z/OS Communications Server

IP CICS Sockets Guide

Version 1 Release 9

SC31-8807-04

z/OS Communications Server

IP CICS Sockets Guide

Version 1 Release 9

SC31-8807-04

Note:
Before using this information and the product it supports, be sure to read the general information under "Notices" on page 585.

Fifth edition (September 2007)

This edition applies to Version 1 Release 9 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:
http://www.ibm.com/servers/eserver/zseries/zos/webqs.html

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.

Contents

Figures

Tables

About this document

This document describes the TCP/IP Socket Interface for CICS® (referred to as CICS TCP/IP for short). It contains an introduction, a guide to initialization, and a guide and reference to writing application programs. Use this document to set up CICS TCP/IP, write application programs, and diagnose problems. The information in this document supports both IPv6 and IPv4. Unless explicitly noted, information describes IPv4 networking protocol. IPv6 support is qualified within the text.

Who should read this document

This document is intended for both system programmers and application programmers who perform any of the following tasks with CICS TCP/IP:

- Setting up CICS TCP/IP
- Writing application programs
- Diagnosing problems

The document assumes that the reader is familiar with the MVS™ operating system, and the C, COBOL, PL/I, or Assembler programming languages. Because the CICS Transaction Server (CICS TS) is a prerequisite for CICS TCP/IP, the document assumes the reader is also familiar with CICS TS.

How this document is organized

This document contains the following topics:

- Chapter 1, "Introduction to CICS TCP/IP," on page 1 provides an overview of CICS TCP/IP.
- Chapter 2, "Setting up and configuring CICS TCP/IP," on page 23 describes the steps required to configure CICS TCP/IP.
- Chapter 3, "Configuring the CICS Domain Name System cache," on page 93 describes how to configure the CICS domain name server cache.
- Chapter 4, "Managing IP CICS sockets," on page 103 explains how to start and stop (enable and disable) CICS TCP/IP.
- Chapter 5, "Writing your own listener," on page 117 discusses writing your own listener.
- Chapter 6, "Application programming guide," on page 123 describes how to write applications that use the sockets application programming interface (API). It describes typical sequences of calls for client, concurrent server (with associated child server processes), and iterative server programs.
- Chapter 7, "C language application programming," on page 157 describes the C language API provided by CICS TCP/IP.
- Chapter 8, "Sockets extended API," on page 223 describes the sockets extended API.
- Appendix A, "Original COBOL application programming interface (EZACICAL)," on page 367 describes the EZACICAL API.
- Appendix B, "Return codes," on page 397 describes system-wide message numbers and codes set by the system calls.

- Appendix C, "GETSOCKOPT/SETSOCKOPT command values," on page 415 provides the decimal or hexadecimal values associated with the GETSOCKOPT/SETSOCKOPT OPTNAMES supported by the APIs discussed in this document.
- Appendix D, "CICS sockets messages," on page 417 contains CICS socket interface messages.
- Appendix E, "Sample programs," on page 463 contains samples of the following programs:
  - EZACICSC - An IPv4 child server
  - EZACICSS - An IPv4 iterative server
  - EZACIC6C - An IPv6 child server
  - EZACIC6S - An IPv6 iterative server
  - EZACICAC - An assembler child server
  - EZACICAS - An assembler iterative server
- Appendix F, "Related protocol specifications," on page 563 lists the related protocol specifications for TCP/IP.
- "Information APARs and technotes" lists information APARs for IP and SNA documents.
- "Accessibility" contains information about features that help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully.
- "Bibliography" contains descriptions of the documents in the z/OS Communications Server library.

How to use this document

To use this document, 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 of 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 607.

Conventions and terminology 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 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

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

Requirement
Dependencies, prerequisites

Result Indicates the outcome

Prerequisite and related information

z/OS Communications Server function is described in the z/OS Communications Server library. Descriptions of those documents are listed in "z/OS Communications Server information" on page 595, in the back of this document.

Required information

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

Related information

This section contains subsections on:
- "Softcopy information"
- "Other documents" on page xxv
- "Redbooks" on page xxvi
- "Where to find related information on the Internet" on page xxvi
- "Using LookAt to look up message explanations" on page xxviii
- "Using IBM Health Checker for z/OS" on page xxviii

Softcopy information

Softcopy publications are available in the following collections:

Titles	Order Number	Description
z/OS V1R9 Collection	SK3T-4269	This is the CD collection shipped with the z/OS product. It includes the libraries for z/OS V1R9, in both BookManager and PDF formats.
z/OS Software Products Collection	SK3T-4270	This CD includes, in both BookManager and PDF formats, the libraries of z/OS software products that run on z/OS but are not elements and features, as well as the *Getting Started with Parallel Sysplex*® bookshelf.
z/OS V1R9 and Software Products DVD Collection	SK3T-4271	This collection includes the libraries of z/OS (the element and feature libraries) and the libraries for z/OS software products in both BookManager and PDF format. This collection combines SK3T-4269 and SK3T-4270.
z/OS Licensed Product Library	SK3T-4307	This CD includes the licensed documents in both BookManager and PDF format.

Titles	Order Number	Description
IBM System z Redbooks Collection	SK3T-7876	The Redbooks selected for this CD series are taken from the IBM Redbooks inventory of over 800 books. All the Redbooks that are of interest to the zSeries platform professional are identified by their authors and are included in this collection. The zSeries subject areas range from e-business application development and enablement to hardware, networking, Linux, solutions, security, parallel sysplex, and many others.

Other documents

For information about z/OS products, refer to *z/OS Information Roadmap* (SA22-7500). The Roadmap describes what level of documents are supplied with each release of z/OS Communications Server, as well as describing each z/OS publication.

Relevant RFCs are listed in an appendix of the IP documents. Architectural specifications for the SNA protocol are listed in an appendix of the SNA documents.

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

Title	Number
DNS and BIND, Fourth Edition, O'Reilly and Associates, 2001	ISBN 0-596-00158-4
Routing in the Internet , Christian Huitema (Prentice Hall PTR, 1995)	ISBN 0-13-132192-7
sendmail, Bryan Costales and Eric Allman, O'Reilly and Associates, 2002	ISBN 1-56592-839-3
SNA Formats	GA27-3136
TCP/IP Illustrated, Volume I: The Protocols, W. Richard Stevens, Addison-Wesley Publishing, 1994	ISBN 0-201-63346-9
TCP/IP Illustrated, Volume II: The Implementation, Gary R. Wright and W. Richard Stevens, Addison-Wesley Publishing, 1995	ISBN 0-201-63354-X
TCP/IP Illustrated, Volume III, W. Richard Stevens, Addison-Wesley Publishing, 1995	ISBN 0-201-63495-3
TCP/IP Tutorial and Technical Overview	GG24-3376
Understanding LDAP	SG24-4986
z/OS Cryptographic Service System Secure Sockets Layer Programming	*SC24-5901*
z/OS Integrated Security Services LDAP Client Programming	*SC24-5924*
z/OS Integrated Security Services LDAP Server Administration and Use	*SC24-5923*
z/OS JES2 Initialization and Tuning Guide	*SA22-7532*
z/OS Problem Management	*G325-2564*
z/OS MVS Diagnosis: Reference	*GA22-7588*
z/OS MVS Diagnosis: Tools and Service Aids	*GA22-7589*
z/OS MVS Using the Subsystem Interface	*SA22-7642*
z/OS Program Directory	*GI10-0670*
z/OS UNIX System Services Command Reference	*SA22-7802*
z/OS UNIX System Services Planning	*GA22-7800*
z/OS UNIX System Services Programming: Assembler Callable Services Reference	*SA22-7803*

Title	Number
z/OS UNIX System Services User's Guide	*SA22-7801*
z/OS XL C/C++ Run-Time Library Reference	*SA22-7821*
System z9 and zSeries OSA-Express Customer's Guide and Reference	*SA22-7935*

Redbooks

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

Title	Number
Communications Server for z/OS V1R8 TCP/IP Implementation, Volume 1: Base Functions, Connectivity, and Routing	SG24-7339
Communications Server for z/OS V1R8 TCP/IP Implementation, Volume 2: Standard Applications	SG24-7340
Communications Server for z/OS V1R8 TCP/IP Implementation, Volume 3: High Availability, Scalability, and Performance	SG24-7341
Communications Server for z/OS V1R8 TCP/IP Implementation, Volume 4: Policy-Based Network Security	SG24-7342
IBM Communication Controller Migration Guide	SG24-6298
IP Network Design Guide	SG24-2580
Managing OS/390 TCP/IP with SNMP	SG24-5866
Migrating Subarea Networks to an IP Infrastructure Using Enterprise Extender	SG24-5957
SecureWay Communications Server for OS/390 V2R8 TCP/IP: Guide to Enhancements	SG24–5631
SNA and TCP/IP Integration	SG24-5291
TCP/IP in a Sysplex	SG24-5235
TCP/IP Tutorial and Technical Overview	GG24-3376
Threadsafe Considerations for CICS	SG24-6351

Where to find related information on the Internet

z/OS

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

http://www.ibm.com/servers/eserver/zseries/zos/

z/OS Internet Library

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

http://www.ibm.com/servers/eserver/zseries/zos/bkserv/

IBM Communications Server product

The primary home page for information about z/OS Communications Server

http://www.software.ibm.com/network/commserver/

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

http://www.software.ibm.com/network/commserver/support/

IBM Systems Center publications

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

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

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)

http://www.ibm.com/support/techdocs/atsmastr.nsf

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 II11334.

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.

Using LookAt to look up message explanations

LookAt is an online facility that lets you look up explanations for most of the IBM messages you encounter, as well as for some system abends and codes. Using LookAt to find information is faster than a conventional search because in most cases LookAt goes directly to the message explanation.

You can use LookAt from these locations to find IBM message explanations for z/OS elements and features, z/VM®, VSE/ESA™, and Clusters for AIX® and Linux™:

- The Internet. You can access IBM message explanations directly from the LookAt Web site at www.ibm.com/servers/eserver/zseries/zos/bkserv/lookat/.
- Your z/OS TSO/E host system. You can install code on your z/OS systems to access IBM message explanations using LookAt from a TSO/E command line (for example: TSO/E prompt, ISPF, or z/OS UNIX System Services).
- Your Microsoft® Windows® workstation. You can install LookAt directly from the z/OS Collection (SK3T-4269) or the *z/OS and Software Products DVD Collection* (SK3T-4271) and use it from the resulting Windows graphical user interface (GUI). The command prompt (also known as the DOS > command line) version can still be used from the directory in which you install the Windows version of LookAt.
- Your wireless handheld device. You can use the LookAt Mobile Edition from www.ibm.com/servers/eserver/zseries/zos/bkserv/lookat/lookatm.html with a handheld device that has wireless access and an Internet browser (for example: Internet Explorer for Pocket PCs, Blazer or Eudora for Palm OS, or Opera for Linux handheld devices).

You can obtain code to install LookAt on your host system or Microsoft Windows workstation from:

- A CD-ROM in the z/OS Collection (SK3T-4269).
- The *z/OS and Software Products DVD Collection* (SK3T-4271).
- The LookAt Web site (click **Download** and then select the platform, release, collection, and location that suit your needs). More information is available in the LOOKAT.ME files available during the download process.

Using IBM Health Checker for z/OS

IBM Health Checker for z/OS is a z/OS component that installations can use to gather information about their system environment and system parameters to help identify potential configuration problems before they impact availability or cause outages. Individual products, z/OS components, or ISV software can provide checks that take advantage of the IBM Health Checker for z/OS framework. This book might refer to checks or messages associated with this component.

For additional information about checks and about IBM Health Checker for z/OS, see *IBM Health Checker for z/OS: User's Guide*. Starting with z/OS V1R4, z/OS users can obtain the IBM Health Checker for z/OS from the z/OS Downloads page at http://www.ibm.com/servers/eservers/zseries/zos/downloads/.

SDSF also provides functions to simplify the management of checks. See *z/OS SDSF Operation and Customization* for additional information.

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:

- Go to the z/OS contact page at:

  http://www.ibm.com/servers/eserver/zseries/zos/webqs.html

  There you will find the feedback page where you can enter and submit your comments.
- 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 you are commenting on (for example, a section number, a page number or a table number).

Summary of changes

Summary of changes
for SC31-8807-04
z/OS Version 1 Release 9

This document contains information previously presented in SC31-8807-03, which supports z/OS Version 1 Release 7.

