></td>
<td>A 4-byte binary field.</td>
<td>A 4-byte binary field.</td>
</tr>
<tr>
<td></td>
<td>To enable, set to a positive value specifying the size of the data portion of the TCP/IP receive buffer.</td>
<td>If enabled, contains a positive value indicating the size of the data portion of the TCP/IP receive buffer.</td>
</tr>
<tr>
<td></td>
<td>To disable, set to a 0.</td>
<td>If disabled, contains a 0.</td>
</tr>
<tr>
<td><strong>SO_RCVTIMEO</strong></td>
<td>This option requires a TIMEVAL structure, which is defined in SYS1.MACLIB(BPXYRLIM) macro. The TIMEVAL structure contains the number of seconds and microseconds specified as fullword binary numbers. The seconds can be a value in the range 0 - 2 678 400 (equal to 31 days), and the microseconds can be a value in the range 0 - 1 000 000 (equal to 1 second). Although TIMEVAL value can be specified using microsecond granularity, the internal TCP/IP timers that are used to implement this function have a granularity of approximately 100 milliseconds.</td>
<td>This option stores a TIMEVAL structure that is defined in the SYS1.MACLIB(BPXYRLIM) macro. The TIMEVAL structure contains the number of seconds and microseconds, which are specified as fullword binary numbers. The number of seconds value that is returned is in the range 0 - 2 678 400 (equal to 31 days). The number of microseconds value that is returned is in the range 0 - 1 000 000.</td>
</tr>
</tbody>
</table>

**SO_RCVBUF**

Use this option to control or determine the size of the data portion of the TCP/IP receive buffer.

The size of the data portion of the receive buffer is protocol-specific, based on the following values prior to any SETSOCKOPT call:

- TCPRCVBufsize keyword on the TCPCONFIG statement in the PROFILE.TCPIP data set for a TCP Socket
- UDPRCVBufsize keyword on the UDPCONFIG statement in the PROFILE.TCPIP data set for a UDP Socket
- The default of 65 535 for a raw socket

**SO_RCVTIMEO**

Use this option to control or determine the maximum length of time that a receive-type function can wait before it completes.

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 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 following receive-type functions are supported:

- READ
- READV
- RECV
- RECVFROM
- RECVMSG
Table 3. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SO_REUSEADDR</strong></td>
<td>A 4-byte binary field.</td>
<td>A 4-byte binary field.</td>
</tr>
<tr>
<td>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. When this option is enabled, the following situations are supported:</td>
<td>To enable, set to 1 or a positive value.</td>
<td>If enabled, contains a 1.</td>
</tr>
<tr>
<td>• 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.</td>
<td>To disable, set to 0.</td>
<td>If disabled, contains a 0.</td>
</tr>
<tr>
<td>• A server with active client connections can be restarted and can bind to its port without having to close all of the client connections.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• For datagram sockets, multicasting is supported so multiple bind() calls can be made to the same class D address and port number.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• 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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SO_SNDBUF</strong></td>
<td>A 4-byte binary field.</td>
<td>A 4-byte binary field.</td>
</tr>
<tr>
<td>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:</td>
<td>To enable, set to a positive value specifying the size of the data portion of the TCP/IP send buffer.</td>
<td>If enabled, contains a positive value indicating the size of the data portion of the TCP/IP send buffer.</td>
</tr>
<tr>
<td>• The TCPSENDBufrsize keyword on the TCPCONFIG statement in the PROFILE.TCPIP data set for a TCP socket</td>
<td>To disable, set to a 0.</td>
<td>If disabled, contains a 0.</td>
</tr>
<tr>
<td>• The UDPSENDBufrsize keyword on the UDPCONFIG statement in the PROFILE.TCPIP data set for a UDP socket</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• The default of 65 535 for a raw socket</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPTNAME options (input)</td>
<td>SETSOCKOPT, OPTVAL (input)</td>
<td>GETSOCKOPT, OPTVAL (output)</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td><strong>SO_SNDTIMEO</strong></td>
<td>This option requires a TIMEVAL structure, which is defined in the SYS1.MACLIB(BPXYRLIM) macro. The TIMEVAL structure contains the number of seconds and microseconds specified as fullword binary numbers. The seconds value is in the range 0 - 2,678,400 (equal to 31 days), and the microseconds value is in the range 0 - 1,000,000 (equal to 1 second). Although the TIMEVAL value can be specified using microsecond granularity, the internal TCP/IP timers that are used to implement this function have a granularity of approximately 100 milliseconds.</td>
<td>This option stores a TIMEVAL structure that is defined in SYS1.MACLIB(BPXYRLIM). The TIMEVAL structure contains the number of seconds and microseconds, which are specified as fullword binary numbers. The number of seconds value that is returned is in the range 0 - 2,678,400 (equal to 31 days). The microseconds value that is returned is in the range 0 - 1,000,000.</td>
</tr>
<tr>
<td><strong>SO_TYPE</strong></td>
<td>N/A</td>
<td>A 4-byte binary field indicating the socket type: X'1' indicates SOCK_STREAM. X'2' indicates SOCK_DGRAM. X'3' indicates SOCK_RAW.</td>
</tr>
<tr>
<td><strong>TCP_KEEPALIVE</strong></td>
<td>A 4-byte binary field.</td>
<td>A 4-byte binary field.</td>
</tr>
</tbody>
</table>

Use this option to control or determine the maximum length of time that a send-type function can remain blocked before it completes.

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.

For a SETSOCKOPT, the following send-type functions are supported:
- SEND
- SENDMSG
- SENDTO
- WRITE
- WRITEV

Use this option to return the socket type.

Use this option to set or determine whether a socket-specific timeout value (in seconds) is to be used in place of a configuration-specific value whenever keep alive timing is active for that socket.

When activated, the socket-specific timer value remains in effect until respecified by SETSOCKOPT or until the socket is closed. See the *z/OS Communications Server: IP Programmer’s Guide and Reference* for more information about the socket option parameters.
Table 3. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP_NODELAY</td>
<td>A 4-byte binary field.</td>
<td>A 4-byte binary field.</td>
</tr>
<tr>
<td></td>
<td>To enable, set to a 0.</td>
<td>If enabled, contains a 0.</td>
</tr>
<tr>
<td></td>
<td>To disable, set to a 1 or nonzero.</td>
<td>If disabled, contains a 1.</td>
</tr>
</tbody>
</table>

**TCP_NODELAY**

Use this option to set or determine whether data sent over the socket is subject to the Nagle algorithm (RFC 896).

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:

```cobol
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.
```

**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.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Authorization</th>
<th>Supervisor state or problem state, any PSW key.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatchable unit mode</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Amode</td>
<td>31-bit or 24-bit.</td>
</tr>
<tr>
<td>Note:</td>
<td>See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary address space control (ASC) mode.</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for interrupts.</td>
</tr>
<tr>
<td>Locks</td>
<td>Unlocked.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>All parameters must be addressable by the caller and in the primary address space.</td>
</tr>
</tbody>
</table>

Figure 35 shows an example of GIVESOCKET call instructions.

```plaintext
WORKING-STORAGE SECTION.
  01 SOC-FUNCTION 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 59(8) BINARY.

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

Figure 35. GIVESOCKET call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 60.

Parameter values set by the application

**SOC-FUNCTION**
A 16-byte character field containing GIVESOCKET. The field is left-justified 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 only be taken by a TAKESOCKET with the same DOMAIN (AF_INET or AF_INET6).

NAME
Specifies an eight-character field, left-justified, 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 Appendix A. Return codes on page 317 for information about ERRNO return codes.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful call.</td>
</tr>
<tr>
<td>−1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

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
The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization</td>
<td>Supervisor state or problem state, any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Amode</td>
<td>31-bit or 24-bit.</td>
</tr>
<tr>
<td>Note</td>
<td>See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary address space control (ASC) mode.</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for interrupts.</td>
</tr>
<tr>
<td>Locks</td>
<td>Unlocked.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>All parameters must be addressable by the caller and in the primary address space.</td>
</tr>
</tbody>
</table>

Figure 36 shows an example of INITAPI call instructions.

```asm
WORKING-STORAGE SECTION.
  01 SOC-FUNCTION PIC X(16) VALUE IS 'INITAPI'.
  01 MAXSOC PIC 9(4) BINARY.
  01 IDENT.
    02 TCPNAME PIC X(8).
    02 ADSNAME PIC X(8).
  01 SUBTASK PIC X(8).
  01 MASNO PIC 9(8) BINARY.
  01 ERNNO PIC 9(8) BINARY.
  01 RETCODE PIC S9(8) BINARY.

PROCEDURE DIVISION.
  CALL 'EZASOKET' USING SOC-FUNCTION MAXSOC IDENT SUBTASK MASNO ERNNO RETCODE.
```

Figure 36. INITAPI call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 60.

**Parameter values set by the application**

**SOC-FUNCTION**
A 16-byte character field containing INITAPI. The field is left-justified 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 65 535 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. 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 multitread or multitask application, a separate copy needs to be loaded for each thread or task. See z/OS Communications Server: IP CICS Sockets Guide 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 Appendix A. Return codes on page 317 for information about ERRNO return codes.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful call.</td>
</tr>
<tr>
<td>-1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

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 4 on page 127 for information about REQARG and RETARG.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Authorization:</th>
<th>Supervisor state or problem state, any PSW key.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN.</td>
</tr>
</tbody>
</table>
Amode: 31-bit or 24-bit.

Note: See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.

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 37 shows an example of IOCTL call instructions.

```
WORKING-STORAGE SECTION.
  01 SOC-FUNCTION PIC X(16) VALUE 'IOCTL'.
  01 S PIC 9(4) BINARY.
  01 COMMAND PIC 9(8) BINARY.
  01 IFREQ.
    03 NAME PIC X(16).
    03 FAMILY PIC 9(4) BINARY.
    03 PORT PIC 9(4) BINARY.
    03 ADDRESS PIC 9(8) BINARY.
    03 RESERVED PIC X(8).
  01 IFREQOUT.
    03 NAME PIC X(16).
    03 FAMILY PIC 9(4) BINARY.
    03 PORT PIC 9(4) BINARY.
    03 ADDRESS PIC 9(8) BINARY.
    03 RESERVED PIC X(8).
  01 GRP-IOCTL-TABLE.
    02 IOCTL-ENTRY OCCURS 100 TIMES.
      03 NAME PIC X(16).
      03 FAMILY PIC 9(4) BINARY.
      03 PORT PIC 9(4) BINARY.
      03 ADDRESS PIC 9(8) BINARY.
      03 NULLS PIC X(8).
  01 IOCTL-REQARG USAGE IS POINTER.
  01 IOCTL-RETARG USAGE IS POINTER.
  01 ERRNO PIC 9(8) BINARY.
  01 RETCODE PIC 9(8) BINARY.

PROCEDURE DIVISION.
  CALL 'EZASOKET' USING SOC-FUNCTION S COMMAND REQARG
                   RETARG ERRNO RETCODE.
```

Figure 37. IOCTL call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 60.

Parameter values set by the application

SOC-FUNCTION
   A 16-byte character field containing IOCTL. The field is left-justified and padded to the right with blanks.
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.
  - When the SIOCGHOMEIF6 IOCTL is issued, the REGARQ must contain a Network Configuration Header. The NETCONFHDR is defined in the SYS1.MACLIB(BPXYIOC6) for assembler 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 4 on page 127**.

**Working-Storage Section.**

```
01 SIOCGHOMEIF6-VAL          pic s9(10) binary value 3222599176.
01 SIOCGHOMEIF6-REDEF        REDEFINES SIOCGHOMEIF6-VAL.
  05 FILLER                  PIC 9(6) COMP.
  05 SIOCGHOMEIF6            PIC 9(8) COMP.
01 IOCTL-RETARG             USAGE IS POINTER.
01 NET-CONF-HDR.             PIC 9(8) BINARY.
  05 NCH-EYE-CATCHER         PIC X(4) VALUE '6NCH'.
  05 NCH-IOCTL               PIC 9(8) BINARY.
  05 NCH-BUFFER-LENTH        PIC 9(8) BINARY.
  05 NCH-BUFFER-PTR          USAGE IS POINTER.
  05 NCH-NUM-ENTRY-RET       PIC 9(8) BINARY.
01 HOME-IF.                  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.
    05 NchNumEntryRet       pic 9(8) binary.
* Allocate storage based on your need.
  03 Allocated-Storage     pic x(nn).
```

**Procedure Division using L1.**

```
move '6NCH' to NchEyeCatcher.
set NchBufferPtr to address of Allocated-Storage.
* Set NchBufferLength to the length of your allocated storage.
move nn to NchBufferLength.
move SIOCGHOMEIF6 to NchIoctl.
Call 'EZASOKET' using socket-ioctl socket-descriptor
SIOCGHOMEIF6 NETCONFHDR NETCONFHDR
errno retcode.
```

**Figure 38. COBOL language example for SIOCGHOMEIF6**

**SIOCIFADDR**

Requests the IPv4 network interface address for a given interface name. For assembler, see the IOCN_IFNAME field in the SYS1.MACLIB(BPXYI0CC) 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.

**SIOCIFBRAADDR**

Requests the IPv4 network interface broadcast address for a given interface name. For assembler, see the IOCN_IFNAME field in the SYS1.MACLIB(BPXYI0CC) 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.

**SIOCIFCONF**

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 only returned for an interface if it has at least one available IP address.

The configuration consists of IF_NAMEINDEX structure, which is defined in SYS1.MACLIB(BPX1IOCC) for the assembler 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-NINDEX, IF-NINAME, and IF-NIEXT) to get the size of the array element.
  3. Add the size of the IF-NITOTALIF and IF-NIENTRY 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.
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 assembler 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.

Figure 39. COBOL language example for SIOCGIFNAMEINDEX
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.

**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 "IP 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 z/OS Communications Server: IP Configuration Guide 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(BPXYI0CC) 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 IP_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.

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, EZBZTLP defines the assembler layout, and EZBZTLSB defines the COBOL layout. For more usage details, see the Application Transparent TLS (AT-TLS) 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 and is used to pass arguments to IOCTL, and RETARG is an output parameter and receives arguments from IOCTL. The REQARG and RETARG parameters are described in Table 4.

<table>
<thead>
<tr>
<th>COMMAND/CODE</th>
<th>SIZE</th>
<th>REQARG</th>
<th>SIZE</th>
<th>RETARG</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIONBIO X'8004A77E'</td>
<td>4</td>
<td>Set socket mode to: X'00'=blocking, X'01'=nonblocking.</td>
<td>0</td>
<td>Not used.</td>
</tr>
<tr>
<td>FIONREAD X'4004A77F'</td>
<td>0</td>
<td>Not used.</td>
<td>4</td>
<td>Number of characters available for read.</td>
</tr>
<tr>
<td>SIOCATMARK X'4004A707'</td>
<td>0</td>
<td>Not used.</td>
<td>4</td>
<td>X'00'= not at OOB data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X'01'= at OOB data.</td>
</tr>
<tr>
<td>SIOCGHOMEIF6 XC014F608'</td>
<td>20</td>
<td>NetConfHdr</td>
<td></td>
<td>See Figure 38 on page 123 NetConfHdr.</td>
</tr>
<tr>
<td>COMMAND/CODE</td>
<td>SIZE</td>
<td>REQARG</td>
<td>SIZE</td>
<td>RETARG</td>
</tr>
<tr>
<td>--------------</td>
<td>------</td>
<td>--------</td>
<td>------</td>
<td>--------</td>
</tr>
<tr>
<td>SIOCGIFADDR X'C020A70D'</td>
<td>32</td>
<td>First 16 bytes - interface name. Last 16 bytes - not used.</td>
<td>32</td>
<td>Network interface address. For assembler, see the IOCN_SADDRIF field in the SYSLINLIB(BPXYOCC) 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.</td>
</tr>
<tr>
<td>SIOCGIFBRDADDR X'C020A712'</td>
<td>32</td>
<td>First 16 bytes - interface name. Last 16 bytes - not used.</td>
<td>32</td>
<td>Network interface address. For assembler, see the IOCN_SADDRIBROADCAST field in the SYSLINLIB(BPXYOCC) API. For COBOL, see the IFR_BROADADDR field in the SEZAINST(EZACOBOL) API. For PL/I, see the IFR_BROADCAST field in the SEZAINST(CBLOCK) API.</td>
</tr>
<tr>
<td>SIOCGIFCONF X'C008A714'</td>
<td>8</td>
<td>Size of RETARG.</td>
<td>See note 1.</td>
<td></td>
</tr>
<tr>
<td>SIOCGIFDSTADDR X'C020A70F'</td>
<td>32</td>
<td>First 16 bytes - interface name. Last 16 bytes - not used.</td>
<td>32</td>
<td>Destination interface address. For assembler, see the IOCN_SADDRIFDEST field in the SYSLINLIB(BPXYOCC) 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.</td>
</tr>
<tr>
<td>SIOCGIFMTU X'C020A726'</td>
<td>32</td>
<td>First 16 bytes - interface name. Last 16 bytes - not used.</td>
<td>32</td>
<td>IPv4 interface MTU (maximum transmission unit). For assembler, see the IOCN_MTUSIZE field in the SYSLINLIB(BPXYOCC) 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.</td>
</tr>
<tr>
<td>SIOCGIFNAMEINDEX X'4000F603'</td>
<td>4</td>
<td>First 4 bytes size of return buffer.</td>
<td>See Figure 39 on page 124 IF-NAMEINDEX.</td>
<td></td>
</tr>
<tr>
<td>SIOCGIPMSFILTER X'C000A724'</td>
<td>–</td>
<td>See IP_MSFILTER structure in macro BPXYOCC. See note 2.</td>
<td>0</td>
<td>Not used</td>
</tr>
<tr>
<td>SIOCGMMSFILTER X'C000F610'</td>
<td>–</td>
<td>See GROUP_FILTER structure in macro BPXYOCC. See note 3</td>
<td>0</td>
<td>Not used</td>
</tr>
<tr>
<td>SIOCSAPPLDATA X'8018D90C'</td>
<td>–</td>
<td>See SETAPPLDATA structure in macro EZBYAPPL</td>
<td>0</td>
<td>Not used</td>
</tr>
<tr>
<td>SIOCSIPMSFILTER X'8000A725'</td>
<td>–</td>
<td>See IP_MSFILTER structure in macro BPXYOCC. See note 2.</td>
<td>0</td>
<td>Not used</td>
</tr>
<tr>
<td>SIOCSMSFILTER X'8000F611'</td>
<td>–</td>
<td>See GROUP_FILTER structure in macro BPXYOCC. See note 3</td>
<td>0</td>
<td>Not used</td>
</tr>
<tr>
<td>SIOCTTLSCTL X'C038D90B'</td>
<td>56</td>
<td>For IOCTL structure layout, see SEZANMAC(EZBZTLSI) for PL/I, SEZANMAC(EZBZTLSI) for assembler, and SEZANMAC(EZBZTLSI) for COBOL.</td>
<td>56</td>
<td>For IOCTL structure layout, see SEZANMAC(EZBZTLSI) for PL/I, SEZANMAC(EZBZTLSI) for assembler, and SEZANMAC(EZBZTLSI) for COBOL.</td>
</tr>
</tbody>
</table>

Notes:
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.

Parameter values returned to the application RETARG

RETARG

Returns an array whose size is based on the value in COMMAND. See Table 4 for information about REQARG and RETARG.
ERRNO
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix A. Return codes on page 317 for information about ERRNO return codes.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful call.</td>
</tr>
<tr>
<td>−1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

The command SIOCGIFCONF returns a variable number of network interface configurations. Figure 40 contains an example of a COBOL II routine that can be used to work with such a structure.

Note: This call can only be programmed in languages that support address pointers. Figure 40 shows a COBOL II example for SIOCGIFCONF.

WORKING-STORAGE SECTION.
77 REQARG PIC 9(8) COMP.
77 COUNT PIC 9(8) COMP VALUE max number of interfaces.
LINKAGE SECTION.
01 RETARG.
   05 IOCTL-TABLE OCCURS 1 TO max TIMES DEPENDING ON COUNT.
      10 NAME PIC X(16).
      10 FAMILY PIC 9(4) BINARY.
      10 PORT PIC 9(4) BINARY.
      10 ADDR PIC 9(8) BINARY.
      10 NULLS PIC X(8).
PROCEDURE DIVISION.
   MULTIPLY COUNT BY 32 GIVING REQARG.
   CALL 'EZASOKET' USING SOC-FUNCTION S COMMAND REQARG RETARG ERRNO RETCODE.

Figure 40. 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.

The following requirements apply to this call:

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 “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.

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 41 shows an example of LISTEN call instructions.

```
WORKING-STORAGE SECTION.
  01 SOC-FUNCTION PIC X(16) VALUE IS 'LISTEN'.
  01 S PIC 9(4) BINARY.
  01 BACKLOG PIC 9(8) BINARY.
  01 ERRNO PIC 9(8) BINARY.
  01 RETCODE PIC S9(8) BINARY.

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

*Figure 41. LISTEN call instruction example*

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 60.

### Parameter values set by the application

**SOC-FUNCTION**
A 16-byte character field containing LISTEN. The field is left-justified 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](https://www.ibm.com/support/knowledgecenter/ST9QVP_11.1.0/com.ibm.zos.v11r1.doc/commserver_ipconfig_ref/commserver_ipconfig_ref.html) for details.

### 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 317 for information about ERRNO return codes.

**RETCODE**
A fullword binary field that returns one of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful call.</td>
</tr>
<tr>
<td>-1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>
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.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Requirement Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization</td>
<td>Supervisor state or problem state, any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Amode</td>
<td>31-bit or 24-bit.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary address space control (ASC) mode.</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for interrupts.</td>
</tr>
<tr>
<td>Locks</td>
<td>Unlocked.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>All parameters must be addressable by the caller and in the primary address space.</td>
</tr>
</tbody>
</table>

Figure 42 on page 132 shows an example of NTOP call instructions.
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 317 for information about ERRNO return codes.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful call.</td>
</tr>
<tr>
<td>-1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

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.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Authorization:</th>
<th>Supervisor state or problem state, any PSW key.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Amode:</td>
<td>31-bit or 24-bit.</td>
</tr>
</tbody>
</table>

**Note:** See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.

<table>
<thead>
<tr>
<th>ASC mode:</th>
<th>Primary address space control (ASC) mode.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt status:</td>
<td>Enabled for interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>Unlocked.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>All parameters must be addressable by the caller and in the primary address space.</td>
</tr>
</tbody>
</table>

Figure 43 on page 134 shows an example of PTON call instructions.
Parameter values set by the application

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAMILY</td>
<td>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.</td>
</tr>
</tbody>
</table>
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 317 for information about ERRNO return codes.

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

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 194 for a subroutine that will translate ASCII input data to EBCDIC.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Authorization:</th>
<th>Supervisor state or problem state, any PSW key.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Amode:</td>
<td>31-bit or 24-bit.</td>
</tr>
</tbody>
</table>

Note: See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.
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 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 44. READ call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 60.

Parameter values set by the application

**SOC-FUNCTION**
A 16-byte character field containing READ. The field is left-justified 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 Appendix A. Return codes on page 317 for information about ERRNO return codes.

**RETCODE**
A fullword binary field that returns one of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A 0 return code indicates that the connection is closed and no data is available.</td>
</tr>
<tr>
<td>&gt;0</td>
<td>A positive value indicates the number of bytes copied into the buffer.</td>
</tr>
<tr>
<td>-1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>
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.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Requirement Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization</td>
<td>Supervisor state or problem state, any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN = HASN</td>
</tr>
<tr>
<td>Amode</td>
<td>31-bit or 24-bit</td>
</tr>
<tr>
<td>Note:</td>
<td>See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary address space control (ASC) mode.</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for interrupts.</td>
</tr>
<tr>
<td>Locks</td>
<td>Unlocked</td>
</tr>
<tr>
<td>Control parameters</td>
<td>All parameters must be addressable by the caller and in the primary address space.</td>
</tr>
</tbody>
</table>

Figure 45 shows an example of READV call instructions.

WORKING-STORAGE SECTION.
01 SOC-FUNCTION PIC X(16) VALUE 'READV'.
01 S PIC 9(4) BINARY.
01 IOVCNT PIC 9(8) BINARY.
01 IOV.
03 BUFFER-ENTRY OCCURS N TIMES.
   05 BUFFER-POINTER USAGE IS POINTER.
   05 RESERVED PIC X(4).
   05 BUFFER_LENGTH PIC 9(8) BINARY.
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 'EZASOCKET' USING SOC-FUNCTION S IOV IOVCNT ERRNO RETCODE.

Figure 45. READV call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 60.
Parameter values set by the application

**SOC-FUNCTION**
A 16-byte character field containing READV. The field is left-justified 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.

**IOV**
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 317 for information about ERRNO return codes.

**RETCODE**
A fullword binary field that returns one of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A 0 return code indicates that the connection is closed and no data is available.</td>
</tr>
<tr>
<td>&gt;0</td>
<td>A positive value indicates the number of bytes copied into the buffer.</td>
</tr>
<tr>
<td>−1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

**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 71 or "IOCTL" on page 119 for a description of how to set nonblocking mode.

For raw sockets, RECV adds a 20-byte header.

**Note:** See "EZACIC05" on page 194 for a subroutine that will translate ASCII input data to EBCDIC.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization</td>
<td>Supervisor state or problem state, any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Amode</td>
<td>31-bit or 24-bit.</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary address space control (ASC) mode.</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for interrupts.</td>
</tr>
<tr>
<td>Locks</td>
<td>Unlocked.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>All parameters must be addressable by the caller and in the primary address space.</td>
</tr>
</tbody>
</table>

**Figure 46** shows an example of RECV call instructions.

```
WORKING-STORAGE SECTION.
  01 SOC-FUNCTION PIC X(16) VALUE IS 'RECV'.
  01 S PIC 9(4) BINARY.
  01 FLAGS PIC 9(8) BINARY.
    88 NO-FLAG VALUE IS 0.
    88 OOB VALUE IS 1.
    88 PEEK VALUE IS 2.
  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 46. RECV call instruction example**

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 60.

**Parameter values set by the application**

**SOC-FUNCTION**
A 16-byte character field containing RECV. The field is left-justified 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:

<table>
<thead>
<tr>
<th>Literal Value</th>
<th>Binary Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO-FLAG</td>
<td>X'00000000'</td>
<td>Read data.</td>
</tr>
<tr>
<td>MSG-OOB</td>
<td>X'00000001'</td>
<td>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.</td>
</tr>
<tr>
<td>MSG-PEEK</td>
<td>X'00000002'</td>
<td>Peek at the data, but do not destroy data. If the peek flag is set, the next receive operation reads the same data.</td>
</tr>
<tr>
<td>MSG-WAITALL</td>
<td>X'00000040'</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 Appendix A. Return codes on page 317 for information about ERRNO return codes.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The socket is closed.</td>
</tr>
<tr>
<td>&gt;0</td>
<td>A positive return code indicates the number of bytes copied into the buffer.</td>
</tr>
<tr>
<td>-1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

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 “FCNTL” on page 71 or “IOCTL” on page 119 for a description of how to set nonblocking mode.

**Note:** See “EZACIC05” on page 194 for a subroutine that will translate ASCII input data to EBCDIC.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Authorization:</strong></td>
<td>Supervisor state or problem state, any PSW key.</td>
</tr>
<tr>
<td><strong>Dispatchable unit mode:</strong></td>
<td>Task.</td>
</tr>
<tr>
<td><strong>Cross memory mode:</strong></td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td><strong>Amode:</strong></td>
<td>31-bit or 24-bit.</td>
</tr>
<tr>
<td><strong>ASC mode:</strong></td>
<td>Primary address space control (ASC) mode.</td>
</tr>
<tr>
<td><strong>Interrupt status:</strong></td>
<td>Enabled for interrupts.</td>
</tr>
<tr>
<td><strong>Locks:</strong></td>
<td>Unlocked.</td>
</tr>
<tr>
<td><strong>Control parameters:</strong></td>
<td>All parameters must be addressable by the caller and in the primary address space.</td>
</tr>
</tbody>
</table>

**Figure 47 on page 142** shows an example of RECVFROM call instructions.
Parameter values set by the application

**SOC-FUNCTION**
A 16-byte character field containing RECVFROM. The field is left-justified 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:

- **NO-FLAG** VALUE IS 0.
- **OOB** VALUE IS 1.
- **PEEK** VALUE IS 2.

---

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 60.

---

Figure 47. RECVFROM call instruction example
### 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'00000040' | 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 Internet 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 Internet 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 Appendix A. Return codes on page 317 for information about ERRNO return codes.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The socket is closed.</td>
</tr>
<tr>
<td>&gt;0</td>
<td>A positive return code indicates the number of bytes of data transferred by the read call.</td>
</tr>
<tr>
<td>-1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

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.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Authorization:</th>
<th>Supervisor state or problem state, any PSW key.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Amode:</td>
<td>31-bit or 24-bit.</td>
</tr>
</tbody>
</table>

Note: See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.

| 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 145 shows an example of RECVMSG call instructions.
WORKING-STORAGE SECTION.
01 SOC-FUNCTION PIC X(16) VALUE IS 'RECVMSG'.
01 S PIC 9(4) BINARY.
01 MSG-HDR.
  03 MSG-NAME USAGE IS POINTER.
  03 MSG-NAME-LEN PIC 9(8) COMP.
  03 IOV USAGE IS POINTER.
  03 IOVCNT USAGE IS POINTER.
  03 MSG-ACCRIGHTS USAGE IS POINTER.
  03 MSG-ACCRIGHTS-LEN USAGE IS POINTER.
01 FLAGS PIC 9(8) BINARY.
  88 NO-FLAG VALUE IS 0.
  88 OOB VALUE IS 1.
  88 PEEK VALUE IS 2.
01 ERRNO PIC 9(8) BINARY.
01 RETCODE PIC S9(8) BINARY.

LINKAGE SECTION.
01 LI.
  03 RECVMSG-IOVECTOR.
    05 IOV1A USAGE IS POINTER.
    05 IOV1AL PIC 9(8) COMP.
    05 IOV1L PIC 9(8) COMP.
    05 IOV2A USAGE IS POINTER.
    05 IOV2AL PIC 9(8) COMP.
    05 IOV2L PIC 9(8) COMP.
    05 IOV3A USAGE IS POINTER.
    05 IOV3AL PIC 9(8) COMP.
    05 IOV3L PIC 9(8) COMP.
  03 RECVMSG-BUFFER1 PIC X(16).
  03 RECVMSG-BUFFER2 PIC X(16).
  03 RECVMSG-BUFFER3 PIC X(16).
  03 RECVMSG-BUFNO PIC 9(8) COMP.

* IPv4 socket address structure.
  03 NAME.
    05 FAMILY PIC 9(4) BINARY.
    05 PORT PIC 9(4) BINARY.
    05 IP-ADDRESS PIC 9(8) BINARY.
    05 RESERVED PIC X(8).

* IPv6 socket address structure.
  03 NAME.
    05 FAMILY PIC 9(4) BINARY.
    05 PORT PIC 9(4) BINARY.
    53 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.

Figure 48. RECVMSG call instruction example (Part 1 of 2)
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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>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:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAMILY</td>
<td>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.</td>
</tr>
<tr>
<td>PORT</td>
<td>Output parameter. A halfword binary number specifying the port number of the sending socket.</td>
</tr>
<tr>
<td>IP-ADDRESS</td>
<td>Output parameter. A fullword binary number specifying the 32-bit IPv4 Internet address of the sending socket.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>Output parameter. An 8-byte reserved field. This field is required, but is not used.</td>
</tr>
</tbody>
</table>

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 Internet 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:
<table>
<thead>
<tr>
<th>Literal Value</th>
<th>Binary Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO-FLAG</td>
<td>X’00000000’</td>
<td>Read data.</td>
</tr>
<tr>
<td>MSG-OOB</td>
<td>X’00000001’</td>
<td>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.</td>
</tr>
<tr>
<td>MSG-PEEK</td>
<td>X’00000002’</td>
<td>Peek at the data, but do not destroy data. If the peek flag is set, the next receive operation reads the same data.</td>
</tr>
<tr>
<td>MSG-WAITALL</td>
<td>X’00000040’</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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 317 for information about ERRNO return codes.

**RETCODE**
A fullword binary field with the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0</td>
<td>Call returned error. See ERRNO field.</td>
</tr>
<tr>
<td>0</td>
<td>Connection partner has closed connection.</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Number of bytes read.</td>
</tr>
</tbody>
</table>

**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:
- Set the read, write, and exception arrays to zeros.
- Specify MAXSOC <= 0.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Authorization:</th>
<th>Supervisor state or problem state, any PSW key.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN.</td>
</tr>
</tbody>
</table>
Amode: 31-bit or 24-bit.

Note: See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.

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.

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 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 selecting requests in a concurrent server program, see z/OS Communications 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 196.

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 would be 0 through 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 49 on page 151 shows an example of SELECT call instructions.
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 assembler language declarations, see "Converting parameter descriptions" on page 60.

Parameter values set by the application

SOC-FUNCTION
A 16-byte character field containing SELECT. The field is left-justified 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.

* The bit mask lengths can be determined from the expression:
  \[((maximum socket number +32)/32 (drop the remainder))\times4

Figure 49. 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 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 Appendix A. Return codes on page 317 for information about ERRNO return codes.

**RETCODE**
A fullword binary field that returns one of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;0</td>
<td>Indicates the sum of all ready sockets in the three masks.</td>
</tr>
</tbody>
</table>
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:
• Set the read, write, and exception arrays to zeros.
• Specify MAXSOC ≤ 0.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Authorization:</th>
<th>Supervisor state or problem state, any PSW key.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Amode:</td>
<td>31-bit or 24-bit.</td>
</tr>
</tbody>
</table>

Note: See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.

<table>
<thead>
<tr>
<th>ASC mode:</th>
<th>Primary address space control (ASC) mode.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt status:</td>
<td>Enabled for interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>Unlocked.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>All parameters must be addressable by the caller and in the primary address space.</td>
</tr>
</tbody>
</table>

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 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 selecting requests in a concurrent server program, see z/OS Communications 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 196.

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 would be 0 through 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.

Figure 50 on page 156 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 below:

WORKING-STORAGE SECTION.
  01 SOC-FUNCTION PIC X(16) VALUE IS 'SELECTEX'.
  01 MAXSOC PIC 9(B) BINARY.
  01 TIMEOUT.
    03 TIMEOUT-SECONDS PIC 9(B) BINARY.
    03 TIMEOUT-MINUTES PIC 9(B) BINARY.
  01 RSNOMSK PIC X(*).
  01 WSNOMSK PIC X(*).
  01 ESNOMSK PIC X(*).
  01 RRETMSK PIC X(*).
  01 WRETMSK PIC X(*).
  01 ERETMSK PIC X(*).
  01 SELECB PIC X(4).
  01 ERRNO PIC 9(B) BINARY.
  01 RETCODE PIC S9(8) BINARY.

Where * is the size of the select mask

PROCEDURE DIVISION.
  CALL 'EZASOKET' USING SOC-FUNCTION MAXSOC TIMEOUT
      RSNOMSK WSNOMSK ESNOMSK
      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.
  01 SOC-FUNCTION PIC X(16) VALUE IS 'SELECTEX'.
  01 MAXSOC PIC 9(B) BINARY.
  01 TIMEOUT.
    03 TIMEOUT-SECONDS PIC 9(B) BINARY.
    03 TIMEOUT-MINUTES PIC 9(B) BINARY.
  01 RSNOMSK PIC X(*).
  01 WSNOMSK PIC X(*).
  01 ESNOMSK PIC X(*).
  01 RRETMSK PIC X(*).
  01 WRETMSK PIC X(*).
  01 ERETMSK PIC X(*).
  01 ECBLIST-PTR USAGE IS POINTER.
  01 ERRNO PIC 9(B) BINARY.
  01 RETCODE PIC S9(8) BINARY.

Where * is the size of the select mask

PROCEDURE DIVISION.
  CALL 'EZASOKET' USING SOC-FUNCTION MAXSOC TIMEOUT
      RSNOMSK WSNOMSK ESNOMSK
      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 50. SELECTEX call instruction example
Parameter values set by the application

**SOC-FUNCTION**
A 16-byte character field containing SELECT. The field is left-justified 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 50000.

**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 Appendix A. Return codes on page 317 for information about ERRNO return codes.
RETCODE
A fullword binary field

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;0</td>
<td>The number of ready sockets.</td>
</tr>
<tr>
<td>0</td>
<td>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.</td>
</tr>
<tr>
<td>-1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

RRETIMSK
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.

WRETIMSK
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.

EREITMSK
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 “EZACIC04” on page 192 for a subroutine that will translate EBCDIC input data to ASCII.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Authorization:</th>
<th>Supervisor state or problem state, any PSW key.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN.</td>
</tr>
</tbody>
</table>
Amode: 31-bit or 24-bit.

Note: See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.

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 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 OOB 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 51. SEND call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 60.

Parameter values set by the application

SOC-FUNCTION
A 16-byte character field containing SEND. The field is left-justified 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:

<table>
<thead>
<tr>
<th>Literal Value</th>
<th>Binary Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO-FLAG</td>
<td>X'00000000'</td>
<td>No flag is set. The command behaves like a WRITE call.</td>
</tr>
<tr>
<td>MSG-OOB</td>
<td>X'00000001'</td>
<td>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.</td>
</tr>
<tr>
<td>MSG-DONTROUTE</td>
<td>X'00000004'</td>
<td>Do not route. Routing is provided by the calling program.</td>
</tr>
</tbody>
</table>
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 317 for information about ERRNO return codes.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>≥0</td>
<td>A successful call. The value is set to the number of bytes transmitted.</td>
</tr>
<tr>
<td>-1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

SENDMSG
The SENDMSG call sends messages on a socket with descriptor S passed in an array of messages.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization:</td>
<td>Supervisor state or problem state, any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Amode:</td>
<td>31-bit or 24-bit.</td>
</tr>
</tbody>
</table>

Note: See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASC mode:</td>
<td>Primary address space control (ASC) mode.</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>Unlocked.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>All parameters must be addressable by the caller and in the primary address space.</td>
</tr>
</tbody>
</table>

Figure 52 on page 161 shows an example of SENDMSG call instructions.
WORKING-STORAGE SECTION.
  01 SOC-FUNCTION PIC X(16) VALUE IS 'SENDMSG'.
  01 S PIC 9(4) BINARY.
  01 MSG-HDR.
    03 MSG-NAME USAGE IS POINTER.
    03 MSG-NAME-LEN PIC 9(8) BINARY.
    03 IOV USAGE IS POINTER.
    03 IOVCNT USAGE IS POINTER.
    03 MSG-ACCRIGHTS USAGE IS POINTER.
    03 MSG-ACCRIGHTS-LEN USAGE IS POINTER.
  01 FLAGS PIC 9(8) BINARY.
    88 NO-FLAG VALUE IS 0.
    88 OOB VALUE IS 1.
    88 DONTROUTE VALUE IS 4.
  01 ERRNO PIC 9(8) BINARY.
  01 RETCODE PIC S9(8) BINARY.
  01 SENDMSG-IPV4ADDR PIC 9(8) BINARY.
  01 SENDMSG-IPV6ADDR.
    05 FILLER PIC 9(16) BINARY.
    05 FILLER PIC 9(16) BINARY.
LINKAGE SECTION.
  01 L1.
  03 SENDMSG-IOVECTOR.
    05 IOV1A USAGE IS POINTER.
    05 IOV1AL PIC 9(8) COMP.
    05 IOV1L PIC 9(8) COMP.
    05 IOV2A USAGE IS POINTER.
    05 IOV2AL PIC 9(8) COMP.
    05 IOV2L PIC 9(8) COMP.
    05 IOV3A USAGE IS POINTER.
    05 IOV3AL PIC 9(8) COMP.
    05 IOV3L PIC 9(8) COMP.
  03 SENDMSG-BUFFER1 PIC X(16).
  03 SENDMSG-BUFFER2 PIC X(16).
  03 SENDMSG-BUFFER3 PIC X(16).
  03 SENDMSG-BUFNO PIC 9(8) COMP.

* IPv4 socket address structure.

  03 NAME.
    05 FAMILY PIC 9(4) BINARY.
    05 PORT PIC 9(4) BINARY.
    05 IP-ADDRESS PIC 9(8) BINARY.
    05 RESERVED PIC X(8) BINARY.

* IPv6 socket address structure.

  03 NAME.
    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.

Figure 52. SENDMSG call instruction example (Part 1 of 2)
**Parameter values set by the application**

**SOC-FUNCTION**
A 16-byte character field containing SENDMSG. The field is left-justified 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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>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:</td>
</tr>
<tr>
<td>FAMILY</td>
<td>Output parameter. A halfword binary number specifying</td>
</tr>
</tbody>
</table>
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 Internet 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 Internet 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
may be specified for any address types and scopes. For a
link scope IPv6-ADDRESS, SCOPE-ID may 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:

<table>
<thead>
<tr>
<th>Literal Value</th>
<th>Binary Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO-FLAG</td>
<td>X'00000000'</td>
<td>No flag is set. The command behaves like a WRITE call.</td>
</tr>
<tr>
<td>MSG-OOB</td>
<td>X'00000001'</td>
<td>Send out-of-band data. (Stream sockets only.)</td>
</tr>
<tr>
<td>MSG-DONTROUTE</td>
<td>X'00000004'</td>
<td>Do not route. Routing is provided by the calling program.</td>
</tr>
</tbody>
</table>

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 317 for information about ERRNO return codes.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>≥0</td>
<td>A successful call. The value is set to the number of bytes transmitted.</td>
</tr>
<tr>
<td>-1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

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 “EZACIC04” on page 192 for a subroutine that will translate EBCDIC input data to ASCII.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Authorization:</th>
<th>Supervisor state or problem state, any PSW key.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Amode:</td>
<td>31-bit or 24-bit.</td>
</tr>
<tr>
<td>Note: See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.</td>
<td></td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary address space control (ASC) mode.</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>Unlocked.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>All parameters must be addressable by the caller and in the primary address space.</td>
</tr>
</tbody>
</table>

Figure 53 on page 166 shows an example of SENDTO call instructions.
For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 60.

**Parameter values set by the application**

**SOC-FUNCTION**
A 16-byte character field containing SENDTO. The field is left-justified 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:

<table>
<thead>
<tr>
<th>Literal Value</th>
<th>Binary Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO-FLAG</td>
<td>X'00000000'</td>
<td>No flag is set. The command behaves like a WRITE call.</td>
</tr>
<tr>
<td>MSG-OOB</td>
<td>X'00000001'</td>
<td>Send out-of-band data. (Stream sockets only.)</td>
</tr>
<tr>
<td>MSG-DONTROUTE</td>
<td>X'00000004'</td>
<td>Do not route. Routing is provided by the calling program.</td>
</tr>
</tbody>
</table>

**NBYTE**
A fullword binary number set to the number of bytes to transmit.
BUFF 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 Internet address.

  RESERVED
    Specifies eight-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 Internet 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 may be specified for any address types and scopes. For a link scope IPv6-ADDRESS, SCOPE-ID may 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 Appendix A. Return codes on page 317 for information about ERRNO return codes.

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

    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.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization</td>
<td>Supervisor state or problem state, any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Amode</td>
<td>31-bit or 24-bit.</td>
</tr>
<tr>
<td>Note: Addressability mode (Amode) considerations</td>
<td>See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.</td>
</tr>
<tr>
<td>ASC mode</td>
<td>Primary address space control (ASC) mode.</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for interrupts.</td>
</tr>
<tr>
<td>Locks</td>
<td>Unlocked.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>All parameters must be addressable by the caller and in the primary address space.</td>
</tr>
</tbody>
</table>

Figure 54 shows an example of SETSOCKOPT call instructions.