The information in this document 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 document 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 document 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

- CICS sockets enhancements, see "Configuring the CICS TCP/IP environment" on page 51.
- Enable application identifier in NMI, SMF, and Netstat, see "Configuring the CICS TCP/IP environment" on page 51.
- Enable application identifier in NMI, SMF, and Netstat, see "TYPE parameter" on page 54.
- New sample for CICS and IMS™ ASCII/EBCDIC translation, see "Using data translation programs for socket call interface" on page 347.
- MLDv2 and IGMPv3 support, see "Structures used in socket calls" on page 160.
- IPv6 scoped address architecture API, see Chapter 7, "C language application programming," on page 157.

Deleted information

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

This document 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 of some content in this document–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 accessibility of information in our documents.

Summary of changes
for SC31-8807-03
z/OS Version 1 Release 7

This document contains information previously presented in SC31-8807-02, which supports z/OS Version 1 Release 5.

The information in this document 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 document 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 document 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

- Application Transparent Transport Layer Security (AT-TLS) exploitation. See "Application Transparent Transport Layer Security" on page 152 for more information.
- Support for CICS Transaction Server (TS) Open Transaction Environment (OTE). See "Open TCB measurements" on page 43 for more information.
- Performance enhancements.

Changed information

- Updated screens for the configuration transaction interface EZAC. See "Customizing the configuration data set" on page 70 for more information.
- Updated screens for the EZAO operator transaction interface. See "IP CICS socket interface management" on page 104 for more information.

This document 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 of some content in this document–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 documents.

Summary of changes
for SC31-8807-02
z/OS Version 1 Release 5

This document contains information previously presented in SC31-8807-01, which supports z/OS Version 1 Release 4. The information in this document supports both IPv6 and IPv4. Unless explicitly noted, information describes IPv4 networking protocol. IPv6 support is qualified within the text.

New information

- EZACIC14 and EZACIC15 data translation programs for EBCDIC and ASCII translation (see “Conversion routines” on page 19, “Data conversion routines” on page 152, “EZACIC14” on page 363, and “EZACIC15” on page 365)
- A topic on configuring the IPv6 listener for IPv6 (see “Rules for configuring the IBM-supplied listener for IPv6” on page 20)
- Optional program definitions for EZACIC6S, EZACIC6C, EZACICAC, and EZACICAS (see “Optional programs, CICS transaction and program definition needed” on page 31)
- File definitions for EZACACHE (see “File definitions” on page 34)
- IPv6 examples for EZACICD (see “Building the configuration data set with EZACICD” on page 51 and “JCL for the configuration macro” on page 66)
- Recommendations for CICS DNS Caching and DNS/WLM support (see Chapter 3, “Configuring the CICS Domain Name System cache,” on page 93)
- IPv6 information throughout the topic about socket addresses (see “Socket addresses” on page 130)
- IPv6 information and examples throughout the topic about listener output format (see “Listener output format” on page 136)
- IPv6 information on listener configuration (see “Writing your own security/transaction link module for the listener” on page 143)
- Information on C structures (see Table 19 on page 160)
- IPv6 information throughout the topic about C socket calls (see “C socket calls” on page 163)
- A topic about address testing macros (see “Address Testing Macros” on page 220)
- IPv6 information throughout the topic about code call instructions (see “Code CALL instructions” on page 226)
- EZACIC09 for TCP/IP bit string processing (see “Bit string processing” on page 348 and “EZACIC09” on page 359)
- EZACICAC, EZACICAS, EZACIC6C, and EZACIC6S sample programs (see “EZACICAC” on page 529, “EZACICAS” on page 540, “EZACIC6C” on page 493, and “EZACIC6S” on page 505)
- CICS sockets messages (see “EZY1218—EZY1366” on page 417)

Changed information

- Information on “Using IBM’s environmental support” on page 117
- Information throughout the topic about code call instructions (see “Code CALL instructions” on page 226)
- CICS resource definition information and examples (see “CICS — Defining CICS TCP/IP resources” on page 26)
- Information about Monitor Control Table entries (see “CICS monitoring” on page 37)
- EZAC and EZAO transaction screens (see “Configuration transaction (EZAC)” on page 70 and “IP CICS socket interface management” on page 104)
- Information on automatically starting and stopping CICS TCP/IP (see “Starting and stopping CICS automatically” on page 103 and “CICS program list table (PLT)” on page 46)
- CICS sockets environment configuration file information throughout “Configuring the CICS TCP/IP environment” on page 51
- The description of TERMAPI has been updated at “TERMAPI” on page 343

- The description of the *max_sock*, MAXSOC, and MAX-SOCK parameters (see “Parameters” on page 203, “Parameter values set by the application” on page 277, and “Parameter values to be set by the application” on page 382)
- Information on TCP/IP host addressing (see “Addressing TCP/IP hosts” on page 5)
- The description of the socket TCP/IP call (see “SOCKET” on page 10)
- The topic about GIVESOCKET and TAKESOCKET calls (see “GIVESOCKET and TAKESOCKET calls” on page 16)
- The topic about conversion routines (see “Conversion routines” on page 19)
- Call for the client application (see Table 8 on page 125)
- IPv6 information about EZACICAL (see Appendix A, “Original COBOL application programming interface (EZACICAL),” on page 367)
- CICS sockets messages (see “EZY1218—EZY1366” on page 417)
- EZACICSC and EZACICSS sample programs (see “EZACICSC” on page 463 and “EZACICSS” on page 472)

Deleted information:

- The SIOCADDRT, SIOCDELRT, SIOCGIFFLAGS, SIOCGIFMETRIC, SIOCGIFNETMASK, SIOCSIFDSTADDR, SIOCSIFFLAGS, and SIOCSIFMETRIC parameters (see “ioctl()” on page 203)

This document 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.

Starting with z/OS V1R4, you will notice changes in the style and structure of some content in this document–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 documents.

Chapter 1. Introduction to CICS TCP/IP

The IP CICS socket API and the IBM supplied listener is IPv4 and IPv6 enabled.

CICS Transaction Server (CICS TS) is an online transaction processing system. Application programs using CICS 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. CICS TCP/IP offers CICS users an alternative to SNA, the TCP/IP family of protocols for those users whose native communications protocol is TCP/IP.

CICS TCP/IP allows remote users to access CICS client/server applications over TCP/IP Internets. Figure 1 shows how these two products give remote users peer-to-peer communication with CICS applications.

It is important to understand that CICS TCP/IP is primarily intended to support ***peer-to-peer*** applications, as opposed to the traditional CICS mainframe interactive applications in which the CICS system contained all program 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 using Telnet. With Telnet, you should be able to access existing 3270-style basic mapping support (BMS) applications without modification and without the need for additional programming. Use CICS TCP/IP when you are developing new peer-to-peer applications in which both ends of the connection are programmable.

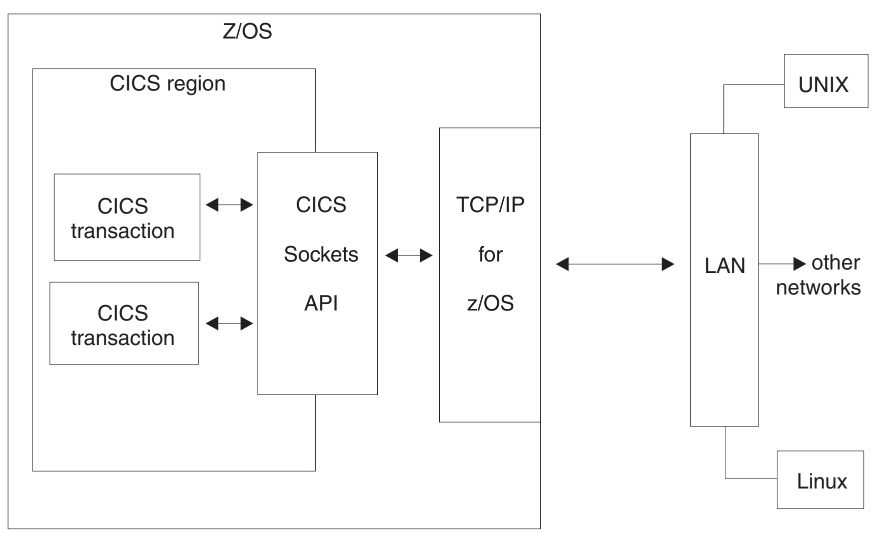


Figure 1. The use of CICS sockets

CICS TCP/IP provides a variant of the Berkeley Software Distribution 4.3 sockets 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 your CICS application programs can use to set up connections, send and receive data, and perform general communications control functions. The programs can be written in COBOL, PL/I, assembler language, or the C language.

TCP/IP Internets

This topic describes some of the basic ideas behind the TCP/IP family of protocols. For more detailed and comprehensive treatments of this subject, see the documents about TCP/IP listed in http://www.ibm.com/servers/eserver/zseries/zos/bkserv/

Like SNA, TCP/IP is a communication protocol 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.

Telnet

TCP/IP Services supports traditional 3270 mainframe interactive (MFI) applications with an emulator function called Telnet (TN3270). For these applications, all program logic is housed in the mainframe, and the remote host uses only that amount of logic necessary to provide basic communication services. Thus, if your requirement is simply to provide access from a remote TCP/IP host to existing CICS MFI applications, you should probably consider Telnet rather than CICS 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 CICS TCP/IP, remote client systems can initiate communications with CICS and cause a CICS transaction to start. It is anticipated that this is the most common mode of operation. (Alternatively, the remote system can act as a server with CICS initiating the conversation.)

TCP, UDP, and IP

TCP/IP is a large family of protocols that is named after its two most important members. Figure 2 on page 3 shows the TCP/IP protocols used by CICS TCP/IP, in terms of the layered Open Systems Interconnection (OSI) model, which is widely used to describe data communication systems. For CICS 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.

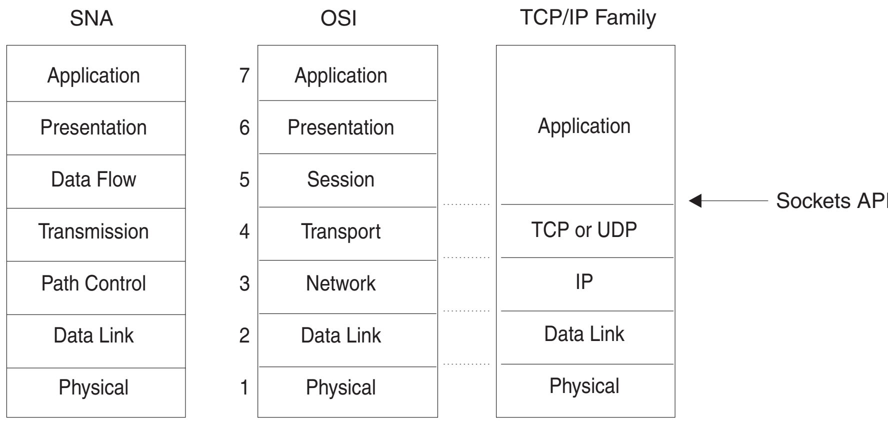


Figure 2. TCP/IP protocols compared to the OSI model and SNA

The protocols implemented by TCP/IP Services and used by CICS TCP/IP are shown in the right hand column 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. 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 enables 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 APIs enable you to:
- Interrogate the network system to get names and status of relevant resources
- Perform system and control functions as required

CICS TCP/IP provides three TCP/IP socket application program interfaces (APIs), similar to those used on UNIX systems. One interfaces to C language programs, the other two to COBOL, PL/I, and assembler language programs.
- **C language**. Historically, TCP/IP has been linked to 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. See Chapter 7, "C language application programming," on page 157 for the sockets calls provided by TCP/IP Services.

- **Sockets Extended API (COBOL, PL/I, assembler language)**. The Sockets Extended API 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. If you are writing new TCP/IP applications in COBOL, PL/I, or assembler language, you might prefer to use the Sockets Extended API. See Chapter 8, "Sockets extended API," on page 223 for details of this interface.
- **Version 2.2.1 (COBOL, PL/I, assembler language)**. This is the API that was offered to users of the original release of CICS TCP/IP. It is similar in use to the Sockets Extended API. The Version 2.2.1 API is available for those who want to maintain Version 2.2.1 programs. This interface is described in Appendix A, "Original COBOL application programming interface (EZACICAL)," on page 367.

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 socket interface to the transport layer protocols, and raw socket interface to the network layer protocols. All three socket types are discussed here for background purposes. While CICS supports stream and datagram sockets, stream sockets provide the most reliable form of data transfer offered by TCP/IP.

Stream sockets transmit data between TCP/IP hosts that are already connected to one another. Data is transmitted in a continuous stream; in other words, there are no record length or new-line character boundaries between data. Communicating processes [1] must agree on a scheme to ensure that both client and server have received all data. One way of doing this is for the sending process to send the length of the data, followed by the data itself. The receiving process reads the length and then loops, accepting data until all of it has been transferred.

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

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

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

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

Addressing TCP/IP hosts

The following topic describes how one TCP/IP host addresses another TCP/IP host. [2]

Address families: An address family defines a specific addressing format. Applications that use the same addressing family have a common scheme for addressing socket endpoints. TCP/IP for CICS supports the AF_INET and the AF_INET6 address family. See the API topic in *z/OS Communications Server: IPv6 Network and Application Design Guide* for more information about IPv6 programming issues.

Socket addresses: A socket address in the AF_INET family contains four fields:

- The name of the address family itself (AF_INET)
- A port
- An IPv4 Internet address
- An eight-byte reserved field

In COBOL, an IPv4 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).
```

A socket address in the AF_INET6 family contains five fields:

- The name of the address family itself (AF_INET6)
- A port
- Flow information indicating traffic class and flow label
- An IPv6 Internet address
- A scope ID indicating link scope

In COBOL, an IPv6 socket address looks like this:

```
01 NAME.
         03 FAMILY      PIC 9(4) BINARY.
         03 PORT        PIC 9(4) BINARY.
         03 FLOWINFO    PIC 9(8) BINARY.
         03 IP-ADDRESS.
            05 FILLER   PIC 9(16) BINARY.
            05 FILLER   PIC 9(16) BINARY.
         03 SCOPE-ID    PIC 9(8) BINARY.
```

Programs, such as servers, that support both AF_INET and AF_INET6 sockets, should code socket address structures using the SOCKADDR layout as described in the SYS1.MACLIB(BPXYSOCK). In COBOL, a socket address structure to support both AF_INET and AF_INET6 looks like this:

```
01  SOCKADDR.
           05 SOCK-FAMILY                 PIC 9(4) BINARY.
              88 SOCK-FAMILY-IS-AFINET    VALUE 2.
```

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.

```
      88 SOCK-FAMILY-IS-AFINET6   VALUE 19.
   05 SOCK-DATA                   PIC X(26).
   05 SOCK-SIN REDEFINES SOCK-DATA.
      10 SOCK-SIN-PORT            PIC 9(4) BINARY.
      10 SOCK-SIN-ADDR            PIC 9(8) BINARY.
      10 FILLER                   PIC X(8).
      10 FILLER                   PIC X(12).
   05 SOCK-SIN6 REDEFINES SOCK-DATA.
      10 SOCK-SIN6-PORT           PIC 9(4) BINARY.
      10 SOCK-SIN6-FLOWINFO       PIC 9(8) BINARY.
      10 SOCK-SIN6-ADDR.
         15 FILLER                PIC 9(16) BINARY.
         15 FILLER                PIC 9(16) BINARY.
      10 SOCK-SIN6-SCOPEID        PIC 9(8) BINARY.
```

The IPv4 or IPv6 socket address structure is in every call that addresses another TCP/IP host.

This structure contains the following fields:

FAMILY
A halfword that defines the addressing family being used. In CICS, FAMILY is set to a value of a decimal 2 (that specifies the AF_INET Internet address family) or a value of a decimal 19 (that specifies the AF_INET6 Internet address family). [3]

PORT Identifies the application port number and must be specified in network byte order.

FLOWINFO
Belongs to the IPv6 socket address structure and is 4 bytes in binary format indicating traffic class and flow label. This field is currently not implemented.

IP-ADDRESS
The Internet address of the network interface used by the application. It must be specified in network byte order.

RESERVED
Belongs to the IPv4 socket address structure and should be set to all zeros.

SCOPE-ID
Belongs to the IPv6 socket address structure and is used to specify link scope for an IPv6 address as an interface index. If specified, and the destination is not link local, then the socket call fails.

Internet (IP) addresses: An Internet address (also known as an IP address) is a 32-bit field that represents an IPv4 network interface or a 128-bit field that represents an IPv6 network interface. An IP address is commonly represented in dotted decimal notation, such as *129.5.25.1*, or in colon-hexadecimal notation, such as *2001:0db8:129:5:25::1*. Every Internet address within an administered AF_INET or AF_INET6 domain must be unique. A common misunderstanding is that a host must have only one Internet address. In fact, a single host can have several Internet addresses, one for each network interface. With IPv6, a single interface can even have multiple addresses, such as link-local, site-local, and global unicast.

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

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

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.

IPv4 Example: An MVS system with an IP address of 129.9.12.7 might have CICS as port 2000, and Telnet as port 23. In this example, a client desiring connection to CICS would issue a CONNECT call, requesting port 2000 at IP address 129.9.12.7.

IPv6 Example: An MVS system with an IPv6 IP address of 2001:0DB8::206:2AFF:FE66:C800 might have CICS as port 2000, and Telnet as port 23. In this example, a client that wants to connect to CICS would issue a CONNECT call, requesting port 2000 at IP address 2001:0DB8::206:2AFF:FE66:C800.

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

Multiple sockets can share the same port and, for CICS, all server applications and the listener share the same port. For client applications, the bind (or connect) socket calls assign a port to the socket that is different from the listener or server port or any other client ports. Normally, client applications do not share ports, but they can if you specify the SO_REUSEADDR socket option.

Domain names: Because dotted decimal or colon-hexadecimal 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, GETHOSTBYADDR, GETADDRINFO, and GETNAMEINFO) that helps 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 for a diagram of the logic flow for a typical client and server. As you study this diagram, keep in mind the fact that a concurrent server typically starts before the client does, and waits for the client to request connection at step **3**. It then continues to wait for additional client requests after the client connection is closed.

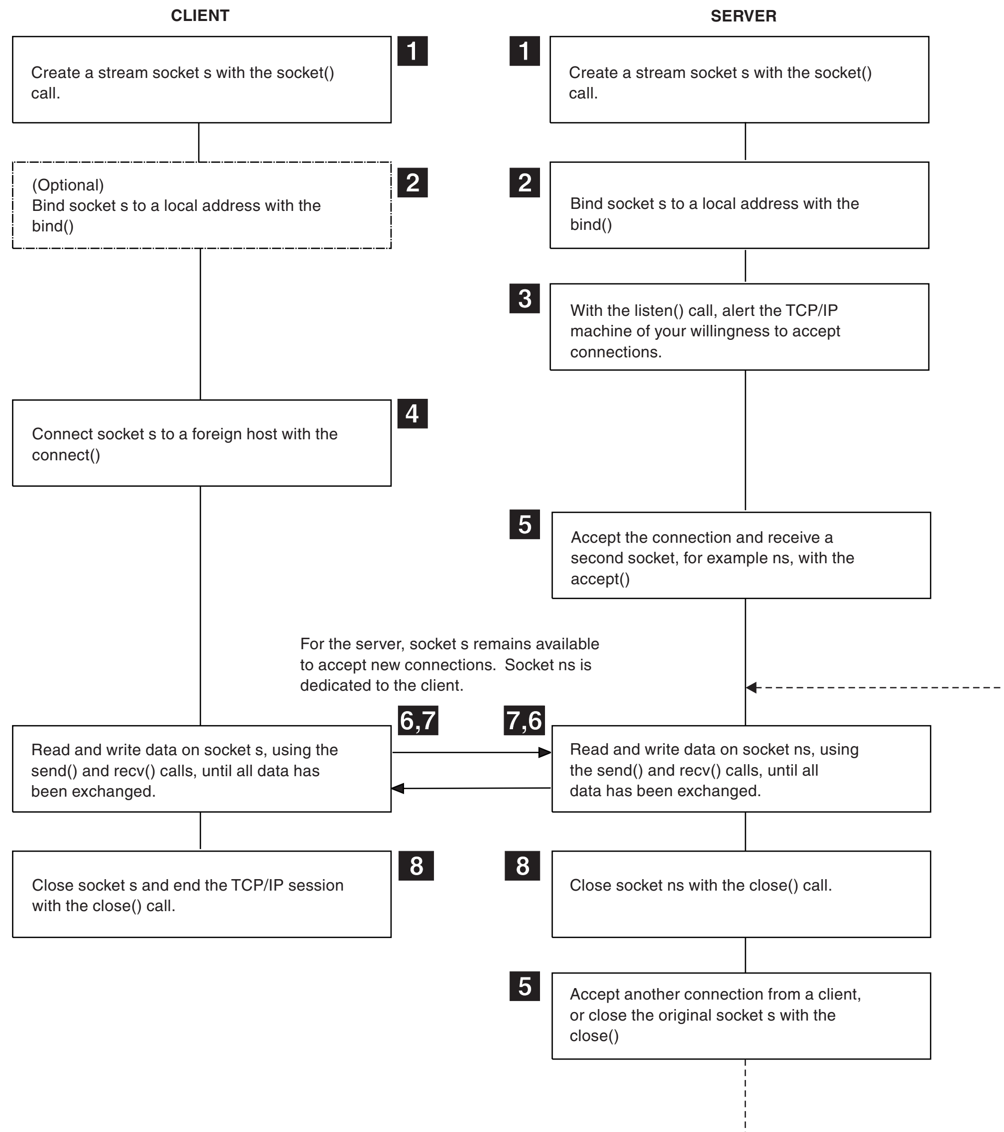


Figure 3. A typical client-server session

Concurrent and iterative servers

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

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

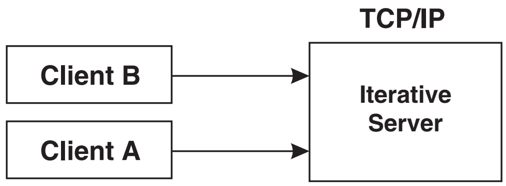


Figure 4. An iterative server

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 CICS listener. It is described in "The IBM listener" on page 134.

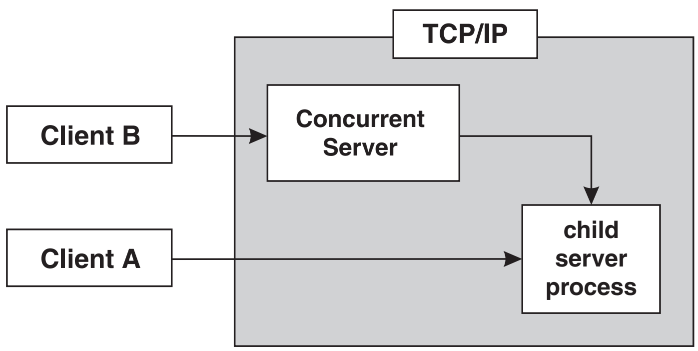


Figure 5. A concurrent server

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

The basic socket calls

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 8, "Sockets extended API," on page 223.

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; the next topic discusses the typical call sequence for a client. This is the logical presentation sequence because the server is usually already in running before the client is started. The step numbers (such as **5**) in this topic refer to the steps in Figure 3 on page 8.

SOCKET

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 of AF_INET or AF_INET6, 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.

For an example of the SOCKET call, see "SOCKET" on page 340.

BIND

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

- The socket descriptor that was just returned by the SOCKET call
- The number of the port on which the server wants to provide its service
- The IP address of the network connection on which the server is listening

If the application wants to receive connection requests from any network interface, the IP address should be set to zeros specifying INADDR_ANY for IPv4 or the IPv6 unspecified address (in6addr_any).

For an example of the BIND call, see "BIND" on page 229.

LISTEN

After the bind, the server has established a specific IP address and port upon which other TCP/IP hosts can request connection. Now it must notify the TCP/IP address space that it intends to listen for connections on this socket. The server does this with the LISTEN **3** call, which puts the socket into passive open mode. Passive open mode describes a socket that can accept connection requests, but cannot be used for communication. A passive open socket is used by a listener program like the CICS 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 are queued.

For an example of the LISTEN call, see "LISTEN" on page 289.

ACCEPT

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

The default mode for Accept is blocking. Accept behavior changes when the socket is nonblocking. 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 nonblocking 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.

For an example of the ACCEPT call, see "ACCEPT" on page 226.

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.

GIVESOCKET and TAKESOCKET

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 16 for more information about these calls.

READ and WRITE

After 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 “READ/WRITE calls — the conversation” on page 13 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.

The SOCKET call

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

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

See “SOCKET” on page 340 for a sample of the SOCKET call.

The 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 **4**.

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

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

See “CONNECT” on page 233 for an example of the CONNECT call.

READ/WRITE calls — the conversation

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

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

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

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

- For an example of the READ call, see “READ” on page 294.
- For an example of the WRITE call, see “WRITE” on page 344.

The 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. For an example of the CLOSE call, see “CLOSE” on page 232.

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.

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 14.