```plaintext
WORKING-STORAGE SECTION.
  01 SOC-FUNCTION PIC X(16) VALUE IS 'SETSOCKOPT'.
  01 S PIC 9(4) BINARY.
  01 OPTNAME PIC 9(B) BINARY.
  01 OPTVAL PIC 9(16) BINARY.
  01 OPTLEN PIC 9(8) BINARY.
  01 ERNNO PIC 9(8) BINARY.
  01 RETCODE PIC 9(8) BINARY.
  01 OPTVAL PIC 9(16) BINARY.
  01 OPTLEN PIC 9(8) BINARY.
  01 ERNNO PIC 9(8) BINARY.
  01 RETCODE PIC 9(8) BINARY.

PROCEDURE DIVISION
  CALL 'EZASOKET' USING SOC-FUNCTION S OPTNAME
                   OPTVAL OPTLEN ERNNO RETCODE.
```

Figure 54. SETSOCKOPT call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 60.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing SETSOCKOPT. The field is left-justified 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 z/OS 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 Appendix A. Return codes on page 317 for information about ERRNO return codes.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful call.</td>
</tr>
<tr>
<td>-1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

Table 5. OPTNAME options for GETSOCKOPT and SETSOCKOPT

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP_ADD_MEMBERSHIP</td>
<td>Contains the IP_MREQ structure as defined in SYSLIB(BPXYSOCK). The IP_MREQ structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 interface address.</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ.</td>
<td></td>
</tr>
</tbody>
</table>
Table 5. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IP_ADD_SOURCE_MEMBERSHIP</strong></td>
<td>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.</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>IP_BLOCK_SOURCE</strong></td>
<td>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.</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>IP_DROP_MEMBERSHIP</strong></td>
<td>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.</td>
<td>N/A</td>
</tr>
</tbody>
</table>

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.

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.

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.
<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IP_DROP_SOURCE_MEMBERSHIP</strong></td>
<td>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.</td>
<td>N/A</td>
</tr>
<tr>
<td>Use this option to enable an application to exit a source multicast group.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>This is an IPv4-only socket option.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>IP_MULTICAST_IF</strong></td>
<td>A 4-byte binary field containing an IPv4 interface address.</td>
<td>A 4-byte binary field containing an IPv4 interface address.</td>
</tr>
<tr>
<td>Use this option to set or obtain the IPv4 interface address used for sending outbound multicast datagrams from the socket application.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>This is an IPv4-only socket option.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use this option to control or determine 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.</td>
<td>To enable, set to 1.</td>
<td>If enabled, will contain a 1.</td>
</tr>
<tr>
<td>To disable, set to 0.</td>
<td>If disabled, will contain a 0.</td>
<td></td>
</tr>
<tr>
<td>This is an IPv4-only socket option.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>IP_MULTICAST_TTL</strong></td>
<td>A 1-byte binary field containing the value of '00’x to ‘FF’x.</td>
<td>A 1-byte binary field containing the value of '00’x to ‘FF’x.</td>
</tr>
<tr>
<td>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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>This is an IPv4-only socket option.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPTNAME options (input)</td>
<td>SETSOCKOPT, OPTVAL (input)</td>
<td>GETSOCKOPT, OPTVAL (output)</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------</td>
<td>-----------------------------</td>
</tr>
</tbody>
</table>

**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.

**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.

**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.
<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IPV6_MULTICAST_HOPS</strong></td>
<td>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. <strong>Note</strong>: 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.</td>
<td>Contains a 4-byte binary value in the range 0 – 255 indicating the number of multicast hops.</td>
</tr>
<tr>
<td><strong>IPV6_MULTICAST_IF</strong></td>
<td>Contains a 4-byte binary field containing an IPv6 interface index number.</td>
<td>Contains a 4-byte binary field containing an IPv6 interface index number.</td>
</tr>
<tr>
<td><strong>IPV6_MULTICAST_LOOP</strong></td>
<td>A 4-byte binary field. To enable, set to 1. To disable, set to 0.</td>
<td>A 4-byte binary field. If enabled, contains a 1. If disabled, contains a 0.</td>
</tr>
<tr>
<td><strong>IPV6_UNICAST_HOPS</strong></td>
<td>Contains a 4-byte binary value specifying the unicast hops. If not specified, then the default is 1 hop. -1 indicates use stack default. 0 – 255 is the valid hop limit range. <strong>Note</strong>: APF authorized applications are permitted to set a hop limit that exceeds the system configured default. CICS applications cannot execute as APF authorized.</td>
<td>Contains a 4-byte binary value in the range 0 – 255 indicating the number of unicast hops.</td>
</tr>
<tr>
<td><strong>IPV6_V6ONLY</strong></td>
<td>A 4-byte binary field. To enable, set to 1. To disable, set to 0.</td>
<td>A 4-byte binary field. If enabled, contains a 1. If disabled, contains a 0.</td>
</tr>
<tr>
<td>OPTNAME options (input)</td>
<td>SETSOCKOPT, OPTVAL (input)</td>
<td>GETSOCKOPT, OPTVAL (output)</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>MCAST_BLOCK_SOURCE</td>
<td>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.</td>
<td>N/A</td>
</tr>
<tr>
<td>MCAST_JOIN_GROUP</td>
<td>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. See SEZAINST(CBLOCK) for the PL/I example of GROUP_REQ. See SEZAINST(EZACOBOL) for the COBOL example of GROUP-REQ.</td>
<td>N/A</td>
</tr>
<tr>
<td>MCAST_JOIN_SOURCE_GROUP</td>
<td>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.</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Table 5. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MCAST_LEAVE_GROUP</strong></td>
<td>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. See SEZAINST(CBLOCK) for the PL/I example of GROUP_REQ. See SEZAINST(EZACOBOL) for the COBOL example of GROUP-REQ.</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Use this option to enable an application to exit a multicast group or exit all sources for a given multicast groups.

| **MCAST_LEAVE_SOURCE_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 |

Use this option to enable an application to exit a source multicast group.

| **MCAST_UNBLOCK_SOURCE** | 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 |

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.
<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SO_ASCII</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use this option to set or determine the translation to ASCII data option. When SO_ASCII is set, data is translated to ASCII. When SO_ASCII is not set, data is not translated to or from ASCII. <strong>Note:</strong> This is a REXX-only socket option.</td>
<td>To enable, set to ON. To disable, set to OFF. <strong>Note:</strong> The optvalue is returned and is optionally followed by the name of the translation table that is used if translation is applied to the data.</td>
<td>If enabled, contains ON. If disabled, contains OFF. <strong>Note:</strong> The optvalue is returned and is optionally followed by the name of the translation table that is used if translation is applied to the data.</td>
</tr>
<tr>
<td><strong>SO_BROADCAST</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use this option to set or determine whether a program can send broadcast messages over the socket to destinations that can receive datagram messages. The default is disabled. <strong>Note:</strong> This option has no meaning for stream sockets.</td>
<td>A 4-byte binary field. To enable, set to 1 or a positive value. To disable, set to 0.</td>
<td>A 4-byte field. If enabled, contains a 1. If disabled, contains a 0.</td>
</tr>
<tr>
<td><strong>SO_DEBUG</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Use SO_DEBUG to set or determine the status of the debug option. The default is disabled. The debug option controls the recording of debug information. **Notes:**
1. This is a REXX-only socket option.
2. This option has meaning only for stream sockets. | To enable, set to ON. To disable, set to OFF. | If enabled, contains ON. If disabled, contains OFF. |
| **SO_EBCDIC**          |                            |                            |
| Use this option to set or determine the translation to EBCDIC data option. When 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:** This is a REXX-only socket option. | To enable, set to ON. To disable, set to OFF. **Note:** The optvalue is returned and is optionally followed by the name of the translation table that is used if translation is applied to the data. | If enabled, contains ON. If disabled, contains OFF. **Note:** The optvalue is returned and is optionally followed by the name of the translation table that is used if translation is applied to the data. |
| **SO_ERROR**           |                            |                            |
| 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. | N/A | A 4-byte binary field containing the most recent ERRNO for the socket. |
Table 5. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SO_KEEPALIVE</td>
<td>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. 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.</td>
<td>A 4-byte binary field.</td>
</tr>
<tr>
<td></td>
<td>Use of the SO_KEEPALIVE option does not guarantee successful completion because TCP/IP only waits the amount of time specified in OPTVAL for SO_KEEPALIVE.</td>
<td></td>
</tr>
<tr>
<td>SO_LINGER</td>
<td>Use this option to control or determine how TCP/IP processes data that has not been transmitted when a CLOSE is issued for the socket. The default is disabled. Notes: 1. This option has meaning only for stream sockets. 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. 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. 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 only waits the amount of time specified in OPTVAL for SO_LINGER.</td>
<td>Contains an 8-byte field containing two 4-byte binary fields. Assembler coding: ONOFF DS F LINGER DS F COBOL coding: 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.</td>
</tr>
</tbody>
</table>
Table 5. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SO_OOBINLINE</td>
<td>A 4-byte binary field.</td>
<td>A 4-byte binary field.</td>
</tr>
<tr>
<td></td>
<td>To enable, set to 1 or a positive value.</td>
<td>If enabled, contains a 1.</td>
</tr>
<tr>
<td></td>
<td>To disable, set to 0.</td>
<td>If disabled, contains a 0.</td>
</tr>
</tbody>
</table>

Use this option to control or determine whether out-of-band data is received. Note: This option has meaning only for stream sockets.

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.

SO_RCVBUF

Use this option to control or determine the size of the data portion of the TCP/IP receive buffer.

The size of the data portion of the receive buffer is protocol-specific, based on the following values prior to any SETSOCKOPT call:

- 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 65 535 for a raw socket

A 4-byte binary field.

To enable, set to a positive value specifying the size of the data portion of the TCP/IP receive buffer.

To disable, set to a 0.

A 4-byte binary field.

If enabled, contains a positive value indicating the size of the data portion of the TCP/IP receive buffer.

If disabled, contains a 0.
Table 5. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SO_RCVTIMEO</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use this option to control or determine the maximum length of time that a receive-type function can wait before it completes. 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 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 following receive-type functions are supported: • READ • READV • RECV • RECEVFROM • RECEVMSG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>This option requires a TIMEVAL structure, which is defined in SYS1.MACLIB(BPXYRLIM) macro. The TIMEVAL structure contains the number of seconds and microseconds specified as fullword binary numbers. The seconds can be a value in the range 0 - 2,678,400 (equal to 31 days), and the microseconds can be a value in the range 0 - 1,000,000 (equal to 1 second). Although TIMEVAL value can be specified using microsecond granularity, the internal TCP/IP timers that are used to implement this function have a granularity of approximately 100 milliseconds.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>This option stores a TIMEVAL structure that is defined in the SYS1.MACLIB(BPXYRLIM) macro. The TIMEVAL structure contains the number of seconds and microseconds, which are specified as fullword binary numbers. The number of seconds value that is returned is in the range 0 - 2,678,400 (equal to 31 days). The number of microseconds value that is returned is in the range 0 - 1,000,000.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 5. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SO_REUSEADDR</strong></td>
<td>A 4-byte binary field.</td>
<td>A 4-byte binary field.</td>
</tr>
<tr>
<td></td>
<td>To enable, set to 1 or a positive value.</td>
<td>If enabled, contains a 1.</td>
</tr>
<tr>
<td></td>
<td>To disable, set to 0.</td>
<td>If disabled, contains a 0.</td>
</tr>
</tbody>
</table>

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.

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.

| **SO_SNDBUF**           | A 4-byte binary field.      | A 4-byte binary field.      |
|                         | 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. |
|                         | To disable, set to 0.        | If disabled, contains a 0.  |

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:

- The TCPSNDBufsize keyword on the TCPCONFIG statement in the PROFILE.TCPIP data set for a TCP socket
- The UDPSENDBufsize keyword on the UDPCONFIG statement in the PROFILE.TCPIP data set for a UDP socket
- The default of 65 535 for a raw socket
Table 5. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SO_SNDTIMEO</td>
<td>This option requires a TIMEVAL structure, which is defined in the SYS1.MACLIB(BPXYRLIM) macro. The TIMEVAL structure contains the number of seconds and microseconds specified as fullword binary numbers. The seconds value is in the range 0 - 2 678 400 (equal to 31 days), and the microseconds value is in the range 0 - 1 000 000 (equal to 1 second). Although the TIMEVAL value can be specified using microsecond granularity, the internal TCP/IP timers that are used to implement this function have a granularity of approximately 100 milliseconds.</td>
<td>This option stores a TIMEVAL structure that is defined in SYS1.MACLIB(BPXYRLIM). The TIMEVAL structure contains the number of seconds and microseconds, which are specified as fullword binary numbers. The number of seconds value that is returned is in the range 0 - 2 678 400 (equal to 31 days). The microseconds value that is returned is in the range 0 - 1 000 000.</td>
</tr>
<tr>
<td>SO_TYPE</td>
<td>N/A</td>
<td>A 4-byte binary field indicating the socket type: X’1’ indicates SOCK_STREAM. X’2’ indicates SOCK_DGRAM. X’3’ indicates SOCK_RAW.</td>
</tr>
<tr>
<td>TCP_KEEPALIVE</td>
<td>A 4-byte binary field.</td>
<td>A 4-byte binary field.</td>
</tr>
</tbody>
</table>

Use this option to control or determine the maximum length of time that a send-type function can remain blocked before it completes.

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.

For a SETSOCKOPT, the following send-type functions are supported:
- SEND
- SENDMSG
- SENDTO
- WRITE
- WRITEV

Use this option to return the socket type.

Use this option to set or determine whether a socket-specific timeout value (in seconds) is to be used in place of a configuration-specific value whenever keep alive timing is active for that socket.

When activated, the socket-specific timer value remains in effect until respecified by SETSOCKOPT or until the socket is closed. See the z/OS Communications Server: IP Programmer’s Guide and Reference for more information about the socket option parameters.
### Table 5. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP_NODELAY</td>
<td>A 4-byte binary field.</td>
<td>A 4-byte binary field.</td>
</tr>
<tr>
<td></td>
<td>To enable, set to a 0.</td>
<td>If enabled, contains a 0.</td>
</tr>
<tr>
<td></td>
<td>To disable, set to a 1 or nonzero.</td>
<td>If disabled, contains a 1.</td>
</tr>
</tbody>
</table>

**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.
```

### 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](https://www.ibm.com/support/docview.wss?uid=swg27023653) to determine the effects of this operation on the outstanding socket calls.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization</td>
<td>Supervisor state or problem state, any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>PASN = HASN.</td>
</tr>
</tbody>
</table>
Amode: 31-bit or 24-bit.

Note: See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.

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 shows an example of SHUTDOWN call instructions.

WORKING-STORAGE SECTION.
  01 SOC-FUNCTION PIC X(16) VALUE IS 'SHUTDOWN'.
  01 S PIC 9(4) BINARY.
  01 HOW PIC 9(8) BINARY.
    88 END-FROM VALUE 0.
    88 END-TO VALUE 1.
    88 END-BOTH VALUE 2.
  01 ERRNO PIC 9(8) BINARY.
  01 RETCODE PIC S9(8) BINARY.

PROCEDURE DIVISION.
  CALL 'EZASOKET' USING SOC-FUNCTION S HOW ERRNO RETCODE.

Figure 55. SHUTDOWN call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 60.

Parameter values set by the application

SOC-FUNCTION
A 16-byte character field containing SHUTDOWN. The field is left-justified 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(END-FROM) Ends further receive operations.</td>
</tr>
<tr>
<td>1</td>
<td>(END-TO)  Ends further send operations.</td>
</tr>
<tr>
<td>2</td>
<td>(END-BOTH)  Ends further send and receive operations.</td>
</tr>
</tbody>
</table>

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 317 for information about ERRNO return codes.

RETCODE
A fullword binary field that returns one of the following:
### 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.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Requirement Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization:</td>
<td>Supervisor state or problem state, any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Amode:</td>
<td>31-bit or 24-bit.</td>
</tr>
</tbody>
</table>

**Note:** See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Requirement Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASC mode:</td>
<td>Primary address space control (ASC) mode.</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>Unlocked.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>All parameters must be addressable by the caller and in the primary address space.</td>
</tr>
</tbody>
</table>

Figure 56 shows an example of SOCKET call instructions.

```assembly
WORKING-STORAGE SECTION.
  01 SOC-FUNCTION PIC X(16) VALUE IS 'SOCKET'.
  * AF_INET
    01 AF PIC 9(8) COMP VALUE 2.
  * AF_INET6
    01 AF PIC 9(8) COMP VALUE 19.
    01 SOCTYPE PIC 9(8) BINARY.
      88 STREAM VALUE 1.
      88 DATAGRAM VALUE 2.
      88 RAW VALUE 3.
    01 PROTO PIC 9(8) BINARY.
    01 ERRNO PIC 9(8) BINARY.
    01 RETCODE PIC 9(8) BINARY.

PROCEDURE DIVISION.
  CALL 'EZASOKET' USING SOC-FUNCTION AF SOCTYPE PROTO ERRNO RETCODE.
```

**Figure 56. SOCKET call instruction example**

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 60.

### Parameter values set by the application

**SOC-FUNCTION**

A 16-byte character field containing SOCKET. The field is left-justified 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Stream sockets provide sequenced, two-way byte streams that are reliable and connection-oriented. They support a mechanism for out-of-band data.</td>
</tr>
<tr>
<td>2</td>
<td>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.</td>
</tr>
<tr>
<td>3</td>
<td>Raw sockets provide the interface to internal protocols (such as IP and ICMP).</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Protocol name</th>
<th>Numeric value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPPROTO_HOPOPT</td>
<td>0</td>
</tr>
<tr>
<td>IPPROTO_TCP</td>
<td>6</td>
</tr>
<tr>
<td>IPPROTO_UDP</td>
<td>17</td>
</tr>
<tr>
<td>IPPROTO_IPV6</td>
<td>41</td>
</tr>
<tr>
<td>IPPROTO_ROUTING</td>
<td>43</td>
</tr>
<tr>
<td>IPPROTO_FRAGMENT</td>
<td>44</td>
</tr>
<tr>
<td>IPPROTO_ESP</td>
<td>50</td>
</tr>
<tr>
<td>IPPROTO_AH</td>
<td>51</td>
</tr>
<tr>
<td>IPPROTO_NONE</td>
<td>59</td>
</tr>
<tr>
<td>IPPROTO_DSTOPTS</td>
<td>60</td>
</tr>
</tbody>
</table>

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 317 for information about ERRNO return codes.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 0 or = 0</td>
<td>Contains the new socket descriptor.</td>
</tr>
<tr>
<td>-1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

TAKE_SOCKET
The TAKE_SOCKET 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 “GIVESOCKET” on page 115 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.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Requirement Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization:</td>
<td>Supervisor state or problem state, any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Amode:</td>
<td>31-bit or 24-bit.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary address space control (ASC) mode.</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>Unlocked.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>All parameters must be addressable by the caller and in the primary address space.</td>
</tr>
</tbody>
</table>

**Figure 57** shows an example of TAKESOCKET call instructions.

```assembly
WORKING-STORAGE SECTION.
  01 SOC-FUNCTION PIC X(16) VALUE IS 'TAKESOCKET'.
  01 SOCRECV 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.

PROCEDURE DIVISION.
  CALL 'EZASOKET' USING SOC-FUNCTION SOCRECV CLIENT ERRNO RETCODE.

**Figure 57. TAKESOCKET call instruction example**
```

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 60.

**Parameter values set by the application**

**SOC-FUNCTION**

A 16-byte character field containing TAKESOCKET. The field is left-justified 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 only acquire 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 317 for information about ERRNO return codes.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>≥ 0</td>
<td>Contains the new socket descriptor.</td>
</tr>
<tr>
<td>−1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

TERMAPI
This call terminates the session created by INITAPI.

The following requirements apply to this call:

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 “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.
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 58 shows an example of TERMAPI call instructions.

WORKING-STORAGE SECTION.
  01 SOC-FUNCTION  PIC X(16)  VALUE IS 'TERMAPI'.

PROCEDURE DIVISION.
  CALL 'EZASOKET' USING SOC-FUNCTION.

Figure 58. TERMAPI call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 60.

Parameter values set by the application

SOC-FUNCTION
  A 16-byte character field containing TERMAPI. The field is left-justified 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 wishes 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 “EZACIC04” on page 192 for a subroutine that will translate EBCDIC output data to ASCII.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization:</td>
<td>Supervisor state or problem state, any PSW key.</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Amode:</td>
<td>31-bit or 24-bit.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> See “Addressability mode (Amode) considerations” under “CALL instruction API environmental restrictions and programming requirements” on page 57.</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary address space control (ASC) mode.</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for interrupts.</td>
</tr>
<tr>
<td>Locks:</td>
<td>Unlocked.</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>All parameters must be addressable by the caller and in the primary address space.</td>
</tr>
</tbody>
</table>

Figure 59 on page 189 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 assembler language declarations, see "Converting parameter descriptions" on page 60.

Parameter values set by the application

SOC-FUNCTION
  A 16-byte character field containing WRITE. The field is left-justified 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 Appendix A, Return codes on page 317 for information about ERRNO return codes.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>≥0</td>
<td>A successful call. A return code greater than 0 indicates the number of bytes of data written.</td>
</tr>
<tr>
<td>−1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

WRITEV

The WRITEV function writes data on a socket from a set of buffers.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Authorization:</th>
<th>Supervisor state or problem state, any PSW key.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task.</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN.</td>
</tr>
<tr>
<td>Amode:</td>
<td>31-bit or 24-bit.</td>
</tr>
<tr>
<td>Note:</td>
<td>See &quot;Addressability mode (Amode) considerations&quot; under &quot;CALL instruction API environmental restrictions and programming requirements&quot; on page 57.</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary address space control (ASC) mode.</td>
</tr>
</tbody>
</table>
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 shows an example of WRITEV call instructions.

```
WORKING-STORAGE SECTION.
  01 SOC-FUNCTION PIC X(16) VALUE 'WRITEV'.
  01 S PIC 9(4) BINARY.
  01 IOVCNT PIC 9(8) BINARY.
  01 IOV.
    03 BUFFER-ENTRY OCCURS N TIMES.
      05 BUFFER-POINTER USAGE IS POINTER.
      05 RESERVED PIC X(4).
      05 BUFFER-LENGTH PIC 9(8) USAGE IS BINARY.
  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 assembler language declarations, see “Converting parameter descriptions” on page 60.

**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.

**IOV** 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 Appendix A. Return codes on page 317 for information about ERRNO return codes.

RETCODE
A fullword binary field.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0</td>
<td>Check ERRNO for an error code.</td>
</tr>
<tr>
<td>0</td>
<td>Connection partner has closed connection.</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Number of bytes sent.</td>
</tr>
</tbody>
</table>

Using data translation programs for socket call interface

In addition to the socket calls, you can use the following utility programs to translate data:

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 z/OS Communications Server: IP Configuration Reference.
- EZACIC05 translates ASCII data to EBCDIC data using the translation table documented in the z/OS 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 68 on page 206.
- EZACIC15 provides an alternative to EZACIC05 and translates ASCII data to EBCDIC data using the translation table documented in Figure 70 on page 208.

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, since bit strings are difficult to decode in COBOL, TCP/IP includes the following:

- 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. Figure 61 shows how EZACIC04 translates a byte of EBCDIC data.

<table>
<thead>
<tr>
<th>ASCII output by EZACIC04</th>
<th>second hex digit of byte of EBCDIC data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>1A</td>
</tr>
<tr>
<td>3</td>
<td>1A</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>26</td>
</tr>
<tr>
<td>6</td>
<td>2D</td>
</tr>
<tr>
<td>7</td>
<td>07</td>
</tr>
<tr>
<td>8</td>
<td>8C</td>
</tr>
<tr>
<td>9</td>
<td>7A</td>
</tr>
<tr>
<td>A</td>
<td>8B</td>
</tr>
<tr>
<td>B</td>
<td>9C</td>
</tr>
<tr>
<td>C</td>
<td>AC</td>
</tr>
<tr>
<td>D</td>
<td>BE</td>
</tr>
<tr>
<td>E</td>
<td>DE</td>
</tr>
</tbody>
</table>

Figure 61. EZACIC04 EBCDIC-to-ASCII table

Figure 62 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.

Figure 62. EZACIC04 call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 60.

OUT-BUFFER

A buffer that contains the following:
- When called, EBCDIC data
- Upon return, ASCII data
LENGTH
    Specifies the length of the data to be translated.
EZACIC05

The EZACIC05 program is used to translate ASCII data to EBCDIC data. EBCDIC data is required by COBOL, PL/I, and assembler language programs. Figure 63 shows how EZACIC05 translates a byte of ASCII data.

```
WORKING-STORAGE SECTION.
   01 IN-BUFFER    PIC X(length of output).
   01 LENGTH      PIC 9(8) BINARY VALUE.