```
WORKING-STORAGE SECTION.
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'SELECT'.
    01  MAXSOC          PIC 9(8) BINARY VALUE 50.
    01  TIMEOUT.
        03  TIMEOUT-SECONDS   PIC 9(8) BINARY.
        03  TIMEOUT-MILLISEC  PIC 9(8) BINARY.
    01  RSNDMASK        PIC X(50).
    01  WSNDMASK        PIC X(50).
    01  ESNDMASK        PIC X(50).
    01  RRETMASK        PIC X(50).
    01  WRETMASK        PIC X(50).
    01  ERETMASK        PIC X(50).
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.

PROCEDURE DIVISION.
     CALL 'EZASOKET' USING SOC-FUNCTION MAXSOC TIMEOUT
                    RSNDMASK WSNDMASK ESNDMASK
                    RRETMASK WRETMASK ERETMASK
                    ERRNO RETCODE.
```

Figure 6. The SELECT call

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

The example also indicates a timeout. If the timeout 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 timeout parameter is nonzero, SELECT only waits the timeout amount of time for at least one socket to become ready under the indicated conditions. This is useful for applications servicing multiple connections that cannot afford to wait for data on a single connection. If the xSNDMASK bits are all zero, SELECT acts as a timer.

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

To maximize performance, a server should only test those sockets that are active. The SELECT call allows an application to select which sockets are 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 3, you have to pass a 4-byte bit string, because this is the minimum length. If the highest number is 32, you must pass 8 bytes (2 fullwords).

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

```
INT(highest socket descriptor / 32) + 1
```

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

Table 1. First fullword passed in a bit string in select

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

In these examples, we use standard assembler numbering notation; the leftmost bit or byte is relative 0.

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'0000002000000001'.

The bits in the second fullword represent the socket descriptor numbers shown in Table 2.

Table 2. Second fullword passed in a bit string in select

Socket descriptor numbers represented by byte	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
Byte 4	63	62	61	60	59	58	57	56
Byte 5	55	54	53	52	51	50	49	48

Table 2. Second fullword passed in a bit string in select (continued)

Socket descriptor numbers represented by byte	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
Byte 6	47	46	45	44	43	42	41	40
Byte 7	39	38	37	36	35	34	33	32

If you develop your program in COBOL or PL/I, the EZACIC06 routine, which is provided as part of TCP/IP Services, makes it easier 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 is 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. An example of the use of the IOCTL call is shown in "IOCTL" on page 278.

The IOCTL call has many functions; establishing blocking mode is only one of its functions. The value in COMMAND determines which function IOCTL performs. The REQARG of 0 specifies nonblocking. (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

Tasks use the GIVESOCKET and TAKESOCKET functions to pass sockets from parent to child.

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

- The socket number it wants to give
- The domain of the socket
- Its own name [5]
- 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 receives 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 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 uses. 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.)

Sockets given and taken must be of the same domain type. When GIVESOCKET is giving an AF_INET socket, then TAKESOCKET must only take an AF_INET socket. When GIVESOCKET is giving an AF_INET6 socket, then TAKESOCKET must only take an AF_IENT6 socket. EBADF is set if the socket taken does not match the domain in the tasksocket() request.

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, dissociates itself from that connection, and so on.

Summary: To summarize, 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 , 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 CICS START/RETRIEVE commands.
- The concurrent server issues the SELECT call to determine when the GIVESOCKET has successfully completed.
- The subtask calls TAKESOCKET with the concurrent server's ID and socket descriptor and uses the resulting socket descriptor for communication with the client.
- When the GIVESOCKET has successfully completed, the concurrent server issues the CLOSE call to complete the handoff.

An example of a concurrent server is the CICS listener. It is described in "The IBM listener" on page 134. Figure 5 on page 9 shows a concurrent server.

What you must have to run CICS TCP/IP

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

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

CICS TCP/IP components

In terms of CICS operation, the CICS TCP/IP feature is a task-related user exit (TRUE) mechanism known as an adapter. The adapting facility that it provides is between application programs that need to access TCP/IP and the manager of the TCP/IP resource.

CICS TCP/IP has the following main components:

- The **stub program** is link-edited to each application program that wants to use it. It intercepts requests issued by the calling application program and causes CICS to pass control to the TRUE.
- The **TRUE** mechanism enables programs to pass calls to the subtask and to the TCP/IP address space.
- CICS TCP/IP supports two methods for accessing TCP/IP
  - The MVS subtask translates commands for accessing TCP/IP into a form acceptable to the TCP/IP resource manager and then passes control to the resource manager. The subtask also handles the MVS waits incurred during socket calls.
  - Using CICS Open Transaction Environment (OTE). The TRUE mechanism accesses TCP/IP directly, not requiring an MVS subtask for blocking commands.
- The **Administration Routine** contains the EXEC CICS ENABLE and DISABLE commands that are used to install and withdraw the TRUE program.
- The **Configuration System** configures the interface and its listeners.

A summary of what CICS TCP/IP provides

Figure 7 on page 19 shows how CICS TCP/IP allows your CICS applications to access the TCP/IP network. It shows that CICS TCP/IP makes the following facilities available to your application programs:

The socket calls

Socket calls are shown in Steps 1 and 2 in Figure 7 on page 19.

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

Call type	**IP CICS TCP API function**
Basic calls:	`ACCEPT`, `BIND`, `CLOSE`, `CONNECT`, `LISTEN`, `SHUTDOWN`

Call type	IP CICS TCP API function
Read/Write calls:	`READ, READV, RECV, RECVFROM, RECVMSG, SEND, SENDMSG, SENDTO, WRITE, WRITEV`
Advanced calls:	`FCNTL, FREEADDRINFO, GETADDRINFO, GETHOSTBYADDR, GETHOSTBYNAME, GETHOSTNAME, GETNAMEINFO, GETPEERNAME, GETSOCKNAME, GETSOCKOPT, IOCTL, NTOP, PTON, SELECT, SELECTEX, SETSOCKOPT`
IBM-specific calls:	`GETCLIENTID, GIVESOCKET, INITAPI, INITAPIX, TAKESOCKET`

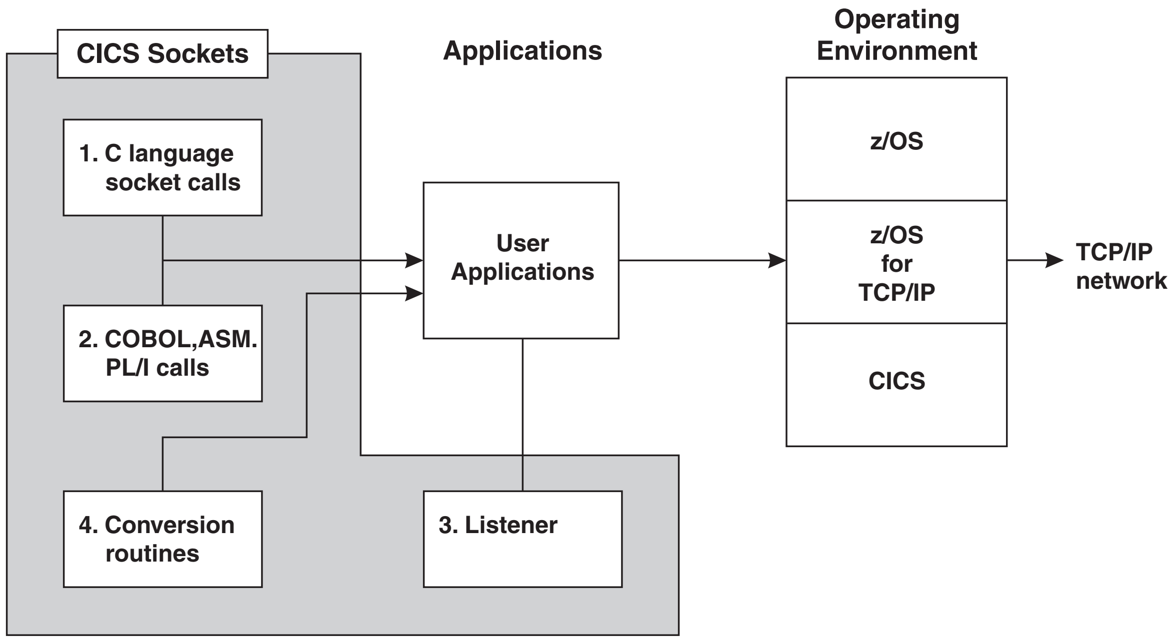


Figure 7. How user applications access TCP/IP networks with CICS TCP/IP (run-time environment)

CICS TCP/IP provides for both connection-oriented and connectionless (datagram) services. CICS does not support the IP (raw socket) protocol.

The listener

CICS TCP/IP includes a concurrent server application, called the IBM Listener, which is a CICS transaction that uses the EZACIC02 program to perform its function.

The IBM listener, EZACIC02, allows for WLM registration and deregistration in support of connection balancing. See *z/OS Communications Server: IP Configuration Reference* for information about BIND-based DNS and connection balancing.

Conversion routines

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

- An EBCDIC-to-ASCII conversion routine that converts EBCDIC data to the ASCII format used in TCP/IP networks and workstations. The routine is run by calling module EZACIC04, which uses an EBCDIC-to-ASCII translation table as described in *z/OS Communications Server: IP Configuration Reference*.

- A corresponding ASCII-to-EBCDIC conversion routine, EZACIC05, which uses an ASCII-to-EBCDIC translation table as described in *z/OS Communications Server: IP Configuration Reference*.
- An alternative EBCDIC-to-ASCII conversion routine. It is run by calling EZACIC14, which uses the translation table listed in "EZACIC14" on page 363.
- A corresponding alternate ASCII-to-EBCDIC conversion routine, EZACIC15, which uses the translation table listed in "EZACIC15" on page 365.

  Tip: A sample translation routine is also supplied in the EZACICTR member of the SEZAINST library. You can modify this member to use alternate EBCDIC-to-ASCII and ASCII-to-EBCDIC translations, including custom translations. See comments in the EZACICTR member for more details.
- 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 or SELECTEX call.
- A routine that decodes the indirectly addressed, variable-length list (hostent structure) returned by the GETHOSTBYADDR and GETHOSTBYNAME calls. This function is provided by calling module EZACIC08.
- A routine that decodes the indirectly addressed, variable-length list (addrinfo structure) returned by the GETADDRINFO call. This function is provided by calling module EZACIC09.

Rules for configuring the IBM-supplied listener for IPv6

The following rules apply when configuring the IBM-supplied listener for IPv6:

- You must enable the z/OS system that the IPv6 listener uses for IPv6. See *z/OS Communications Server: IP Configuration Reference* for information on IPv6 system configuration.
- Because an IPv6 enabled listener uses the GIVESOCKET API function to give an IPv6 socket to a child server transaction, you must enable that child server transaction program to use IPv6 sockets. This requires that all API functions that use a socket address structure be changed to use the larger IPv6 socket address structure. See Chapter 7, "C language application programming," on page 157 or Chapter 8, "Sockets extended API," on page 223 for more information.

  If the listener gives the accepted socket to the child server program, the child server program must be able to take that socket. If the listener is defined as an INET6 listener, the EBADF errno is issued if the child server's TAKESOCKET is AF_INET. If the listener is defined as an INET listener, the EBADF errno is issued if the child server's TAKESOCKET is AF_INET6.
- The Security/Transaction Exit program allows the user to examine and change certain pieces of data that are passed to the child server program by the listener.

  Table 3 illustrates the listener configuration in contrast with the connected client's address family and indicates the contents of the IPv4 and IPv6 IP address fields presented to the Security/Transaction Exit.

Table 3. Security/Transaction Exit program information fields

Listener's AF configuration	Connected Client's AF	Exit's Address Family	Exit's Client's IPv4 address	Exit's Client's IPv6 address	Exit's Listener's IPv4 address	Exit's Listener's IPv6 address
not specified	AF_INET	AF_INET	IPv4 addr	zeros	IPv4 addr	zeros
AF_INET	AF_INET	AF_INET	IPv4 addr	zeros	IPv4 addr	zeros
AF_INET6	AF_INET	AF_INET6	zeros	IPv4 mapped IPv6 addr	zeros	IPv4 mapped IPv6 addr

Table 3. Security/Transaction Exit program information fields (continued)

Listener's AF configuration	Connected Client's AF	Exit's Address Family	Exit's Client's IPv4 address	Exit's Client's IPv6 address	Exit's Listener's IPv4 address	Exit's Listener's IPv6 address
AF_INET6	AF_INET6	AF_INET6	zeros	IPv6 addr	zeros	IPv6 addr

Chapter 2. Setting up and configuring CICS TCP/IP

This topic describes the steps required to configure CICS TCP/IP.

It is assumed that both CICS and TCP/IP Services are already installed and operating on MVS.

Before you can start CICS TCP/IP, you need to do the following:

Task	See
Modify the CICS job stream to enable CICS TCP/IP startup.	"MVS JCL — Modifying CICS startup"
Define additional files, programs, maps, and transient data queues to CICS using resource definition online (RDO) and the CICS resource management utility DFHCSDUP commands.	"CICS — Defining CICS TCP/IP resources" on page 26
Modify TCP/IP Services data sets.	"TCP/IP services — Modifying data sets" on page 49
Use the configuration macro (EZACICD), to build the TCP Configuration data set.	"Building the configuration data set with EZACICD" on page 51
Use the configuration transaction (EZAC) to customize the Configuration data set.	"Customizing the configuration data set" on page 70
Note: You can modify the data set while CICS is running by using EZAC. See "Configuration transaction (EZAC)" on page 70.	

MVS JCL — Modifying CICS startup

Figure 8 on page 24 illustrates the modifications required in the CICS startup job stream to enable CICS TCP/IP startup. The numbers in the right margin of the JCL correspond to the modifications that follow.