PROCEDURE DIVISION.
   CALL 'EZACIC05' USING IN-BUFFER LENGTH.
```

Figure 63. EZACIC05 ASCII-to-EBCDIC table

Figure 64 shows an example of EZACIC05 call instructions.

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 60.

IN-BUFFER

A buffer that contains the following:

- When called, ASCII data
- Upon return, EBCDIC data
LENGTH
   Specifies the length of the data to be translated.
EZACICO6

The SELECT 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 assembler language program EZACICO6 to translate them to character strings to be used with the SELECT call.

Figure 65 shows an example of EZACICO6 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.
    05 CTOB PIC X(4) VALUE 'CTOB'.
    05 BTOC PIC X(4) VALUE 'BTOC'.
  01 CHAR-MASK-LENGTH PIC 9(8) COMP VALUE nn.

PROCEDURE CALL (to convert from character to binary)
  CALL 'EZACICO6' USING CTOB
    BIT-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. EZACICO6 call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 60.

TOKEN
  Specifies a 16-character identifier. This identifier is required and it must be the first parameter in the list.

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 right-most bit in the first fullword represents socket 0 and the left-most bit represents
socket 31. The right-most bit in the second fullword represents socket 32
and the left-most 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful call.</td>
</tr>
<tr>
<td>-1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

**Examples:** If you want to use the SELECT call to test sockets 0, 5, and 32, and
you are using a character array to represent the sockets, you must set the
appropriate characters in the character array to 1. In this example, index positions
1, 6 and 33 in the character array are set to 1. Then you can call EZACIC06 with
the COMMAND parameter set to CTOB. When EZACIC06 returns, the first
fullword of BIT-MASK contains B'00000000000000000000000000100001' to indicate
that sockets 0 and 5 will be checked. The second word of BIT-MASK contains
B'00000000000000000000000000000001' to indicate that socket 32 will be checked.
These instructions process the bit string shown in the following example:

```
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.
TEST-socket-EXIT.
EXIT.
```
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 Internet addresses.

TCP/IP uses indirect addressing to connect the variable number of alias names and Internet addresses in the HOSTENT structure that are returned by the GETHOSTBYADDR AND GETHOSTBYNAME calls.

If you are coding in PL/I or assembler 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 Internet 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:
   • 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 Internet address type, always 2 for AF_INET
   • The host Internet address length, always 4 for AF_INET
   • The number of host Internet addresses for this host
   • The host Internet address sequence number
   • The host Internet address
3. If the GETHOSTBYADDR or GETHOSTBYNAME call returns more than one alias name or host Internet address, the application program should repeat the call to EZACIC08 until all alias names and host Internet addresses have been retrieved.

[Figure 66 on page 199] shows an example of EZACIC08 call instructions.
Parameter values set by the application

**HOSTENT-ADDR**  
This fullword binary field must contain the address of the HOSTENT structure (as returned by the GETHOSTBYxxxx 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 Internet 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 Internet addresses returned by this instance of the call.

**HOSTADDR-SEQ**
This halfword binary field contains the sequence number of the host Internet address currently found in HOSTADDR-VALUE.

**HOSTADDR-VALUE**
This fullword binary field contains a host Internet address.

**RETURN-CODE**
This fullword binary field contains the EZACIC08 return code:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion.</td>
</tr>
<tr>
<td>-1</td>
<td>HOSTENT address is not valid.</td>
</tr>
<tr>
<td>-2</td>
<td>A value of HOSTALIAS-SEQ is not valid.</td>
</tr>
<tr>
<td>-3</td>
<td>A value of HOSTADDR-SEQ is not valid.</td>
</tr>
</tbody>
</table>
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 assembler 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 EZACIC09 and passes it the address of the next address information structure as referenced by the NEXT argument. EZACIC09 processes the structure and returns the following:
   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 202 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-canonicalnameok pic 9(8) binary value 2.
  02 ai-numerichost pic 9(8) binary value 4.
  02 ai-numericerv pic 9(8) binary value 8.
  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.

Figure 67. EZACIC09 call instruction example (Part 1 of 3)
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.

Figure 67. EZACIC09 call instruction example (Part 2 of 3)
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 socket-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 67. EZACIC09 call instruction example (Part 3 of 3)

For equivalent PL/I and assembler language declarations, see "Converting
parameter descriptions" on page 60.

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful call.</td>
</tr>
<tr>
<td>−1</td>
<td>Invalid RES address.</td>
</tr>
</tbody>
</table>
The EZACIC14 program is an alternative to EZACIC04, which translates EBCDIC data to ASCII data. Figure 68 shows how EZACIC14 translates a byte of EBCDIC data.

<table>
<thead>
<tr>
<th>ASCII output by EZACIC14</th>
<th>second hex digit of byte of EBCDIC data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>86</td>
</tr>
<tr>
<td>3</td>
<td>90</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>26</td>
</tr>
<tr>
<td>6</td>
<td>7F</td>
</tr>
<tr>
<td>7</td>
<td>8D</td>
</tr>
<tr>
<td>8</td>
<td>B0</td>
</tr>
<tr>
<td>9</td>
<td>B5</td>
</tr>
<tr>
<td>A</td>
<td>B4</td>
</tr>
<tr>
<td>B</td>
<td>AC</td>
</tr>
<tr>
<td>C</td>
<td>7B</td>
</tr>
<tr>
<td>D</td>
<td>7D</td>
</tr>
<tr>
<td>E</td>
<td>5C</td>
</tr>
<tr>
<td>F</td>
<td>30</td>
</tr>
</tbody>
</table>

Figure 68. EZACIC14 EBCDIC-to-ASCII table

Figure 69 shows an example of EZACIC14 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.

Figure 69. EZACIC14 call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 60.

OUT-BUFFER
A buffer that contains the following:
- When called, EBCDIC data
- Upon return, ASCII data
LENGTH
   Specifies the length of the data to be translated.
EZACIC15

The EZACIC15 program is an alternative to EZACIC05, which translates ASCII data to EBCDIC data. Figure 70 shows how EZACIC15 translates a byte of ASCII data.

<table>
<thead>
<tr>
<th>EBCDIC output by EZACIC15</th>
<th>second hex digit of byte of ASCII data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
</tr>
</tbody>
</table>

Figure 70. EZACIC15 ASCII-to-EBCDIC table

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.

Figure 71. EZACIC15 call instruction example

OUT-BUFFER
A buffer that contains the following:
- When called, ASCII data
- Upon return, EBCDIC data
LENGTH
  Specifies the length of the data to be translated.
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 dataset:

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EZASOKPS</td>
<td>PL/I call interface sample IPv4 server program</td>
</tr>
<tr>
<td>EZASOKPC</td>
<td>PL/I call interface sample IPv4 client program</td>
</tr>
<tr>
<td>EZASO6PS</td>
<td>PL/I call interface sample IPv6 server program</td>
</tr>
<tr>
<td>EZASO6PC</td>
<td>PL/I call interface sample IPv6 client program</td>
</tr>
<tr>
<td>CBLOCK</td>
<td>PL/I common variables</td>
</tr>
<tr>
<td>EZACOBOL</td>
<td>COBOL common variables</td>
</tr>
<tr>
<td>EZASO6CS</td>
<td>COBOL call interface sample IPv6 server program</td>
</tr>
<tr>
<td>EZASO6CC</td>
<td>COBOL call interface sample IPv6 client program</td>
</tr>
</tbody>
</table>

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
EZASOKPS: PROC OPTIONS(MAIN);

/* INCLUDE CBLOCK - common variables */
%include CBLOCK;

ID.TCPNAME = 'TCPIP';  /* Set TCP to use */
ID.ADSNAME = 'EZASOKPS';  /* and address space name */
open file(driver);

EZASOKPS: PROC OPTIONS(MAIN);

/* 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 */

Figure 72. EZASOKPS PL/1 sample server program for IPv4 (Part 1 of 4)
call ezasoket(SOCKET, AF_INET, TYPE_STREAM, PROTO, ERRNO, RETCODE);
if retcode < 0 then do;
  msg = blank; /* clear field */
  msg = 'FAIL: socket, stream, internet' || errno;
  write file(driver) from (msg);
  goto getout;
end;
else sock_stream = retcode;

/* Execute BIND */
name_id.port = 8888;
name_id.address = '01234567'BX; /* internet address */
call ezasoket(BIND, SOCK_STREAM, NAME_ID, ERRNO, RETCODE);
if retcode < 0 then do;
  msg = blank; /* clear field */
  msg = 'FAIL: bind' || errno;
  write file(driver) from (msg);
  goto getout;
end;

/* Execute GETSOCKNAME */
name_id.port = 8888;
name_id.address = '01234567'BX; /* internet address */
call ezasoket(GETSOCKNAME, SOCK_STREAM, NAME_ID, ERRNO, RETCODE);
msg = blank; /* clear field */
if retcode < 0 then do;
  msg = 'FAIL: getsockname, stream, internet' || errno;
  write file(driver) from (msg);
end;
else do;
  msg = 'getsockname = ' || name_id.address;
  write file(driver) from (msg);
end;

/* Execute LISTEN */

Figure 72. EZASOKPS PL/1 sample server program for IPv4 (Part 2 of 4)
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                                                  */
/*                                                                  */
/*********************************************************************/

name_id.port = 8888;
name_id.address = '01234567'BX; /* internet address */
call ezasoket(ACCEPT, SOCK_STREAM,
    NAME_ID, ERRNO, RETCODE);
msg = blank; /* clear field */
if retcode < 0 then do;
    msg = 'FAIL: accept' || errno;
    write file(driver) from (msg);
end;
else do;
    accpsock = retcode;
    msg = 'accept socket=' | | accpsock;
    write file(driver) from (msg);
end;

/*********************************************************************/
/*                                                                  */
/* Execute READ                                                   */
/*                                                                  */
/*********************************************************************/

nbyte = length(bufin);
call ezasoket(READ, ACCPSOCK,
    NBYTE, BUFIN, ERRNO, RETCODE);
msg = blank; /* clear field */
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);
    bufout = bufin;
    nbyte = retcode;
end;

/*********************************************************************/
/*                                                                  */
/*                                                                  */
/*********************************************************************/

Figure 72. EZASOKPS PL/1 sample server program for IPv4 (Part 3 of 4)
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
EZASOKPC: PROC OPTIONS(MAIN);
/* INCLUDE CBLOCK - common variables */
%include CBLOCK;

ID.TCPNAME = 'TCPIP'; /* Set TCP to use */
ID.ADSNAME = 'EZASOKPC'; /* and address space name */
open file(driver);

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;

call ezasoket(SOCKET, AF_INET, TYPE_STREAM, PROTO,
    ERRNO, RETCODE);
if retcode < 0 then do;
    msg = blank; /* clear field */
    msg = 'FAIL: socket, stream, internet' || errno;
    write file(driver) from (msg);
end;

Figure 73. EZASOKPC PL/1 sample client program for IPv4 (Part 1 of 3)
goto getout;
end;
sock_stream = retcode;  /* save socket descriptor */

/**************************************************************************/
/* Execute CONNECT */
/**************************************************************************/

name_id.port = 8888;
name_id.address = '01234567'BX;  /* internet address */
call ezasoket(CONNECT, SOCK_STREAM, NAME_ID, ERRNO, RETCODE);
if retcode < 0 then do;
  msg = blank;  /* clear field */
  msg = 'FAIL: connect, stream, internet' || errno;
  write file(driver) from (msg);
  goto getout;
end;

/**************************************************************************/
/* Execute GETPEERNAME */
/**************************************************************************/

call ezasoket(GETPEERNAME, SOCK_STREAM, NAME_ID, ERRNO, RETCODE);
msg = blank;  /* clear field */
if retcode < 0 then do;
  msg = 'FAIL: getpeername' || errno;
  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);
msg = blank;  /* clear field */
if retcode < 0 then do;
  msg = 'FAIL: write' || errno;
  write file(driver) from (msg);
end;
else do;
  msg = 'write = ' || bufout;

Figure 73. EZASOKPC PL/1 sample client program for IPv4 (Part 2 of 3)
Sample code for IPv6 server program

The EZAS06PS 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

write file(driver) from (msg);
end;

*******************************************************************************/
/*
/* Execute READ */
/*
/*******************************************************************************/

nbyte = length(bufin);
call ezasoket(READ, SOCK_STREAM, NBYTE, BUFIN, ERRNO, RETCODE);

msg = blank; /* clear field */
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:
h ow = 2;
call ezasoket(SHUTDOWN, SOCK_STREAM, HOW, ERRNO, RETCODE);
if retcode < 0 then do;
    msg = blank; /* clear field */
    msg = 'FAIL: shutdown' || errno;
    write file(driver) from (msg);
end;

*******************************************************************************/
/*
/* Execute TERMAPI */
/*
*******************************************************************************/

call ezasoket(TERMAPI);

close file(driver);
end ezasokpc;

Figure 73. EZASOKPC PL/1 sample client program for IPv4 (Part 3 of 3)
- INITAPI
- LISTEN
- NTOP
- PTON
- READ
- SOCKET
- TERMAPI
- WRITE
EZASO6PS: PROC OPTIONS(MAIN);
/* INCLUDE CBLOCK - common variables */
%include CBLOCK;
ID.TCPNAME = 'TCPCS'; /* Set TCP to use */
ID.ADSNAME = 'EZASO6PS'; /* 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, MAXSOC, ID, SUBTASK, MAXSNO, ERRNO, RETCODE);
if retcode < 0 then do;
   msg = 'FAIL: socket' || errno;
   write file(driver) from (msg);
   goto getout;
end;
/* Figure 74. EZASO6PS PL/1 sample server program for IPv6 (Part 1 of 6) */
call ezasoket(SOCKET, AF_INET6, TYPE_STREAM, PROTO,
ERRNO, RETCODE);
if retcode < 0 then do;
    msg = blank;  /* clear field */
    msg = 'FAIL: socket, stream, internet' || errno;
    write file(driver) from (msg);
    goto getout;
end;
else sock_stream = retcode;
/*********************************************************************/
/**/ /* Execute PTON */
/**/ /* */
/*********************************************************************/
PRESENTABLE_ADDR = IPV6_LOOPBACK;  /* Set IP address to use */
PRESENTABLE_ADDR_LEN = LENGTH(PRESENTABLE_ADDR) ;  /* and its length */
call ezasoket(PTON, AF_INET6, PRESENTABLE_ADDR,
PRESENTABLE_ADDR_LEN, NUMERIC_ADDR,
ERRNO, RETCODE);
if retcode < 0 then do;
    msg = blank;  /* clear field */
    msg = 'FAIL: pton' || errno;
    write file(driver) from (msg);
    goto getout;
end;
name6_id.address = NUMERIC_ADDR;  /* IPV6 internet address */
/*********************************************************************/
/**/ /* Execute GETHOSTNAME */
/**/ /* */
/*********************************************************************/
call ezasoket(GETHOSTNAME, HOSTNAME_LEN, HOSTNAME,
ERRNO, RETCODE);
msg = blank;  /* clear field */
if retcode < 0 then do;
    msg = 'FAIL: gethostname' || errno;
    write file(driver) from (msg);
    goto getout;
end;
else do;
    msg = 'gethostname='| | HOSTNAME;
    write file(driver) from (msg);
    GAI_NODE = HOSTNAME;  /* Set host name for getaddrinfo to use */
end;
/*********************************************************************/
/**/ /* Execute GETADDRINFO */
/**/ /* */
/*********************************************************************/
GAI_SERVLEN = 0;  /* set service length */
GAI_HINTS.FLAGS = ai_CANONNAMEOK;  /* Request canonical name */
HINTS = ADDR(GAI_HINTS);  /* Set results pointer */
Figure 74. EZAASO6PS PL/1 sample server program for IPv6 (Part 2 of 6)
call ezasoket(GETADDRINFO,
   GAI_NODE, GAI_NODELEN,
   GAI_SERVICE, GAI_SERVLEN,
   HINTS, RES,
   CANONNAME_LEN,
   ERRNO, RETCODE);

msg = blank; /* clear field */
if retcode < 0 then do;
   msg = 'FAIL: getaddrinfo' || errno;
   write file(driver) from (msg);
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; /* clear field */
   if retcode < 0 then do;
      msg = 'FAIL: freeaddrinfo' || errno;
      write file(driver) from (msg);
   end;
end; /* end from getaddrinfo */
   /*****************************************************************************/
   /* */
   /* Execute BIND */
   /* */
   /*****************************************************************************/

name6_id.port = 8888;
call ezasoket(BIND, SOCK_STREAM, NAME6_ID,
   ERRNO, RETCODE);
   if retcode < 0 then do;

Figure 74. EZASO6PS PL/1 sample server program for IPv6 (Part 3 of 6)
msg = 'FAIL: bind' || errno;
write file(driver) from (msg);
goto getout;
end;

/***************************************************************
/*
/* Execute GETSOCKNAME
/*
/***************************************************************
call ezasoket(GETSOCKNAME, SOCK_STREAM,
    NAME6_ID, ERRNO, RETCODE);
msg = blank; /* clear field */
if retcode < 0 then do;
    msg = 'FAIL: getsockname, stream, internet' || errno;
    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
/*
/***************************************************************
call ezasoket(ACCEPT, SOCK_STREAM,
    NAME6_ID, ERRNO, RETCODE);
msg = blank; /* clear field */
if retcode < 0 then do;
    msg = 'FAIL: accept' || errno;
    write file(driver) from (msg);
end;
else do;
    accpsock = retcode;
    msg = 'accept socket = ' || accpsock;
    write file(driver) from (msg);
end;

Figure 74. EZASO6PS PL/1 sample server program for IPv6 (Part 4 of 6)
call ezasoket(NTOP, AF_INET6, NUMERIC_ADDR,
     PRESENTABLE_ADDR, PRESENTABLE_ADDR_LEN,
     ERRNO, RETCODE);

msg = blank; /* clear field */
if retcode < 0 then do;
msg = 'FAIL: ntop' || errno;
write file(driver) from (msg);
goto getout;
end;
else do;
msg = 'presentable address = ' || PRESENTABLE_ADDR;
write file(driver) from (msg);
end;
/* */

nbyte = length(bufin);
call ezasoket(READ, ACCPSOCK,
     NBYTE, BUFIN, ERRNO, RETCODE);

msg = blank; /* clear field */
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);
bufout = bufin;
nbyte = retcode;
end;
/* */

/* */
call ezasoket(WRITE, ACCPSOCK, NBYTE, BUFOUT,
     ERRNO, RETCODE);

msg = blank; /* clear field */
if retcode < 0 then do;
msg = 'FAIL: write' || errno;
write file(driver) from (msg);
end;
else do;
msg = 'write = ' || bufout;

Figure 74. EZASO6PS PL/1 sample server program for IPv6 (Part 5 of 6)
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
EZASO6PC: PROC OPTIONS(MAIN);

/* INCLUDE CBLOCK - common variables */
% include CBLOCK;

ID.TCPNAME = 'TCPCS'; /* Set TCP to use */
ID.ADSNAME = 'EZASO6PS'; /* and address space name */
open file(driver);

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;

call ezasoket(SOCKET, AF_INET6, TYPE_STREAM, PROTO, ERRNO, RETCODE);
if retcode < 0 then do;
    msg = blank; /* clear field */
    msg = 'FAIL: socket, stream, internet' || errno;
    write file(driver) from (msg);
end;

Figure 75. EZASO6PC PL/1 sample client program for IPv6 (Part 1 of 4)
goto getout;
end;
sock_stream = retcode; /* save socket descriptor */

/*********************************************************/
/* Execute PTON */
/*********************************************************/
PRESENTABLE_ADDR = IPV6_LOOPBACK; /* Set the address to use */
PRESENTABLE_ADDR_LEN = LENGTH(PRESENTABLE_ADDR); /* and it's length */
call ezasoket(PTON, AF_INET6, PRESENTABLE_ADDR,
PRESENTABLE_ADDR_LEN, NUMERIC_ADDR,
ERRNO, RETCODE);
msg = blank; /* clear field */
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 internet address */

/*********************************************************/
/* Execute CONNECT */
/*********************************************************/
name6_id.port = 8888;
call ezasoket(CONNECT, SOCK_STREAM, NAME6_ID,
ERRNO, RETCODE);
if retcode < 0 then do;
  msg = blank; /* clear field */
  msg = 'FAIL: connect, stream, internet' || errno;
  write file(driver) from (msg);
  goto getout;
end;

/*********************************************************/
/* Execute GETPEERNAME */
/*********************************************************/
call ezasoket(GETPEERNAME, SOCK_STREAM,
NAME6_ID, ERRNO, RETCODE);
msg = blank; /* clear field */
if retcode < 0 then do;
  msg = 'FAIL: getpeername' || errno;
  write file(driver) from (msg);
  goto getout;
end;

/*********************************************************/
/* Execute GETNAMEINFO */
/*********************************************************/
call ezasoket(GETNAMEINFO, SOCK_STREAM,
NAME6_ID, ERRNO, RETCODE);
msg = blank; /* clear field */
if retcode < 0 then do;
  msg = 'FAIL: getnameinfo' || errno;
  write file(driver) from (msg);
end;

Figure 75. EZASO6PC PL/1 sample client program for IPv6 (Part 2 of 4)
NAMELEN = 28 ; /* Set length of NAME */
GNI_HOST = blank; /* Clear Host name */
GNI_HOSTLEN = LENGTH(GNI_HOST); /* Set Host name length */
GNI_SERVICE = blank; /* Clear Service name */
GNI_SERVLEN = LENGTH(GNI_SERVICE); /* Set Service name length */
GNI_FLAGS = NI_NAMEREQD; /* Set an error if name is not found */
call ezasoket(GETNAMEINFO, NAME6_ID, NAMELEN,
            GNI_HOST, GNI_HOSTLEN,
            GNI_SERVICE, GNI_SERVLEN,
            GNI_FLAGS,
            ERRNO, RETCODE);

msg = blank; /* clear field */
if retcode < 0 then do;
   msg = 'FAIL: getnameinfo' || errno; 
   write file(driver) from (msg);
end;
else do;
   msg = 'getnameinfo host=' || GNI_HOST ; 
   write file(driver) from (msg);
   msg = 'getnameinfo service=' || GNI_SERVICE ; 
   write file(driver) from (msg);
end;

bufout = message;
nbyte = length(message);
call ezasoket(WRITE, SOCK_STREAM, NBYTE, BUFOUT,
            ERRNO, RETCODE);

msg = blank; /* clear field */
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;

nbyte = length(bufin);
call ezasoket(READ, SOCK_STREAM,
            NBYTE, BUFIN, ERRNO, RETCODE);
msg = blank; /* clear field */

Figure 75. EZASO6PC PL/1 sample client program for IPv6 (Part 3 of 4)
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; /* clear field */
    msg = 'FAIL: shutdown' || errno;
    write file(driver) from (msg);
end;

/***************************************************************
/* Execute TERMAPI                                          */
/***************************************************************
call ezasoket(TERMAPI);
close file(driver);
end ezaso6pc;

Figure 75. EZASO6PC PL/I sample client program for IPv6 (Part 4 of 4)

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.
DCL ABS BUILTIN;
DCL ADDR BUILTIN;
DCL ACCEPT CHAR(16) INIT('ACCEPT'); /* temporary ACCEPT socket */
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 */
DCL ai_PASSIVE BIT(32) INIT('00000001'BX); /* flag: getaddrinfo hints */
DCL ai_CANONNAMEOK BIT(32) INIT('00000002'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);

Figure 76. CBLOCK PL/1 common variables (Part 1 of 12)
/* flag: getaddrinfo hints */
DCL ai_ALL BIT(32) INIT('00000020'BX);
/* flag: getaddrinfo hints */
DCL ai_ADDRCONFIG BIT(32) INIT('00000040'BX);
/* flag: getaddrinfo hints */
DCL ai_ALLFLAGMASK BIT(32) INIT('FFFFFF80'BX);
DCL ALIAS CHAR(255); /* alternate NAME */
DCL APITYPE FIXED BIN(15) INIT(2); /* default API type */
DCL BACKLOG FIXED BIN(31); /* max length of pending queue */
DCL BADNAME CHAR(20); /* temporary name */
DCL BIND CHAR(16) INIT('BIND');
DCL BIT BUILTIN;
DCL BITZERO BIT(1); /* bit zero value */
DCL BLANK255 CHAR(255) INIT(' '); /* */
DCL BLANK CHAR(100) INIT(' '); /* */
DCL BUF CHAR(80) INIT(' '); /* macro READ/WRITE buffer */
DCL BUFF CHAR(15) INIT(' '); /* short buffer */
DCL BUFSIZE CHAR(32767) INIT(' '); /* buffer */
DCL CANONNAME_LEN FIXED BIN(31); /* getaddrinfo canonical name length */
DCL CLIENT, /* socket addr of connection peer */
  2 DOMAIN FIXED BIN(31) INIT(2), /* domain of client (AF_INET) */
  2 NAME CHAR(8) INIT(' '), /* addr identifier for client */
  2 TASK CHAR(8) INIT(' '), /* task identifier for client */
  2 RESERVED CHAR(20) INIT(' '); /* reserved */
DCL CLOSE CHAR(16) INIT('CLOSE');
DCL COUNT FIXED BIN(31) INIT(100); /* elements in GRP_IOCTL_TABLE */
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(RECSIZE(100)) RECORD;
DCL ERR FIXED BIN(31); /* error number variable */
DCL ERRNO FIXED BIN(31) INIT(0); /* error number */
DCL ESNDMSK CHAR(4); /* check for pending */
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 EZNTL CHAR(16) INIT('FCNTL');
DCL FIONREAD BIT(32) INIT('400447F'BX); /* flag: #readable bytes */
DCL FLAGS FIXED BIN(31) INIT(0); /* default: no flags */

Figure 76. CBLOCK PL/1 common variables (Part 2 of 12)
/* 1 = OOB, SEND OUT-OF-BAND*/
/* 4 = DON'T ROUTE */
DCL FREEADDRINFO CHAR(16) INIT('FREEADDRINFO');
DCL GAI_NODE CHAR(255) INIT(' '); /* getaddrinfo node */
DCL GAI_NODELEN FIXED BIN(31) INIT(255); /* getaddrinfo node length */
DCL GAI_SERVICE CHAR(32) INIT(' '); /* getaddrinfo service */
DCL GAI_SERVLEN FIXED BIN(31) INIT(32); /* getaddrinfo service */
DCL GAI_HINTS FIXED BIN(31); /* getaddrinfo hints addrinfo */
2 FLAGS FIXED BIN(31) INIT(0), /* hints flags */
2 AF FIXED BIN(31) INIT(0), /* hints family */
2 SOCTYPE FIXED BIN(31) INIT(0), /* hints socket type */
2 PROTO FIXED BIN(31) INIT(0), /* hints protocol */
2 NAMELEN FIXED BIN(31) INIT(0),
1 2 * CHAR(4),
1 2 * CHAR(4),
1 2 CANONNAME FIXED BIN(31) INIT(0),
1 2 * CHAR(4),
1 2 NAME FIXED BIN(31) INIT(0),
1 2 * CHAR(4),
1 2 NEXT FIXED BIN(31) INIT(0);
DCL 1 GAI_ADDRINFO BASED(RES), /* getaddrinfo RES addrinfo */
2 FLAGS FIXED BIN(31),
2 AF FIXED BIN(31),
2 SOCTYPE FIXED BIN(31),
2 PROTO FIXED BIN(31),
2 NAMELEN FIXED BIN(31), /* RES socket address struct length*/
1 2 * CHAR(4),
1 2 * CHAR(4),
1 2 CANONNAME POINTER, /* RES canonical name */
1 2 * CHAR(4),
1 2 NAME POINTER, /* RES socket address structure */
1 2 * CHAR(4),
1 2 NEXT POINTER; /* RES next addrinfo, zero if none.*/
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_ADDRINFO.NAME),
2 NAME FIXED BIN(31) INIT(0),
1 2 * CHAR(4),
1 2 * CHAR(4),
1 2 CANONNAME FIXED BIN(31) INIT(0),
1 2 * CHAR(4),
1 2 NAME FIXED BIN(31) INIT(0),
1 2 * CHAR(4),
1 2 NEXT FIXED BIN(31) INIT(0);

Figure 76. CBLOCK PL/1 common variables (Part 3 of 12)
2 LEN BIT(8),
2 FAMILY BIT(8),
2 PORT BIT(16),
2 FLOWINFO FIXED BIN(31),
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('GETHOSTBYADDR');
DCL GETHOSTBYNAME CHAR(16) INIT('GETHOSTBYNAME');
DCL GETHOSTNAME CHAR(16) INIT('GETHOSTNAME');
DCL GETHOSTID CHAR(16) INIT('GETHOSTID');
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 GNI_FLAGS FIXED BIN(31); /* getnameinfo flags */
DCL GNI_HOST CHAR(255); /* getnameinfo host */
DCL GNI_HOSTLEN FIXED BIN(31); /* getnameinfo host length */
DCL GNI_SERVICE CHAR(32); /* getnameinfo service */
DCL GNI_SERVLLEN FIXED BIN(31); /* getnameinfo service length */
DCL 1 GROUP_FILTER4 BASED, /* Group_Filter for IPv4 */
2 GF4_HEADER, /* Header portion */
3 GF4_INTERFACE FIXED BIN(31), /* Interface index */
3 * CHAR(4), /* Padding */
3 GF4_GROUP, /* Group Multi Address */
4 GF4_SOCK_LEN BIT(8), /* Socket len */
4 GF4_SOCK_FAMILY BIT(8), /* Socket family */
4 GF4_SOCK_SIN_PORT BIT(16), /* Socket port */
4 GF4_SOCK_SIN_ADDR BIT(32), /* Socket address */
3 GF4_RESERVED1 CHAR(8), /* Unused */
3 GF4_FMODE FIXED BIN(31), /* Filter mode */
3 GF4_NMSRC FIXED BIN(31), /* Num of sources */
2 GF4_SLIST CHAR(0); /* Source list */
DCL 1 GF4_SRCENTRY BASED, /* Source Entry */
2 GF4_SRCAADDR, /* Source IP address */
3 GF4_SOCK_LEN BIT(8), /* Socket len */
3 GF4_SOCK_FAMILY BIT(8), /* Socket family */
3 GF4_SOCK_SIN_PORT BIT(16), /* Socket port */
3 GF4_SOCK_SIN_ADDR BIT(32), /* Socket address */
3 GF4_RESERVED1 CHAR(8), /* Unused */
3 * CHAR(112), /* */
DCL 1 GROUP_FILTER6 BASED, /* Group Filter for IPv6 */
2 GF6_HEADER, /* Header portion */
3 GF6_SLIST CHAR(0); /* Source list */
DCL 1 GF6_SRCENTRY BASED, /* Source Entry */
2 GF6_SRCAADDR, /* Source IP address */
3 GF6_SOCK_LEN BIT(8), /* Socket len */
3 GF6_SOCK_FAMILY BIT(8), /* Socket family */
3 GF6_SOCK_SIN_PORT BIT(16), /* Socket port */
3 GF6_SOCK_SIN_ADDR BIT(32), /* Socket address */
3 GF6_RESERVED1 CHAR(8), /* Unused */
3 * CHAR(112), /* */
DCL 1 GROUP_FILTER6 BASED, /* Group Filter for IPv6 */
2 GF6_HEADER, /* Header portion */
3 GF6_INTERFACE FIXED BIN(31), /* Interface index */

Figure 76. CBLOCK PL/1 common variables (Part 4 of 12)
Figure 76. CBLOCK PL/1 common variables (Part 5 of 12)
Figure 76. CBLOCK PL/1 common variables (Part 6 of 12)
DCL ICMP FIXED BIN(31) INIT(2); /* prototype icmp ??? */
DCL 1 ID, /* */
  2 TCPNAME CHAR(8) INIT('TCPIP'), /* remote address space */
  2 ADSNAME CHAR(8) INIT('USER9'); /* local address space */
DCL IDENT POINTER; /* TCP/IP Addr Space */
DCL IFCONF CHAR(255); /* configuration structure */
DCL 1 IF_NAMEINDEX,
  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);
DCL 1 IF_NAMEINDEXENTRY,
  2 IF_NINDEX FIXED BIN(31), /* Interface Index */
  2 IF_NINAME CHAR(16), /* Interface Name, blank padded */
  2 IF_NEXT,
    3 IF_NINAMETERM CHAR(1), /* Null for C for Name len=16 */
    3 IF_RESERVED CHAR(3); /* Reserved */
DCL 1 IFREQ, /* Interface Structure */
  2 IFR_NAME CHAR(16), /* Interface Name, blank padded */
  2 IFR_IFA UNION,
    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), /* Socket Port */
      4 IFR_ADDR_ADDR BIT(32), /* Socket Address */
    4 IFR_ADDR_RSVD CHAR(8), /* Socket Reserved */
    3 IFR_DSTADDR, /* Interface Dest IP Addr */
      4 IFR_DSTADDR_LEN BIT(8), /* Socket Len */
      4 IFR_DSTADDR_FAMILY BIT(8), /* Socket Family */
      4 IFR_DSTADDR_PORT BIT(16), /* Socket Port */
      4 IFR_DSTADDR_ADDR BIT(32), /* Socket Address */
    4 IFR_DSTADDR_RSVD CHAR(8), /* Socket Reserved */
    3 IFR_BROADADDR, /* Interface Broadcast IP Addr */
      4 IFR_BROADADDR_LEN BIT(8), /* Socket Len */
      4 IFR_BROADADDR_FAMILY BIT(8), /* Socket Family */
      4 IFR_BROADADDR_PORT BIT(16), /* Socket Port */
      4 IFR_BROADADDR_ADDR BIT(32), /* Socket Address */
    4 IFR_BROADADDR_RSVD CHAR(8), /* Socket Reserved */
    3 IFR_FLAGS BIT(16), /* Interface Flags */
    3 IFR_METRIC FIXED BIN(31), /* Interface Metric */
    3 IFR_DATA FIXED BIN(31), /* Interface Data */
    3 IFR_MTU FIXED BIN(31); /* Interface MTU */
/* The following constants are for use with the IFR_FLAGS field */
/* in structure IFREQ. */
DCL IFF_UP BIT(16) INIT('0001'BX); /* interface is UP */
DCL IFF_BROADCAST BIT(16) INIT('0002'BX); /* broadcast addr valid */
DCL IFF_DEBUG BIT(16) INIT('0004'BX); /* turn on debugging */
DCL IFF_LOOPBACK 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_NOARP 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 */

Figure 76. CBLOCK PL/1 common variables (Part 7 of 12)
DCL IFF_MULTICAST 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 */
DCL INDEX BUILTIN;
DCL IOCTL CHAR(16) INIT('IOCTL'); /* ioctl command */
DCL IOCTL_CMD FIXED BIN(31); /* ioctl command */
DCL IOCTL_REQARG POINTER; /* send pointer to data area*/
DCL IOCTL_RETARG POINTER; /* return pointer to data area*/
DCL IOCTL_REQ00 FIXED BIN(31); /* command request argument */
DCL IOCTL_REQ04 FIXED BIN(31); /* command request argument */
DCL IOCTL_REQ08 FIXED BIN(31); /* command request argument */
DCL IOCTL_REQ32 CHAR(32) INIT(' '); /* command request argument */
DCL IOCTL_RET00 FIXED BIN(31); /* command return argument */
DCL IOCTL_RET04 FIXED BIN(31); /* command return argument */
DCL INITAPI CHAR(16) INIT('INITAPI'); /* */
DCL IP FIXED BIN(31) INIT(1); /* prototype ip ??? */
DCL 1 IP_MREQ,
 2 IMR_MULTIADDR BIT(32), /* IP multicast addr of group */
 2 IMR_INTERFACE BIT(32); /* 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), /* IP multicast addr of grp */
 2 IMRS_SOURCEADDR BIT(32), /* IP source addr */
 2 IMRS_INTERFACE BIT(32); /* local IP addr of intf */
DCL 1 IP_MSFILTER BASED, /* IP_MsFilter */
 2 IMSF_HEADER, /* Header portion */
 3 IMSF_MULTIADDR BIT(32), /* Multicast address */
 3 IMSF_INTERFACE BIT(32), /* Interface address */
 3 IMSF_FMODE FIXED BIN(31), /* Filter mode */
 3 IMSF_NUMSRC FIXED BIN(31), /* Num of sources */
 2 IMSF_SLIST CHAR(0); /* Source list */
DCL 1 IMSF_SRCENTRY BASED, /* Source Entry */
 2 IMSF_SRCADDR BIT(32); /* Source IP address */
DCL IP_MULTICAST_TTL BIT(32) INIT('00100003'BX);
DCL IP_MULTICAST_LOOP BIT(32) INIT('00100004'BX);
DCL IP_MULTICAST_IF BIT(32) INIT('00100007'BX);
DCL IP_ADD_MEMBERSHIP BIT(32) INIT('00100005'BX);
DCL IP_DROP_MEMBERSHIP BIT(32) INIT('00100006'BX);
DCL IP_BLOCK_SOURCE BIT(32) INIT('0010000A'BX);
DCL IP_ADD_SOURCE_MEMBERSHIP BIT(32) INIT('0010000C'BX);
DCL IP_DROP_SOURCE_MEMBERSHIP BIT(32) INIT('0010000D'BX);
DCL IPRES POINTER; /* EZACIC09 RES addrinfo ptr */
DCL IPV6_JOIN_GROUP BIT(32) INIT('00010005'BX);
DCL IPV6_LEAVE_GROUP BIT(32) INIT('00010006'BX);
DCL IPV6_LOOPBACK CHAR(3) INIT('::1');
DCL IPV6_MULTICAST_HOPS BIT(32) INIT('00010009'BX);
DCL IPV6_MULTICAST_IF BIT(32) INIT('00010007'BX);
DCL IPV6_MULTICAST_LOOP BIT(32) INIT('00010004'BX);

Figure 76. CBLOCK PL/1 common variables (Part 8 of 12)
DCL IPV6_UNICAST_HOPS BIT(32) INIT('00010003'BX);
DCL IPV6_VONLY BIT(32) INIT('0001000A'BX);
DCL J FIXED BIN(15); /* loop index */
DCL K FIXED BIN(15); /* loop index */
DCL LENGTH BUILTIN;
DCL LABL CHAR(9);
DCL LISTEN CHAR(16) INIT('LISTEN'); /* max descriptor assigned */
DCL 1 MAXSOC_INPUT FIXED BIN(31) INIT(0);
DCL 1 MAXSOC_FWD,
2 MAXSOC_IGNORE FIXED BIN(15) INIT(0), /* largest sock # checked */
2 MAXSOC FIXED BIN(15) INIT(255);
DCL MCAST_JOIN_GROUP BIT(32) INIT('00100028'BX);
DCL MCAST_LEAVE_GROUP BIT(32) INIT('00100029'BX);
DCL MCAST_JOIN_SOURCE_GROUP BIT(32) INIT('0010002A'BX);
DCL MCAST_LEAVE_SOURCE_GROUP BIT(32) INIT('0010002B'BX);
DCL MCAST_BLOCK_SOURCE BIT(32) INIT('0010002C'BX);
DCL MCAST_UNBLOCK_SOURCE BIT(32) INIT('0010002D'BX);
DCL MCAST_EXCLUDE BIT(32) INIT('00000001'BX);
DCL MCAST_INCLUDE BIT(32) INIT('00000000'BX);
DCL MCAST_NUMSRC_MAX BIT(32) INIT('00000040'BX);
DCL MESSAGE CHAR(50) INIT('I love my 1 @ Rottweiler!'); /* message */
DCL MSG CHAR(100) INIT(' '); /* message text */
DCL 1 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 */
2 RESERVED CHAR(8); /* reserved */
DCL 1 NAME6_ID, /* socket addr of connection peer */
2 FAMILY FIXED BIN(15) INIT(19), /* NAMELN IGNORED & FAMILY */
2 PORT BIT(16), /* port # */
2 FLOWINFO FIXED BIN(31), /* Flow info */
2 ADDRESS CHAR(16), /* IPv6 internet address */
2 SCOPEID FIXED BIN(31); /* Scope ID */
DCL NAME CHAR(255) VARYING; /* name field, long */
DCL NAMES CHAR(24); /* name field, short */
DCL NAMELEN FIXED BIN(31); /* length of name/alias field */
DCL NBYTE FIXED BIN(31); /* Number of bytes in buffer */
DCL 1 NETCONFHDR, /* Network Configuration Hdr */
2 NCHEYECATCHER CHAR(4) INIT('6NCH'), /* Eye Catcher '6NCH' */
2 NCHIOCTL BIT(32) INIT('0014F600'BX), /* The IOCTL being processed */
/* with this instance of the */
/* NetConfHdr. (RAS item) */
2 NCHBUFFERLENGTH FIXED BIN(31) INIT(3200), /* Buffer Length */
2 NCHBUFFERPR POINTER, /* Buffer Pointer */
2 NCHNUMENTRYRET FIXED BIN(31); /* Number of HomeIF returned via */
/* SIOCGRHOMIF6 or the number of*/
/* GR6RtEntry's returned via */
/* SIOCGR6RTABLE. */
DCL NI_NOFQDN FIXED BIN(31) INIT(1);

Figure 76. CBLOCK PL/1 common variables (Part 9 of 12)
Figure 76. CBLOCK PL/1 common variables (Part 10 of 12)
DCL SAVEFAM FIXED BIN(15);    /* temporary family name     */
DCL SELECB CHAR(4) INIT('1');
DCL SELECT CHAR(16) INIT('SELECT');
DCL SELECTEX CHAR(16) INIT('SELECTEX');
DCL SEND CHAR(16) INIT('SEND');
DCL SENDMSG CHAR(16) INIT('SENDMSG');
DCL SENDTO CHAR(16) INIT('SENDTO');
DCL SETADEYE1 CHAR(8) INIT('SETAPPLD');
DCL SETADVER FIXED BIN(15) INIT(1);
DCL SETADCONTLEN FIXED BIN(15) INIT(48);
DCL SETADBUFLEN FIXED BIN(15) INIT(40);
DCL SETAPPLDATA,
  2 SETAD_EYE1 CHAR(8),
  2 SETAD_VER FIXED BIN(15),
  2 SETAD_LEN FIXED BIN(15),
  2 * CHAR(4),
  2 SETAD_PTR64 ,
  3 SETAD_PTRHW CHAR(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 SHUTDOWN CHAR(16) INIT('SHUTDOWN');
DCL SIOCATMARK BIT(32) INIT('4004A707'BX); /* flag: out-of-band data*/
DCL SIOCGHOMEF6 BIT(32) INIT('C014F608'BX); /* flag: netw int config*/
DCL SIOCGIFADDR BIT(32) INIT('C020A70D'BX); /* flag: network int addr*/
DCL SIOCGIFCONF BIT(32) INIT('C008A714'BX); /* flag: set rout metr*/
DCL SIOCGIFMTU BIT(32) INIT('C020A726'BX); /* flag: get intf mtu */
DCL SIOCGIFNAMEINDEX BIT(32) INIT('4000F603'BX); /* flag: name and indexes */
DCL SIOCGIFNETMASK BIT(32) INIT('C020A715'BX); /* flag: network mask*/
DCL SIOCSIFMETRIC BIT(32) INIT('8020A718'BX); /* flag: set rout metr*/
DCL SIOCSAPPDATA BIT(32) INIT('B018D090'BX); /* Set APPLDATA */
DCL SIOCIPMSFILTER BIT(32) INIT('C000A724'BX); /* flag: get multicast src filter */
DCL SIOCIPMSFILTER BIT(32) INIT('8000A725'BX); /* flag: set multicast src filter */
DCL SIOCIPMSFILTER BIT(32) INIT('C000F610'BX); /* flag: get multicast src filter */
DCL SIOCIPMSFILTER 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 SIOCIPMSFILTER BIT(32) INIT('C038D90B'BX) */

Figure 76. CBLOCK PL/1 common variables (Part 11 of 12)
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.