```
//DFHSTART PROC START='AUTO',
// INDEX1='cicshlq', High-level qualifier(s) for CICS/TS run time libs.
// INDEX2='cicshlq', High-level qualifier(s) for CICS/TS load libraries
// SYSHLQ='systemhlq', High-level qualifier(s) for z/OS system datasets
// TCPHLQ='tcpiphlq',  High-level qualifier(s) for z/OS TCP/IP datasets
// USRHLQ='userhlq',   High-level qualifier(s) for user libraries
// REGNAM='1A',        Region name
// REG='0K',           Storage required
// DUMPTR='NO',        Dump/Trace analysis required, YES or NO
// RUNCICS='YES',      CICS startup required, YES or NO
// OUTC='*',           Print output class
// JVMMEMBR='DFHJVM',  JVM member
// SIP=P               Suffix of DFH$SIP member in the SYSIN dataset
//*************************************************************
//*******************  EXECUTE CICS  ************************
//*************************************************************
//CICS    EXEC PGM=DFHSIP,REGION=&REG,TIME=1440
// PARM='START=&START,SYSIN'
//*
//SYSIN    DD DISP=SHR,
// DSN=&INDEX1..SYSIN(DFH$SIP&SIP)  <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< 4 5
//*
//DFHCMACD DD DSN=&INDEX..DFHCMACD,DISP=SHR
//*************************************************************
//*        THE CICS STEPLIB CONCATENATION
//*        If Language Environment is required, the SCEERUN
//*        dataset is needed in STEPLIB or LNKLST
//*************************************************************
//STEPLIB  DD DSN=&INDEX2..SDFHAUTH,DISP=SHR
//         DD DSN=CEE.SCEERUN,DISP=SHR
//         DD DSN=&USRHLQ..LINKLIB,DISP=SHR
//*************************************************************
//*        THE CICS LIBRARY (DFHRPL) CONCATENATION
//*        If Language Environment is required, the SCEECICS
//*        and SCEERUN datasets are needed in DFHRPL.
//*        Refer to the Systems Definition Guide for
//*        information on how to run with the native
//*        runtime environments such as VS COBOL II.
//*************************************************************
//DFHRPL   DD DSN=&INDEX2..SDFHLOAD,DISP=SHR
//         DD DSN=CEE.SCEECICS,DISP=SHR
//         DD DSN=CEE.SCEERUN,DISP=SHR
//         DD DSN=&USRHLQ..CICS.LOAD,DISP=SHR
//         DD DSN=&TCPHLQ..SEZATCP,DISP=SHR  <<<<<<<<<<<<<<<<<<<<<<<<<< 1
//         DD DSN=&USRHLQ..CICS.TABLLIB,DISP=SHR
//*        THE AUXILIARY TEMPORARY STORAGE DATASET
//DFHTEMP  DD DISP=SHR,
// DSN=&INDEX1..CNTL.CICS &REGNAM..DFHTEMP
//*        THE INTRAPARTITION DATASET
//DFHINTRA DD DISP=SHR,
// DSN=&INDEX1..CNTL.CICS &REGNAM..DFHINTRA
//*        THE AUXILIARY TRACE DATASETS
//DFHAUXT  DD DISP=SHR,DCB=BUFNO=5,
// DSN=&INDEX1..CICS &REGNAM..DFHAUXT
//DFHBUXT  DD DISP=SHR,DCB=BUFNO=5,
// DSN=&INDEX1..CICS &REGNAM..DFHBUXT
//*        THE CICS LOCAL CATALOG DATASET
```

Figure 8. JCL for CICS startup with the TCP/IP socket interface (Part 1 of 2)

```
//DFHLCD   DD DISP=SHR,
// DSN=&INDEX1..CICS &REGNAM.DFHLCD
//*        THE CICS GLOBAL CATALOG DATASET
//DFHGCD   DD DISP=SHR,
// DSN=&INDEX1..CICS &REGNAM..DFHGCD
//*            AMP=('BUFND=5,BUFNI=20,BUFSP=122880')
//*        THE CICS LOCAL REQUEST QUEUE DATASET
//DFHLRQ   DD DISP=SHR,
// DSN=&INDEX1..CICS &REGNAM..DFHLRQ
//*        DATASETS FOR JVM SUPPORT
//DFHCJVM  DD DUMMY
//DFHJVM   DD DISP=SHR,
// DSN=&INDEX2..SDFHENV (&JVMMEMBR)
//*        EXTRAPARTITION DATASETS
//DFHCXRF  DD SYSOUT=&OUTC
//LOGUSR   DD SYSOUT=&OUTC,DCB=(DSORG=PS,RECFM=V,BLKSIZE=136)
//MSGUSR   DD SYSOUT=&OUTC,DCB=(DSORG=PS,RECFM=V,BLKSIZE=136)
//*        IP CICS SOCKET INTERFACE MSGS
//TCPDATA  DD SYSOUT=&OUTC,DCB=(DSORG=PS,RECFM=V,BLKSIZE=136) <<<<<<<  2
//*        RESOLVER TRACE
//SYSTCPT  DD DSN=&USRHLQ..RES.TRACE,DISP=SHR <<<<<<<<<<<<<<<<<<<<<<<  6
//*SYSTCPT  DD SYSOUT=&OUTC <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<  6
//SYSPRINT DD SYSOUT=&OUTC
//SYSTCPD  DD DSN=&SYSHLQ..TCPPARMS(TCPDATA),DISP=SHR <<<<<<<<<<<<<<<  3
//CEEMSG   DD SYSOUT=&OUTC <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<  7
//CEEOUT   DD SYSOUT=&OUTC <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<  7
//*        THE DUMP DATASETS
//DFHDMPA  DD DISP=SHR,
// DSN=&INDEX1..CICS &REGNAM..DFHDMPA
//DFHDMPB  DD DISP=SHR,
// DSN=&INDEX1..CICS &REGNAM..DFHDMPB
//PRINTER  DD SYSOUT=&OUTC,DCB=BLKSIZE=121
//*        THE CICS SYSTEM DEFINITION DATASET
//DFHCSD   DD DISP=SHR,
// DSN=&INDEX1..DFHCSD
```

Figure 8. JCL for CICS startup with the TCP/IP socket interface (Part 2 of 2)

The z/OS Communication Server TCP/IP data set prefix names might have been modified during installation. When you see the prefix *hlq* in this document, substitute the prefix used in your installation.

The following are the required modifications to the startup of CICS:

1. You must concatenate the data set SEZATCP to the DFHRPL DD. This data set contains all the other IP CICS TCP/IP modules.
2. Add a TCPDATA DD entry for the IP CICS sockets output messages (see “Transient data definition” on page 36).
3. The SYSTCPD DD explicitly identifies which data set is to be used to obtain the parameters defined by TCPIP.DATA. This is used to select the stack you want to use if there are multiple TCP/IP stacks on this system. See *z/OS Communications Server: IP Configuration Guide* for further information.
4. The CICS System Initialization Table (SIT) override might contain the following. See the *CICS System Definition Guide*, in the CICS system initialization secttion for more information on setting CICS SIT parameters:
   - `GMTEXT= WELCOME TO CICS/TS WITH z/OS CS TCP/IP SOCKETS INTERFACE`
   - `MCT=SO`

     If you want IP CICS sockets to provide performance data then include the IP CICS Sockets Monitor Control Table (MCT) entries in your MCT along with any appropriate monitor SIT controls.

- `PLTPI=SI`

  If you want IP CICS sockets to start at Program Load Table (PLT) phase 2 then include EZACIC20 in an appropriate startup PLT.
- `PLTSD=SD`

  If you want IP CICS sockets to shutdown at PLT phase 1, then include EZACIC20 in an appropriate shutdown PLT.
- `PLTPIUSR=PLTUSER`

  PLT User ID. Specify the appropriate user ID to start the IP CICS socket interface and listeners.

5. The following CICS SIT parameters affect the IP CICS socket interface when it is configured to use the CICS Open Transaction Environment. CICS/TS V2R2 or later is required for this support.
   - `MAXOPENTCBS=50`

     When specifying the EZACICD TYPE=CICS,OTE=YES configuration option, carefully consider this value; it is the size of the CICS managed open API, L8, TCB pool. This pool is used by the IP CICS socket interface and other open API-enabled task-related user exits such as DB2®. Use the CEMT SET DISPATCHER command to dynamically alter this value.
   - `FORCEQR`

     User programs that are defined to CICS as THREADSAFE are executed on the quasi-reentrant TCB. Use the CEMT SET SYSTEM command to dynamically alter this value.
6. Write the Resolver trace to either a dataset or JES spool.
7. The information is used by IP CICS C Sockets API programs for user messages.

CICS — Defining CICS TCP/IP resources

The following CICS definitions must be made:
- Transactions
- Programs (see “Program definitions” on page 28)
- Basic Mapping Support (BMS) mapset (EZACICM, shown in Figure 23 on page 30)
- Files (see “File definitions” on page 34)
- Transient data queues (see “Transient data definition” on page 36)

To ensure that the CICS system definition (CSD) file contains all necessary socket-related resource definitions, you should execute a CSD upgrade (DFHCSDUP) using member EZACICCT in SEZAINST. See *CICS Resource Definition Guide* for information about DFHCSDUP.

Note: For the enhanced listener, more temporary storage is needed to support passing a larger amount of data to the security/transaction exit and to the child server. Depending upon the size of the data defined in the listener configuration, temporary storage should be adjusted accordingly.

Transaction definitions

Figures 9, 10, 11, and 12 show the CICS CSD update (DFHCSDUP) commands to define the four transactions. These commands can be found in *hlq*.SEZAINST(EZACICCT).

EZAC Configure the socket interface

EZAO Enable the socket interface

EZAP Internal transaction that is invoked during termination of the socket interface

CSKL Listener task

Note: This is a single listener. Each listener in the same CICS region needs a unique transaction ID.

Note: In the following definitions we have suggested priority of 255. This ensures timely transaction dispatching, and (in the case of CSKL) maximizes the connection rate of clients requesting service.

Using storage protection

When running with CICS 3.3.0 or higher on a storage-protection-enabled machine, the EZAP, EZAO, and CSKL transactions must be defined with TASKDATAKEY(CICS). If this is not done, EZAO fails with an ASRA abend code indicating an incorrect attempt to overwrite the CDSA by EZACIC01. The *CICS Customization Guide* contains more information on storage protection with task-related user exits (TRUEs).

In Figure 10 on page 27, Figure 11 on page 27, and Figure 12 on page 27 note that, if the machine does not support storage protection or is not enabled for storage protection, TASKDATAKEY(CICS) is ignored and does not cause an error.