```cobol
/* flag: ttls */
DCL SOCK FIXED BIN(15); /* socket descriptor */
DCL SOCKET CHAR(16) INIT('SOCKET');
DCL SOCK_DATAGRAM FIXED BIN(15); /* socket descriptor datagram */
DCL SOCK_RAW FIXED BIN(15); /* 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_REUSEADDR FIXED BIN(31) INIT(4); /* toggle, local address reuse*/
DCL SO_SNDBUF FIXED BIN(31) INIT(4097);
DCL SO_SNDTIMEO BIT(32) INIT('00001006'BX);
DCL SO_TYPE FIXED BIN(31) INIT(4104); /* return type of socket */
DCL SO_TIMESTAMP BIT(32) INIT('00001005'BX);
DCL SUBSTR BUILTIN;
DCL SUBTASK CHAR(8) INIT('ANYNAME'); /* task/path identifier */
DCL SYNCKEY CHAR(16) INIT('SYNC');
DCL TAKESOCKET CHAR(16) INIT('TAKESOCKET');
DCL TASK CHAR(16) INIT('TAKESOCKET');
DCL TERMAPI CHAR(16) INIT('TERMAPI'); /* */
DCL TIME BUILTIN;
DCL 1 TIMEOUT,
  2 TIME_SEC FIXED BIN(31), /* value in secs */
  2 TIME_MSEC FIXED BIN(31); /* value in milliseconds */
DCL 1 TIMEVAL,
  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 length connectionless */
DCL TYPE_RAW FIXED BIN(31) INIT(3); /* internal protocol interface */
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('00000008'BX);
DCL TCP_NODELAY BIT(32) INIT('00000001'BX);
```

Figure 76. CBLOCK PL/1 common variables (Part 12 of 12)
**MODULE NAME: EZACOBOL - COBOL COMMON VARIABLES**

**Copyright: Licensed Materials - Property of IBM**

"Restricted Materials of IBM"

5694-A01

Copyright IBM Corp. 2007, 2009

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

Status: CSV1R11

***************************************************************************

**COBOL COMMON VARIABLES**

***************************************************************************

* Socket option values.

  01 IP-ADD-MEMBERSHIP PIC X(4) VALUE '00100005'.
  01 IP-ADD-SOURCE-MEMBERSHIP PIC X(4) VALUE '0010000C'.
  01 IP-BLOCK-SOURCE PIC X(4) VALUE '0010000A'.
  01 IP-DROP-MEMBERSHIP PIC X(4) VALUE '00100006'.
  01 IP-DROP-SOURCE-MEMBERSHIP PIC X(4) VALUE '0010000D'.
  01 IP-MULTICAST-IF PIC X(4) VALUE '00100009'.
  01 IP-MULTICAST-LOOP PIC X(4) VALUE '00100004'.
  01 IP-MULTICAST-TTL PIC X(4) VALUE '00100003'.
  01 IP-UNBLOCK-SOURCE PIC X(4) VALUE '00100008'.
  01 IPV6-JOIN-GROUP PIC X(4) VALUE '00100005'.
  01 IPV6-JOIN-SOURCE-GROUP PIC X(4) VALUE '00100006'.
  01 IPV6-MULTICAST-HOPS PIC X(4) VALUE '00100009'.
  01 IPV6-MULTICAST-IF PIC X(4) VALUE '00100007'.
  01 IPV6-MULTICAST-LOOP PIC X(4) VALUE '00100004'.
  01 IPV6-UNICAST-HOPS PIC X(4) VALUE '00100003'.
  01 IPV6-V6ONLY PIC X(4) VALUE '0010000A'.
  01 MCAST-BLOCK-SOURCE PIC X(4) VALUE '0010002C'.
  01 MCAST-JOIN-GROUP PIC X(4) VALUE '00100028'.
  01 MCAST-JOIN-SOURCE-GROUP PIC X(4) VALUE '0010002A'.
  01 MCAST-LEAVE-GROUP PIC X(4) VALUE '00100029'.
  01 MCAST-LEAVE-SOURCE-GROUP PIC X(4) VALUE '0010002B'.
  01 MCAST-UNBLOCK-SOURCE PIC X(4) VALUE '0010002D'.

Figure 77. EZACOBOL COBOL common variables (Part 1 of 8)
01 SO-RCVTIMEO PIC X(4) VALUE X'00001006'.
01 SO-SNDTIMEO PIC X(4) VALUE X'00001005'.

* IOCTL Commands
* 01 SIOCGIFMTU PIC X(4) VALUE X'C020A726'.
01 SIOCGIPMSFILTER PIC X(4) VALUE X'C000A724'.
01 SIOCSIPMSFILTER PIC X(4) VALUE X'8000A725'.
01 SIOCGMSFILTER PIC X(4) VALUE X'C000F610'.
01 SIOCSMSFILTER PIC X(4) VALUE X'8000F611'.
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 PIC X(1).
  05 SS-FAMILY PIC X(1).
  05 SS-DATA PIC X(126).

* IP-MREQ for IP_ADD_MEMBERSHIP and IP_DROP_MEMBERSHIP
* 01 IP-MREQ.
  05 IMR-MULTIADDR PIC 9(8) BINARY.
  05 IMR-INTERFACE 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.
  05 IMR-MULTIADDR PIC 9(8) BINARY.
  05 IMR-SOURCEADDR PIC 9(8) BINARY.
  05 IMR-INTERFACE PIC 9(8) BINARY.

* IPV6-MREQ for IPV6_JOIN_GROUP and IPV6_LEAVE_GROUP
* 01 IPV6-MREQ.
  05 IPV6MR-MULTIADDR.
    10 FILLER PIC 9(16) BINARY.
    10 FILLER PIC 9(16) BINARY.
  05 IPV6MR-INTERFACE PIC 9(8) BINARY.

* GROUP-REQ for
  * MCAST_JOIN_GROUP
  * MCAST_LEAVE_GROUP
* 01 GROUP-REQ.
  05 GR-INTERFACE PIC 9(8) BINARY.
  05 FILLER PIC X(4).
  05 GR-GROUP PIC X(128).

Figure 77. EZACOBOL COBOL common variables (Part 2 of 8)
05 GR-GROUP-R REDEFINES GR-GROUP.
 10 GR-GROUP-(sock-len) PIC X(1).
 10 GR-GROUP-SOCK-FAMILY PIC X(1).
 10 GR-GROUP-SOCK-DATA PIC X(26).
 10 GR-GROUP-SOCK-SIN REDEFINES GR-GROUP-SOCK-DATA.
   15 GR-GROUP-SOCK-SIN-PORT PIC 9(4) BINARY.
   15 GR-GROUP-SOCK-SIN-ADD PIC 9(8) BINARY.
   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-SIN6-SCOPEID PIC 9(8) BINARY.
 10 FILLER PIC X(100).

* GROUP-SOURCE-REQ for
* MCAST_BLOCK_SOURCE
* MCAST_UNBLOCK_SOURCE
* MCAST_JOIN_SOURCE_GROUP
* MCAST_LEAVE_SOURCE_GROUP
* 
01 GROUP-SOURCE-REQ.
 05 GSR-INTERFACE PIC 9(8) BINARY.
 05 FILLER PIC X(4).
 05 GSR-GROUP PIC X(128).
 05 GSR-GROUP-R REDEFINES GSR-GROUP.
   10 GSR-GROUP-sock-len PIC X(1).
   10 GSR-GROUP-SOCK-FAMILY PIC X(1).
   10 GSR-GROUP-SOCK-DATA PIC X(26).
   10 GSR-GROUP-SOCK-SIN REDEFINES GSR-GROUP-SOCK-DATA.
     15 GSR-GROUP-SOCK-SIN-PORT PIC 9(4) BINARY.
     15 GSR-GROUP-SOCK-SIN-ADD PIC 9(8) BINARY.
     15 FILLER PIC X(8).
     15 FILLER PIC X(12).
   10 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.
     15 GSR-GROUP-SOCK-SIN6-ADDR.
     20 FILLER PIC 9(16) BINARY.
     20 FILLER PIC 9(16) BINARY.
     15 GSR-GROUP-SOCK-SIN6-SCOPEID PIC 9(8) BINARY.
   10 FILLER PIC X(100).
 05 GSR-SOURCE PIC X(128).
 05 GSR-SOURCE-R REDEFINES GSR-SOURCE.
   10 GSR-SOURCE-sock-len 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-ADD PIC 9(8) BINARY.
     15 FILLER PIC X(8).
     15 FILLER PIC X(12).

Figure 77. EZACOBOL COBOL common variables (Part 3 of 8)
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.
   20 FILLER PIC 9(16) BINARY.
   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.
77 MCAST-EXCLUDE PIC 9(8) BINARY VALUE 1.
77 MCAST-NUMSRC-MAX PIC 9(8) BINARY VALUE 64.

* IP-MSFILTER
*
01 IP-MSFILTER.
  02 IMSF-HEADER.
   03 IMSF-MULTIIDADDR PIC 9(8) BINARY.
   03 IMSF-INTERFACE PIC 9(8) BINARY.
   03 IMSF-FMODE 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.
   03 IMSF-SRCENTRY OCCURS 1 TO 64 TIMES
      DEPENDING ON IMSF-NUMSRC.
      05 IMSF-SRCADDR PIC 9(8) BINARY.

* GROUP-FILTER
*
01 GROUP-FILTER.
  02 GF-HEADER.
   03 GF-INTERFACE PIC 9(8) BINARY.
   03 FILLER PIC X(4).
   03 GF-GROUP PIC X(128).
   03 GF-GROUP-R REDEFINES GF-GROUP.
      05 GF-GROUP-SOCK-LEN PIC X(1).
      05 GF-GROUP-SOCK-FAMILY 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).
      10 FILLER PIC X(12).
      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.
      05 FILLER PIC X(100).
      03 GF-FMODE PIC 9(8) BINARY.

Figure 77. EZACOBOL COBOL common variables (Part 4 of 8)
* Structure for setting APPLDATA when using the SIOCSAPPLDATA ioctl. *

```cobol
77 SETADEYE1 PIC X(8) VALUE 'SETAPPLD'.
77 SETADVER PIC 9(4) BINARY VALUE 1.
01 SETAPPLDATA.
  02 SETAD-EYE1 PIC X(8).
  02 SETAD-VER PIC 9(4) BINARY.
  02 SETAD-LEN PIC 9(4) BINARY.
  02 FILLER PIC X(4).
  02 SETAD-PTR64 PIC 9(16) BINARY.
  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. *

```cobol
77 SETADEYE2 PIC X(8) VALUE 'APPLDATA'.
01 SETADCONTAINER.
  02 SETAD-EYE2 PIC X(8).
  02 SETAD-BUFFER PIC X(40).
```

* TIMEVAL for SO_RCVTIMEO and SO_SNDTIMEO *

```cobol
01 TIMEVAL.
  02 TV-SEC PIC 9(8) BINARY.
  02 TV-USEC PIC 9(8) BINARY.
```

---

Figure 77. EZACOBOL COBOL common variables (Part 5 of 8)
* 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 PIC X(1).
    10 IFR-ADDR-FAMILY PIC X(1).
    10 IFR-ADDR-PORT PIC 9(4) BINARY.
    10 IFR-ADDR-ADDR PIC 9(8) BINARY.
    10 FILLER PIC X(8).
  05 IFR-DSTADDR REDEFINES IFR-IFR.
    10 IFR-DSTADDR-LEN PIC X(1).
    10 IFR-DSTADDR-FAMILY PIC X(1).
    10 IFR-DSTADDR-PORT PIC 9(4) BINARY.
    10 IFR-DSTADDR-ADDR PIC 9(8) BINARY.
    10 FILLER PIC X(8).
  05 IFR-BROADADDR REDEFINES IFR-IFR.
    10 IFR-BROADADDR-LEN PIC X(1).
    10 IFR-BROADADDR-FAMILY PIC X(1).
    10 IFR-BROADADDR-PORT PIC 9(4) BINARY.
    10 IFR-BROADADDR-ADDR PIC 9(8) BINARY.
    10 FILLER PIC X(8).
  05 IFR-FLAGS-R REDEFINES IFR-IFR.
    10 IFR-FLAGS PIC X(2).
    10 FILLER PIC X(14).
  05 IFR-METRIC-R REDEFINES IFR-IFR.
    10 IFR-METRIC PIC 9(8) BINARY.
    10 FILLER PIC X(12).
  05 IFR-DATA-R REDEFINES IFR-IFR.
    10 IFR-DATA PIC 9(8) BINARY.
    10 FILLER PIC X(12).
  05 IFR-MTU-R REDEFINES IFR-IFR.
    10 IFR-MTU PIC 9(8) BINARY.
    10 FILLER PIC X(12).

* Constants for use with the IFR_FLAGS field in structure IFREQ.
* 
  01 IFF-UP PIC X(2) VALUE X'0001'.
  01 IFF-BROADCAST PIC X(2) VALUE X'0002'.
  01 IFF-DEBUG PIC X(2) VALUE X'0004'.
  01 IFF-LOOPBACK PIC X(2) VALUE X'0008'.
  01 IFF-POINTOPOINT PIC X(2) VALUE X'0010'.
  01 IFF-NOTRAILERS PIC X(2) VALUE X'0020'.
  01 IFF-RUNNING PIC X(2) VALUE X'0040'.
  01 IFF-NOARP PIC X(2) VALUE X'0080'.
  01 IFF-PROMISC PIC X(2) VALUE X'0100'.
  01 IFF-ALLMULTI PIC X(2) VALUE X'0200'.
  01 IFF-MULTICAST PIC X(2) VALUE X'0400'.
  01 IFF-POINTOMULTIPT PIC X(2) VALUE X'0800'.
  01 IFF-BRIDGE PIC X(2) VALUE X'1000'.
  01 IFF-SNAP PIC X(2) VALUE X'2000'.
  01 IFF-VIRTUAL PIC X(2) VALUE X'4000'.
  01 IFF-SAMEHOST PIC X(2) VALUE X'8000'.

Figure 77. EZACOBOL COBOL common variables (Part 6 of 8)
Figure 77. EZACOBOL COBOL common variables (Part 7 of 8)
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

Figure 77. EZACOBOL COBOL common variables (Part 8 of 8)
Identification Division.
**********************************************************
Program-id. EZASO6CS.
**********************************************************
Environment Division.
**********************************************************
Data Division.
**********************************************************
Working-storage Section.
*****************************************************************
* Socket interface function codes
*****************************************************************
01 soket-functions.
   02 soket-accept pic 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-fcntl pic x(16) value 'FCNTL '.
   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-gethostbyname pic x(16) value 'GETHOSTBYADDR '.
   02 soket-gethostname pic x(16) value 'GETHOSTNAME '.
   02 soket-gethostid pic x(16) value 'GETHOSTID '.
   02 soket-getpeergroupname pic x(16) value 'GETPEERGROUPNAME '.
   02 soket-getpeername pic x(16) value 'GETPEERNAME '.
   02 soket-getsockname pic x(16) value 'GETSOCKNAME '.

Figure 78. EZASO6CS COBOL call interface sample IPv6 server program (Part 1 of 13)
02 soket-getsockopt pic x(16) value 'GETSOCKOPT'.
02 soket-givesocket pic x(16) value 'GIVESOCKET'.
02 soket-initapi pic x(16) value 'INITAPI'.
02 soket-ioctl pic x(16) value 'IOCTL'.
02 soket-listen pic x(16) value 'LISTEN'.
02 soket-ntop pic x(16) value 'NTOP'.
02 soket-pton pic x(16) value 'PTON'.
02 soket-read pic x(16) value 'READ'.
02 soket-rerecv pic x(16) value 'RECV'.
02 soket-rerecvfrom pic x(16) value 'RECVFROM'.
02 soket-select pic x(16) value 'SELECT'.
02 soket-send pic x(16) value 'SEND'.
02 soket-sendto pic x(16) value 'SENDTO'.
02 soket-setsockopt pic x(16) value 'SETSOCKOPT'.
02 soket-shutdown pic x(16) value 'SHUTDOWN'.
02 soket-socket pic x(16) value 'SOCKET'.
02 soket-takesocket pic x(16) value 'TAKESOCKET'.
02 soket-termapi pic x(16) value 'TERMAPI'.
02 soket-write pic x(16) value 'WRITE'.

*---------------------------------------------------------------*
* Work variables                                               *
*---------------------------------------------------------------*
01 errno pic 9(8) binary value zero.
01 retcode pic s9(8) binary value zero.
01 client-ipaddr-dotted pic x(15) value space.
01 server-ipaddr-dotted pic x(15) value space.
01 ezacomm-function pic x value space.
 88 CONNECTED value 'Y'.
01 saved-message-id pic x(8) value space.
 88 close-down-message-received value 'CLSDWN*'.
01 Terminate-Options pic x value space.
 88 Opened-API value 'A'.
 88 Opened-Socket value 'S'.
01 saved-message-id-len pic 9(8) Binary value 8.
01 Cur-time .
 02 Hour pic 9(2).
 02 Minute pic 9(2).
 02 Second pic 9(2).
 02 Hund-Sec pic 9(2).
01 S pic 9(4) comp.

*---------------------------------------------------------------*
* Variables used for the INITAPI call                          *
*---------------------------------------------------------------*
01 maxsoc-fwd pic 9(8) Binary.
01 maxsoc-rdf redefines maxsoc-fwd.
 02 filler pic x(2).
 02 maxsoc pic 9(4) Binary.
01 initapi-ident.
 05 tcpname pic x(8) Value 'TCP  CS '.
 05 asname pic x(8) Value space.
01 subtask pic x(8) value 'EZASO6CS'.
01 maxsn0 pic 9(8) Binary Value 1.

*---------------------------------------------------------------*
* Variables returned by the GETCLIENTID Call                   *
*---------------------------------------------------------------*

Figure 78. EZASO6CS COBOL call interface sample IPv6 server program (Part 2 of 13)
01 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.

*---------------------------------------------------------------*
* Variables used for the SOCKET call *
*---------------------------------------------------------------*
  01 AF-INET pic 9(8) Binary Value 2.
  01 AF-INET6 pic 9(8) Binary Value 19.
  01 SOCK-STREAM pic 9(8) Binary Value 1.
  01 SOCK-DATAGRAM pic 9(8) Binary Value 2.
  01 SOCK-RAW pic 9(8) Binary Value 3.
  01 IPPROTO-IP pic 9(8) Binary Value zero.
  01 IPPROTO-TCP pic 9(8) Binary Value 6.
  01 IPPROTO-UDP pic 9(8) Binary Value 17.
  01 IPPROTO-IPV6 pic 9(8) Binary Value 41.
  01 socket-descriptor pic 9(4) Binary Value zero.

*---------------------------------------------------------------*
* 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 *
*---------------------------------------------------------------*
  01 node-name pic x(255).
  01 node-name-len pic 9(8) binary.
  01 service-name pic x(32).
  01 service-name-len pic 9(8) binary.
  01 canonical-name-len pic 9(8) binary.
  01 ai-passive pic 9(8) binary value 1.
  01 ai-canonnameok pic 9(8) binary value 2.
  01 ai-numerichost pic 9(8) binary value 4.
  01 ai-numericerv pic 9(8) binary value 8.
  01 ai-v4mapped pic 9(8) binary value 16.
  01 ai-all pic 9(8) binary value 32.
  01 ai-addrconfig pic 9(8) binary value 64.

*---------------------------------------------------------------*
* Variables used for the BIND call *
*---------------------------------------------------------------*
  01 server-socket-address.
    05 server-family pic 9(4) Binary value 19.
    05 server-port pic 9(4) Binary value 1031.
    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.
  01 NBYTE PIC 9(8) COMP value 80.
  01 BUF PIC X(80).
  01 BACKLOG PIC S9(8) COMP VALUE 10.

*---------------------------------------------------------------*

Figure 78. EZASO6CS COBOL call interface sample IPv6 server program (Part 3 of 13)
* Variables used/returned by the EZACICO9 call *

01 input-addrinfo-ptr usage is pointer.
01 output-name-len pic 9(8) binary.
01 output-canonical-name pic x(256).
01 output-name usage is pointer.
01 output-next-addrinfo usage is pointer.

* Variables used for the LISTEN call *

01 backlog-level pic 9(4) Binary Value zero.

* Variables used for the ACCEPT call *

01 socket-descriptor-new pic 9(4) Binary Value zero.

* Variables used for the NTOP/PTON call *

01 IN6ADDR-ANY pic x(45) value '::'.
01 IN6ADDR-LOOPBACK pic x(45) value '::1'.
01 ntop-family pic 9(8) Binary.
01 pton-family pic 9(8) Binary.
01 presentable-addr pic x(45) value spaces.
01 presentable-addr-len pic 9(4) Binary value 45.
01 numeric-addr.
   05 filler pic 9(16) Binary Value 0.
   05 filler pic 9(16) Binary Value 0.

* Variables used by the RECV Call *

01 client-socket-address.
   05 client-family pic 9(4) Binary Value 19.
   05 client-port pic 9(4) Binary Value 1032.
   05 client-flowinfo pic 9(8) Binary Value zero.
   05 client-ipaddr.
      10 filler pic 9(16) Binary Value 0.
      10 filler pic 9(16) Binary Value 0.
   05 client-scopeid pic 9(8) Binary Value zero.

* Buffer and length field for recv and send operation *

01 send-request-len pic 9(8) Binary Value zero.
01 read-request-len pic 9(8) Binary Value zero.
01 read-buffer pic x(4000) value space.
01 filler redefines read-buffer.
   05 message-id pic x(8).
   05 filler 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.

Figure 78. EZASO6CS COBOL call interface sample IPv6 server program (Part 4 of 13)
* Error message for socket interface errors *

*---------------------------------------------------------------*
77 failure pic S9(8) comp.
01 ezaerror-msg.
  05 filler pic x(9) value 'Function='.
  05 ezaerror-function pic x(16) value space.
  05 filler pic x value ' '.
  05 filler pic x(8) value 'Retcode='.
  05 ezaerror-retcode pic ---99.
  05 filler pic x value ' '.
  05 filler pic x(9) value 'Errorno='.
  05 ezaerror-errno pic zzz99.
  05 filler pic x value ' '.
  05 ezaerror-text pic x(99) value ' '.

*================
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 results-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.
    output-ip-sock-data redefines
    05 output-ipv4-ipaddr pic 9(8) Binary.
    05 filler pic x(20).
  03 output-ipv6-sock-data redefines

Figure 78. EZASO6CS COBOL call interface sample IPv6 server program (Part 5 of 13)
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.
*=============================================*

*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
* PROCEDURE CONTROLS *
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*

  Perform Initialize-API thru Initialize-API-Exit.
  Perform Get-ClientID thru Get-ClientID-Exit.
  Perform Sockets-Descriptor thru Sockets-Descriptor-Exit.
  Perform Presentation-To-Numeric thru
       Presentation-To-Numeric-Exit.
  Perform Get-Host-Name thru Get-Host-Name-Exit.
  Perform Get-Address-Info thru Get-Address-Info-Exit.
  Perform Bind-Socket thru Bind-Socket-Exit.
  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.
  Perform Write-Message thru Write-Message-Exit.
  Perform Close-Socket thru Exit-Now.

*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
* 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 maxsoc error retcode.
  Move 'initapi failed' to ezaerror-text.
  If retcode < 0 move 12 to failure.
  Perform Return-Code-Check thru Return-Code-Exit.
  Move 'A' to Terminate-Options.
  Initialize-API-Exit.
  Exit.

*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
* Let us see the client-id *
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*

Figure 78. EZASO6CS COBOL call interface sample IPv6 server program (Part 6 of 13)
*---------------------------------------------------------------*
Get-ClientID.
  move soket-getclientid to ezaerror-function.
  Call 'EZASOKET' using soket-getclientid clientid errno
       retcode.
  Display 'Client ID = ' clientid-name
       'task= ' clientid-task.
  Move 'Getclientid failed' to ezaerror-text.
  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.

Figure 78. EZASO6CS COBOL call interface sample IPv6 server program (Part 7 of 13)
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 0 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 = ' node-name.
    display 'Node name length = ' 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.
    display 'GETADDRINFO Input fields: '
    display 'Node name = ' node-name.
    display 'Service name = ' service-name.
    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 ezaerror-function.
    set hints-addrinfo-ptr to address of results-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
       display 'Address of results addrinfo is '
          results-addrinfo-ptr.
       set address of results-addrinfo to results-addrinfo-ptr
       set input-addrinfo-ptr to address of results-addrinfo

Figure 78. EZASO6CS COBOL call interface sample IPv6 server program (Part 8 of 13)
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.
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 ip address'.
set address of output-ip-name to output-name.
move results-ai-family to ntopt-family.
display 'ntop-family = ' ntopt-family.
if ntopt-family = AF-INET then
daIl into IPv4 address
Figure 78. EZASO6CS COBOL call interface sample IPv6 server program (Part 9 of 13)
move output-ipv4-ipaddr to numeric-addr
move 16 to presentable-addr-len
else
display 'Formatting ipv6 address'
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.
Move 'Bind call failed' to ezaerror-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.

Figure 78. EZASO6CS COBOL call interface sample IPv6 server program (Part 10 of 13)
* Accept a connection request *

Accept-Connection.
  Move 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
    errno 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 EZASO6SC' to buf.
  Call 'EZASOKET' using soket-write socket-descriptor-new

Figure 78. EZASO6CS COBOL call interface sample IPv6 server program (Part 11 of 13)
nbyte buf
errno retcode.
Move '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
errno retcode.
Accept cur-time from time.
Display cur-time ' EZASO6CS : CLOSE RETCODE=' RETCODE
 ' ERRNO= ' ERRNO.
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.

*---------------------------------------------------------------*
* 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.

*---------------------------------------------------------------*

Figure 78. EZASO6CS COBOL call interface sample IPv6 server program (Part 12 of 13)
* 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 78. EZASO6CS COBOL call interface sample IPv6 server program (Part 13 of 13)

**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
Identification Division.

Program-id. EZASO6CC.

Environment Division.

Data Division.

Working-storage Section.

Socket interface function codes

01 socket-functions.
   02 socket-accept pic x(16) value 'ACCEPT' .
   02 socket-bind pic x(16) value 'BIND' .
   02 socket-close pic x(16) value 'CLOSE' .
   02 socket-connect pic x(16) value 'CONNECT' .
   02 socket-fcntl pic x(16) value 'FCNTL' .
   02 socket-freeaddrinfo pic x(16) value 'FREEADDRINFO' .
   02 socket-getaddrinfo pic x(16) value 'GETADDRINFO' .
   02 socket-getclientid pic x(16) value 'GETCLIENTID' .
   02 socket-gethostbyname pic x(16) value 'GETHOSTBYNAME' .
   02 socket-gethostid pic x(16) value 'GETHOSTID' .
   02 socket-gethostname pic x(16) value 'GETHOSTNAME' .
   02 socket-getnameinfo pic x(16) value 'GETNAMEINFO' .
   02 socket-getpeername pic x(16) value 'GETPEERNAME' .
02 soket-getsockname pic x(16) value 'GETSOCKNAME'.
02 soket-getsockopt pic x(16) value 'GETSOCKOPT'.
02 soket-givesocket pic x(16) value 'GIVESOCKET'.
02 soket-initapi pic x(16) value 'INITAPI'.
02 soket-ioct1 pic x(16) value 'IOCTL'.
02 soket-listen pic x(16) value 'LISTEN'.
02 soket-nstop pic x(16) value 'NSTOP'.
02 soket-pton pic x(16) value 'PTON'.
02 soket-read pic x(16) value 'READ'.
02 soket-recc pic x(16) value 'RECV'.
02 soket-reccfrom pic x(16) value 'RECVFROM'.
02 soket-select pic x(16) value 'SELECT'.
02 soket-send pic x(16) value 'SEND'.
02 soket-sendto pic x(16) value 'SENDTO'.
02 soket-setsockopt pic x(16) value 'SETSOCKOPT'.
02 soket-shutdown pic x(16) value 'SHUTDOWN'.
02 soket-socket pic x(16) value 'SOCKET'.
02 soket-takesocket pic x(16) value 'TAKESOCKET'.
02 soket-termapi pic x(16) value 'TERMAPI'.
02 soket-write pic x(16) value 'WRITE'.

* Work variables *

01 errno pic 9(8) binary value zero.
01 retcode pic 9(8) binary value zero.
01 index-counter pic 9(8) binary value zero.
01 buffer-element.
   05 buffer-element-nbr pic 9(5).
   05 filler pic x(3) value space.
01 server-ipaddr-dotted pic x(15) value space.
01 client-ipaddr-dotted pic x(15) value space.
01 close-server pic 9(8) binary value zero.
   88 close-server-down value 1.
01 Connect-Flag pic x value space.
   88 CONNECTED value 'Y'.
01 Client-Server-Flag pic x value space.
   88 CLIENTS value 'C'.
   88 SERVERS value 'S'.
01 Terminate-Options pic x value space.
   88 Opened-API value 'A'.
   88 Opened-Socket value 'S'.
01 timer-accum pic 9(8) binary value zero.
01 timer-interval pic 9(8) binary value 2000.
01 Cur-time.
   02 Hour pic 9(2).
   02 Minute pic 9(2).
   02 Second pic 9(2).
   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.
01 maxsoc-rdf redefines maxsoc-fwd.
   02 filler pic x(2).

Figure 79. EZASO6CC COBOL call interface sample IPv6 client program (Part 2 of 9)
02 maxsoc pic 9(4) Binary.
01 initapi-ident.
  05 tcpname pic x(8) Value 'TCPCS'.
  05 asname pic x(8) Value space.
01 subtask pic x(8) Value 'EZSO6CC'.
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                      *
*---------------------------------------------------------------*
01 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.

*---------------------------------------------------------------*
* Variables returned by the GETNAMEINFO Call                      *
*---------------------------------------------------------------*
01 name-len pic 9(8) Binary.
01 host-name pic x(255).
01 host-name-len pic 9(8) Binary.
01 service-name pic x(32).
01 service-name-len pic 9(8) Binary.
01 name-info-flags pic 9(8) Binary value 0.
01 ni-nofqdn pic 9(8) Binary value 1.
01 ni-numerichost pic 9(8) Binary value 2.
01 ni-namereqd pic 9(8) Binary value 4.
01 ni-numericserver pic 9(8) Binary value 8.
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.
01 SOCK-STREAM pic 9(8) Binary Value 1.
01 SOCK-DATAGRAM pic 9(8) Binary Value 2.
01 SOCK-RAW pic 9(8) Binary Value 3.
01 IPPROTO-IP pic 9(8) Binary Value zero.
01 IPPROTO-TCP pic 9(8) Binary Value 6.
01 IPPROTO-UDP pic 9(8) Binary Value 17.
01 IPPROTO-IPV6 pic 9(8) Binary Value 41.
01 socket-descriptor pic 9(4) Binary Value zero.

*---------------------------------------------------------------*
* Server socket address structure                                *
*---------------------------------------------------------------*
01 server-socket-address.
  05 server-afinet pic 9(4) Binary Value 19.
  05 server-port pic 9(4) Binary Value 1031.
  05 server-flowinfo pic 9(8) Binary Value zero.
  10 filler pic 9(16) Binary Value 0.
  10 filler pic 9(16) Binary Value 0.

Figure 79. EZASO6CC COBOL call interface sample IPv6 client program (Part 3 of 9)
05 server-scopeid pic 9(8) Binary Value zero.
01 NBYTE PIC 9(8) COMP value 80.
01 BUF PIC X(80).

* Variables used by the BIND Call *
*---------------------------------------------------------------*
01 client-socket-address.
  05 client-family pic 9(4) Binary Value 19.
  05 client-port pic 9(4) Binary Value 1032.
  05 client-flowinfo pic 9(8) Binary Value 0.
  05 client-ipaddr.
  10 filler pic 9(16) Binary Value 0.
  10 filler pic 9(16) Binary Value 0.
  05 client-scopeid pic 9(8) Binary Value 0.

* Buffer and length fields for send operation *
*---------------------------------------------------------------*
01 send-request-length pic 9(8) Binary value zero.
01 send-buffer.
  05 send-buffer-total pic x(4000) value space.
  05 closedown-message redefines send-buffer-total.
  10 closedown-id pic x(8).
  10 filler 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 '::'.
01 IN6ADDR-LOOPBACK pic x(45) value '::1'.
01 presentable-addr pic x(45) value spaces.
01 presentable-addr-len pic 9(4) Binary value 45.
01 numeric-addr.
  05 filler pic 9(16) Binary Value 0.
  05 filler pic 9(16) Binary Value 0.

* Buffer and length fields for recv operation *
*---------------------------------------------------------------*
01 read-request-length pic 9(8) Binary value zero.
01 read-buffer pic x(4000) value space.

* Other fields for send and recvfrom 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 *
*---------------------------------------------------------------*
01 ezaerror-msg.
  05 filler pic x(9) Value 'Function='.
  05 ezaerror-function pic x(16) Value space.
  05 filler pic x value ' '.
  05 filler pic x(8) Value 'Retcode='.

Figure 79. EZASO6CC COBOL call interface sample IPv6 client program (Part 4 of 9)
05 ezaerror-retcode pic --99.
05 filler pic x value ' '.
05 filler pic x(9) value 'Errorno='.
05 ezaerror-errno pic z99.
05 filler pic x value ' '.
05 ezaerror-text pic x(50) value ' '.

Linkage Section.
*================

*=============================================*
Procedure Division.
*=============================================*

*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
* PROCEDURE CONTROLS *
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*

Perform Initialize-API thru Initialize-API-Exit.
Perform Get-Client-ID thru Get-Client-ID-Exit.
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 thru Read-Message-Exit.
Perform Shutdown-Receive thru Shutdown-Receive-Exit.
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
        subtask maxsno errno retcode.
Move 'Initapi failed' to ezaerror-text.
If retcode < 0 move 12 to failure.
Perform Return-Code-Check thru Return-Code-Exit.
Move 'A' to Terminate-Options.
Initialize-API-Exit.
Exit.

*---------------------------------------------------------------*
* Let us see the client-id *
*---------------------------------------------------------------*

Get-Client-ID.
Move soket-getclientid to ezaerror-function.
Call 'EZASOKET' using soket-getclientid clientid errno retcode.

Figure 79. EZASO6CC COBOL call interface sample IPv6 client program (Part 5 of 9)
Display 'Our client ID = ' clientid-name ' ' clientid-task.
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-Check.
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 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 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.

*---------------------------------------------------------------*
* CONNECT *
*---------------------------------------------------------------*
Connect-Socket.
Move space to Connect-Flag.
Move zeros to errno retcode.
move soket-connect to ezaerror-function.
CALL 'EZASOKET' USING SOKET-CONNECT socket-descriptor
server-socket-address errno retcode.
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.
Exit.

Figure 79. EZASO6CC COBOL call interface sample IPv6 client program (Part 6 of 9)
* 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 ezaerror-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.
  Move 'Message from EZASO6CC' to buf.
  Call 'EZASOKET' 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.

Figure 79. EZASO6CC COBOL call interface sample IPv6 client program (Part 7 of 9)
Exit.

*---------------------------------------------------------------*
* 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
        errno 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 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
        errno retcode.

Figure 79. EZASO6CC COBOL call interface sample IPv6 client program (Part 8 of 9)
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 79. EZASO6CC COBOL call interface sample IPv6 client program (Part 9 of 9)
Chapter 8. IMS Listener samples

This topic includes sample programs using the IMS Listener. The following samples are included:

- “IMS TCP/IP control statements”
- “Sample program explicit-mode” on page 275
- “Sample program implicit-mode” on page 285
- “Sample program - IMS MPP client” on page 294

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

The following is 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, refer to the 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=,DBDL=,
// NBA=,OBA=,IMSID=IMS1,AGN=,VSFX=,VFREE=,
// SSM=,PREINIT=,ALTID=,PWFI=N,
// APARM=
//*
//REGION EXEC PGM=DFSSRC00,REGION=&RGN,;
// TIME=1440,DPRTY=(12,0),
// PARM=(MSG,&CL1&CL2&CL3&CL4,;
// &OPT&Ovla&SPIE&VALCK&Tlim&PCB,;
// &PRLD,&STIMER,&SOD,&DBDL,&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=TCP/IP.SEZATCP,DISP=SHR
```
JCL for linking the IMS Listener

The following examples are JCL that can be used to link the IMS Listener.

EZAIMSCZ JCLIN
//EZAIMSCZ JOB (accounting,information),programmer.name, // MSGLEVEL=(1,1),MSGCLASS=A,CLASS=A //*************************************************************** //NOTE: ANY ZONE UPDATED WITH THE LINK COMMAND OR CROSS-ZONE * // 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 //*************************************************************** //RESLIB DD DISP=SHR,DSN=imshlq.RESLIB //SDFSRESL DD DISP=SHR,DSN=imshlq.SDFSRESL //*************************************************************** //SMPCS1 dd dsn=zos.global.csi,disp=old //SYST1 DD UNIT=SYSDA,SPACE=(1700,(900,200)) //SYST2 DD UNIT=SYSDA,SPACE=(1700,(600,100)) //SYST3 DD UNIT=SYSDA,SPACE=(1700,(600,100)) //SYST4 DD UNIT=SYSDA,SPACE=(1700,(600,100)) //SMPWRK1 DD UNIT=SYSDA,SPACE=(8800,(75,0,216)), 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) //Figure 80. Cross zone Lnk IMS application interface (Part 1 of 2)
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
```
Sample program explicit-mode

The following is 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 TAKE_SOCKET 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;
{
unsigned short port; /* port client will connect to */
char buf[lim]; /* send receive buffers 0 -3 */
char buf1[lim];
char buf2[lim];
char buf3[lim];
struct hostent *hostnm; /* server host name information */
struct sockaddr_in server; /* server address */

/*
 * Check Arguments Passed. Should be hostname and port.
 */
if (argc != 3)
{
    printf("Usage: %s hostname port\n", argv[0]);
    exit(1);
}

/*
 * The host name is the first argument. Get the server address.
 */
hostnm = gethostbyname(argv[1]);
if (hostnm == (struct hostent *) 0)
{
    printf("Gethostbyname failed\n");
    exit(2);
}

/*
Figure 81. Sample C client to drive IMS Listener (Part 1 of 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-&gt;h_addr);
```

* Get a stream socket.

```
if ((s = socket(AF_INET, SOCK_STREAM, 0)) &lt; 0)
{
    tcperror("Socket()");
    exit(3);
}
```

* Connect to the server.

```
if (connect(s, (struct sockaddr *)&amp;server, sizeof(server)) &lt; 0)
{
    tcperror("Connect()");
    exit(4);
}
```

if (send(s, buf, sizeof(buf), 0) &lt; 0)
{
    tcperror("Send()");
    exit(5);
}

printf("send one complete\n");
```

* The server sends message #1. Receive it into buffer1

```
if (recv(s, buf1, sizeof(buf1), 0) &lt; 0)
{
    tcperror("Recv()");
    exit(6);
}
```

printf("receive one complete\n");
```

Figure 81. Sample C client to drive IMS Listener (Part 2 of 3)
Sample explicit-mode server program (Assembler language)

```assembly
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);
}
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 81. Sample C client to drive IMS Listener (Part 3 of 3)
EZASVAS2 CSECT ENTRY POINT
USING EZASVAS2,BASE ADDRESSABILITY
SAVE (14,12) SAVE DL/I REGS
LR BASE,15
ST R13,SAVEAREA+4 SAVE AREA CHAINING
LA R13,SAVEAREA 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

* 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 R4,8(R0,R1) LOAD ADDR OF ALT PCB
LA R4,0(R0,R4) REMOVE HIGH ORDER BIT
* USING ALTPCB2,R4 ADDRESSABILITY
* LA R5,IOAREAIN
LA R7,IOAREAOT POINT TO OUTPUT AREA FOR TCPIP
GUCALL DS 0H GET UNIQUE CALL
*******************************************************************
* Get Transaction-initiation message containing Sockets data *
*******************************************************************
CALL ASMTDLI,(GUFUNCT,(2),(5)),VL GET TIM
CLC STATUS(L'STATUS),=CL2'QC' IF NO MESSAGES
BE GOBACK RETURN TO IMS
* CLC STATUS(L'STATUS),=CL2' IF BLANK OK
BNE ERRRTN SOME WRONG HERE
* XR R6,R6 CLEAR REG
BAL R6,INITAPI GO INSERT SEGMENT
B GUCALL SET RETURN ADDRESS

* INITAPI DS 0H
* Set up for INITAPI
MVC TCPNAME(L'TCPNAME),TIMTCPAS TCP Address space
MVC ASDNAME(L'ASDNAME),TIMSAS 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

Figure 82. Sample assembler IMS server (Part 1 of 6)
Figure 82. Sample assembler IMS server (Part 2 of 6)
Figure 82. Sample assembler IMS server (Part 3 of 6)
MVC OTLTIME(L'OTLTIME),DCTIME MOVE IN TIME
UNPK OTDATE,CDATE MAKE TIME & DATE
OI OTDATE+7,X'F0' EBCDIC
UNPK OTTIME,CTIME
OI OTTIME+7,X'F0'
XR R9,R9 GET READY
L R9,INPUTMSN INPUT COUNT
CVD R9,DLBWORK INPUT COUNT
UNPK OTINPUTN,DLBWORK INPUT COUNT
OI OTINPUTN+7,X'F0', FIX SIGN
MVC OTFILL(L'OTFILL),=28X'40' FILL CHAR
MVC OTLTERM(L'OTLTERM),LTERMN ADD TERMINAL

* *
* ** CALL ASMTDLI,(ISRTFUNCT,(3),(7),USER1),VL *
* ** XC IOAREADOT(L'IOAREADOT),IOAREADOT *
** BR R6
* ** ERRRTN DS 0H SOME WRONG HERE *
* ** L R13,4(R13) RETURN (14,12),RC=8 RELOAD DL/I REGS & RETURN *
** RETURN (14,12),RC=8 RELOAD DL/I REGS & RETURN *
* ** L R13,4(R13) RETURN (14,12),RC=0 RELOAD DL/I REGS & RETURN *

DS 0D
PSBS DS 3F
SPACE 1
BASE EQU 12
RC EQU 15
R0 EQU 0
R1 EQU 1
R2 EQU 2
R3 EQU 3
R4 EQU 4
R5 EQU 5
R6 EQU 6
R7 EQU 7
R8 EQU 8
R9 EQU 9
R10 EQU 10
R11 EQU 11
R12 EQU 12
R13 EQU 13
R14 EQU 14
R15 EQU 15
SPACE 1

DS 0F
SAVEAREA DC 18F'0'

* Figure 82. Sample assembler IMS server (Part 4 of 6)
Figure 82. Sample assembler IMS server (Part 5 of 6)
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMLEN</td>
<td>Length of trans init msg</td>
</tr>
<tr>
<td>TIMRSV</td>
<td>reserved set to zeros</td>
</tr>
<tr>
<td>TIMID</td>
<td>LISTENER ID set to LISTNR</td>
</tr>
<tr>
<td>TIMLAS</td>
<td>LISTENER addr space name</td>
</tr>
<tr>
<td>TIMLTD</td>
<td>LISTENER taskid for takesocket</td>
</tr>
<tr>
<td>TIMSAS</td>
<td>SERVER addr space name</td>
</tr>
<tr>
<td>TIMSTD</td>
<td>SERVER TASK ID user in initapi</td>
</tr>
<tr>
<td>TIMSD</td>
<td>socket given in LISTENER used in tasksocket</td>
</tr>
<tr>
<td>TIMTCPAS</td>
<td>TCPIP addr space name</td>
</tr>
<tr>
<td>TIMDT</td>
<td>Data type of client</td>
</tr>
<tr>
<td></td>
<td>ASCII(0) or EBCDIC(1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTLTH</td>
<td>logical terminal name</td>
</tr>
<tr>
<td>OTLSRV</td>
<td>reserved for IMS</td>
</tr>
<tr>
<td>OTLTERM</td>
<td>status code</td>
</tr>
<tr>
<td>OTINPUTN</td>
<td>current date YYDDD</td>
</tr>
<tr>
<td>OTCITM</td>
<td>current time HHMMSSST</td>
</tr>
<tr>
<td>OTINPUTN</td>
<td>sequence number</td>
</tr>
<tr>
<td>OTMSGDN</td>
<td>message out desc name</td>
</tr>
<tr>
<td>USERID</td>
<td>user id of source</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTPCB1</td>
<td>alternate pcb</td>
</tr>
<tr>
<td>ALTERM1</td>
<td>destination name</td>
</tr>
<tr>
<td>ALSTAT1</td>
<td>status code</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTPCB2</td>
<td>alternate pcb</td>
</tr>
<tr>
<td>ALTERM2</td>
<td>destination name</td>
</tr>
<tr>
<td>ALSTAT2</td>
<td>status code</td>
</tr>
</tbody>
</table>

Figure 82. Sample assembler IMS server (Part 6 of 6)
Sample program implicit-mode

The following is 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 ASMTDLDI) 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 <bsdtypes.h>
#include <in.h>
#include <socket.h>
#include <netdb.h>
#include <stdio.h>

/* * Client Main. */
main(argc, argv)
int argc;
char **argv;
{ unsigned short port; /* port client will connect to */
  struct sktmsg
  { 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);
  }

  printf("Usage: %s program name\n", argv[0]);

  /* Figure 83. Sample C client to drive IMS Listener (Part 1 of 5) */
/* 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]);

/*
 * 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);
}

/*
 * Put a message into the buffer.
 */
msgbuff.msgdat[0] = '\0';
msgbuff.msgsrv = 0;
msgbuff.msglen = 20;
strncat(msgbuff.msgtrn,"*TRNREQ*",
lim-strlen(msgbuff.msgdat)-1);
strncat(msgbuff.msgdat,"DLSI101 ",

Figure 83. Sample C client to drive IMS Listener (Part 2 of 5)
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.datrv = 0;
strncat(datbuff.datdat,"THIS IS FIRST TEXT MESSAGE SEND TO SERVER ",
    lim-strlen(datbuff.datdat)-1);
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");

/*
 * Put a text message into the buffer.
 */
datbuff.datdat??(0??)='\0';
datbuff.datlen = 47;
datbuff.datrv = 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);
}

printf("\n");
printf(datbuff.datdat);
printf("\n");
printf("send for 2nd test message complete\n");

Figure 83. Sample C client to drive IMS Listener (Part 3 of 5)
/*
 * 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(0);
}

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(0);
}

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");

/*

Figure 83. Sample C client to drive IMS Listener (Part 4 of 5)
Sample implicit-mode server program (Assembler language)

Figure 83. Sample C client to drive IMS Listener (Part 5 of 5)
EZASVAS1 CSECT ENTRY POINT
USING EZASVAS1,BASE ADDRESSABILITY
SAVE (I4,12) SAVE DL/I REGS
LR BASE,15
ST R13,SAVEAREA+4 SAVE AREA CHAINING
LA R13,SAVEAREA 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

* 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 R4,8(R0,R1) LOAD ADDR OF ALT PCB
LA R4,0(R0,R4) REMOVE HIGH ORDER BIT
* USING ALTPCB2,R4 ADDRESSABILITY
* LA R5,IOAREAIN
LA R7,IOAREAOT POINT TO OUTPUT AREA
* GUCALL DS 0H GET UNIQUE CALL
* *
* CALL ASMADLI,(GUFUNCT,(2),(5)),VL
*
* CLC STATUS(L’STATUS),=CL2’QC’ IF NO MESSAGES
BE GOBACK RETURN TO IMS
* CLC STATUS(L’STATUS),=CL2’ ‘ IF BLANK OK
BNE ERRRTN SOME WRONG HERE
* *
* XR R6,R6 CLEAR REG
LA R6,GNCALL SET RETURN ADDRESS
BAL R6,ISRTCALL GO INSERT SEGMENT
* GNCALL DS 0H GET NEXT CALL
* *
* CALL ASMADLI,(GNFUNCT,(2),(5)),VL
*
* CLC STATUS(L’STATUS),=CL2’QD’ IF NO MORE SEGMENTS
BE GUCALL RETURN TO IMS
CLC STATUS(L’STATUS),=CL2’ ‘ IF NO MORE SEGMENTS
BNE ERRRTN SOME WRONG HERE

Figure 84. Sample assembler IMS server (Part 1 of 4)
Figure 84. Sample assembler IMS server (Part 2 of 4)
Chapter 8. IMS Listener samples 293

Figure 84. Sample assembler IMS server (Part 3 of 4)
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 is 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.

Figure 84. Sample assembler IMS server (Part 4 of 4)
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

```assembler
EZAIMSC3 CSECT
EZAIMSC3 AMODE ANY
EZAIMSC3 RMODE ANY

GBLB &TRACE ASSEMBLER VARIABLE TO CONTROL TRACE GENERATION
&TRACE SETB 1 1=TRACE ON 0=TRACE OFF
GBLB &SUBTR ASSEMBLER VARIABLE TO CONTROL SUBTRACE
&SUBTR SETB 0 1=SUBTRACE ON 0=SUBTRACE OFF

*---------------------------------------------------------------------*
** MODULE NAME: EZAIMSC3 *
** *
** Copyright: Licensed Materials - Property of IBM *
** *
** "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: 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 *
```

Figure 85. Sample of IMS program as a client (Part 1 of 10)
**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
- **SERVPORT DC H'5000'** Port Address of Server
- **SOCTASK DC F'0'** Task number for this client
- **SERVELN 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 DC CL16'INITAPI'**
- **GETHSTID DC CL16'GETHOSTID'**
- **SOCKET DC CL16'SOCKET'**
- **GHBN DC CL16'GETHOSTBYNAME'**
- **CONNECT DC CL16'CONNECT'**
- **READ DC CL16'READ'**
- **WRITE DC CL16'WRITE'**
- **CLOSE DC CL16'CLOSE'**
- **TERMAPI DC CL16'TERMAPI'**

**Beginning of program execution statements**

**Beginning of program**

```assembly
STM R14,R12,12(R13) Save callers registers
LR R3,R15 Move base reg to R3
L R4,R4BASE Add R4 as second base reg
DROP R15 Tell assembler to drop R15 as base
USING SOC0000,R3,R4 Tell assembler to use R3 and R4 as X
base registers
LR R7,R13 Save address of previous save area
LA R12,SOCSTG Move address of program stg to R12
LA R13,SOCSTGL Move length of program stg to R13
SR R14,R14 Clear R14
SR R15,R15 Clear R15
MVCL R12,R14 Clear program storage
LA R13,SOCSTG Move address of program stg to R13
USING SOCSTG,R13 Tell assembler about storage
ST R7,SOCSAVEL Save address of lower save area
ST R13,B(R7) Complete save area chain
SOC00200 DS 0H

* Build message for console
* MVC MSGID,MSGIC Initialize first part of message
```

Figure 85. Sample of IMS program as a client (Part 2 of 10)
L R0,SOCTASK        Get task number
CVD R0,DWORK        Convert task number to decimal
UNPK MSGTD,DWORK+5(3) Convert decimal to character
OI MSGTD+4,X'F0'    Clear sign
MVC MSG2D,MSG2CS    Move 'Started' to message
LA R6,MSG          Put text address in R6
MVC MSGLEN,=AL2(MSGTL) Put length of text in msg hdr.
MVC WTOLIST,WTOPROT Move prototype WTO to list form
WTO TEXT=(R6),      Write message to operator X
   MF=(E,WTOLIST)
   * Issue INITAPI Call to connect to interface
   * MVC SOCTASKC(3),=CL3'SOC' Build Task Identifier
   * MVC SOCTASKC+3(5),MSGTD
   * MVC MSG2D,MSG2C1 Move 'INITAPI' to message
   * MVC MAXSOC,=AL2(50) Initialize MAXSOC field
   * MVC ASTCPNAM,=CL8'TCPV3 ' Initialize TCP Name
   * MVC ASCLNAME,=CL8'TCPCLINT' Initialize AS Name
   * CALL EZASOKET,  X
      (INITAPI,MAXSOC,ASIDENT,SOCTASKC,HISOC,ERRNO,  X
      RETCODE), X
   * VL Specify variable parameter list
   * L R6,RETCODE      Check for successful call
   * C R6,='F'0'       Is it less than zero
   * BL SOCERR        Yes, go display error and terminate
   * AIF (NOT &TRACE).TRACE01
   * TRACE ENTRY FOR INITAPI TRACE TYPE = 1
   * LA R6,MSG         Put text address in R6
   * MVC MSGLEN,=AL2(MSGTL) Put length of text in msg hdr.
   * WTO TEXT=(R6),    Write message to operator X
   * MF=(E,WTOLIST)
   .TRACE01 ANOP
   * * Issue GETHOSTID Call to obtain internet address of host
   * * MVC MSG2D,MSG2C8 Move 'GTHSTID' to message
   * * CALL EZASOKET, X
      (GETHOSTID,SERVIADD), X
   * VL Specify Variable parameter list
   * * AIF (NOT &TRACE).TRACE08
   * * TRACE ENTRY FOR GETHOSTID TRACE TYPE = 8
   * * LA R6,MSG       Put text address in R6
   * * MVC MSGLEN,=AL2(MSGTL) Put length of text in msg hdr.
   * * WTO TEXT=(R6),  Write message to operator X
   * * MF=(E,WTOLIST)
   .TRACE08 ANOP
   * * Issue SOCKET Call to obtain a socket descriptor
   * * MVC MSG2D,MSG2C2 Move 'SOCKET' to message

Figure 85. Sample of IMS program as a client (Part 3 of 10)
MVC AF,'F'2' Address Family = Internet
MVC SOCTYPE,'F'1' Type = Stream Sockets
XC PROTO,PROTO Clear protocol field
*
CALL EZASOKET, Issue SOCKET Call X
(SOCKET,AF,SOCTYPE,PROTO,ERRNO,RETCODE), X
VL Specify variable parameter list
*
L R6,RETCODE Check for successful call
C R6,'F'0' Is it less than zero
BL SOCERR Yes, go display error and terminat
AIF (NOT &TRACE).TRACE02
* TRACE ENTRY FOR SOCKET TRACE TYPE = 2
LA R6,MSG Put text address in R6
MVC MSGLEN,=AL2(MSGTL) Put length of text in msg hdr.
WTO TEXT=(R6), Write message to operator X
MF=(E,WTOLIST)
*.TRACE02 ANOP
*
Get socket descriptor number
*
L R6,RETCODE Descriptor number returned
STH R6,SOCDESC Save it
*
Issue CONNECT Command to Connect to Server
*
MVC SSOCAF,'H'2' Set AF=INET
MVC SSOCPORT,SERVPORT Move Port Number
MVC SSOCINET,SERVIADD Move Internet Address of Server
MVC MSG2D,MSG2C4 Move 'CONNECT' to message
*
CALL EZASOKET, Issue CONNECT Call X
(CONNECT,SOCDESC,SERVSOC,ERRNO,RETCODE), X
VL Specify variable parameter list
*
L R6,RETCODE Check for successful call
C R6,'F'0' Is it less than zero
BL SOCERR Yes, go display error and terminat
AIF (NOT &TRACE).TRACE04
* TRACE ENTRY FOR CONNECT TRACE TYPE = 4
LA R6,MSG Put text address in R6
MVC MSGLEN,=AL2(MSGTL) Put length of text in msg hdr.
WTO TEXT=(R6), Write message to operator X
MF=(E,WTOLIST)
*.TRACE04 ANOP
*
Send initial message to server
*
MVC BUFFER(L'MSG1),MSG1 Move Message to Buffer
LA R6,L'MSG1 Get length of message
ST R6,DATALLEN Put length in data field
MVC MSG2D,MSG2C5 Move 'WRITE' to message
*
CALL EZASOKET, Issue WRITE Call X
(WRITE,SOCDESC,DATALLEN,BUFFER,ERRNO,RETCODE), X

Figure 85. Sample of IMS program as a client (Part 4 of 10)
VL
L R6,RETCODE Check for successful call
C R6,=F'0' Is it less than zero
BL SOCERR Yes, go display error and terminate
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='
MVI MSG3S,C+' Move sign
L R6,RETCODE Get return code value
CVD R6,DWORK Convert it to decimal
UNPK MSG4D,DWORK+4(4) Unpack it
OI MSG4D+6,X'F0' Correct the sign
LA R6,MSG Put text address in R6
WTO TEXT=(R6), Write message to operator X
MF=(E,WTOLIST)

.TRACE05 ANOP
*
* Read response to initial message
*
MVC MSG2D,MSG2C6 Move 'READ' to message
LA R6,L'BUFFER Get length of buffer
ST R6,DATALEN Put length in data field
*
CALL EZASOKET, Issue READ Call X
(READ,SOCDESC,DATALEN,BUFFER,ERRNO,RETCODE), X
VL Specify variable parameter list
*
L R6,RETCODE Check for successful call
C R6,=F'0' Is it less than zero
BL SOCERR Yes, go display error and terminate
AIF (NOT &TRACE).TRACE06
* TRACE ENTRY FOR READ TRACE TYPE = 6
MVC MSGLEN,=AL2(MSGTL+18) Put length of text in msg hdr.
MVC MSG3D,ERR3C ' RETCODE='
MVI MSG3S,C+' Move sign
L R6,RETCODE Get return code value
CVD R6,DWORK Convert it to decimal
UNPK MSG4D,DWORK+4(4) Unpack it
OI MSG4D+6,X'F0' Correct the sign
LA R6,MSG Put text address in R6
WTO TEXT=(R6), Write message to operator X
MF=(E,WTOLIST)

.TRACE06 ANOP
*
* Send second message to server
*
MVC BUFFER(L'MSG2),MSG2 Move Message to Buffer
LA R6,L'MSG2 Get length of message
ST R6,DATALEN Put length in data field
MVC MSG2D,MSG2C5 Move 'WRITE' to message
*
CALL EZASOKET, Issue WRITE Call X
(WRITE,SOCDESC,DATALEN,BUFFER,ERRNO,RETCODE), X

Figure 85. Sample of IMS program as a client (Part 5 of 10)
VL

*  
L  R6,RETCODE Check for successful call  
C  R6,=F'0' Is it less than zero  
BL  SOCERR Yes, go display error and terminate

AIF (NOT &TRACE).TRACE15  
* TRACE ENTRY FOR WRITE TRACE TYPE = 5
MVC  MSGLEN,=AL2(MSGTL+18) Put length of text in msg hdr.  
MVC  MSG3D,ERR3C ' RETCODE='  
MVI  MSG3S,C'+.' Move sign  
em  R6,RETCODE Get return code value  
CVD  R6,DWORK Convert it to decimal  
UNPK  MSG4D,DWORK+4(4) Unpack it  
OI  MSG4D+6,X'F0' Correct the sign  
LA  R6,MSG Put text address in R6  
WTO  TEXT=(R6), Write message to operator X  
MF=(E,WTOLIST)

.TRACE15 ANOP  
L  R6,RETCODE Check for successful call  
C  R6,=F'0' Is it less than zero  
BL  SOCERR Yes, go display error and terminate

* Read response to second message
*  
MVC  MSG2D,MSG2C6 Move 'READ' to message  

CALL  EZASOKET, Issue READ Call X  
(READ,SOCDESC,SOCMSGL,BUFFER,ERRNO,RETCODE), X  
VL Specify variable parameter list

*  
L  R6,RETCODE Check for successful call  
C  R6,=F'0' Is it less than zero  
BL  SOCERR Yes, go display error and terminate

AIF (NOT &TRACE).TRACE16  
* TRACE ENTRY FOR READ TRACE TYPE = 6
MVC  MSGLEN,=AL2(MSGTL+18) Put length of text in msg hdr.  
MVC  MSG3D,ERR3C ' RETCODE='  
MVI  MSG3S,C'+.' Move sign  
em  R6,RETCODE Get return code value  
CVD  R6,DWORK Convert it to decimal  
UNPK  MSG4D,DWORK+4(4) Unpack it  
OI  MSG4D+6,X'F0' Correct the sign  
LA  R6,MSG Put text address in R6  
WTO  TEXT=(R6), Write message to operator X  
MF=(E,WTOLIST)

.TRACE16 ANOP  
*  
* Send End message to server
*  
MVC  BUFFER(L'ENDMSG),ENDMSG Move end message to buffer  
LA  R6,L'ENDMSG Get length of message  
ST  R6,SOCMSGL Put length in length field  
MVC  MSG2D,MSG2C5 Move 'WRITE' to message

Figure 85. Sample of IMS program as a client (Part 6 of 10)
CALL EZASOKET, Issue WRITE Call
(WRITE, SOCDESC, SOCMSTG, BUFFER, ERRNO, RETCODE),
VL
*
  L R6,RETCODE    Check for successful 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 WRITE TRACE TYPE = 5
  MVC MSGLEN, =AL2(MSGTL+18) Put length of text in msg hdr.
  MVC MSG3D, ERR3C    ' RETCODE='
  MVI MSG3S, C'+'    Move sign
  L R6,RETCODE      Get return code value
  CVD R6, DWORK      Convert it to decimal
  UNPK MSG4D, DWORK+4(4) Unpack it
  OI MSG4D+6, X'F0'  Correct the sign
  LA R6, MSG        Put text address in R6
  WTO TEXT=(R6),    Write message to operator
  MF=(E,WTOLIST)

.TRACE25 ANOP
*
  Read response to end message
*
  MVC MSG2D, MSG2C6 Move 'READ' to message
*
  CALL EZASOKET, Issue READ Call
  (READ, SOCDESC, SOCMSTG, BUFFER, ERRNO, RETCODE),
  VL Specify variable parameter list
*
  L R6,RETCODE    Check for successful call
  C R6,=F'0'      Is it less than zero
  BL SOCERR       Yes, go display error and terminate
AIF (NOT &TRACE).TRACE26
* TRACE ENTRY FOR READ TRACE TYPE = 6
  MVC MSGLEN, =AL2(MSGTL+18) Put length of text in msg hdr.
  MVC MSG3D, ERR3C    ' RETCODE='
  MVI MSG3S, C'+'    Move sign
  L R6,RETCODE      Get return code value
  CVD R6, DWORK      Convert it to decimal
  UNPK MSG4D, DWORK+4(4) Unpack it
  OI MSG4D+6, X'F0'  Correct the sign
  LA R6, MSG        Put text address in R6
  WTO TEXT=(R6),    Write message to operator
  MF=(E,WTOLIST)

.TRACE26 ANOP
*
  Close socket
*
  MVC MSG20, MSG2C7 Move 'CLOSE' to message
*
  CALL EZASOKET, Issue CLOSE Call
  (CLOSE, SOCDESC, ERRNO, RETCODE),
  VL Specify variable parameter list
*
  L R6,RETCODE    Check for successful call

Figure 85. Sample of IMS program as a client (Part 7 of 10)
C R6,=F'0' Is it less than zero
BL SOCERR Yes, go display error and terminate
AIF (NOT &TRACE).TRACE07
* TRACE ENTRY FOR CLOSE TRACE TYPE = 7
LA R6,MSG Put text address in R6
MVC MSGLEN,=AL2(MSGTL) Put length of text in msg hdr.
WTO TEXT=(R6), Write message to operator
MF=(E,WTOLIST)
.TRACE07 ANOP
* Terminate Connection to API
* CALL EZASOKET, Issue TERMAPI Call
   (TERMAPI), Specify variable parameter list
* Issue console message for task termination
* MVC MSG2D,MSG2CE Move 'Ended' to message
LA R6,MSG Put text address in R6
MVC MSGLEN,=AL2(MSGTL) Put length of text in msg hdr.
WTO TEXT=(R6), Write message to operator
MF=(E,WTOLIST)
* Return to Caller
* L R13,SOCSAVEL
LM R14,R12,12(R13)
BR R14
* Write error message to operator and ABENDSOC1
* SOCERR DS 0H Write error message to operator
MVC ERR1D,MSG1D 'IMSTCPCL, TASK #'
MVC ERR2D,MSG2D Move task number to message
MVC ERR3D,MSG3D Call Type
MVC ERR3S,C'-' Move sign which is always minus
MVC ERR5D,ERR5C ' ' ERRNO= '
L R6,RETCODE Get return code value
CVD R6,DWORK Convert it to decimal
UNPK ERR4D,DWORK+4(4) Unpack it
OI ERR4D+6,X'F0' Correct the sign
L R6,ERRNO Get errno value
CVD R6,DWORK Convert it to decimal
UNPK ERR6D,DWORK+4(4) Unpack it
OI ERR6D+6,X'F0' Correct the sign
LA R6,ERR Put text address in R6
MVC ERRLEN,=AL2(ERRTL) Put length of text in msg hdr.
WTO TEXT=(R6), Write message to operator
MF=(E,WTOLIST)
ABEND DS 0H
DC H'0' Force ABEND
WTOPROT WTO TEXT=, List form of WTO Macro
MF=L

Figure 85. Sample of IMS program as a client (Part 8 of 10)
Figure 85. Sample of IMS program as a client (Part 9 of 10)
Sample server program for IMS MPP client
EZASVAS3 CSECT
EZASVAS3 AMODE ANY
EZASVAS3 RMODE ANY

GBLB &TRACE ASSEMBLER VARIABLE TO CONTROL TRACE GENERATION
&TRACE SETB 1 1=TRACE ON 0=TRACE OFF
GBLB &SUBTR ASSEMBLER VARIABLE TO CONTROL SUBTRACE
&SUBTR SETB 0 1=SUBTRACE ON 0=SUBTRACE OFF

*---------------------------------------------------------------------*
**
* MODULE NAME: EZASVAS3 *
* *
* Copyright: Licensed Materials - Property of IBM *
* *
* "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 CSV1RS 020604 BKELSEY : Support 64K sockets *
* $F1= RBBASE CSV1R11 080612 Herr : Cleaned up >72 lines *
* *
*---------------------------------------------------------------------*

SOC0000 DS 0H
USING *,R15  Tell assembler to use reg 15
B SOC00100  Branch to startup address
DC CL14'SERVEREYECATCH'
ASIDENT DS 0F  Address Space Identifier for initapi
ASTCPNAM DC CL8'TCPV3  ' Name of TCP/IP Address Space

Figure 86. Sample of IMS program as a server (Part 1 of 11)
ASCLNAME DC CLB'CALLSRVER'  Our name as known to TCP/IP
TIMEOUT DS 0F  Timeout value for select
TIMESEC DC F'100'  Timeout value in seconds
TIMEMSEC DC F'0'  Timeout value in milliseconds
BUFLEN EQU 1000  Set length of I/O buffers
R4BASE DC A(SOC0000+4096)
SOC00100 DS 0H  Beginning of program
STM R14,R12,12(R13)  Save callers registers
LR R3,R15  Move base reg to R3
L R4,R4BASE  Add R4 as second base reg
DROP R15  Tell assembler to drop R15 as base
USING SOC0000,R3,R4  Tell assembler to use R3 and R4 as base
LA R6,SOCSSTG  Clear program storage
LA R7,SOCSSTGL
SR R14,R14
SR R15,R15
MVCL R6,R14  Save address of higher save area
LA R7,SOCSAVE  Complete save area chain
ST R7,(R13)  Tell caller where our save area is
LA R13,SOCSAVE  Point R13 at our save area
MVI ENDSW,X'00'  Clear end-of-transmission switch

* Build message for console *
MVC MSG1D,MSG1C  Initialize first part of message
MVC MSGTD,=CL5'00000'  Move subtask number from clientid
MVC MSG2D,MSG2CS  Move 'Started' to message
LA R6,MSG  Put text address in R6
MVC MSGLEN,=AL2(MSGTL)  Put length of text in msg hdr.
MVC WTOLIST,WTOPROT  Move prototype WTO to list form
WTO TEXT=(R6), Write message to operator
MF=(E,WTOLIST)

* Issue INITAPI Call to connect to interface *
MVC SOCTASKC,=CL8'TAS00000'  Give subtask a name
MVC MSG2D,MSG2C00  Move 'INITAPI' to message
MVC MAXSOC,=AL2(50)  Initialize MAXSOC parameter
CALL EZASOKET,
(INITAPI,MAXSOC,ASIDENT,SOCTASKC,HISOC,ERRNO,
RETCODE), X
VL

L R6,RETCODE  Check for successful call
C R6,'F'0'  Is it less than zero
BL SOCERR  Yes, go display error and terminate
AIF (NOT &TRACE).TRACE00

* TRACE ENTRY FOR INITAPI  TRACE TYPE = 0 *
LA R6,MSG  Put text address in R6
MVC MSGLEN,=AL2(MSGTL)  Put length of text in msg hdr.
WTO TEXT=(R6), Write message to operator
MF=(E,WTOLIST)

Figure 86. Sample of IMS program as a server (Part 2 of 11)
**.TRACE00 ANOP**

* Issue SOCKET Call to obtain socket to listen on

* MVC MSG2D,MSG2C25 Move 'SOCKET' to message
  MVC AF,=F'2' Initialize AF to '2' (INET)
  MVC SOCTYPE,=F'1' Specify stream sockets
  MVC PROTO,=F'0' Protocol is ignored for stream

* CALL EZASOKET, Issue SOCKET CALL
  (SOCKET,AF,SOCTYPE,PROTO,ERRNO,RETCODE), X
  VL

* L R6,RETCODE Check for successful call
  C R6,=F'0' Is it less than zero
  BL SOCERR Yes, go display error and terminate

* TRACE ENTRY FOR SOCKET TRACE TYPE = 25
  LA R6,MSG Put text address in R6
  MVC MSGLEN,=AL2(MSGTL) Put length of text in msg hdr.
  WTO TEXT=(R6), Write message to operator X
  MF=(E,WTOLIST)

**.TRACE25 ANOP**

L R0,RETCODE Get descriptor number of socket
STH R0,LISTSOC Save it

* Issue GETHOSTID call to determine our internet address

* MVC MSG2D,MSG2C07 Move 'GETHOSTID' to message

* CALL EZASOKET, Issue GETHOSTID Call
  (GETHOSTID,RETCODE),VL

* AIF (NOT &TRACE).TRACE07

**.TRACE07 ANOP**

L R0,RETCODE Get internet address of host
ST R0,SINETADR Save it

* Issue BIND call to establish port

* MVC MSG2D,MSG2C02 Move 'BIND' to message
  MVC SPORT,=H'5000' Move port number to structure
  MVC SAF,=H'2' Move AF (INET) to structure

* CALL EZASOKET, Issue BIND Call
  (BIND,LISTSOC,SOCKNAME,ERRNO,RETCODE), X
  VL

L R6,RETCODE Check for successful call
C R6,=F'0' Is it less than zero
BL SOCERR Yes, go display error and terminate

Figure 86. Sample of IMS program as a server (Part 3 of 11)
AIF (NOT &TRACE).TRACE02
TRACE ENTRY FOR BIND TRACE TYPE = 02
LA R6,MSG       Put text address in R6
MVC MSGLEN,=AL2(MSGTL)    Put length of text in msg hdr.
WTO TEXT=(R6),        Write message to operator X
MF=(E,WTOLIST)

TRACE02 ANOP

* Issue LISTEN call to establish backlog of connection requests
*
MVC MSG2D,MSG2C13    Move 'LISTEN' to message
MVC BACKLOG,'F'5'    Set backlog to 5
*
CALL EZASOKET, Issue LISTEN Call X
   (LISTEN,LISTSOC,BACKLOG,ERRNO,RETCODE),VL
L R6,RETCODE        Check for successful call
C R6,'F'0'         Is it less than zero
BL SOCERR         Yes, go display error and terminate