```
DEFINE TRANSACTION(EZAC)
DESCRIPTION(CONFIGURE SOCKETS INTERFACE)
GROUP(SOCKETS)
PROGRAM(EZACIC23)
TASKDATALOC(ANY) TASKDATAKEY(USER)
```

Figure 9. EZAC, transaction to configure the socket interface

```
DEFINE TRANSACTION(EZAO)
DESCRIPTION(ENABLE SOCKETS INTERFACE)
GROUP(SOCKETS)
PROGRAM(EZACIC00) PRIORITY(255)
TASKDATALOC(ANY) TASKDATAKEY(CICS)
```

Figure 10. EZAO, transaction to enable the socket interface

```
DEFINE TRANSACTION(EZAP)
DESCRIPTION(DISABLE SOCKETS INTERFACE)
GROUP(SOCKETS)
PROGRAM(EZACIC22) PRIORITY(255)
TASKDATALOC(ANY) TASKDATAKEY(CICS)
```

Figure 11. EZAP, transaction to disable the socket interface

```
DEFINE TRANSACTION(CSKL)
DESCRIPTION(LISTENER TASK)
GROUP(SOCKETS)
PROGRAM(EZACIC02) PRIORITY(255)
TASKDATALOC(ANY) TASKDATAKEY(CICS)
```

Figure 12. CSKL, Listener task transaction

Notes:

1. Use of the IBM-supplied listener is not required.
2. You can use a transaction name other than CSKL.
3. The TASKDATALOC values for EZAO and EZAP and the TASKDATALOC value for CSKL must all be the same.

4. The user ID invoking the EZAO transaction to activate or deactivate the IP CICS socket interface requires the UPDATE access to the EXITPROGRAM resource when CICS command security is active. The user ID invoking the EZAC transaction requires the UPDATE access to the EXITPROGRAM resource to allow the EZAC transaction to perform an IPv6 run-time check when the AF is changed to INET6. Failure to have at least the UPDATE access to the EXITPROGRAM resource causes the IP CICS socket interface and listener to not start or not stop.

Program definitions

Three categories of program are or could be required to support CICS TCP/IP:
- Required programs, CICS definition needed
- Optional programs, CICS definition needed
- Required programs, CICS definition not needed

Required programs, CICS definition needed

You need to define the following 11 programs and 1 mapset to run CICS TCP/IP, or to provide supporting functions:

EZACICM
Has all the maps used by the transactions that enable and disable CICS TCP/IP.

EZACICME
The U.S. English text delivery module.

EZACIC00
The connection manager program. It provides the enabling and disabling of CICS TCP/IP through the transactions EZAO and EZAP.

EZACIC01
The task related user exit (TRUE).

EZACIC02
The listener program that is used by the transaction CSKL. This transaction is started when you enable CICS TCP/IP through the EZAO transaction.

Note: While you do not need to use the IBM-supplied listener, you do need to provide a listener function.

EZACIC12
The module that performs WLM registration and deregistration functions for CICS sockets.

EZACIC20
The initialization and termination front-end module for CICS sockets.

EZACIC21
The initialization module for CICS sockets.

EZACIC22
The termination module for CICS sockets.

EZACIC23
The primary module for the configuration transaction (EZAC).

EZACIC24
The message delivery module for transactions EZAC and EZAO.

EZACIC25
The Domain Name Server (DNS) cache module.

The following figures show sample RDO definitions of these programs.

Using storage protection: When running with CICS 3.3.0 or higher on a storage-protection-enabled machine, all the required CICS TCP/IP programs (EZACIC00, EZACIC01, and EZACIC02) must have EXECKEY(CICS) as part of their definitions. The *CICS Customization Guide* contains more information on storage protection with TRUEs.

Figures 13, 14, and 15 show EZACIC00, EZACIC01, and EZACIC02 defined with EXECKEY(CICS). Note that, if the machine does not support storage protection or is not enabled for storage protection, EXECKEY(CICS) is ignored and does not cause an error.

```
DEFINE PROGRAM(EZACIC00)
DESCRIPTION(PRIMARY PROGRAM FOR TRANSACTION EZAO)
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(CICS)
RELOAD(NO) RESIDENT(NO) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(TRANSIENT)
```

Figure 13. EZACIC00, connection manager program

```
DEFINE PROGRAM(EZACIC01)
DESCRIPTION(TASK RELATED USER EXIT <TRUE> )
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(CICS)
RELOAD(NO) RESIDENT(YES) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(NORMAL)
```

Figure 14. EZACIC01, task related user exit program

```
DEFINE PROGRAM(EZACIC02)
DESCRIPTION(IBM LISTENER)
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(CICS)
CONCURRENCY(THREADSAFE)
```

Figure 15. EZACIC02, listener program

```
DEFINE PROGRAM(EZACIC12)
DESCRIPTION(WORK LOAD MANGER REGISTRATION / DEREGISTRATION)
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(CICS)
RELOAD(NO) RESIDENT(NO) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(TRANSIENT)
CONCURRENCY(THREADSAFE)
```

Figure 16. EZACIC12, WLM registration and deregistration module for CICS sockets

```
DEFINE PROGRAM(EZACIC20)
DESCRIPTION(INITIALIZATION/TERMINATION FOR CICS SOCKETS)
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(CICS)
RELOAD(NO) RESIDENT(NO) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(TRANSIENT)
```

Figure 17. EZACIC20, front-end module for CICS sockets

```
DEFINE PROGRAM(EZACIC21)
DESCRIPTION(INITIALIZATION MODULE FOR CICS SOCKETS)
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(CICS)
RELOAD(NO) RESIDENT(YES) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(TRANSIENT)
```

Figure 18. EZACIC21, initialization module for CICS sockets

```
DEFINE PROGRAM(EZACIC22)
DESCRIPTION(TERMINATION MODULE FOR CICS SOCKETS)
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(CICS)
RELOAD(NO) RESIDENT(NO) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(TRANSIENT)
```

Figure 19. EZACIC22, termination module for CICS sockets

```
DEFINE PROGRAM(EZACIC23)
DESCRIPTION(PRIMARY MODULE FOR TRANSACTION EZAC)
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(USER)
RELOAD(NO) RESIDENT(NO) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(TRANSIENT)
```

Figure 20. EZACIC23, primary module for transaction EZAC

```
DEFINE PROGRAM(EZACIC24)
DESCRIPTION(MESSAGE DELIVERY MODULE FOR CICS SOCKETS)
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(CICS)
RELOAD(NO) RESIDENT(NO) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(TRANSIENT)
```

Figure 21. EZACIC24, message delivery module for CICS sockets

```
DEFINE PROGRAM(EZACIC25)
DESCRIPTION(CACHE MODULE FOR THE DOMAIN NAME SERVER)
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(USER)
RELOAD(NO) RESIDENT(YES) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(NORMAL)
```

Figure 22. EZACIC25, domain name server cache module

```
DEFINE MAPSET(EZACICM)
DESCRIPTION(MAPSET FOR CICS SOCKETS INTERFACE)
GROUP(SOCKETS)
RESIDENT(NO) USAGE(TRANSIENT) USELPACOPY(NO)
STATUS(ENABLED)
```

Figure 23. EZACICM, maps used by the EZAO transaction

```
DEFINE PROGRAM(EZACICME)
DESCRIPTION(US ENGLISH TEXT DELIVERY MODULE)
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(CICS)
RELOAD(NO) RESIDENT(YES) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(NORMAL)
CONCURRENCY(THREADSAFE)
```

Figure 24. EZACICME, U.S. English text delivery module

Optional programs, CICS transaction and program definition needed

The following six programs are optional. They are the supplied samples. They are also in SEZAINST:

EZACICSC
: A sample IPv4 child server that works with the IPv4 listener (EZACIC02). See "EZACICSC" on page 463.

EZACICSS
: A sample IPv4 iterative server. EZACICSS establishes the connection between CICS and TCP/IP stacks, and receives client requests from workstations. See "EZACICSS" on page 472.

EZACIC6C
: A sample IPv6 child server that works with either a standard or enhanced IPv6 listener (EZACIC02). See "EZACIC6C" on page 493.

EZACIC6S
: A sample IPv6 iterative server. EZACIC6S establishes the connection between CICS and TCP/IP stacks, and receives client requests from workstations. See "EZACIC6S" on page 505.

EZACICAC
: A sample assembler child server that works with either a standard or enhanced, IPv4 or IPv6 listener (EZACIC02). See "EZACICAC" on page 529.

EZACICAS
: A sample assembler iterative server that establishes the connection between CICS and TCP/IP stacks, and accepts either ASCII or EBCDIC, IPv4 or IPv6 (if IPv6 is enabled on the system) client connection requests. See "EZACICAS" on page 540.

If these sample programs are used, they require DFHCSDUP definitions as shown in Figure 26 on page 32, Figure 25, Figure 28 on page 32, Figure 27 on page 32, Figure 29 on page 32, and Figure 30 on page 33.

```
DEFINE TRANSACTION(SRV1)
DESCRIPTION(SAMPLE STARTED SERVER)
GROUP(SOCKETS)
PROGRAM(EZACICSC)
TASKDATALOC(ANY) TASKDATAKEY(USER)

DEFINE PROGRAM(EZACICSC)
DESCRIPTION(SAMPLE STARTED SERVER)
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(USER)
RELOAD(NO) RESIDENT(NO) USELPACOPY(NO)
LANGUAGE(COBOL) STATUS(ENABLED) USAGE(NORMAL)
CONCURRENCY(THREADSAFE)
```

Figure 25. EZACICSC, sample IPv4 child server transaction and program definitions

```
DEFINE TRANSACTION(SRV2)
DESCRIPTION(SAMPLE SERVER)
GROUP(SOCKETS)
PROGRAM(EZACICSS)
TASKDATALOC(ANY) TASKDATAKEY(USER)

DEFINE PROGRAM(EZACICSS)
DESCRIPTION(SAMPLE SERVER FOR TRANSACTION SRV2 )
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(USER)
RELOAD(NO) RESIDENT(NO) USELPACOPY(NO)
LANGUAGE(COBOL) STATUS(ENABLED) USAGE(NORMAL)
```

Figure 26. EZACICSS, sample iterative IPv4 server transaction and program definitions

```
DEFINE TRANSACTION(SRV3)
DESCRIPTION(SAMPLE IPV6 CHILD SERVER)
GROUP(SOCKETS)
PROGRAM(EZACIC6C)
TASKDATALOC(ANY) TASKDATAKEY(USER)

DEFINE PROGRAM(EZACIC6C)
DESCRIPTION(SAMPLE IPV6 CHILD SERVER)
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(USER)
RELOAD(NO) RESIDENT(NO) USELPACOPY(NO)
LANGUAGE(COBOL) STATUS(ENABLED) USAGE(NORMAL)
CONCURRENCY(THREADSAFE)
```

Figure 27. EZACIC6C, sample IPv6 child server transaction and program definitions

```
DEFINE TRANSACTION(SRV4)
DESCRIPTION(SAMPLE IPV6 SERVER)
GROUP(SOCKETS)
PROGRAM(EZACIC6S)
TASKDATALOC(ANY) TASKDATAKEY(USER)

DEFINE PROGRAM(EZACIC6S)
DESCRIPTION(SAMPLE IPV6 SERVER FOR TRANSACTION SRV4)
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(USER)
RELOAD(NO) RESIDENT(NO) USELPACOPY(NO)
LANGUAGE(COBOL) STATUS(ENABLED) USAGE(NORMAL)
```

Figure 28. EZACIC6S, sample iterative IPv6 server transaction and program definitions

```
DEFINE TRANSACTION(SRV5)
DESCRIPTION(SAMPLE ASSEMBLER CHILD SERVER)
GROUP(SOCKETS)
PROGRAM(EZACICAC)
TASKDATALOC(ANY) TASKDATAKEY(USER)

DEFINE PROGRAM(EZACICAC)
DESCRIPTION(SAMPLE ASSEMBLER CHILD SERVER)
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(USER)
RELOAD(NO) RESIDENT(NO) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(NORMAL)
CONCURRENCY(THREADSAFE)
```

Figure 29. EZACICAC, sample assembler child server transaction and program definitions

```
DEFINE TRANSACTION(SRV6)
DESCRIPTION(SAMPLE ASSEMBLER SERVER)
GROUP(SOCKETS)
PROGRAM(EZACICAS)
TASKDATALOC(ANY) TASKDATAKEY(USER)

DEFINE PROGRAM(EZACICAS)
DESCRIPTION(SAMPLE ASSEMBLER SERVER FOR TRANSACTION SRV6 )
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(USER)
RELOAD(NO) RESIDENT(NO) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(NORMAL)
```

Figure 30. EZACICAS, sample assembler server transaction and program definitions

Required programs, CICS definition not needed

The following programs do not need to be defined to CICS.

EZACICAL
: The application stub that invokes the TRUE and passes on the CICS application's socket call. This program is in SEZATCP.

EZACIC03
: The MVS subtask that passes data between the CICS socket task and the transport interface into TCP/IP for MVS. This program is in SEZALOAD.

  Note: If the SEZALOAD load library is included in the LINKLIST, then it does not need to be in the STEPLIB concatenation.

EZACIC07
: The application stub that handles the C API for non-reentrant programs. This program is in SEZATCP.

EZACIC17
: The application stub that handles the C API for reentrant programs. This program is in SEZATCP.

Threadsafe enablement

The following programs can be defined to CICS as threadsafe. This is particulary important when the IP CICS socket interface is using the CICS Open Transaction Environment. See "TYPE parameter" on page 54 for more information on configuring the IP CICS socket interface to use CICS Open Transaction Environment.

EZACIC02
: Enables the listener to initially execute on an open API TCB. Some TCB switching still occurs because CICS commands that are not threadsafe are used.

EZACIC12
: Enables the Work Load Manager registration/deregistration program to initially execute on an open API TCB. Some TCB switching still occurs because CICS commands that are not threadsafe are used. The listener links to this program only when WLMGN1, WLMGN2, or WLMGN3 is specified.

EZACICME
: Enables the message module to initially execute on an open API TCB. Some TCB switching still occurs because CICS commands that are not threadsafe are used.

Sample programs: EZACICSC, EZACIC6C, EZACICAC

These sample child servers contain logic to determine when the IP CICS socket interface is threadsafe, and executes the interface accordingly.

Use the DFHCSDUP commands in SEZAINST(EZACICPT) to change the CICS CONCURRENCY setting for these program definitions on a CICS/TS V2R2 or later system. EZACICPT was originally a duplicate of EZACICCT. It is being reused to contain the ALTER PROGRAM commands.

```
ALTER PROGRAM(EZACIC02)
       DESCRIPTION(IBM LISTENER THREADSAFE)
       GROUP(SOCKETS)
       CONCURRENCY(THREADSAFE)
ALTER PROGRAM(EZACIC12)
       DESCRIPTION(WORKLOAD MGR REGISTRATION / DEREGISTRATION THREADSAFE)
       GROUP(SOCKETS)
       CONCURRENCY(THREADSAFE)
ALTER PROGRAM(EZACICME)
       DESCRIPTION(US ENGLISH TEXT DELIVERY MODULE THREADSAFE)
       GROUP(SOCKETS)
       CONCURRENCY(THREADSAFE)
ALTER PROGRAM(EZACICSC)
       DESCRIPTION(SAMPLE IPV4 CHILD SERVER THREADSAFE)
       GROUP(SOCKETS)
       CONCURRENCY(THREADSAFE)
ALTER PROGRAM(EZACIC6C)
       DESCRIPTION(SAMPLE IPV6 CHILD SERVER THREADSAFE)
       GROUP(SOCKETS)
       CONCURRENCY(THREADSAFE)
ALTER PROGRAM(EZACICAC)
       DESCRIPTION(SAMPLE ASSEMBLER CHILD SERVER THREADSAFE)
       GROUP(SOCKETS)
       CONCURRENCY(THREADSAFE)
```

Figure 31. ALTER PROGRAM instructions

Use the CEDA INSTALL command to install the new PROGRAM definitions in your CICS system. When you put a new version of the program in your library, you do not need to install the definition again, unless attributes specified on the definition have changed. To make the new version available, use the CEMT transaction:

```
CEMT SET PROGRAM(pgmid) NEWCOPY
```

File definitions

The updates to CICS TCP/IP include two files: EZACONFG, the sockets configuration file, and EZACACHE, which is required if you want to use the Domain Name Server Cache function (EZACIC25).

EZACONFG

Use the following DFHCSDUP commands to define EZACONFG file. The numbers correspond to the notes that follow.

```
DEFINE FILE(EZACONFG)
DESCRIPTION(CICS SOCKETS CONFIGURATION FILE)
GROUP(SOCKETS)
DSNAME(EZACONFG)  1  LSRPOOLID(1) DSNSHARING(ALLREQS)
STRINGS(01)

REMOTESYSTEM(....) REMOTENAME(........)
RECORDSIZE(....) KEYLENGTH(...)  2

OPENTIME(STARTUP)  4 STATUS(ENABLED)
DISPOSITION(SHARE) TABLE(NO) RECORDFORMAT(V)
READ(YES) BROWSE(YES) ADD(NO)
DELETE(NO) UPDATE(NO)  3
DATABUFFERS(2) INDEXBUFFERS(1) JNLSYNCWRITE(NO)
```

Figure 32. DFHCSDUP commands to define EZACONFG

Notes:

1. Choose a DSName to fit installation standards.
2. If you want to have EZACONFG reside in a file owning region (FOR) and be accessed indirectly from an application owning region (AOR), the systems programmer must assure that no CICS socket modules can execute directly in the FOR. That is, do not install any CICS TCP/IP resources other than EZACONFG in the FOR. Otherwise, EZACONFG can become disabled and is not accessible from the AOR
3. If you want to have the EZAC transaction residing in an AOR and indirectly accessing EZACONFG in the FOR, the ADD, DELETE, and UPDATE parameters in the FOR's file definition must be set to YES. The FOR therefore is the only CICS region that can open EZACONFG. Thus, no sharing of EZACONFG between different CICS regions is possible.
4. Specify OPENTIME(FIRSTREF) to reduce the overhead that is incurred when CICS opens non-essential datasets during CICS startup.

EZACACHE

If you want to use the Domain Name Server Cache function (EZACIC25), this definition is required.

Recommendations: The following recommendations apply when defining EZACACHE:

- If you require improved performance for Domain Name Server lookups for both IPv4 and IPv6 resources, you should consider configuring a caching-only BIND 9 name server on the local system. Doing this has the following benefits:
  - After a hostname is resolved, it is cached locally, allowing all other applications running in the system to retrieve this information without incurring the overhead of network communications.
  - A caching domain name server honors the time to live (TTL) value that indicates when a resource record's information should expire.
  - BIND 9 supports caching of both IPv4 and IPv6 resources.
  - IBM recommends that a caching-only BIND 9 name server be used to support both IPv4 and IPv6 names.
- Do not attempt to share a cache file.
- If the server intends to use WLM connection balancing, it is recommended that the client does not cache DNS names. Connection balancing relies on up-to-date information about current capacity of hosts in the sysplex. If DNS names are retrieved from a cache instead of the DNS/WLM name server, connections are made without regard for current host capacity, degrading the effectiveness of

connection balancing. Of course, not caching names can mean more IP traffic, which in some cases can outweigh the benefits of connection balancing.

See *z/OS Communications Server: IP Configuration Reference* for information about caching issues.

- DNS/WLM continues to support CICS listeners wanting to participate in work load balancing for IPv4 clients. IPv6 enabled listeners are still able to participate in work load balancing for their IPv4 clients and IPv6 clients. DNS/WLM is not possible when using IPv6 addresses because DNS/WLM is only supported on the BIND 4.9.3 server. Use a DNS/BIND 9 caching-only server to support IPv6 addresses.

  If you want to support IPv6 clients and DNS/WLM (for IPv4 clients), set up a caching-only BIND 9 name server to support both IPv4 and IPv6 addresses and keep your BIND 4.9.3 name server in the sysplex for DNS/WLM support. Have your IPv6-enabled client get the IPv4 address for the participating listener from the DNS/WLM server and then convert the returned IPv4 address to an IPv4-mapped IPv6 address. Use this address to connect to the IPv6-enabled listener. Note that this is not a true IPv6 connection as DNS/WLM because does not give an IPv6 address. Clients that want to connect to the server over an IPv6 network should use an IPv6 address.

Use the following DFHCSDUP commands to define EZACACHE file:

```
DEFINE FILE(EZACACHE)
DESCRIPTION(DOMAIN NAME SERVER CACHE CONFIGURATION FILE)
GROUP(SOCKETS)
DSNAME(EZACACHE) 1 LSRPOOLID(1) DSNSHARING(ALLREQS)
STRINGS(20) 2 OPENTIME(STARTUP) STATUS(ENABLED)
DISPOSITION(OLD) TABLE(USER) RECORDFORMAT(V)
READ(YES) BROWSE(YES) ADD(YES)
DELETE(YES) UPDATE(YES) MAXNUMRECS(4000)
DATABUFFERS(060) 3 INDEXBUFFERS(2000) 4 JNLSYNCWRITE(NO)
TABLE(USER) 5 MAXNUMRECS(4000) 6
```

Figure 33. DFHCSDUP commands to define EZACACHE

Notes:

1. Choose a DSName to fit installation standards.
2. For strings, specify the maximum number of concurrent users.
3. Databuffers should equal strings multiplied by two.
4. Indexbuffers equals the number of records in the index set.
5. Although it is optional, it is recommended that you specify TABLE(USER) because it makes the process run faster. For more information on data tables, see *CICS Resource Definition Guide*.
6. Maxnumrecs equals the maximum number of destinations queried.

Transient data definition

Figure 34 on page 37 shows the DFHCSDUP commands required to define the TCPM transient data queue for CICS TCP/IP. For more information about DFHCSDUP commands, see *CICS Resource Definition Guide*.

The destination TCPM can be changed. If it is changed, it must match the name specified in the ERRORTD parameter of the EZAC DEFINE CICS, the EZACICD TYPE=CICS, or both (see “Building the configuration data set with EZACICD” on page 51).

```
DEFINE TDQUEUE(TCPM) GROUP(SOCKETS)
DESCRIPTION(USED FOR SOCKETS MESSAGES)
TYPE(EXTRA)
DATABUFFERS(1)
DDNAME(TCPDATA)
ERROROPTION(IGNORE)
OPENTIME(INITIAL)
TYPEFILE(OUTPUT)
RECORDSIZE(132)
RECORDFORMAT(VARIABLE)
BLOCKFORMAT(UNBLOCKED)
DISPOSITION(SHR)

DEFINE TDQUEUE(TRAA) GROUP(SOCKETS)
DESCRIPTION(USED FOR SOCKETS APPLICATION)
TYPE(INTRA)
ATIFACILITY(FILE)
TRIGGERLEVEL(1)
TRANSID(TRAA)
```

Figure 34. CICS TCP/IP Transient Data Queue definitions

The listener writes to the TCPM queue while CICS TCP/IP is enabled. In addition to this, your own sockets applications can write to this queue using EXEC CICS WRITEQ TD commands. It is recommended that an extrapartition transient data queue be defined, as shown in Figure 34.

The CICS startup JCL must include a DD statement for the extrapartition transient data queue being defined (as in Figure 8 on page 24, line **3**).

The listener transaction can start a server using a transient data queue, as described in “Listener input format” on page 135. The intrapartition transient data queue definition in Figure 34 shows an entry for an application that is started using the trigger-level mechanism of destination control.

CICS monitoring

The CICS Sockets Feature uses the CICS Monitoring Facility to collect data about its operation. There are two collection points: the Task Related User Exit (TRUE) and the listener. This data is collected as Performance Class Data. The TRUE uses Event Monitoring Points (EMPs) with the identifier EZA01 and the listener uses Event Monitoring Points (EMPs) with the identifier EZA02. If the Monitor Control Table entries are not defined, the following records are written to the CICS internal trace when CICS Socket calls are made:

```
*EXC* - Invalid monitoring point
```

When this occurs, the TRUE mechanism and the listener disable use of this specific EMP and no further data is written to SMF. An EMP is dependent on its associated entry in the MCT, so when an EMP is disabled it must be re-enabled as follows:

1. By adding entries to the Monitor Control table
2. Restarting CICS
3. Starting IP CICS socket interface and listener

You can tailor your MCT to only monitor events required by your installation. This can be done by only supplying the MCT entries you require as the TRUE and the listener disables those not coded and continue to execute EMPs matching the entries in the MCT.

See the *CICS Performance Guide* for more information on the CICS monitoring facility.

Event monitoring points for the TRUE

The TRUE monitors call activity plus use of reusable, attached or OTE tasks. The call activity is monitored by the following classes of calls:

- Initialization (INITAPI or other first call)
- Read (inbound data transfer) calls
- Write (outbound data transfer) calls
- Select calls
- All other calls

There are counters and clocks for each of these classes. In addition, there are counters for use of reusable tasks, attached tasks and the use of open API tasks.

- Counter/Clock 1 - Initialization Call
- Counter/Clock 2 - Read Call
- Counter/Clock 3 - Write Call
- Counter/Clock 4 - Select Call
- Counter/Clock 5 - Other Call
- Counter 6 - Use of a reusable task
- Counter 7 - Use of an attached task
- Counter 8 - Use of an open API, L8, TCB
- Counter 9 - Number of times at TCBLIM

The following Monitor Control Table (MCT) entries use the event monitoring points in the performance class used by the Task Related User Exit (TRUE). These entries are in *hlq*.SEZAINST(EZACIMCT).