AIF (NOT &TRACE).TRACE13
TRACE ENTRY FOR LISTEN TRACe TYPE = 13
LA R6,MSG       Put text address in R6
MVC MSGLEN,=AL2(MSGTL)    Put length of text in msg hdr.
WTO TEXT=(R6),        Write message to operator X
MF=(E,WTOLIST)

TRACE13 ANOP

* Issue SELECT call to wait on connection request
*
MVC MSG2D,MSG2C19    Move 'SELECT' to message
MVC SELSOC,'F'31'    Maximum number of sockets
MVC WSNOMASK,'F'0'    Not checking for writes
MVC ESNDMASK,'F'0'    Not checking for exceptions
LA R0,1        Put 1 in rightmost position of R0
LH R1,LISTSOC    Put listener socket number in R1
SLL R0,0(R1)    Create mask for read
ST R0,RSNDMASK    Put value in mask field
*
CALL EZASOKET, Issue SELECT Call X
   (SELECT,SELSOC,TIMEOUT,RSNDMASK,WSNOMASK,ESNDMASK, X
    RRETMASK,WERETMASK,ERETMASK,ERRNO,RETCODE), X
   VL
L R6,RETCODE        Check for successful call
C R6,'F'0'         Is it less than zero
BL SOCERR         Yes, go display error and terminate

AIF (NOT &TRACE).TRACE19
TRACE ENTRY FOR SELECT TRACE TYPE = 19
LA R6,MSG       Put text address in R6
MVC MSGLEN,=AL2(MSGTL)    Put length of text in msg hdr.
WTO TEXT=(R6),        Write message to operator X
MF=(E,WTOLIST)

TRACE19 ANOP

Figure 86. Sample of IMS program as a server (Part 4 of 11)
* Issue ACCEPT call to accept a new connection
* MVC MSG2D,MSG2C01 Move 'ACCEPT' to message
MVC NS,=F'4' Use socket 4 for connection socket
* CALL EZASOKET, Issue ACCEPT Call
(ACCEPT,LISTSOC,SOCKNAME,ERRNO,RETCODE), VL
L R6,RETCODE Check for successful call
C R6,=F'0' Is it less than zero
BL SOCERR Yes, go display error and terminate
* AIF (NOT &TRACE).TRACE01
* TRACE ENTRY FOR ACCEPT TRACE TYPE = 01
LA R6,MSG Put text address in R6
MVC MSGLEN,=AL2(MSGTL) Put length of text in msg hdr.
WTO TEXT=(R6), Write message to operator X
MF=(E,WTOLIST)
.TRACE01 ANOP
L 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
* LA R6,L'RESPMSG Get length of buffer
ST R6,DATALEN Put length in data field
MVC MSG2D,MSG2C14 Move 'READ' to message
XC FLAGS,FLAGS Clear the FLAGS field
* CALL EZASOKET, Issue READ Call
(READ,CONNSOC,DATALEN,BUFFER,ERRNO,RETCODE),VL
L R6,RETCODE Check for successful call
C R6,=F'0' Is it less than zero
BL SOCERR Yes, go display error and terminate
* AIF (NOT &TRACE).TRAC14A
* TRACE ENTRY FOR READ TRACE TYPE = 14
LA R6,MSG Put text address in R6
MVC MSGLEN,=AL2(MSGTL) Put length of text in msg hdr.
WTO TEXT=(R6), Write message to operator X
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
ST R6,DATALEN Put length in data field
XC FLAGS,FLAGS Clear FLAGS field
MVC MSG2D,MSG2C26 Move 'WRITE' to message
* CALL EZASOKET, Issue WRITE call
(WRITE,CONNSOC,DATALEN,BUFFER,ERRNO,RETCODE),VL
* Figure 86. Sample of IMS program as a server (Part 5 of 11)
L R6,RETCODE Check for successful call
C R6,=F'0' Is it less than zero
BL SOCCERR Yes, go display error and terminate
AIF (NOT &TRACE).TRAC26A

* TRACE ENTRY FOR WRITE TRACE TYPE = 22
LA R6,MSG Put text address in R6
MVC MSGLEN,=AL2(MSGTL) Put length of text in msg hdr.
WTO TEXT=(R6), Write message to operator X
MF=(E,WTOLIST)

.TRAC26A ANOP
SOC0300 DS 0H
*
* Read Message from Client
*
MVC MSG2D,MSG2C14 Move 'READ' to message
LA R0,L'BUFFER Get length of buffer
ST R0,DATALEN Use it for data length
XC FLAGS,FLAGS Clear FLAGS field
*
CALL EZASOKET,
(READ,CONNSOC,DATALEN,BUFFER,ERRNO,RETCODE),VL
*
L R6,RETCODE Check for successful call
C R6,=F'0' Is it less than zero
BNH SOCCERR Yes, go display error and terminate
AIF (NOT &TRACE).TRAC14B

* TRACE ENTRY FOR RECV TRACE TYPE = 14
LA R6,MSG Put text address in R6
MVC MSGLEN,=AL2(MSGTL) Put length of text in msg hdr.
WTO TEXT=(R6), Write message to operator X
MF=(E,WTOLIST)

.TRAC14B ANOP
CLC BUFFER(3),=CL3'END' Was this last record
BNE SOC0350 No
MVI ENDSW,C'E' Yes, set end-of-transmission switch

SOC0350 DS 0H
*
* Send Response to Client
*
MVC MSG2D,MSG2C26 Move 'WRITE' to message
MVC DATALEN,RETCODE Get message length from previous call
XC FLAGS,FLAGS Clear FLAGS field
*
CALL EZASOKET,
(WRITE,CONNSOC,DATALEN,BUFFER,ERRNO,RETCODE),VL
*
L R6,RETCODE Check for successful call
C R6,=F'0' Is it less than zero
BNH SOCCERR Yes, go display error and terminate
AIF (NOT &TRACE).TRAC26B

* TRACE ENTRY FOR SEND TRACE TYPE = 26
LA R6,MSG Put text address in R6
MVC MSGLEN,=AL2(MSGTL) Put length of text in msg hdr.
WTO TEXT=(R6), Write message to operator X
MF=(E,WTOLIST)

Figure 86. Sample of IMS program as a server (Part 6 of 11)
Figure 86. Sample of IMS program as a server (Part 7 of 11)
Write error message to operator

* Write error message to operator

SOCERR DS 0H
MVC ERR1D,MSG1D 'SERVER, TASK #'
MVC ERRTD,MSGTD Move task number to message
MVC ERR2D,MSG2D Call Type
MVC ERR3D,MSG3C ' RETCODE= '
MVI ERR3S,C'-'
MVC ERR5D,MSG5C ' ERRNO= '
L R6,RETCODE Get return code value
CVD R6,DWORK Convert it to decimal
UNPK ERR4D,DWORK+4(4) Unpack it
OI ERR4D+6,X'F0' Correct the sign
L R6,ERRNO Get errno value
CVD R6,DWORK Convert it to decimal
UNPK ERR6D,DWORK+4(4) Unpack it
OI ERR6D+6,X'F0' Correct the sign
LA R6,ERR Put text address in R6
MVC ERRLEN,=AL2(ERRTL) Put length of text in msg hdr.
WTO TEXT=(R6), Write message to operator X
MF=(E,WTOLIST)

* Return to Caller

* L R13,SOCSAVEH
* LM R14,R12,12(R13)
* BR R14
ABEND DS 0H
DC H'0' Force ABEND

*---------------------------------------------------------------------*
* Constants *
*---------------------------------------------------------------------*
WTOPROT WTO TEXT=, List form of WTO Macro X
MF=L
WTOPROTL EQU =-WTOPROT Length of WTO Prototype
MSG1C DC CL17'SERVER, TASK #'
MSG2CS DC CL8' STARTED'
MSG2CE DC CL8' ENDED '
ERR3C DC CL10' RETCODE= '
ERR5C DC CL8' ERRNO= '
MSG2C00 DC CL8' INITAPI'
MSG2C01 DC CL8' ACCEPT '
MSG2C02 DC CL8' BIND '
MSG2C03 DC CL8' CLOSE '
MSG2C03A DC CL8' CLOSE2 '
MSG2C07 DC CL8' GTHSTID'
MSG2C13 DC CL8' LISTEN '
MSG2C14 DC CL8' READ '
MSG2C19 DC CL8' SELECT '
MSG2C25 DC CL8' SOCKET '
MSG2C26 DC CL8' WRITE '
MSG2C32 DC CL8' TAKESKTI'

Figure 86. Sample of IMS program as a server (Part 8 of 11)
RESPMGS DC CL50'FIRST RESPONSE FROM SERVER '

*---------------------------------------------------------------------*
* Constants used for call types                                      *
*---------------------------------------------------------------------*
INITAPI DC CL16'INITAPI'
BIND DC CL16'BIND'
LISTEN DC CL16'LISTEN'
ACCEPT DC CL16'ACCEPT'
READ DC CL16'READ'
SELECT DC CL16'SELECT'
WRITE DC CL16'WRITE'
SOCKET DC CL16'SOCKET'
GETHSTID DC CL16'GETHSTID'
TERMAPI DC CL16'TERMAPI'

*---------------------------------------------------------------------*
* Program Storage Area                                                *
*---------------------------------------------------------------------*
SOCSTG DS 0F PROGRAM STORAGE
SOCSAVE DS 0F Save Area
SOCSAVE1 DS F Word for high-level languages
SOCSAVEH DS F Address of previous save area
SOCSAVEL DS F Address of next save area
SOCSAV14 DS F Reg 14
SOCSAV15 DS F Reg 15
SOCSAV0 DS F Reg 0
SOCSAV1 DS F Reg 1
SOCSAV2 DS F Reg 2
SOCSAV3 DS F Reg 3
SOCSAV4 DS F Reg 4
SOCSAV5 DS F Reg 5
SOCSAV6 DS F Reg 6
SOCSAV7 DS F Reg 7
SOCSAV8 DS F Reg 8
SOCSAV9 DS F Reg 9
SOCSAV10 DS F Reg 10
SOCSAV11 DS F Reg 11
SOCSAV12 DS F Reg 12
SOCSAV13 DS F Reg 13
PARMADDR DS F Address of parameter list
GWAADDR DS F Address of Global Work Area
TIEADDR DS F Address of Task Information Element
LISTSOC DS H Socket number used for listen
CONNROC DS H Socket number created by accept
SOCMSGN DS F Number of messages to be exchanged
SOCMSGL DS F Length of messages to be exchanged
SOCTASKC DS CL8 Character task identifier
HISOC DS F Highest socket descriptor available
SERVLLEN DS H
SERVSOC DS 0F Socket Address of Server
SERVAF DS H Address Family of Server = 2
SERVPORF DS H Port Address of Server
SERVIAADD DS F Internet Address of Server
ENDSW DS C End of transmission switch
MSG DS 0F Message area

Figure 86. Sample of IMS program as a server (Part 9 of 11)
Figure 86. Sample of IMS program as a server (Part 10 of 11)
**WTO output from sample program**

**Client Output**

- 13.29.18 JOB00084 IEF403I SOCCALLS - STARTED - TIME=13.29.18
- 13.29.18 JOB00084 +SERVER, TASK # 00000 STARTED
- 13.29.19 JOB00084 +SERVER, TASK # 00000 INITAPI
- 13.29.19 JOB00084 +SERVER, TASK # 00000 SOCKET
- 13.29.19 JOB00084 +SERVER, TASK # 00000 GTHSTID
- 13.29.19 JOB00084 +SERVER, TASK # 00000 BIND
- 13.29.20 JOB00084 +SERVER, TASK # 00000 LISTEN
- 13.29.41 JOB00084 +SERVER, TASK # 00000 SELECT
- 13.29.41 JOB00084 +SERVER, TASK # 00000 ACCEPT
- 13.29.41 JOB00084 +SERVER, TASK # 00000 READ
- 13.29.41 JOB00084 +SERVER, TASK # 00000 WRITE
- 13.29.41 JOB00084 +SERVER, TASK # 00000 READ
- 13.29.41 JOB00084 +SERVER, TASK # 00000 WRITE
- 13.29.41 JOB00084 +SERVER, TASK # 00000 READ
- 13.29.42 JOB00084 +SERVER, TASK # 00000 CLOSE
- 13.29.42 JOB00084 +SERVER, TASK # 00000 CLOSE2
- 13.29.42 JOB00084 +SERVER, TASK # 00000 ENDED

**Server Output**

- 13.27.45 JOB00082 IEF403I MESSAGE - STARTED - TIME=13.27.45
- 13.29.41 JOB00082 +IMSTCPCL, TASK # 00000 STARTED
- 13.29.41 JOB00082 +IMSTCPCL, TASK # 00000 INITAPI
- 13.29.41 JOB00082 +IMSTCPCL, TASK # 00000 SOCKET
- 13.29.41 JOB00082 +IMSTCPCL, TASK # 00000 GTHSTID
- 13.29.41 JOB00082 +IMSTCPCL, TASK # 00000 CONNECT
- 13.29.41 JOB00082 +IMSTCPCL, TASK # 00000 WRITE RETCODE= +0000016
- 13.29.41 JOB00082 +IMSTCPCL, TASK # 00000 READ RETCODE= +0000050
- 13.29.41 JOB00082 +IMSTCPCL, TASK # 00000 WRITE RETCODE= +0000016
- 13.29.41 JOB00082 +IMSTCPCL, TASK # 00000 READ RETCODE= +0000016

---

Figure 86. Sample of IMS program as a server (Part 11 of 11)
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 6. Sockets ERRNOs

<table>
<thead>
<tr>
<th>Error number</th>
<th>Message name</th>
<th>Socket type</th>
<th>Error description</th>
<th>Programmer’s response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EAI_NONAME</td>
<td>GETADDRINFO</td>
<td>NODE or HOST cannot be found.</td>
<td>Ensure the NODE or HOST name can be resolved.</td>
</tr>
<tr>
<td>1</td>
<td>EDOM</td>
<td>All</td>
<td>Argument too large</td>
<td>Check parameter values of the function call.</td>
</tr>
<tr>
<td>1</td>
<td>EPERM</td>
<td>All</td>
<td>Permission is denied. No owner exists.</td>
<td>Check that TCP/IP is still active; check protocol value of socket () call.</td>
</tr>
<tr>
<td>1</td>
<td>EPERM</td>
<td>IOCTL (SIOCTTLSCTL requesting both TTLS_INIT_CONNECTION and TTLS_RESET_SESSION or both TTLS_INIT_CONNECTION and TTLS_RESET_CIPHER)</td>
<td>The combination of requests specified is not permitted.</td>
<td>Request TTLS_RESET_SESSION and TTLS_RESET_CIPHER only when TTLS_INIT_CONNECTION has been previously requested for the connection.</td>
</tr>
</tbody>
</table>
## ERRNOs

Table 6. Sockets ERRNOs (continued)

<table>
<thead>
<tr>
<th>Error number</th>
<th>Message name</th>
<th>Socket type</th>
<th>Error description</th>
<th>Programmer's response</th>
</tr>
</thead>
</table>
| 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 or TTLS_STOP_CONNECTION  
  - The TTLS_STOP_CONNECTION option was requested along with TTLS_RESET_SESSION or TTLS_RESET_CIPHER  
  - The TTLS_ALLOW_HSTIMEOUT option was requested without TTLS_INIT_CONNECTION | Request TTLS_RESET_SESSION and TTLS_RESET_CIPHER only when TTLS_INIT_CONNECTION and TTLS_STOP_CONNECTION are not requested. Always request TTLS_INIT_CONNECTION when TTLS_ALLOW_HSTIMEOUT is requested. Use separate SIOCTTLSCTL ioctls to request TTLS_INIT_CONNECTION and TTLS_STOP_CONNECTION. |
| 2            | EAI_AGAIN    | FREADDRINFO, 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. |
| 3            | EAI_FAIL     | FREADDRINFO, GETADDRINFO, GETNAMEINFO | This is an unrecoverable error. NODELEN, HOSTLEN, or SERVLEN is incorrect. For FREADDRINFO, 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. |
Table 6. Sockets ERRNOs (continued)

<table>
<thead>
<tr>
<th>Error number</th>
<th>Message name</th>
<th>Socket type</th>
<th>Error description</th>
<th>Programmer's response</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>EAI_OVERFLOW</td>
<td>GETNAMEINFO</td>
<td>The output buffer for the host name or service name was too small.</td>
<td>Increase the size of the buffer to 255 characters, which is the maximum size permitted.</td>
</tr>
<tr>
<td>4</td>
<td>EINTR</td>
<td>All</td>
<td>A system call was interrupted.</td>
<td>Check that the socket connection and TCP/IP are still active.</td>
</tr>
<tr>
<td>5</td>
<td>EAI_FAMILY</td>
<td>GETADDRINFO</td>
<td>The AF or the FAMILY is incorrect.</td>
<td>Correct the AF or the FAMILY.</td>
</tr>
<tr>
<td>5</td>
<td>EIO</td>
<td>All</td>
<td>An I/O error occurred.</td>
<td>Check status and contents of source database if this occurred during a file access.</td>
</tr>
<tr>
<td>6</td>
<td>EAI_MEMORY</td>
<td>GETADDRINFO</td>
<td>The resolver cannot obtain storage to process the host name.</td>
<td>Contact your system administrator.</td>
</tr>
<tr>
<td>6</td>
<td>ENXIO</td>
<td>All</td>
<td>The device or driver was not found.</td>
<td>Check status of the device attempting to access.</td>
</tr>
<tr>
<td>7</td>
<td>E2BIG</td>
<td>All</td>
<td>The argument list is too long.</td>
<td>Check the number of function parameters.</td>
</tr>
<tr>
<td>7</td>
<td>EAI_BADFLAGS</td>
<td>GETADDRINFO</td>
<td>FLAGS has an incorrect value.</td>
<td>Correct the FLAGS.</td>
</tr>
<tr>
<td>8</td>
<td>EAI_SERVICE</td>
<td>GETADDRINFO</td>
<td>The SERVICE was not recognized for the specified socket type.</td>
<td>Correct the SERVICE.</td>
</tr>
<tr>
<td>8</td>
<td>ENOEXEC</td>
<td>All</td>
<td>An EXEC format error occurred.</td>
<td>Check that the target module on an exec call is a valid executable module.</td>
</tr>
<tr>
<td>9</td>
<td>EAI_SOCKTYPE</td>
<td>GETADDRINFO</td>
<td>The SOCTYPE was not recognized.</td>
<td>Correct the SOCTYPE.</td>
</tr>
<tr>
<td>9</td>
<td>EBADF</td>
<td>All</td>
<td>An incorrect socket descriptor was specified.</td>
<td>Check socket descriptor value. It might be currently not in use or incorrect.</td>
</tr>
<tr>
<td>9</td>
<td>EBADF</td>
<td>GIVESOCKET</td>
<td>The socket has already been given. The socket domain is not AF_INET or AF_INET6.</td>
<td>Check the validity of function parameters.</td>
</tr>
<tr>
<td>9</td>
<td>EBADF</td>
<td>SELECT</td>
<td>One of the specified descriptor sets is an incorrect socket descriptor.</td>
<td>Check the validity of function parameters.</td>
</tr>
<tr>
<td>9</td>
<td>EBADF</td>
<td>TAKESOCKET</td>
<td>The socket has already been taken.</td>
<td>Check the validity of function parameters.</td>
</tr>
<tr>
<td>9</td>
<td>EAI_SOCKTYPE</td>
<td>GETADDRINFO</td>
<td>The SOCTYPE was not recognized.</td>
<td>Correct the SOCTYPE.</td>
</tr>
<tr>
<td>10</td>
<td>ECHILD</td>
<td>All</td>
<td>There are no children.</td>
<td>Check if created subtasks still exist.</td>
</tr>
<tr>
<td>11</td>
<td>EAGAIN</td>
<td>All</td>
<td>There are no more processes.</td>
<td>Retry the operation. Data or condition might not be available at this time.</td>
</tr>
</tbody>
</table>
## ERRNOs

**Table 6. Sockets ERRNOs (continued)**

<table>
<thead>
<tr>
<th>Error number</th>
<th>Message name</th>
<th>Socket type</th>
<th>Error description</th>
<th>Programmer's response</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>ENOMEM</td>
<td>All</td>
<td>There is not enough storage.</td>
<td>Check the validity of function parameters.</td>
</tr>
<tr>
<td>13</td>
<td>EACCES</td>
<td>All</td>
<td>Permission denied, caller not authorized.</td>
<td>Check access authority of file.</td>
</tr>
<tr>
<td>13</td>
<td>EACCES</td>
<td>Takesocket</td>
<td>The other application (listener) did not give the socket to your application. Permission denied, caller not authorized.</td>
<td>Check access authority of file.</td>
</tr>
<tr>
<td>13</td>
<td>EACCES</td>
<td>IOCTL (SIOCTTLSCTL)</td>
<td>The IOCTL is requesting a function that requires that the socket be mapped to policy that specifies ApplicationControlled On.</td>
<td>Check policy and add ApplicationControlled On if the application should be permitted to issue the controlled SIOCTTLSCTL functions.</td>
</tr>
<tr>
<td>14</td>
<td>EFAULT</td>
<td>All</td>
<td>An incorrect storage address or length was specified.</td>
<td>Check the validity of function parameters.</td>
</tr>
<tr>
<td>14</td>
<td>EFAULT</td>
<td>All</td>
<td>All EZASMI macros when using an asynchronous exit routine. The exit routine has abnormally ended (ABEND condition).</td>
<td>Correct the error in the routine's code. Add an ESTAE routine to the exit.</td>
</tr>
<tr>
<td>14</td>
<td>EFAULT</td>
<td>IOCTL (SIOCSAPPLDATA)</td>
<td>An abend occurred while attempting to copy the SetADcontainer structure from the address provided in the SetAD_ptr field.</td>
<td>Check the validity of function parameters.</td>
</tr>
<tr>
<td>15</td>
<td>ENOTBLK</td>
<td>All</td>
<td>A block device is required.</td>
<td>Check device status and characteristics.</td>
</tr>
<tr>
<td>16</td>
<td>EBUSY</td>
<td>All</td>
<td>Listen has already been called for this socket. Device or file to be accessed is busy.</td>
<td>Check if the device or file is in use.</td>
</tr>
<tr>
<td>17</td>
<td>EEXIST</td>
<td>All</td>
<td>The data set exists.</td>
<td>Remove or rename existing file.</td>
</tr>
<tr>
<td>18</td>
<td>EXDEV</td>
<td>All</td>
<td>This is a cross-device link. A link to a file on another file system was attempted.</td>
<td>Check file permissions.</td>
</tr>
<tr>
<td>19</td>
<td>ENODEV</td>
<td>All</td>
<td>The specified device does not exist.</td>
<td>Check file name and if it exists.</td>
</tr>
<tr>
<td>20</td>
<td>ENOTDIR</td>
<td>All</td>
<td>The specified directory is not a directory.</td>
<td>Use a valid file that is a directory.</td>
</tr>
<tr>
<td>21</td>
<td>EISDIR</td>
<td>All</td>
<td>The specified directory is a directory.</td>
<td>Use a valid file that is not a directory.</td>
</tr>
<tr>
<td>22</td>
<td>EINVAL</td>
<td>All types</td>
<td>An incorrect argument was specified.</td>
<td>Check the validity of function parameters.</td>
</tr>
<tr>
<td>22</td>
<td>EINVAL</td>
<td>Multicast Source filter APIs</td>
<td>Mix of any-source, source-specific or full-state APIs</td>
<td>Specify the correct type of APIs.</td>
</tr>
<tr>
<td>Error number</td>
<td>Message name</td>
<td>Socket type</td>
<td>Error description</td>
<td>Programmer’s response</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------</td>
<td>-------------</td>
<td>------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>22</td>
<td>EINVAL</td>
<td>MCAST_JOIN_GROUP, MCAST_JOIN_SOURCE_GROUP, MCAST_LEAVE_GROUP, MCAST_LEAVE_SOURCE_GROUP, MCAST_UNBLOCK_SOURCE, SIOCGMSFILTER, SIOCSMSFILTER</td>
<td>The socket address family or the socket length of the input multicast group or the source IP address is not correct.</td>
<td>Specify the correct value.</td>
</tr>
<tr>
<td>22</td>
<td>EINVAL</td>
<td>SIOCSMSFILTER, SIOCSIPMSFILTER</td>
<td>The specified filter mode is not correct.</td>
<td>Specify the correct value.</td>
</tr>
<tr>
<td>23</td>
<td>ENFILE</td>
<td>All</td>
<td>Data set table overflow occurred.</td>
<td>Reduce the number of open files.</td>
</tr>
<tr>
<td>24</td>
<td>EMFILE</td>
<td>All</td>
<td>The socket descriptor table is full.</td>
<td>Check the maximum sockets specified in MAXDESC().</td>
</tr>
<tr>
<td>25</td>
<td>ENOTTY</td>
<td>All</td>
<td>An incorrect device call was specified.</td>
<td>Check specified IOCTL() values.</td>
</tr>
<tr>
<td>26</td>
<td>ETXTBSY</td>
<td>All</td>
<td>A text data set is busy.</td>
<td>Check the current use of the file.</td>
</tr>
<tr>
<td>27</td>
<td>EFBIG</td>
<td>All</td>
<td>The specified data set is too large.</td>
<td>Check size of accessed dataset.</td>
</tr>
<tr>
<td>28</td>
<td>ENOSPC</td>
<td>All</td>
<td>There is no space left on the device.</td>
<td>Increase the size of accessed file.</td>
</tr>
<tr>
<td>29</td>
<td>EPIPE</td>
<td>All</td>
<td>An incorrect seek was attempted.</td>
<td>Check the offset parameter for seek operation.</td>
</tr>
<tr>
<td>30</td>
<td>EROFS</td>
<td>All</td>
<td>The data set system is Read only.</td>
<td>Access data set for read only operation.</td>
</tr>
<tr>
<td>31</td>
<td>EMLINK</td>
<td>All</td>
<td>There are too many links.</td>
<td>Reduce the number of links to the accessed file.</td>
</tr>
<tr>
<td>32</td>
<td>EPIPE</td>
<td>All</td>
<td>The connection is broken. For socket write/send, peer has shut down one or both directions.</td>
<td>Reconnect with the peer.</td>
</tr>
<tr>
<td>32</td>
<td>EPIPE</td>
<td>IOCTL (SIOCTTLSCTL requesting TTLS_INIT_CONNECTION, TTLS_RESET_CIPHER, or TTLS_STOP_CONNECTION)</td>
<td>The TCP connection is not in the established state.</td>
<td>Issue the SIOCTTLSCTL IOCTL when the socket is connected.</td>
</tr>
<tr>
<td>33</td>
<td>EDOM</td>
<td>All</td>
<td>The specified argument is too large.</td>
<td>Check and correct function parameters.</td>
</tr>
<tr>
<td>34</td>
<td>ERANGE</td>
<td>All</td>
<td>The result is too large.</td>
<td>Check function parameter values.</td>
</tr>
<tr>
<td>35</td>
<td>EWOULDBLOCK</td>
<td>Accept</td>
<td>The socket is in nonblocking mode and connections are not queued. This is not an error condition.</td>
<td>Reissue Accept().</td>
</tr>
</tbody>
</table>
### ERRNOs

**Table 6. Sockets ERRNOs (continued)**

<table>
<thead>
<tr>
<th>Error number</th>
<th>Message name</th>
<th>Socket type</th>
<th>Error description</th>
<th>Programmer's response</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>EWOULDBLOCK</td>
<td>IOCTL (SIOCTTLSCTL)</td>
<td>The handshake is in progress and the socket is a nonblocking socket.</td>
<td>For a nonblocking socket, you can wait for the handshake to complete by issuing Select or Poll for Socket Writable.</td>
</tr>
<tr>
<td>35</td>
<td>EWOULDBLOCK</td>
<td>Read Recvfrom</td>
<td>The socket is in nonblocking mode and read data is not available. This is not an error condition.</td>
<td>Issue a select on the socket to determine when data is available to be read or reissue the Read()/Recvfrom().</td>
</tr>
<tr>
<td>35</td>
<td>EWOULDBLOCK</td>
<td>All receive calls (RECV, RECVMSG, RECVFROM, READY, READ), when the socket is set with the SO_RCVTIMEO socket option</td>
<td>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.</td>
<td>The application should reissue the receive call.</td>
</tr>
<tr>
<td>35</td>
<td>EWOULDBLOCK</td>
<td>Send Sendto Write</td>
<td>The socket is in nonblocking mode and buffers are not available.</td>
<td>Issue a select on the socket to determine when data is available to be written or reissue the Send(), Sendto(), or Write().</td>
</tr>
<tr>
<td>35</td>
<td>EWOULDBLOCK</td>
<td>All send calls (SEND, SENDMSG, SENDTO, WRITEV, WRITE), when the socket is set with the SO_SNDTIMEO socket option</td>
<td>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.</td>
<td>The application should reissue the send call.</td>
</tr>
<tr>
<td>36</td>
<td>EINPROGRESS</td>
<td>Connect</td>
<td>The socket is marked nonblocking and the connection cannot be completed immediately. This is not an error condition.</td>
<td>See the Connect() description for possible responses.</td>
</tr>
<tr>
<td>36</td>
<td>EINPROGRESS</td>
<td>IOCTL (SIOCTTLSCTL requesting TTLS_INIT_CONNECTION or TTLS_STOP_CONNECTION)</td>
<td>The handshake is already in progress and the socket is a nonblocking socket.</td>
<td>For a nonblocking socket, you can wait for the handshake to complete by issuing Select or Poll for Socket Writable.</td>
</tr>
<tr>
<td>37</td>
<td>EALREADY</td>
<td>Connect</td>
<td>The socket is marked nonblocking and the previous connection has not been completed.</td>
<td>Reissue Connect().</td>
</tr>
<tr>
<td>37</td>
<td>EALREADY</td>
<td>IOCTL (SIOCTTLSCTL requesting TTLS_INIT_CONNECTION or TTLS_STOP_CONNECTION)</td>
<td>For TTLS_INIT_CONNECTION, the socket is already secure. For TTLS_STOP_CONNECTION, the socket is not secure.</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Table 6. Sockets ERRNOs (continued)

<table>
<thead>
<tr>
<th>Error number</th>
<th>Message name</th>
<th>Socket type</th>
<th>Error description</th>
<th>Programmer’s response</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>EALREADY</td>
<td>Maxdesc</td>
<td>A socket has already been created calling Maxdesc() or multiple calls to Maxdesc().</td>
<td>Issue Getablesize() to query it.</td>
</tr>
<tr>
<td>37</td>
<td>EALREADY</td>
<td>Setibmopt</td>
<td>A connection already exists to a TCP/IP image. A call to SETIBMOPR (IBMTCPI_IMAGE), has already been made.</td>
<td>Only call Setibmopt() once.</td>
</tr>
<tr>
<td>38</td>
<td>ENOTSOCK</td>
<td>All</td>
<td>A socket operation was requested on a nonsocket connection. The value for socket descriptor was not valid.</td>
<td>Correct the socket descriptor value and reissue the function call.</td>
</tr>
<tr>
<td>39</td>
<td>EDESTADDRREQ</td>
<td>All</td>
<td>A destination address is required.</td>
<td>Fill in the destination field in the correct parameter and reissue the function call.</td>
</tr>
<tr>
<td>40</td>
<td>EMSGSIZE</td>
<td>Sendto, Sendmsg, Send Write for Datagram (UDP) or RAW sockets</td>
<td>The message is too long. It exceeds the IP limit of 64K or the limit set by the setsockopt() call.</td>
<td>Either correct the length parameter, or send the message in smaller pieces.</td>
</tr>
<tr>
<td>41</td>
<td>EPROTOTYPE</td>
<td>All</td>
<td>The specified protocol type is incorrect for this socket.</td>
<td>Correct the protocol type parameter.</td>
</tr>
<tr>
<td>41</td>
<td>EPROTOTYPE</td>
<td>IOCTL (SIOCTTLSCTL)</td>
<td>Socket is not a TCP socket.</td>
<td>Issue the SIOCTTLSCTL IOCTL on TCP sockets only.</td>
</tr>
<tr>
<td>41</td>
<td>EPROTOTYPE</td>
<td>IOCTL (SIOCSAPPLDATA)</td>
<td>The request was not successful. The socket is not a stream (TCP) socket.</td>
<td>Issue the SIOCSAPPLDATA IOCTL on TCP sockets only.</td>
</tr>
<tr>
<td>42</td>
<td>ENOPROTOOPT</td>
<td>Getsockopt, Setsockopt</td>
<td>The socket option specified is incorrect or the level is not SOL_SOCKET. Either the level or the specified optname is not supported.</td>
<td>Correct the level or optname.</td>
</tr>
<tr>
<td>42</td>
<td>ENOPROTOOPT</td>
<td>Getibmssockopt, Setibmsockopt</td>
<td>Either the level or the specified optname is not supported.</td>
<td>Correct the level or optname.</td>
</tr>
<tr>
<td>43</td>
<td>EPROTONOSUPPORT</td>
<td>Socket</td>
<td>The specified protocol is not supported.</td>
<td>Correct the protocol parameter.</td>
</tr>
<tr>
<td>44</td>
<td>ESOCKTNOSUPPORT</td>
<td>All</td>
<td>The specified socket type is not supported.</td>
<td>Correct the socket type parameter.</td>
</tr>
<tr>
<td>45</td>
<td>EOPNOTSUPP</td>
<td>IOCTL</td>
<td>The specified IOCTL command is not supported by this socket API.</td>
<td>Correct the IOCTL COMMAND.</td>
</tr>
<tr>
<td>45</td>
<td>EOPNOTSUPP</td>
<td>GETSOCKOPT</td>
<td>The specified GETSOCKOPT OPTNAME option is not supported by this socket API.</td>
<td>Correct the GETSOCKOPT OPTNAME option.</td>
</tr>
</tbody>
</table>
### Table 6. Sockets ERRNOs (continued)

<table>
<thead>
<tr>
<th>Error number</th>
<th>Message name</th>
<th>Socket type</th>
<th>Error description</th>
<th>Programmer’s response</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>EOPNOTSUPP</td>
<td>IOCTL</td>
<td>Mapped policy indicates that AT-TLS is not enabled for the connection.</td>
<td>Modify the policy to enable AT-TLS for the connection.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IOCTL</td>
<td>The specified flags are not supported on this socket type or protocol.</td>
<td>Correct the FLAG.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Accept</td>
<td>The selected socket is not a stream socket.</td>
<td>Use a valid socket.</td>
</tr>
</tbody>
</table>
| 45           | EOPNOTSUPP   | Listen      | The socket does not support the Listen call. | Change the type on the Socket() call when the socket was created.  
Listen() only supports a socket type of SOCK_STREAM. |
|              |              | Getibmopt   | 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(). |
| 46           | EAFNOSUPPORT | 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 | Bind        | 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. |
|              |              | Connect     | 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 | Getclient   | The socket specified by the socket descriptor parameter was not created in the AF_INET domain. | Correct by ensuring a Listen() was not issued before the Connect(). |
|              |              | 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 6. Sockets ERRNOs (continued)

<table>
<thead>
<tr>
<th>Error number</th>
<th>Message name</th>
<th>Socket type</th>
<th>Error description</th>
<th>Programmer’s response</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>EADDRINUSE</td>
<td>Bind</td>
<td>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.</td>
<td>If you want to reuse the same address, use Setsockopt() with SO_REUSEADDR. Refer to 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.</td>
</tr>
<tr>
<td>48</td>
<td>EADDRINUSE</td>
<td>IP_ADD_MEMBERSHIP, IP_ADD_SOURCE_MEMBERSHIP, IPV6_JOIN_GROUP, MCAST_JOIN_GROUP, MCAST_JOIN_SOURCE_GROUP</td>
<td>The specified multicast address and interface address (or interface index) pair is already in use.</td>
<td>Correct the specified multicast address, interface address, or interface index.</td>
</tr>
<tr>
<td>49</td>
<td>EADDRNOTAVAIL</td>
<td>Bind</td>
<td>The specified address is incorrect for this host.</td>
<td>Correct the function address parameter.</td>
</tr>
<tr>
<td>49</td>
<td>EADDRNOTAVAIL</td>
<td>Connect</td>
<td>The calling host cannot reach the specified destination.</td>
<td>Correct the function address parameter.</td>
</tr>
<tr>
<td>49</td>
<td>EADDRNOTAVAIL</td>
<td>Multicast APIs</td>
<td>The specified multicast address, interface address, or interface index is not correct.</td>
<td>Correct the specified address.</td>
</tr>
<tr>
<td>49</td>
<td>EADDRNOTAVAIL</td>
<td>IP_BLOCK_SOURCE, IP_ADD_SOURCE_MEMBERSHIP, MCAST_BLOCK_SOURCE, MCAST_JOIN_SOURCE_GROUP</td>
<td>A duplicate source IP address is specified on the multicast group and interface pair.</td>
<td>Correct the specified source IP address.</td>
</tr>
<tr>
<td>49</td>
<td>EADDRNOTAVAIL</td>
<td>IP_UNBLOCK_SOURCE, IP_DROP_SOURCE_MEMBERSHIP, MCAST_UNBLOCK_SOURCE, MCAST_LEAVE_SOURCE_GROUP</td>
<td>A previously blocked source multicast group cannot be found.</td>
<td>Correct the specified address.</td>
</tr>
<tr>
<td>50</td>
<td>ENETDOWN</td>
<td>All</td>
<td>The network is down.</td>
<td>Retry when the connection path is up.</td>
</tr>
<tr>
<td>51</td>
<td>ENETUNREACH</td>
<td>Connect</td>
<td>The network cannot be reached.</td>
<td>Ensure that the target application is active.</td>
</tr>
<tr>
<td>52</td>
<td>ENETRESET</td>
<td>All</td>
<td>The network dropped a connection on a reset.</td>
<td>Reestablish the connection between the applications.</td>
</tr>
<tr>
<td>53</td>
<td>ECONNABORTED</td>
<td>All</td>
<td>The software caused a connection abend.</td>
<td>Reestablish the connection between the applications.</td>
</tr>
<tr>
<td>54</td>
<td>ECONNRESET</td>
<td>All</td>
<td>The connection to the destination host is not available.</td>
<td>N/A</td>
</tr>
<tr>
<td>Error number</td>
<td>Message name</td>
<td>Socket type</td>
<td>Error description</td>
<td>Programmer’s response</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>54</td>
<td>ECONNRESET</td>
<td>Send Write</td>
<td>The connection to the destination host is not available.</td>
<td>The socket is closing. Issue Send() or Write() before closing the socket.</td>
</tr>
<tr>
<td>55</td>
<td>ENOBUFS</td>
<td>All</td>
<td>No buffer space is available.</td>
<td>Check the application for massive storage allocation call.</td>
</tr>
<tr>
<td>55</td>
<td>ENOBUFS</td>
<td>Accept</td>
<td>Not enough buffer space is available to create the new socket.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>55</td>
<td>ENOBUFS</td>
<td>Send Sendto Write</td>
<td>Not enough buffer space is available to send the new message.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>55</td>
<td>ENOBUFS</td>
<td>IOCTL</td>
<td>The buffer size provided is too small.</td>
<td>For TTLS_Version1 use the returned certificate length to allocate a larger buffer and reissue IOCTL with the larger buffer.</td>
</tr>
<tr>
<td>55</td>
<td>ENOBUFS</td>
<td>Takesocket</td>
<td>Not enough buffer space is available to create the new socket.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>55</td>
<td>ENOBUFS</td>
<td>IOCTL</td>
<td>There is no storage available to store the associated data.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>55</td>
<td>ENOBUFS</td>
<td>IP_BLOCK_SOURCE, IP_ADD_SOURCE_, MEMBERSHIP, MCAST_BLOCK_SOURCE, MCAST_JOIN_SOURCE_, GROUP, SIOCSPIFILTER, SIOCSMSFILTER, setipv4sourcefilter</td>
<td>A maximum of 64 source filters can be specified per multicast address, interface address pair.</td>
<td>Remove unneeded source IP addresses and reissue the command.</td>
</tr>
<tr>
<td>56</td>
<td>EISCONN</td>
<td>Connect</td>
<td>The socket is already connected.</td>
<td>Correct the socket descriptor on Connect() or do not issue a Connect() twice for the socket.</td>
</tr>
<tr>
<td>57</td>
<td>ENOTCONN</td>
<td>All</td>
<td>The socket is not connected.</td>
<td>Connect the socket before communicating.</td>
</tr>
<tr>
<td>57</td>
<td>ENOTCONN</td>
<td>IOCTL</td>
<td>The socket is not connected.</td>
<td>Issue the SIOCTTLSCTL IOCTL only after the socket is connected.</td>
</tr>
<tr>
<td>58</td>
<td>ESHUTDOWN</td>
<td>All</td>
<td>A Send cannot be processed after socket shutdown.</td>
<td>Issue read/receive before shutting down the read side of the socket.</td>
</tr>
<tr>
<td>59</td>
<td>ETOOMANYREFS</td>
<td>All</td>
<td>There are too many references. A splice cannot be completed.</td>
<td>Call your system administrator.</td>
</tr>
</tbody>
</table>
### ERRNOs

<table>
<thead>
<tr>
<th>Error number</th>
<th>Message name</th>
<th>Socket type</th>
<th>Error description</th>
<th>Programmer’s response</th>
</tr>
</thead>
<tbody>
<tr>
<td>59</td>
<td>ETOOMANYREFS</td>
<td>IP_ADD_MEMBERSHIP, IP_ADD_SOURCE_MEMBERSHIP, MCAST_JOIN_GROUP, MCAST_JOIN_SOURCE_GROUP, IPV6_JOIN_GROUP</td>
<td>A maximum of 20 multicast groups per single UDP socket or a maximum of 256 multicast groups per single RAW socket can be specified.</td>
<td>Remove unneeded multicast groups and reissue the command.</td>
</tr>
<tr>
<td>60</td>
<td>ETIMEDOUT</td>
<td>Connect</td>
<td>The connection timed out before it was completed.</td>
<td>Ensure the server application is available.</td>
</tr>
<tr>
<td>61</td>
<td>ECONNREFUSED</td>
<td>Connect</td>
<td>The requested connection was refused.</td>
<td>Ensure server application is available and at specified port.</td>
</tr>
<tr>
<td>62</td>
<td>ELOOP</td>
<td>All</td>
<td>There are too many symbolic loop levels.</td>
<td>Reduce symbolic links to specified file.</td>
</tr>
<tr>
<td>63</td>
<td>ENAMETOOLONG</td>
<td>All</td>
<td>The file name is too long.</td>
<td>Reduce size of specified file name.</td>
</tr>
<tr>
<td>64</td>
<td>EHOSTDOWN</td>
<td>All</td>
<td>The host is down.</td>
<td>Restart specified host.</td>
</tr>
<tr>
<td>65</td>
<td>EHOSTUNREACH</td>
<td>All</td>
<td>There is no route to the host.</td>
<td>Set up network path to specified host and verify that host name is valid.</td>
</tr>
<tr>
<td>66</td>
<td>ENOTEMPTY</td>
<td>All</td>
<td>The directory is not empty.</td>
<td>Clear out specified directory and reissue call.</td>
</tr>
<tr>
<td>67</td>
<td>EPROCLIM</td>
<td>All</td>
<td>There are too many processes in the system.</td>
<td>Decrease the number of processes or increase the process limit.</td>
</tr>
<tr>
<td>68</td>
<td>EUSERS</td>
<td>All</td>
<td>There are too many users on the system.</td>
<td>Decrease the number of users or increase the user limit.</td>
</tr>
<tr>
<td>69</td>
<td>EDQUOT</td>
<td>All</td>
<td>The disk quota has been exceeded.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>70</td>
<td>ESTALE</td>
<td>All</td>
<td>An old NFS® data set handle was found.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>71</td>
<td>EREMOTE</td>
<td>All</td>
<td>There are too many levels of remote in the path.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>72</td>
<td>ENOSTR</td>
<td>All</td>
<td>The device is not a stream device.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>73</td>
<td>ETIME</td>
<td>All</td>
<td>The timer has expired.</td>
<td>Increase timer values or reissue function.</td>
</tr>
<tr>
<td>74</td>
<td>ENOSR</td>
<td>All</td>
<td>There are no more stream resources.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>75</td>
<td>ENOMSG</td>
<td>All</td>
<td>There is no message of the desired type.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>76</td>
<td>EBADMSG</td>
<td>All</td>
<td>The system cannot read the message.</td>
<td>Verify that z/OS Communications Server installation was successful and that message files were properly loaded.</td>
</tr>
<tr>
<td>77</td>
<td>EIDRM</td>
<td>All</td>
<td>The identifier has been removed.</td>
<td>Call your system administrator.</td>
</tr>
</tbody>
</table>
## ERRNOs

### Table 6. Sockets ERRNOs (continued)

<table>
<thead>
<tr>
<th>Error number</th>
<th>Message name</th>
<th>Socket type</th>
<th>Error description</th>
<th>Programmer's response</th>
</tr>
</thead>
<tbody>
<tr>
<td>78</td>
<td>EDEADLK</td>
<td>All</td>
<td>A deadlock condition has occurred.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>78</td>
<td>EDEADLK</td>
<td>Select</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>79</td>
<td>ENOLCK</td>
<td>All</td>
<td>No record locks are available.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>80</td>
<td>ENONET</td>
<td>All</td>
<td>The requested machine is not on the network.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>81</td>
<td>ERREMOTE</td>
<td>All</td>
<td>The object is remote.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>82</td>
<td>ENOLINK</td>
<td>All</td>
<td>The link has been severed.</td>
<td>Release the sockets and reinitialize the client-server connection.</td>
</tr>
<tr>
<td>83</td>
<td>EADV</td>
<td>All</td>
<td>An ADVERTISE error has occurred.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>84</td>
<td>ESRMNT</td>
<td>All</td>
<td>An SRMOUNT error has occurred.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>85</td>
<td>ECOMM</td>
<td>All</td>
<td>A communication error has occurred on a Send call.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>86</td>
<td>EPROTO</td>
<td>All</td>
<td>A protocol error has occurred.</td>
<td>Call your system administrator.</td>
</tr>
</tbody>
</table>
| 86           | EPROTO       | IOCTL (SIOCTTLSCTL request in TTLS_RESET_SESSION TTLS_RESET_CIPHER 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_RESET_CIPHER or TTLS_STOP_CIPHER was requested on a connection that is secured using SSL version 2
- TTLS_ALLOW_HSTIMEOUT was requested but the policy has the HandshakeRole value client or the HandshakeTimeout value is 0. | Call your system administrator. |
| 87           | EMULTIHOP    | All         | A multi-hop address link was attempted. | Call your system administrator. |
### ERRNOs

**Table 6. Sockets ERRNOs (continued)**

<table>
<thead>
<tr>
<th>Error number</th>
<th>Message name</th>
<th>Socket type</th>
<th>Error description</th>
<th>Programmer’s response</th>
</tr>
</thead>
<tbody>
<tr>
<td>88</td>
<td>EDOTDOT</td>
<td>All</td>
<td>A cross-mount point was detected. This is not an error.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>89</td>
<td>EREMCHG</td>
<td>All</td>
<td>The remote address has changed.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>90</td>
<td>ECONNCLOSED</td>
<td>All</td>
<td>The connection was closed by a peer.</td>
<td>Check that the peer is running.</td>
</tr>
<tr>
<td>113</td>
<td>EBADF</td>
<td>All</td>
<td>Socket descriptor is not in correct range. The maximum number of socket descriptors is set by MAXDESC(). The default range is 0–49.</td>
<td>Reissue function with corrected socket descriptor.</td>
</tr>
<tr>
<td>113</td>
<td>EBADF</td>
<td>Bind socket</td>
<td>The socket descriptor is already being used.</td>
<td>Correct the socket descriptor.</td>
</tr>
<tr>
<td>113</td>
<td>EBADF</td>
<td>Givesocket</td>
<td>The socket has already been given. The socket domain is not AF_INET.</td>
<td>Correct the socket descriptor.</td>
</tr>
<tr>
<td>113</td>
<td>EBADF</td>
<td>Select</td>
<td>One of the specified descriptor sets is an incorrect socket descriptor.</td>
<td>Correct the socket descriptor. Set on Select() or Selectex().</td>
</tr>
<tr>
<td>113</td>
<td>EBADF</td>
<td>Takesocket</td>
<td>The socket has already been taken.</td>
<td>Correct the socket descriptor.</td>
</tr>
<tr>
<td>113</td>
<td>EBADF</td>
<td>Accept</td>
<td>A Listen() has not been issued before the Accept().</td>
<td>Issue Listen() before Accept().</td>
</tr>
<tr>
<td>121</td>
<td>EINVAL</td>
<td>All</td>
<td>An incorrect argument was specified.</td>
<td>Check and correct all function parameters.</td>
</tr>
<tr>
<td>121</td>
<td>EINVAL</td>
<td>IOCTL (SIOCSPAPPLDATA)</td>
<td>The input parameter is not a correctly formatted SetApplData structure.</td>
<td>Check and correct all function parameters.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• The SetAD_eye1 value is not valid.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• The SetAD_ver value is not valid.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• The storage pointed to by SetAD_ptr does not contain a correctly formatted SetADcontainer structure.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• The SetAD_eye2 value is not valid.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• The SetAD_len value contains an incorrect length for the SetAD_ver version of the SetADcontainer structure.</td>
<td></td>
</tr>
<tr>
<td>122</td>
<td>ECLOSED</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>126</td>
<td>ENMELONG</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### ERRNOs

**Table 6. Sockets ERRNOs (continued)**

<table>
<thead>
<tr>
<th>Error number</th>
<th>Message name</th>
<th>Socket type</th>
<th>Error description</th>
<th>Programmer’s response</th>
</tr>
</thead>
<tbody>
<tr>
<td>134</td>
<td>ENOSYS</td>
<td>IOCTL</td>
<td>The function is not implemented</td>
<td>Either configure the system to support the <code>ioctl</code> command or remove the <code>ioctl</code> command from your program.</td>
</tr>
<tr>
<td>134</td>
<td>ENOSYS</td>
<td>IOCTL - siocgifnameindex</td>
<td>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.</td>
<td>Either configure the system to support the <code>ioctl</code> command or remove the <code>ioctl</code> command from your program.</td>
</tr>
<tr>
<td>136</td>
<td>ENOTEMPT</td>
<td>All</td>
<td></td>
<td></td>
</tr>
<tr>
<td>145</td>
<td>E2BIG</td>
<td>All</td>
<td>The argument list is too long.</td>
<td>Eliminate excessive number of arguments.</td>
</tr>
<tr>
<td>156</td>
<td>EMVSINITIAL</td>
<td>All</td>
<td>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 may not be defined or may contain errors such as an improper HOME() directory specification.</td>
<td>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.</td>
</tr>
<tr>
<td>157</td>
<td>EMISSED</td>
<td>All</td>
<td>An MVS environmental or internal error occurred.</td>
<td></td>
</tr>
<tr>
<td>157</td>
<td>EMVSERR</td>
<td>All</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1002</td>
<td>EIBMSOCKOUTOF RANGE</td>
<td>Socket</td>
<td>A socket number assigned by the client interface code is out of range.</td>
<td>Check the socket descriptor parameter.</td>
</tr>
<tr>
<td>1003</td>
<td>EIBMSOCKINUSE</td>
<td>Socket</td>
<td>A socket number assigned by the client interface code is already in use.</td>
<td>Use a different socket descriptor.</td>
</tr>
<tr>
<td>1004</td>
<td>EIBMIUCVERR</td>
<td>All</td>
<td>The request failed because of an IUCV error. This error is generated by the client stub code.</td>
<td>Ensure IUCV/VMCF is functional.</td>
</tr>
<tr>
<td>1008</td>
<td>EIBMCONFLICT</td>
<td>All</td>
<td>This request conflicts with a request already queued on the same socket.</td>
<td>Cancel the existing call or wait for its completion before reissuing this call.</td>
</tr>
<tr>
<td>1009</td>
<td>EIBMCANCELLED</td>
<td>All</td>
<td>The request was canceled by the CANCEL call.</td>
<td>Informational, no action needed.</td>
</tr>
<tr>
<td>Error number</td>
<td>Message name</td>
<td>Socket type</td>
<td>Error description</td>
<td>Programmer’s response</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------</td>
</tr>
<tr>
<td>1011</td>
<td>EIBMBADTCPNAME</td>
<td>All</td>
<td>A TCP/IP name that is not valid was detected.</td>
<td>Correct the name specified in the IBM_TCPIMAGE structure.</td>
</tr>
<tr>
<td>1011</td>
<td>EIBMBADTCPNAME</td>
<td>Setibmopt</td>
<td>A TCP/IP name that is not valid was detected.</td>
<td>Correct the name specified in the IBM_TCPIMAGE structure.</td>
</tr>
<tr>
<td>1011</td>
<td>EIBMBADTCPNAME</td>
<td>INITAPI</td>
<td>A TCP/IP name that is not valid was detected.</td>
<td>Correct the name specified on the IDENT option TCPNAME field.</td>
</tr>
<tr>
<td>1012</td>
<td>EIBMBADREQUESTCODE</td>
<td>All</td>
<td>A request code that is not valid was detected.</td>
<td>Contact your system administrator.</td>
</tr>
<tr>
<td>1013</td>
<td>EIBMBADCONNECTIONSTATE</td>
<td>All</td>
<td>A connection token that is not valid was detected; bad state.</td>
<td>Verify TCP/IP is active.</td>
</tr>
<tr>
<td>1014</td>
<td>EIBMUNAUTHORIZEDCALLER</td>
<td>All</td>
<td>An unauthorized caller specified an authorized keyword.</td>
<td>Ensure user ID has authority for the specified operation.</td>
</tr>
<tr>
<td>1015</td>
<td>EIBMBADCONNECTIONMATCH</td>
<td>All</td>
<td>A connection token that is not valid was detected. There is no such connection.</td>
<td>Verify TCP/IP is active.</td>
</tr>
<tr>
<td>1016</td>
<td>EIBMTCPABEND</td>
<td>All</td>
<td>An abend occurred when TCP/IP was processing this request.</td>
<td>Verify that TCP/IP has restarted.</td>
</tr>
<tr>
<td>1023</td>
<td>EIBMTERMERROR</td>
<td>All</td>
<td>Encountered a terminating error while processing.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>1026</td>
<td>EIBMINVDELETE</td>
<td>All</td>
<td>Delete requestor did not create the connection.</td>
<td>Delete the request from the process that created it.</td>
</tr>
<tr>
<td>1027</td>
<td>EIBMINVSOCKET</td>
<td>All</td>
<td>A connection token that is not valid was detected. No such socket exists.</td>
<td>Call your system programmer.</td>
</tr>
<tr>
<td>1028</td>
<td>EIBMINVTCPCONNECTION</td>
<td>All</td>
<td>Connection terminated by TCP/IP. The token was invalidated by TCP/IP.</td>
<td>Reestablish the connection to TCP/IP.</td>
</tr>
<tr>
<td>1032</td>
<td>EBMSCALLINPROGRESS</td>
<td>All</td>
<td>Another call was already in progress.</td>
<td>Reissue after previous call has completed.</td>
</tr>
<tr>
<td>1036</td>
<td>EIBMNOACTIVETCP</td>
<td>All</td>
<td>TCP/IP is not installed or not active.</td>
<td>Correct TCP/IP name used.</td>
</tr>
<tr>
<td>1036</td>
<td>EIBMNOACTIVETCP</td>
<td>Select</td>
<td>EIBMNOACTIVETCP</td>
<td>Ensure TCP/IP is active.</td>
</tr>
<tr>
<td>1036</td>
<td>EIBMNOACTIVETCP</td>
<td>Getibmopt</td>
<td>No TCP/IP image was found.</td>
<td>Ensure TCP/IP is active.</td>
</tr>
<tr>
<td>1037</td>
<td>EIBMINVTSRBUSERDATA</td>
<td>All</td>
<td>The request control block contained data that is not valid.</td>
<td>Call your system programmer.</td>
</tr>
<tr>
<td>1038</td>
<td>EIBMINVUSERDATA</td>
<td>All</td>
<td>The request control block contained user data that is not valid.</td>
<td>Check your function parameters and call your system programmer.</td>
</tr>
</tbody>
</table>
Table 6. Sockets ERRNOs (continued)

<table>
<thead>
<tr>
<th>Error number</th>
<th>Message name</th>
<th>Socket type</th>
<th>Error description</th>
<th>Programmer’s response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1040</td>
<td>EIBMSELECTEXPOST</td>
<td>SELECTEX</td>
<td>SELECTEX passed an ECB that was already posted.</td>
<td>Check whether the user's ECB was already posted.</td>
</tr>
<tr>
<td>1112</td>
<td>ECANCEL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2001</td>
<td>EINVALDRXSOCKETCALL</td>
<td>REXX&lt;sup&gt;™&lt;/sup&gt;</td>
<td>A syntax error occurred in the RXSOCKET parameter list.</td>
<td>Correct the parameter list passed to the REXX socket call.</td>
</tr>
<tr>
<td>2002</td>
<td>ECONSOLEINTERRUPT</td>
<td>REXX</td>
<td>A console interrupt occurred.</td>
<td>Retry the task.</td>
</tr>
<tr>
<td>2003</td>
<td>ESUBTASKINVALID</td>
<td>REXX</td>
<td>The subtask ID is incorrect.</td>
<td>Correct the subtask ID on the INITIALIZE call.</td>
</tr>
<tr>
<td>2004</td>
<td>ESUBTASKALREADYACTIVE</td>
<td>REXX</td>
<td>The subtask is already active.</td>
<td>Only issue the INITIALIZE call once in your program.</td>
</tr>
<tr>
<td>2005</td>
<td>ESUBTASKALNOTACTIVE</td>
<td>REXX</td>
<td>The subtask is not active.</td>
<td>Issue the INITIALIZE call before any other socket call.</td>
</tr>
<tr>
<td>2006</td>
<td>ESOCKNETNOTALLOCATED</td>
<td>REXX</td>
<td>The specified socket could not be allocated.</td>
<td>Increase the user storage allocation for this job.</td>
</tr>
<tr>
<td>2007</td>
<td>EMAXSOCKETSREACHED</td>
<td>REXX</td>
<td>The maximum number of sockets has been reached.</td>
<td>Increase the number of allocate sockets, or decrease the number of sockets used by your program.</td>
</tr>
<tr>
<td>2009</td>
<td>ESOCKETNOTDEFINED</td>
<td>REXX</td>
<td>The socket is not defined.</td>
<td>Issue the SOCKET call before the call that fails.</td>
</tr>
<tr>
<td>2011</td>
<td>EDOMAINSERVERFAILURE</td>
<td>REXX</td>
<td>A Domain Name Server failure occurred.</td>
<td>Call your MVS system programmer.</td>
</tr>
<tr>
<td>2012</td>
<td>EINVALVALIDNAME</td>
<td>REXX</td>
<td>An incorrect <em>name</em> was received from the TCP/IP server.</td>
<td>Call your MVS system programmer.</td>
</tr>
<tr>
<td>2013</td>
<td>EINVALVALIDCLIENTID</td>
<td>REXX</td>
<td>An incorrect <em>clientid</em> was received from the TCP/IP server.</td>
<td>Call your MVS system programmer.</td>
</tr>
<tr>
<td>2014</td>
<td>EINVALVALIDFILENAME</td>
<td>REXX</td>
<td>An error occurred during NUCEXT processing.</td>
<td>Specify the correct translation table file name, or verify that the translation table is valid.</td>
</tr>
<tr>
<td>2016</td>
<td>EHOSTNOTFOUND</td>
<td>REXX</td>
<td>The host is not found.</td>
<td>Call your MVS system programmer.</td>
</tr>
<tr>
<td>2017</td>
<td>EIPADDRNOTFOUND</td>
<td>REXX</td>
<td>Address not found.</td>
<td>Call your MVS system programmer.</td>
</tr>
<tr>
<td>3412</td>
<td>ENODATA</td>
<td></td>
<td>Message does not exist.</td>
<td></td>
</tr>
<tr>
<td>3416</td>
<td>ELINKED</td>
<td></td>
<td>Stream is linked.</td>
<td></td>
</tr>
<tr>
<td>3419</td>
<td>ERECURSE</td>
<td></td>
<td>Recursive attempt rejected.</td>
<td></td>
</tr>
<tr>
<td>3420</td>
<td>EASYNC</td>
<td></td>
<td>Asynchronous I/O scheduled. This is a normal, internal event that is NOT returned to the user.</td>
<td></td>
</tr>
<tr>
<td>Error number</td>
<td>Message name</td>
<td>Socket type</td>
<td>Error description</td>
<td>Programmer’s response</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
<td>-------------</td>
<td>-------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>3448</td>
<td>EUNATCH</td>
<td></td>
<td>The protocol required to support the specified address family is not available.</td>
<td></td>
</tr>
<tr>
<td>3464</td>
<td>ETERM</td>
<td></td>
<td>Operation terminated.</td>
<td></td>
</tr>
<tr>
<td>3474</td>
<td>EUNKNOWN</td>
<td></td>
<td>Unknown system state.</td>
<td></td>
</tr>
<tr>
<td>3495</td>
<td>EBADOBJ</td>
<td></td>
<td>You attempted to reference a object that does not exist.</td>
<td></td>
</tr>
<tr>
<td>3513</td>
<td>EOUTOFSTATE</td>
<td></td>
<td>Protocol engine has received a command that is not acceptable in its current state.</td>
<td></td>
</tr>
</tbody>
</table>
ERRNOs
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.

You can request RFCs through electronic mail, from the automated Network Information Center (NIC) mail server, by sending a message to service@nic.ddn.mil with a subject line of RFC nnnn for text versions or a subject line of RFC nnnn.PS for PostScript versions. To request a copy of the RFC index, send a message with a subject line of RFC INDEX.

For more information, contact nic@nic.ddn.mil or at:

Government Systems, Inc.
Attn: Network Information Center
14200 Park Meadow Drive
Suite 200
Chantilly, VA 22021

Hard copies of all RFCs are available from the NIC, either individually or by subscription. Online copies are available at the following Web address:

http://www.rfc-editor.org/rfc.html

See "Internet drafts" on page 350 for draft RFCs implemented in this and previous Communications Server releases.

Many features of TCP/IP Services are based on the following RFCs:

<table>
<thead>
<tr>
<th>RFC</th>
<th>Title and Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFC 652</td>
<td>Telnet output carriage-return disposition option D. Crocker</td>
</tr>
<tr>
<td>RFC 653</td>
<td>Telnet output horizontal tabstops option D. Crocker</td>
</tr>
<tr>
<td>RFC 654</td>
<td>Telnet output horizontal tab disposition option D. Crocker</td>
</tr>
<tr>
<td>RFC 655</td>
<td>Telnet output formfeed disposition option D. Crocker</td>
</tr>
<tr>
<td>RFC 657</td>
<td>Telnet output vertical tab disposition option D. Crocker</td>
</tr>
<tr>
<td>RFC 658</td>
<td>Telnet output linefeed disposition D. Crocker</td>
</tr>
<tr>
<td>RFC 698</td>
<td>Telnet extended ASCII option T. Mock</td>
</tr>
<tr>
<td>RFC 726</td>
<td>Remote Controlled Transmission and Echoing Telnet option J. Postel, D. Crocker</td>
</tr>
<tr>
<td>RFC 727</td>
<td>Telnet logout option M.R. Crispin</td>
</tr>
<tr>
<td>RFC 732</td>
<td>Telnet Data Entry Terminal option J.D. Day</td>
</tr>
<tr>
<td>RFC 733</td>
<td>Standard for the format of ARPA network text messages D. Crocker, J. Vital, K.T. Pogran, D.A. Henderson</td>
</tr>
<tr>
<td>RFC</td>
<td>Title</td>
</tr>
<tr>
<td>-------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>734</td>
<td>SUPDUP Protocol</td>
</tr>
<tr>
<td>735</td>
<td>Revised Telnet byte macro option</td>
</tr>
<tr>
<td>736</td>
<td>Telnet SUPDUP option</td>
</tr>
<tr>
<td>749</td>
<td>Telnet SUPDUP—Output option</td>
</tr>
<tr>
<td>765</td>
<td>File Transfer Protocol specification</td>
</tr>
<tr>
<td>768</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>779</td>
<td>Telnet send-location option</td>
</tr>
<tr>
<td>783</td>
<td>TFTP Protocol (revision 2)</td>
</tr>
<tr>
<td>791</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>792</td>
<td>Internet Control Message Protocol</td>
</tr>
<tr>
<td>793</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>820</td>
<td>Assigned numbers</td>
</tr>
<tr>
<td>821</td>
<td>Simple Mail Transfer Protocol</td>
</tr>
<tr>
<td>822</td>
<td>Standard for the format of ARPA Internet text messages</td>
</tr>
<tr>
<td>823</td>
<td>DARPA Internet gateway</td>
</tr>
<tr>
<td>826</td>
<td>Ethernet Address Resolution Protocol: Or converting network protocol</td>
</tr>
<tr>
<td></td>
<td>addresses to 48.bit Ethernet address for transmission on Ethernet</td>
</tr>
<tr>
<td>854</td>
<td>Telnet Protocol Specification</td>
</tr>
<tr>
<td>855</td>
<td>Telnet Option Specification</td>
</tr>
<tr>
<td>856</td>
<td>Telnet Binary Transmission</td>
</tr>
<tr>
<td>857</td>
<td>Telnet Echo Option</td>
</tr>
<tr>
<td>858</td>
<td>Telnet Suppress Go Ahead Option</td>
</tr>
<tr>
<td>859</td>
<td>Telnet Status Option</td>
</tr>
<tr>
<td>860</td>
<td>Telnet Timing Mark Option</td>
</tr>
<tr>
<td>861</td>
<td>Telnet Extended Options: List Option</td>
</tr>
<tr>
<td>862</td>
<td>Echo Protocol</td>
</tr>
<tr>
<td>863</td>
<td>Discard Protocol</td>
</tr>
<tr>
<td>864</td>
<td>Character Generator Protocol</td>
</tr>
<tr>
<td>865</td>
<td>Quote of the Day Protocol</td>
</tr>
<tr>
<td>868</td>
<td>Time Protocol</td>
</tr>
<tr>
<td>877</td>
<td>Standard for the transmission of IP datagrams over public data networks</td>
</tr>
<tr>
<td>883</td>
<td>Domain names: Implementation specification</td>
</tr>
<tr>
<td>884</td>
<td>Telnet terminal type option</td>
</tr>
<tr>
<td>885</td>
<td>Telnet end of record option</td>
</tr>
<tr>
<td>894</td>
<td>Standard for the transmission of IP datagrams over Ethernet networks</td>
</tr>
<tr>
<td>896</td>
<td>Congestion control in IP/TCP internetworks</td>
</tr>
</tbody>
</table>
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 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 1014  XDR: External Data Representation standard Sun Microsystems
RFC 1027  Using ARP to implement transparent subnet gateways S. Carl-Mitchell, J. Quarterman
RFC 1032  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

Appendix B. Related protocol specifications  337
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 1058  Routing Information Protocol C. Hedrick
RFC 1060  Assigned numbers J. Reynolds, J. Postel
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
RFC 1112  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  Hitchhiker’s 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 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
<p>| 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 |
| 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 &quot;New Internet User&quot; Questions  | G. Malkin, A. Marine |
| RFC 1327 | Mapping between X.400 (1988)/ISO 10021 and RFC 822  | S. Hardcastle-Kille |</p>
<table>
<thead>
<tr>
<th>RFC</th>
<th>Title</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1340</td>
<td>Assigned Numbers</td>
<td>J. Reynolds, J. Postel</td>
</tr>
<tr>
<td>1344</td>
<td>Implications of MIME for Internet Mail Gateways</td>
<td>N. Bornstein</td>
</tr>
<tr>
<td>1349</td>
<td>Type of Service in the Internet Protocol Suite</td>
<td>P. Almquist</td>
</tr>
<tr>
<td>1350</td>
<td>The TFTP Protocol (Revision 2)</td>
<td>K.R. Sollins</td>
</tr>
<tr>
<td>1351</td>
<td>SNMP Administrative Model</td>
<td>J. Davin, J. Galvin, K. McCloghrie</td>
</tr>
<tr>
<td>1352</td>
<td>SNMP Security Protocols</td>
<td>J. Galvin, K. McCloghrie, J. Davin</td>
</tr>
<tr>
<td>1353</td>
<td>Definitions of Managed Objects for Administration of SNMP Parties</td>
<td>K. McCloghrie, J. Davin, J. Galvin</td>
</tr>
<tr>
<td>1354</td>
<td>IP Forwarding Table MIB</td>
<td>F. Baker</td>
</tr>
<tr>
<td>1356</td>
<td>Multiprotocol Interconnect on X.25 and ISDN in the Packet Mode</td>
<td>A. Malis, D. Robinson, R. Ullmann</td>
</tr>
<tr>
<td>1358</td>
<td>Charter of the Internet Architecture Board (IAB)</td>
<td>L. Chapin</td>
</tr>
<tr>
<td>1363</td>
<td>A Proposed Flow Specification</td>
<td>C. Partridge</td>
</tr>
<tr>
<td>1368</td>
<td>Definition of Managed Objects for IEEE 802.3 Repeater Devices</td>
<td>D. McMaster, K. McCloghrie</td>
</tr>
<tr>
<td>1372</td>
<td>Telnet Remote Flow Control Option</td>
<td>C. L. Hedrick, D. Borman</td>
</tr>
<tr>
<td>1374</td>
<td>IP and ARP on HIPPI</td>
<td>J. Renwick, A. Nicholson</td>
</tr>
<tr>
<td>1381</td>
<td>SNMP MIB Extension for X.25 LACP</td>
<td>D. Throop, F. Baker</td>
</tr>
<tr>
<td>1382</td>
<td>SNMP MIB Extension for the X.25 Packet Layer</td>
<td>D. Throop</td>
</tr>
<tr>
<td>1387</td>
<td>RIP Version 2 Protocol Analysis</td>
<td>G. Malkin</td>
</tr>
<tr>
<td>1388</td>
<td>RIP Version 2 Carrying Additional Information</td>
<td>G. Malkin</td>
</tr>
<tr>
<td>1389</td>
<td>RIP Version 2 MIB Extensions</td>
<td>G. Malkin, F. Baker</td>
</tr>
<tr>
<td>1390</td>
<td>Transmission of IP and ARP over FDDI Networks</td>
<td>D. Katz</td>
</tr>
<tr>
<td>1393</td>
<td>Traceroute Using an IP Option</td>
<td>G. Malkin</td>
</tr>
<tr>
<td>1398</td>
<td>Definitions of Managed Objects for the Ethernet-Like Interface Types</td>
<td>F. Kastenholz</td>
</tr>
<tr>
<td>1408</td>
<td>Telnet Environment Option</td>
<td>D. Borman, Ed.</td>
</tr>
<tr>
<td>1413</td>
<td>Identification Protocol</td>
<td>M. St. Johns</td>
</tr>
<tr>
<td>1416</td>
<td>Telnet Authentication Option</td>
<td>D. Borman, ed.</td>
</tr>
<tr>
<td>1420</td>
<td>SNMP over IPX</td>
<td>S. Bostock</td>
</tr>
<tr>
<td>1428</td>
<td>Transition of Internet Mail from Just-Send-8 to 8bit-SMTP/MIME</td>
<td>G. Vaudreuil</td>
</tr>
<tr>
<td>1445</td>
<td>Administrative Model for version 2 of the Simple Network Management Protocol (SNMPv2)</td>
<td>J. Galvin, K. McCloghrie</td>
</tr>
<tr>
<td>1447</td>
<td>Party MIB for version 2 of the Simple Network Management Protocol (SNMPv2)</td>
<td>K. McCloghrie, J. Galvin</td>
</tr>
</tbody>
</table>
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 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
RFC 1646  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-MIME Transport 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 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 1832 | XDR: External Data Representation Standard R. Srinivasan |
| RFC 1833 | Binding Protocols for ONC RPC Version 2 R. Srinivasan |
| RFC 1850 | 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 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 Notifications  K. Moore, G. Vaudreuil
RFC 1901  Introduction to Community-based SNMPv2  J. Case, K. McCloghrie, M. Rose, S. Waldbusser
RFC 1912  Common DNS Operational and Configuration Errors  D. Barr
RFC 1918  Address Allocation for Private Internets  Y. Rekhter, B. Moskowitz, D. Karrenberg, G.J. de Groot, E. Lear
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 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 2018 | TCP Selective Acknowledgement Options M. Mathis, J. Mahdavi, S. Floyd, A. Romanow |
| RFC 2026 | The Internet Standards Process — Revision 3 S. Bradner |
| RFC 2033 | Local Mail Transfer Protocol J. Myers |
| RFC 2034 | SMTP Service Extension for Returning Enhanced Error CodesN. Freed |
| 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 |
| RFC 2080 | 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 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 |
| 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 2271 | An Architecture for Describing SNMP Management Frameworks D. Harrington, R. Presuhn, B. Wijnen |
| RFC 2273 | SNMPv3 Applications D. Levi, P. Meyer, B. Stewart |
| 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 |
| RFC 2385 | Protection of BGP Sessions via the TCP MD5 Signature OptionA. 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 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 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 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
RFC 2523  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
Appendix B. Related protocol specifications
A DNS RR for specifying the location of services (DNS SRV) A. Gubrandsen, P. Vixie, L. Esibov

Simple Mail Transfer Protocol J. Klensin, Ed.

Internet Message Format P. Resnick, Ed.

TELNET KERMIT OPTION J. Altman, F. da Cruz

Secret Key Transaction Authentication for DNS (TSIG) P. Vixie, O. Gudmundsson, D. Eastlake 3rd, B. Wellington

Textual Conventions for Internet Network Addresses M. Daniele, B. Haberman, S. Routhier, J. Schoenwaelder

Deliver By SMTP Service Extension D. Newman

DNS Extensions to Support IPv6 Address Aggregation and Renumbering M. Crawford, C. Huitema

The Naming Authority Pointer (NAPTR) DNS Resource Record M. Mealling, R. Daniel

SMTP Service Extension for Command Pipelining N. Freed

Secret Key Establishment for DNS (TKEY RR) D. Eastlake, 3rd

Telnet Authentication Option T. Ts’o, ed., J. Altman

Telnet Authentication: Kerberos Version 5 T. Ts’o

Telnet Data Encryption Option T. Ts’o

Telnet Encryption: DES 64 bit Cipher Feedback T. Ts’o

Telnet Encryption: DES 64 bit Output Feedback T. Ts’o

Analysis of an Equal-Cost Multi-Path Algorithm C. Hopps

IP Version 6 Management Information Base for The Multicast Listener Discovery Protocol B. Haberman, R. Wozella

Policy Core Information Model—Version 1 Specification B. Moore, E. Ellesson, J. Strassner, A. Westerinen

Delegation of IP6.ARPA R. Bush

The BSD Syslog Protocol C. Lonvick

SMTP Service Extension for Secure SMTP over Transport Layer Security P. Hoffman

DNSSEC and IPv6 A6 aware server/resolver message size requirements O. Gudmundsson

Textual Conventions for Internet Network Addresses M. Daniele, B. Haberman, S. Routhier, J. Schoenwaelder


Internet Group Management Protocol, Version 3 B. Cain, S. Deering, I. Kouvelas, B. Fenner, A. Thyagarajan

Increasing TCP’s Initial Window M. Allman, S. Floyd, C. Partridge

Introduction and Applicability Statements for Internet-Standard Management Framework J. Case, R. Mundy, D. Partain, B. Stewart
<table>
<thead>
<tr>
<th>RFC</th>
<th>Title</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(SNMP) Management Frameworks</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Protocol (SNMP)</td>
<td></td>
</tr>
<tr>
<td>3414</td>
<td>User-based Security Model (USM) for version 3 of the Simple Network</td>
<td>U. Blumenthal, B. Wijnen</td>
</tr>
<tr>
<td></td>
<td>Management Protocol (SNMPv3)</td>
<td></td>
</tr>
<tr>
<td>3415</td>
<td>View-based Access Control Model (VACM) for the Simple Network</td>
<td>B. Wijnen, R. Presuhn, K. McCloghrie</td>
</tr>
<tr>
<td></td>
<td>Management Protocol (SNMP)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Management Protocol (SNMP)</td>
<td>S. Waldbusser</td>
</tr>
<tr>
<td></td>
<td>(SNMP)</td>
<td>S. Waldbusser</td>
</tr>
<tr>
<td>3418</td>
<td>Management Information Base (MIB) for the Simple Network</td>
<td>R. Presuhn, J. Case, K. McCloghrie, M. Rose,</td>
</tr>
<tr>
<td></td>
<td>Management Protocol (SNMP)</td>
<td>S. Waldbusser</td>
</tr>
<tr>
<td>3419</td>
<td>Textual Conventions for Transport Addresses</td>
<td>M. Daniele, J. Schoenwaelder</td>
</tr>
<tr>
<td>3484</td>
<td>Default Address Selection for Internet Protocol version 6 (IPv6)</td>
<td>R. Draves</td>
</tr>
<tr>
<td>3493</td>
<td>Basic Socket Interface Extensions for IPv6</td>
<td>R. Gilligan, S. Thomson, J. Bound, J. McCann,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>W. Stevens</td>
</tr>
<tr>
<td>3513</td>
<td>Internet Protocol Version 6 (IPv6) Addressing Architecture</td>
<td>R. Hinden, S. Deering</td>
</tr>
<tr>
<td>3526</td>
<td>More Modular Exponential (MODP) Diffie-Hellman groups for Internet</td>
<td>T. Kivinen, M. Kojo</td>
</tr>
<tr>
<td></td>
<td>Key Exchange (IKE)</td>
<td></td>
</tr>
<tr>
<td>3542</td>
<td>Advanced Sockets Application Programming Interface (API) for IPv6</td>
<td>W. Richard Stevens, M. Thomas, E. Nordmark,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T. Jinmei</td>
</tr>
<tr>
<td>3569</td>
<td>An Overview of Source-Specific Multicast (SSM)</td>
<td>S. Bhattacharyya, Ed.</td>
</tr>
<tr>
<td></td>
<td>Internet-standard Network Management Framework</td>
<td></td>
</tr>
<tr>
<td>3602</td>
<td>The AES-CBC Cipher Algorithm and Its Use with IPsec</td>
<td>S. Frankel, R. Glenn, S. Kelly</td>
</tr>
<tr>
<td>3629</td>
<td>UTF-8, a transformation format of ISO 10646</td>
<td>R. Kermode, C. Vicisano</td>
</tr>
<tr>
<td>3658</td>
<td>Delegation Signer (DS) Resource Record (RR)</td>
<td>O. Gudmundsson</td>
</tr>
<tr>
<td>3678</td>
<td>Socket Interface Extensions for Multicast Source Filters</td>
<td>D. Thaler, B. Fenner, B. Quinn</td>
</tr>
<tr>
<td>3715</td>
<td>IPsec-Network Address Translation (NAT) Compatibility Requirements</td>
<td>B. Aboba, W. Dixon</td>
</tr>
</tbody>
</table>
Internet drafts

Internet drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Other groups may also distribute working documents as Internet drafts. You can see Internet drafts at http://www.ietf.org/ID.html.

Several areas of IPv6 implementation include elements of the following Internet drafts and are subject to change during the RFC review process.
Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification

A. Conta, S. Deering
Appendix C. Accessibility

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. Refer to z/OS TSO/E Primer, z/OS TSO/E User’s Guide, and z/OS ISPF User’s Guide Vol I 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

z/OS information is accessible using screen readers with the BookServer/Library Server versions of z/OS books in the Internet library at [www.ibm.com/systems/z/os/zos/bkserv/](http://www.ibm.com/systems/z/os/zos/bkserv/)
Notices

IBM may not offer all of the products, services, or features discussed in this document. 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 give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

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

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, 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.

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

Site Counsel
IBM Corporation
P.O. Box 12195
3039 Cornwallis Road
Research Triangle Park, North Carolina 27709-2195
U.S.A

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 information 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 measurement 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.

All IBM prices shown are IBM’s suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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 illustrates 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. 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 IBM’s application programming interfaces.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

IBM is required to include the following statements in order to distribute portions of this document and the software described herein to which contributions have been made by The University of California. Portions herein © Copyright 1979, 1980, 1983, 1986, Regents of the University of California. Reproduced by permission. Portions herein were developed at the Electrical Engineering and Computer Sciences Department at the Berkeley campus of the University of California under the auspices of the Regents of the University of California.

Portions of this publication relating to RPC are Copyright © Sun Microsystems, Inc., 1988, 1989.

Some portions of this publication relating to X Window System** are Copyright © 1987, 1988 by Digital Equipment Corporation, Maynard, Massachusetts, and the Massachusetts Institute Of Technology, Cambridge, Massachusetts. All Rights Reserved.

Some portions of this publication relating to X Window System are Copyright © 1986, 1987, 1988 by Hewlett-Packard Corporation.

Permission to use, copy, modify, and distribute the M.I.T., Digital Equipment Corporation, and Hewlett-Packard Corporation portions of this software and its documentation for any purpose without fee is hereby granted, provided that the above copyright notice appears in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the names of M.I.T., Digital, and Hewlett-Packard not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. M.I.T., Digital, and Hewlett-Packard make no representation about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.


Copyright © 1988, 1993 The Regents of the University of California. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. All advertising materials mentioning features or use of this software must display the following acknowledgement:
   This product includes software developed by the University of California, Berkeley and its contributors.

4. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software program contains code, and/or derivatives or modifications of code originating from the software program "Popper." Popper is Copyright ©1989-1991 The Regents of the University of California, All Rights Reserved. Popper was created by Austin Shelton, Information Systems and Technology, University of California, Berkeley.

Permission from the Regents of the University of California to use, copy, modify, and distribute the "Popper" software contained herein for any purpose, without fee, and without a written agreement is hereby granted, provided that the above copyright notice and this paragraph and the following two paragraphs appear in all copies. HOWEVER, ADDITIONAL PERMISSIONS MAY BE NECESSARY FROM OTHER PERSONS OR ENTITIES, TO USE DERIVATIVES OR MODIFICATIONS OF POPPER.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THE POPPER SOFTWARE, OR ITS DERIVATIVES OR MODIFICATIONS, AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE POPPER SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATIONS TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

Copyright © 1983 The Regents of the University of California. All rights reserved.
Redistribution and use in source and binary forms are permitted provided that the
above copyright notice and this paragraph are duplicated in all such forms and
that any documentation, advertising materials, and other materials related to such
distribution and use acknowledge that the software was developed by the
University of California, Berkeley. The name of the University may not be used to
endorse or promote products derived from this software without specific prior
written permission. THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT
LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE.

Copyright © 1991, 1993 The Regents of the University of California. All rights
reserved.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:
1. Redistributions of source code must retain the above copyright notice, this list
   of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this
   list of conditions and the following disclaimer in the documentation and/or
   other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must
   display the following acknowledgement:
   This product includes software developed by the University of
   California, Berkeley and its contributors.
4. Neither the name of the University nor the names of its contributors may be
   used to endorse or promote products derived from this software without
   specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS
IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHELL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF
THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.

Copyright © 1990 by the Massachusetts Institute of Technology

Export of this software from the United States of America may require a specific
license from the United States Government. It is the responsibility of any person or
organization contemplating export to obtain such a license before exporting.

WITHIN THAT CONSTRAINT, permission to use, copy, modify, and distribute this
software and its documentation for any purpose and without fee is hereby granted,
provided that the above copyright notice appear in all copies and that both that
copyright notice and this permission notice appear in supporting documentation,
and that the name of M.I.T. not be used in advertising or publicity pertaining to
distribution of the software without specific, written prior permission. Furthermore
if you modify this software you must label your software as modified software and
not distribute it in such a fashion that it might be confused with the original M.I.T.
software. M.I.T. makes no representations about the suitability of this software for
any purpose. It is provided "as is" without express or implied warranty.

Copyright © 1998 by the FundsXpress, INC. All rights reserved.

Export of this software from the United States of America may require a specific
license from the United States Government. It is the responsibility of any person or
organization contemplating export to obtain such a license before exporting.

WITHIN THAT CONSTRAINT, permission to use, copy, modify, and distribute this
software and its documentation for any purpose and without fee is hereby granted,
provided that the above copyright notice appear in all copies and that both that
copyright notice and this permission notice appear in supporting documentation,
and that the name of FundsXpress not be used in advertising or publicity
pertaining to distribution of the software without specific, written prior
permission. FundsXpress makes no representations about the suitability of this
software for any purpose. It is provided "as is" without express or implied
warranty.

THIS SOFTWARE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE.


Permission to use, copy, modify, and distribute this software for any purpose with
or without fee is hereby granted, provided that the above copyright notice and this
permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND INTERNET SOFTWARE
CONSORTIUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS
SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS. IN NO EVENT SHALL INTERNET SOFTWARE CONSORTIUM
BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL
DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF
USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN
CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Copyright © 1995-1998 Eric Young (eay@cryptsoft.com) All rights reserved.

This package is an SSL implementation written by Eric Young (eay@cryptsoft.com).
The implementation was written so as to conform with Netscape’s SSL.

This library is free for commercial and non-commercial use as long as the
following conditions are adhered to. The following conditions apply to all code
found in this distribution, be it the RC4, RSA, lhash, DES, etc., code; not just the
SSL code. The SSL documentation included with this distribution is covered by the
same copyright terms except that the holder is Tim Hudson (tjh@cryptsoft.com).

Copyright remains Eric Young’s, and as such any Copyright notices in the code are
not to be removed. If this package is used in a product, Eric Young should be
given attribution as the author of the parts of the library used. This can be in the
form of a textual message at program startup or in documentation (online or
textual) provided with the package.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the copyright notice, this list of
   conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this
   list of conditions and the following disclaimer in the documentation and/or
   other materials provided with the distribution.

3. All advertising materials mentioning features or use of this software must
display the following acknowledgement: "This product includes cryptographic
software written by Eric Young (eay@cryptsoft.com)". The word 'cryptographic'
can be left out if the routines from the library being used are not cryptographic
related.

4. If you include any Windows specific code (or a derivative thereof) from the
apps directory (application code) you must include acknowledgement:
   "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"

THIS SOFTWARE IS PROVIDED BY ERIC YOUNG "AS IS" AND ANY EXPRESS
OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.

The license and distribution terms for any publicly available version or derivative
of this code cannot be changed. i.e. this code cannot simply be copied and put
under another distribution license [including the GNU Public License.]

This product includes cryptographic software written by Eric Young.


Permission to use, copy, modify, and distribute this software for any purpose with
or without fee is hereby granted, provided that the above copyright notice and this
permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND INTERNET SOFTWARE
CONSORTIUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS
SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS. IN NO EVENT SHALL INTERNET SOFTWARE CONSORTIUM
BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL
DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF
USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN
CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
Copyright © 2004 IBM Corporation and its licensors, including Sendmail, Inc., and the Regents of the University of California. All rights reserved.

Copyright © 1999,2000,2001 Compaq Computer Corporation

Copyright © 1999,2000,2001 Hewlett-Packard Company

Copyright © 1999,2000,2001 IBM Corporation

Copyright © 1999,2000,2001 Hummingbird Communications Ltd.


Copyright © 1999,2000,2001 Sun Microsystems, Inc.

Copyright © 1999,2000,2001 The Open Group

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

X Window System is a trademark of The Open Group.

If you are viewing this information softcopy, photographs and color illustrations may not appear.

You can obtain softcopy from the z/OS Collection (SK3T-4269), which contains BookManager and PDF formats.
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.

The following terms are trademarks of other companies:

Intel is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

PostScript is a registered trademark of Adobe Systems Incorporated in the United States, and/or other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.
Bibliography

This bibliography contains descriptions of the documents in the z/OS
Communications Server library.

z/OS Communications Server documentation is available in the following forms:
• Online at the z/OS Internet Library web page at [www.ibm.com/systems/z/os/zos/bkserv/](http://www.ibm.com/systems/z/os/zos/bkserv/)
• In softcopy on CD-ROM collections. See “Softcopy information” on page xvii.

z/OS Communications Server library updates


z/OS Communications Server information

z/OS Communications Server product information is grouped by task in the following tables.

### Planning

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS Communications Server: New Function Summary</td>
<td>GC31-8771</td>
<td>This document is intended to help you plan for new IP for SNA function, 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.</td>
</tr>
<tr>
<td>z/OS Communications Server: IPv6 Network and Application Design Guide</td>
<td>SC31-8885</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Resource definition, configuration, and tuning

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS Communications Server: IP Configuration Guide</td>
<td>SC31-8775</td>
<td>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 in conjunction with the z/OS Communications Server: IP Configuration Reference.</td>
</tr>
</tbody>
</table>
# Configuration Reference

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
</table>
| z/OS Communications Server: IP Configuration Reference  | SC31-8776 | This document presents information for people who want to administer and maintain IP. Use this document in conjunction with the z/OS Communications Server: IP Configuration Guide. The information in this document includes:  
  - TCP/IP configuration data sets  
  - Configuration statements  
  - Translation tables  
  - SMF records  
  - Protocol number and port assignments                  |
| z/OS Communications Server: SNA Network Implementation Guide | SC31-8777 | This document presents the major concepts involved in implementing an SNA network. Use this document in conjunction with the z/OS Communications Server: SNA Resource Definition Reference. |
| z/OS Communications Server: SNA Resource Definition Reference | SC31-8778 | 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 in conjunction with the z/OS Communications Server: SNA Network Implementation Guide. |
| z/OS Communications Server: SNA Resource Definition Samples | SC31-8836 | 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    | SC31-8833 | This document is for system 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

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS Communications Server: IP User’s Guide and Commands</td>
<td>SC31-8780</td>
<td>This document describes how to use TCP/IP applications. It contains requests that allow a user to log on to a remote host using Telnet, transfer data sets using FTP, send and receive electronic mail, print on remote printers, and authenticate network users.</td>
</tr>
<tr>
<td>z/OS Communications Server: IP System Administrator’s Commands</td>
<td>SC31-8781</td>
<td>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.</td>
</tr>
<tr>
<td>z/OS Communications Server: SNA Operation</td>
<td>SC31-8779</td>
<td>This document serves as a reference for programmers and operators requiring detailed information about specific operator commands.</td>
</tr>
<tr>
<td>z/OS Communications Server: Quick Reference</td>
<td>SX75-0124</td>
<td>This document contains essential information about SNA and IP commands.</td>
</tr>
</tbody>
</table>
## Customization

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
</table>
| z/OS Communications Server: SNA Customization | SC31-6854 | This document enables you to customize SNA, and includes the following:  
- 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

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference</td>
<td>SC31-8788</td>
<td>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.</td>
</tr>
<tr>
<td>z/OS Communications Server: IP CICS Sockets Guide</td>
<td>SC31-8807</td>
<td>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.</td>
</tr>
<tr>
<td>z/OS Communications Server: IP IMS Sockets Guide</td>
<td>SC31-8830</td>
<td>This document is for programmers who want application programs that use the IMS TCP/IP application development services provided by IBM's TCP/IP Services.</td>
</tr>
<tr>
<td>z/OS Communications Server: IP Programmer’s Guide and Reference</td>
<td>SC31-8787</td>
<td>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.</td>
</tr>
<tr>
<td>z/OS Communications Server: SNA Programming</td>
<td>SC31-8829</td>
<td>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.</td>
</tr>
<tr>
<td>z/OS Communications Server: SNA Programmer’s LU 6.2 Guide</td>
<td>SC31-8811</td>
<td>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.)</td>
</tr>
<tr>
<td>z/OS Communications Server: SNA Programmer’s LU 6.2 Reference</td>
<td>SC31-8810</td>
<td>This document provides reference material for the SNA LU 6.2 programming interface for host application programs.</td>
</tr>
<tr>
<td>z/OS Communications Server: CSM Guide</td>
<td>SC31-8808</td>
<td>This document describes how applications use the communications storage manager.</td>
</tr>
</tbody>
</table>
### Title Number Description

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS Communications Server: CMIP Services and Topology Agent Guide</td>
<td>SC31-8828</td>
<td>This document describes the Common Management Information Protocol (CMIP) programming interface for application programmers to use in coding CMIP application programs. The document provides guide and reference information about CMIP services and the SNA topology agent.</td>
</tr>
</tbody>
</table>

### Diagnosis

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS Communications Server: IP Diagnosis Guide</td>
<td>GC31-8782</td>
<td>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.</td>
</tr>
<tr>
<td>z/OS Communications Server: ACF/TAP Trace Analysis Handbook</td>
<td>GC23-8588</td>
<td>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.</td>
</tr>
<tr>
<td>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</td>
<td>GC31-6850, GC31-6851</td>
<td>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.</td>
</tr>
<tr>
<td>z/OS Communications Server: SNA Data Areas Volume 1 and z/OS Communications Server: SNA Data Areas Volume 2</td>
<td>GC31-6852, GC31-6853</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Messages and codes

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
</table>
| z/OS Communications Server: SNA Messages | SC31-8790 | 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) | SC31-8783 | This volume contains TCP/IP messages beginning with EZA. |
| z/OS Communications Server: IP Messages Volume 2 (EZB, EZD) | SC31-8784 | This volume contains TCP/IP messages beginning with EZB or EZD. |
| z/OS Communications Server: IP Messages Volume 3 (EZY) | SC31-8785 | This volume contains TCP/IP messages beginning with EZY. |
| z/OS Communications Server: IP Messages Volume 4 (EZZ, SNM) | SC31-8786 | This volume contains TCP/IP messages beginning with EZZ and SNM. |
| z/OS Communications Server: IP and SNA Codes | SC31-8791 | This document describes codes and other information that appear in z/OS Communications Server messages. |
Index

A
accept 24
ACCEPT (call) 61
accessibility 353
active sockets 53
active sockets queue 31
ADDRSPC parameter 52
ADDRSPCPEF parameter 53
AF parameter on call interface, on SOCKET 184
alternate PCB 26
APPC 1
application data 26, 31
application data, explicit mode
data translation 36
end-of-message indicator 36
format 36
network byte order 36
application data, explicit-mode
protocol 43, 44
translation 43, 44
application data, implicit-mode
data translation 38, 46
end-of-message 46
end-of-message indicator 38
format 38, 46
Application types
3270 1
client-server 1
ASCII to EBCDIC translation 36
ASMADELI 48
Assist module
role of 23
trade-offs 23
use of IMS message queue 23

B
BACKLOG parameter 53
BACKLOG parameter on call interface, LISTEN call 130
backlog queue 31
backlog queue, length 53
bb status code 46, 48
Berkeley Sockets
BSD 4.3 3
big-endian 36
BIND 24
BIND (call) 63
bit-mask on call interface, on EZACIC06 call 196
bit-mask-length on call interface, on EZACIC06 call 197
BMP 52
BUF parameter on call socket interface
on GETIBMOPT 92
on READ 136
on RECV 140
on RECVFROM 143
on SEND 160
on SENDTO 167
on WRITE 189
buffer full 41

C
C language 3
list of calls 21
CADLI 48
CALL Instruction Interface for Assembler, PL/I, and COBOL 57
Call Instructions for Assembler, PL/I, and COBOL Programs
ACCEPT 61
BIND 63
CLOSE 66
CONNECT 68
EZACIC04 192
EZACIC05 194
EZACIC06 196
EZACIC08 198
FCNTL 71
GETCLIENTID 82
GETHOSTBYADDR 84
GETHOSTBYNAME 86
GETHOSTID 89
GETHOSTNAME 89
GETIBMOPT 91
GETPEERNAME 97
GETSOCKNAME 99
GETSOCKOPT 101
GIVESOCKET 115
INITAPI 117
IOCTL 119
LISTEN 129
READ 135
READV 137
RECV 138
RECVFROM 140
RECVMSG 144
SELECT 148
SELECTEX 153
SENDMSG 160
SENDTO 164
TAKESOCKET 185
TERMAPI 187
WRITE 188
WRITEV 189
Call Instructions for Assembler, PL/I, and COBOL Programs
EZACIC14 206
EZACIC15 208
call interface sample PL/I programs 210
call sequence, explicit-mode client 36
CBLADLI 48
CHAR-MASK parameter on call interface, on EZACIC06 196
call server 12
CHNG 26
call client
defined 35
explicit-mode 35
logic flow 35
call client sequence, implicit-mode 37
CLIENT parameter on call socket interface
on GETCLIENTID 83

© Copyright IBM Corp. 1994, 2009
CLIENT parameter on call socket interface (continued)
on GIVESOCKET 116
on TAKESOCKET 187
client-server 1
client/server processing 6
COBOL language
  list of calls 21
codes, RSM reason 41
COMMAND parameter on call interface, IOCTL call 121
COMMAND parameter on call socket interface
  on EZACIC06 197
  on FCNTL 72
  on GETIBMOPT 91
COMM 43, 44
commit database updates 26
commit, explicit-mode 35
Communications Server for z/OS, online information xix
complete-status message 41
concurrent server
  defined 11
  illustrated 11, 12
configuring IMS TCP/IP 56
connection, how established 24
corversation, TCP/IP 24
CSMOKY 39, 41
CSMOKY message 37

D
data translation
  explicit-mode 36
  data translation, socket interface 191
    ASCII to EBCDIC 194
    bit-mask to character 196
    character to bit-mask 196
    EBCDIC to ASCII 192, 206
data, application 26, 31
database calls 26
database updates, commit 26
DataLen 54
DataType 54
disability 353
DNS, online information xx

E
EBCDIC to ASCII translation 36
ERETMSK parameter on call interface, on SELECT 152
ERRNO parameter on call socket interface (continued)
on READV 138
on RECV 140
on RECVFROM 144
on RECVMSG 148
on SELECT 152
on SELECTEX 157
on SEND 160
on SENDMSG 164
on SENDTO 167
on SOCKET 185
on TAKESOCKET 187
on WRITE 189
on WRITEV 191
ERRNO parameter on macro socket interface
  on FCNTL 73, 82
ESDNMASK parameter on call interface, on SELECT 152
EWOULDBLOCK error return, call interface calls
  RECV 139
  RECVFROM 141
explicit-mode 3
explicit-mode client
  application data format 36
  call sequence 36
  data format 36
  data translation 36
  network byte order 36
explicit-mode server
  application data 43
  call sequence 43
  I/O PCB 43
  PL/I programming 43
  TIM 43
  transaction-initiation message 43
EZACIC04, call interface, EBCDIC to ASCII translation 192
EZACIC05, call interface, ASCII to EBCDIC translation 194
EZACIC06 19
EZACIC06, call interface, bit-mask translation 196
EZACIC08, HOSTENT structure interpreter utility 198
EZACIC09, RES structure interpreter utility 201
EZACIC14, call interface, EBCDIC to ASCII translation 206
EZACIC15, call interface, ASCII to EBCDIC translation 208
EZASOKET
  Assembler language call format 59
  COBOL language call format 59
  PL/I language call format 59

F
FCNTL (call) 71
FLAGS parameter on call socket interface
  on RECV 140
  on RECVFROM 142
  on RECVMSG 147
  on SEND 159
  on SENDMSG 164
  on SENDTO 166
FNDELAY flag on call interface, on FCNTL 72

G
GETCLIENTID (call) 82
GETHOSTBYADDR (call) 84
GETHOSTBYNAME (call) 86
GETHOSTID (call) 89
GETHOSTNAME (call) 89
GETIBMOPT (call) 91
GETPEERNAME (call) 97
GETSOCKNAME (call) 99
GETSOCKOPT (call) 101
GIVESOCKET 26
GIVESOCKET (call) 115

hlq.PROFILE.TCPIP data set 55
hlq.TCPIP.DATA data set 56
HOSTADDR parameter on call interface, on GETHOSTBYADDR 84
HOSTENT parameter on call socket interface
on GETHOSTBYADDR 85
on GETHOSTBYNAME 87
HOSTENT structure interpreter parameters, on EZACIC08 199
HOW parameter on call interface, on SHUTDOWN 183

I/O Area size 48
I/O PCB in explicit-mode server 45
IBM Software Support Center, contacting xiii
IDENT parameter on call interface, INITAPI call 118
implicit mode 3
implicit-mode
client 37
call sequence 37
logic flow 37
complete status message 37
CSM 37
data stream 37
transaction-request message 37
TRM 37
implicit-mode client
application data stream 39
application data, format 39
call sequence 39
data format 39
data translation 39
end-of-message indicator 39
logic flow 39
implicit-mode server
application data 46
Assist module 46
call sequence 46
I/O PCB 46
PL/I programming 46
implicit mode

IMSLSECX, Listener security exit name 54
IN-BUFFER parameter on call interface, EZACIC05 call 194
Information APARS xvii
initapi 43, 45
INITAPI(call) 117
INQY 26
Internet, finding z/OS information online xix

IOCTL (call) 119
IOV parameter on call socket interface
on READV 138
on WRITEV 190
IOVCNT parameter on call socket interface
on READV 138
on RECVMSG 147
on SENDMSG 164
on WRITEV 190
IP protocol 7
IpAddr 54
ISRT 46
iterative server
defined 11
illustrated 12

K
keyboard 353

L
length of backlog queue 53
LENGTH parameter on call socket interface
on EZACIC04 193
on EZACIC05 195
on EZACIC14 206
on EZACIC15 208
license, patent, and copyright information 355
LISTEN 24
LISTEN (call) 129
Listener call sequence 31
Listener configuration file
LISTENER statement 52
TCPIP statement 52
TRANSACTION statement 52
Listener ReasonCode 54
Listener RetnCode 54
Listener startup parameters 52
Listener statement 53
LISTNR 45
little-endian 36
LookAt xvii
LTERM name 49
LU 6.2 1

M
mainframe
education xvii
MAXACTSKT 31
MAXACTSKT parameter 53
MAXSNO parameter on call interface, INITAPI call 119
MAXSOC parameter on call socket interface
on INITAPI 118
on SELECT 151
on SELECTEX 157
MAXTRANS parameter 53
Message Format Services 1
Message format services (MFS) 31
message queue 23, 24, 26
message queue, use of 31
messages
complete-status message 41
MFS 1
MODE=SNGL 43
MSG parameter on call socket interface
  on RECVMSG 146
  on SENDMSG 162
multiple connection requests 31

NAME parameter on call socket interface
  on ACCEPT 62
  on BIND 65
  on CONNECT 70
  on GETHOSTBYNAME 87
  on GETHOSTNAME 90
  on GETPEERNAME 98
  on GETSOCKNAME 100
  on GECVFROM 143
NAMELEN parameter on call socket interface
  on GETHOSTBYNAME 87
  on GETHOSTNAME 90
NBYTE parameter on call socket interface
  on READ 136
  on RECV 140
  on RECVFROM 143
  on SEND 160
  on SENDTO 166
  on WRITE 189
network byte order 36

O
OSI 6
OUT-BUFFER parameter on call interface, on EZACIC04 192
OUT-BUFFER parameter on call interface, on EZACIC14 206
OUT-BUFFER parameter on call interface, on EZACIC15 208
output area size 48
Overview 1

P
pending activity 18
pending exception 19
pending read 19
PL/I coding 42
PL/I programs, required statement 60
PLIADLI 48
Port 54
port numbers
  reserving port numbers 55
PORT parameter 53
ports
  compared with sockets 9
  reserving port numbers 55
prerequisite information xvii
program variable definitions, call interface
  assembler definition 60
  COBOL PIC 60
  PL/I declare 60
  VS COBOL II PIC 60
PROTO parameter on call interface, on SOCKET 185
PURG call 48

QC status code 46, 48
QD status code 46, 48

R
READ 26
READ (call) 135
READV (call) 137
ReasnCode, Listener 54
reason codes 41
RECV (call) 138
RECFROM (call) 140
RECVMSG (call) 144
REQARG and RETARG parameter on call socket interface
  on FCNTL 72
  on IOCTL 127
REQSTS 39
request-status message 39
Request-status message 35
requirements for IMS TCP/IP 21
RETCODE parameter on call interface, on IOCTL 128
RETCODE parameter on call socket interface
  on ACCEPT 63
  on BIND 66
  on CLOSE 68
  on CONNECT 71
  on EZACIC06 197
  on FCNTL 72
  on GETCLIENTID 83
  on GETHOSTBYADDR 85
  on GETHOSTBYNAME 87
  on GETHOSTID 89
  on GETHOSTNAME 90
  on GETIBMOPT 93
  on GETPEERNAME 99
  on GETSOCKNAME 101
  on GETSOCKOPT 103
  on GIVESOCKET 117
  on INITAPI 119
  on IOCTL 129
  on LISTEN 130
  on READ 136
  on READV 138
  on RECV 140
  on RECFROM 144
  on RECVMSG 148
  on SELECT 152
  on SELECTEX 158
  on SEND 160
  on SENDMSG 164
  on SENDTO 167
  on SETSOCKOPT 169
  on SHUTDOWN 183
  on SOCKET 185
  on TAKESOCKET 187
  on WRITE 189
  on WRITEV 191
RETCODE parameter on macro socket interface
  on FCNTL 73, 82
RetnCode, Listener 54
return codes
  call interface 61
return codes, I/O PCB
  bb 49
  EA 49
  EB 49
  EC 49
  QC 49
  QD 49
  ZZ 49
RFC (request for comments) 335
RFC (request for comments) (continued)
   accessing online xix
ROLB call  49
RRETMSK parameter on call interface, on SELECT  152
RSM  35
RSM reason codes  41
RSMId  39
RSMLen  39
RSMRetCod  39
RSMRsnCod  39
RSMRsv  39
RSNDMSK parameter on call interface, on SELECT  151

S
S, defines socket descriptor on macro interface
   on FCNTL  73, 76, 94
S, defines socket descriptor on socket interface
   on ACCEPT  62
   on BIND  65
   on CLOSE  68
   on CONNECT  69
   on FCNTL  72
   on GETPEERNAME  98
   on GETSOCKNAME  100
   on GETSOCKOPT  102
   on GIVESOCKET  116
   on IOCTL  121
   on LISTEN  130
   on READ  136
   on READV  138
   on RECV  139
   on RECVFROM  142
   on RECVMSG  146
   on SEND  159
   on SENDMSG  162
   on SENDTO  166
   on SETSOCKOPT  168
   on SHUTDOWN  183
   on WRITE  189
   on WRITEV  190
sample programs
   call interface
      CBLOCK, PL/I  228
      client, PL/I  214
      server, PL/I  210
security exit  24
security exit reason codes  41
security exit, data passed by Listener  54
security exit, Listener  54
security exit, return codes  54
SELECT (call)  148
select mask  18
SELECTEX (call)  153
SEND (call)  158
SENMMSG (call)  160
SENDTO (call)  164
server call sequence, explicit-mode  43
server programming, logic flow  43
server, defined  35
server, explicit mode
   see explicit mode server  43
SETSOCKOPT (call)  168
shortcut keys  353
SHUTDOWN (call)  182
SNA  1
SN A protocols
   compared with SNA  5
   compared with TCP/IP  5
SOCKET (call)  184
Socket interface  3
sockets
   compared with ports  9
   introduction  7
Sockets  1
Sockets Extended API  8
SOCTYPE parameter on call interface, TAKESOCKET
   call  186
SOCTYPE parameter on call interface, on SOCKET  185
software information xvii
SUBTASK parameter on call interface, INITAPI call  119
SYNC  26
syntax diagram, how to read  xiv

T
takesocket  26, 43, 45
TAKESOCKET (call)  185
TCP protocol  7
TCP/IP
   online information xix
   protocol specifications  335
TCP/IP for MVS, modifying data sets
   modifying data sets  55
TCP/IP protocols  6
TCP/IP Services  21
TCP/IP statement  52
Technotes xvii
TELNET  1
TERMAPI (call)  187
TIM  26, 45
TIMDataType  45
TIMEOUT parameter on call interface, on SELECT  151
TIMEOUT parameter on call socket interface
   on SELECTEX  157
TIMId  45
TIMLen  45
TIMListTaskID  43
TIMListAddrSpC  43, 45
TIMListTaskID  45
TIMRsv  45
TIMSktDesc  43, 45
TIMServAddrSpC  43, 45
TIMServTaskID  43, 45
TIMTCPAddrSpC  43, 45
TN3270  1
TOKEN parameter on call interface, on EZACIC06  196
trademark information  363
TRANCODE  23, 24
Transaction code  23
transaction name, IMS  53
transaction not defined  41
transaction request message  24
TRANSACTION statement  53
transaction unavailable  41
transaction verification  54
Transaction-initiation message  45
transaction-request message  39
Transaction-request message  35
TransNam  54
TRM  24, 35, 39
TRM bad format  41
TRMId  39

Index  373
U

UDP protocol  7
updates, database commit  26
use of HOSTENT structure interpreter, EZACIC08  198
Userdata  54
utility programs  191
   EZACIC04  192
   EZACIC05  194
   EZACIC06  196
   EZACIC08  198
   EZACIC14  206
   EZACIC15  208

V

verification, transaction  54
VTAM  1
VTAM, online information  xix

W

WRETMSK parameter on call interface, on SELECT  152
WRITE (call)  188
write()  26, 31
WRITEV (call)  189
WSNDMSK parameter on call interface, on SELECT  152

Z

z/OS Basic Skills information center  xvii
z/OS Basic Skills Information Center  xvii
z/OS, documentation library listing  365
ZZ status code  48
Communicating Your Comments to IBM

If you especially like or dislike anything about this document, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this document. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

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

Please send your comments to us in either of the following ways:

• If you prefer to send comments by FAX, use this number: 1+919-254-1258
• If you prefer to send comments electronically, use this address:
  - comsvrcf@us.ibm.com
• If you prefer to send comments by post, use this address:
  
  International Business Machines Corporation
  Attn: z/OS Communications Server Information Development
  P.O. Box 12195, 3039 Cornwallis Road
  Department AKCA, Building 501
  Research Triangle Park, North Carolina 27709-2195

Make sure to include the following in your note:
• Title and publication number of this document
• Page number or topic to which your comment applies.