```
         DFHMCT TYPE=INITIAL,SUFFIX=SO
*
* ENTRIES FOR IP CICS SOCKETS TASK-RELATED USER EXIT
*
         DFHMCT TYPE=EMP,ID=(EZA01.01),CLASS=PERFORM,                  X
               PERFORM=SCLOCK(1),                                      X
               CLOCK=(1,INIT,READ,WRITE,SELECT,OTHER)
         DFHMCT TYPE=EMP,ID=(EZA01.02),CLASS=PERFORM,                  X
               PERFORM=PCLOCK(1)
*
* SOCKET FUNCTIONS READING DATA
*
         DFHMCT TYPE=EMP,ID=(EZA01.03),CLASS=PERFORM,                  X
               PERFORM=SCLOCK(2)
         DFHMCT TYPE=EMP,ID=(EZA01.04),CLASS=PERFORM,                  X
               PERFORM=PCLOCK(2)
*
* SOCKET FUNCTIONS WRITING DATA
*
         DFHMCT TYPE=EMP,ID=(EZA01.05),CLASS=PERFORM,                  X
               PERFORM=SCLOCK(3)
         DFHMCT TYPE=EMP,ID=(EZA01.06),CLASS=PERFORM,                  X
               PERFORM=PCLOCK(3)
*
* SOCKET FUNCTIONS SELECTING SOCKETS
*
         DFHMCT TYPE=EMP,ID=(EZA01.07),CLASS=PERFORM,                  X
               PERFORM=SCLOCK(4)
         DFHMCT TYPE=EMP,ID=(EZA01.08),CLASS=PERFORM,                  X
               PERFORM=PCLOCK(4)
*
* OTHER SOCKET FUNCTIONS
*
         DFHMCT TYPE=EMP,ID=(EZA01.09),CLASS=PERFORM,                  X
               PERFORM=SCLOCK(5)
         DFHMCT TYPE=EMP,ID=(EZA01.10),CLASS=PERFORM,                  X
               PERFORM=PCLOCK(5)
*
* CICS TASK TERMINATION
*
         DFHMCT TYPE=EMP,ID=(EZA01.13),CLASS=PERFORM,                  X
               PERFORM=(MLTCNT(1,5)),                                  X
               COUNT=(1,TINIT,TREAD,TWRITE,TSELECT,TOTHER)
*
* REUSABLE SUBTASK POOL
*
         DFHMCT TYPE=EMP,ID=(EZA01.11),CLASS=PERFORM,                  X
               PERFORM=ADDCNT(6,1),                                    X
               COUNT=(6,REUSABLE,ATTACHED,OPENAPI,TCBLIM)
*
* DYNAMICALLY DEFINED SUBTASKS
```

Figure 35. The Monitor Control Table (MCT) for TRUE (Part 1 of 2)

```
*
         DFHMCT TYPE=EMP,ID=(EZA01.12),CLASS=PERFORM,                  X
               PERFORM=ADDCNT(7,1)
*
* OPEN API
*
         DFHMCT TYPE=EMP,ID=(EZA01.15),CLASS=PERFORM,                  X
               PERFORM=ADDCNT(8,1)
*
* TCBLIM
*
         DFHMCT TYPE=EMP,ID=(EZA01.16),CLASS=PERFORM,                  X
               PERFORM=ADDCNT(9,1)
*
* CICS TASK INTERFACE TERMINATION
*
         DFHMCT TYPE=EMP,ID=(EZA01.14),CLASS=PERFORM,                  X
               PERFORM=(MLTCNT(10,4)),                                 X
               COUNT=(10,TREUSABL,TATTACHE,TOPENAPI,TTCBLIM)
```

Figure 35. The Monitor Control Table (MCT) for TRUE (Part 2 of 2)

In the ID parameter, the following specifications are used:

(EZA01.01)
Start of Initialization Call

(EZA01.02)
End of Initialization Call

(EZA01.03)
Start of Read Call

(EZA01.04)
End of Read Call

(EZA01.05)
Start of Write Call

(EZA01.06)
End of Write Call

(EZA01.07)
Start of Select Call

(EZA01.08)
End of Select Call

(EZA01.09)
Start of Other Call

(EZA01.10)
End of Other Call

(EZA01.11)
First call to Interface Using Reusable Task

(EZA01.12)
First call to Interface Using Attached Task

(EZA01.13)
CICS Task Termination

(EZA01.14)
CICS socket interface Termination

(EZA01.15)
First call to Interface Using an open API TCB

(EZA01.16)
Number of times at TCBLIM

Event monitoring points for the listener

The listener monitors the activities associated with connection acceptance and server task startup. Since it uses the TRUE, the data collected by the TRUE can be used to evaluate listener performance.

The listener counts the following events:

- Number of Connection Requested Accepted
- Number of Transactions Started
- Number of Transactions Rejected Due To Invalid Transaction ID
- Number of Transactions Rejected Due To Disabled Transaction
- Number of Transactions Rejected Due To Disabled Program
- Number of Transactions Rejected Due To Givesocket Failure
- Number of Transactions Rejected Due To Negative Response from Security Exit
- Number of Transactions Not Authorized to Run
- Number of Transactions Rejected Due to I/O Error
- Number of Transactions Rejected Due to No Space
- Number of Transactions Rejected Due to TD Length Error

The following Monitor Control Table (MCT) entries use the event-monitoring points in the performance class used by the listener. These entries can be found in *hlq*.SEZAINST(EZACIMCL).

```
* ENTRIES FOR IP CICS SOCKETS LISTENER
*
*
* NUMBER OF TIMES ACCEPT COMPLETED SUCCESSFULLY
*
         DFHMCT TYPE=EMP,ID=(EZA02.01),CLASS=PERFORM,                  X
               PERFORM=ADDCNT(1,1),COUNT=(1,CONN)
*
* NUMBER OF CHILD SERVER TASKS STARTED
*
         DFHMCT TYPE=EMP,ID=(EZA02.02),CLASS=PERFORM,                  X
               PERFORM=ADDCNT(2,1),COUNT=(2,STARTED)
*
* NUMBER OF REQUESTS FOR UNDEFINED CHILD SERVER TRANSACTIONS
*
         DFHMCT TYPE=EMP,ID=(EZA02.03),CLASS=PERFORM,                  X
               PERFORM=ADDCNT(3,1),COUNT=(3,INVALID)
*
* NUMBER OF REQUESTS FOR DISABLED CHILD SERVER TRANSACTIONS
*
         DFHMCT TYPE=EMP,ID=(EZA02.04),CLASS=PERFORM,                  X
               PERFORM=ADDCNT(4,1),COUNT=(4,DISTRAN)
*
* NUMBER OF REQUESTS FOR DISABLED CHILD SERVER PROGRAMS
*
         DFHMCT TYPE=EMP,ID=(EZA02.05),CLASS=PERFORM,                  X
               PERFORM=ADDCNT(5,1),COUNT=(5,DISPROG)
*
* NUMBER OF GIVESOCKET FAILURES
*
         DFHMCT TYPE=EMP,ID=(EZA02.06),CLASS=PERFORM,                  X
               PERFORM=ADDCNT(6,1),COUNT=(6,GIVESOKT)
*
* NUMBER OF TRMS REJECTED BY THE SECURITY/USER EXIT
*
         DFHMCT TYPE=EMP,ID=(EZA02.07),CLASS=PERFORM,                  X
               PERFORM=ADDCNT(7,1),COUNT=(7,SECEXIT)
*
* NUMBER OF TIME CHILD SERVER TRANSACTION NOT AUTHORIZED
*
         DFHMCT TYPE=EMP,ID=(EZA02.08),CLASS=PERFORM,                  X
               PERFORM=ADDCNT(8,1),COUNT=(8,NOTAUTH)
*
* NUMBER OF TRMS TD QUEUE I/O ERROR
*
         DFHMCT TYPE=EMP,ID=(EZA02.09),CLASS=PERFORM,                  X
               PERFORM=ADDCNT(9,1),COUNT=(9,IOERR)
*
* NUMBER OF TIMES NO SPACE ON CHILD SERVER TD QUEUE
*
          DFHMCT TYPE=EMP,ID=(EZA02.10),CLASS=PERFORM,                 X
               PERFORM=ADDCNT(10,1),COUNT=(10,NOSPACE)
```

Figure 36. The Monitor Control Table (MCT) for listener (Part 1 of 2)

```
*
* NUMBER OF TIMES LENGTH ERROR ON CHILD SERVER TD QUEUE
*
         DFHMCT TYPE=EMP,ID=(EZA02.11),CLASS=PERFORM,                  X
               PERFORM=ADDCNT(11,1),COUNT=(11,LENERR)
*
* LISTENER TERMINATION
*
         DFHMCT TYPE=EMP,ID=(EZA02.12),CLASS=PERFORM,                  X
               PERFORM=(MLTCNT(12,11)),                                X
               COUNT=(12,TCONN,TSTARTED,TINVALID,TDISTRAN,TDISPROG,TGIVX
               ESOK,TSECEXIT,TNOTAUTH,TIOERR,TNOSPACE,TLENERR)
         DFHMCT TYPE=FINAL
         END
```

Figure 36. The Monitor Control Table (MCT) for listener (Part 2 of 2)

In the ID parameter, the following specifications are used:

(EZA02.01)
Completion of ACCEPT call

(EZA02.02)
Completion of CICS transaction initiation

(EZA02.03)
Detection of Invalid Transaction ID

(EZA02.04)
Detection of Disabled Transaction

(EZA02.05)
Detection of Disabled Program

(EZA02.06)
Detection of Givesocket Failure

(EZA02.07)
Transaction Rejection by Security Exit

(EZA02.08)
Transaction Not Authorized

(EZA02.09)
I/O Error on Transaction Start

(EZA02.10)
No Space Available for TD Start Message

(EZA02.11)
TD Length Error

(EZA02.12)
Program Termination

Open TCB measurements

When migrating IP CICS sockets-enabled applications to exploit the CICS Transaction Server Open Transaction Environment it is important to consider that the CPU usage is spent on both the QR TCB and the L8 TCB.

The time spent on the QR TCB can be used on the following:

- Task startup
- Processing a non-threadsafe CICS command

- Processing application code when switched back to the QR TCB
- Processing non-threadsafe subprograms
- Final task processing

The time spent on the L8 TCB can be used on the following:

- OPEN TCB processing
- Processing the EZASOKET call
- Running the application code
- Processing threadsafe CICS commands
- Processing threadsafe subprograms
- TCP/IP processing the socket call

If the application makes use of other non-CICS resources that are enabled to exploit OTE (such as DB2) then that CPU usage time is also accumulated against the QR and L8 TCBs.

If IP CICS sockets is not using OTE, then all the CPU time that is used to process the EZASOKET call occurs on the private MVS subtasks and shows up on the SMF 30 record.

If IP CICS sockets is using OTE, then the CPU time that is used to process the EZASOKET call shows up for the CICS transaction.

The following figure shows a EZASOKET threadsafe transaction. The numbers correspond to the list that follows.

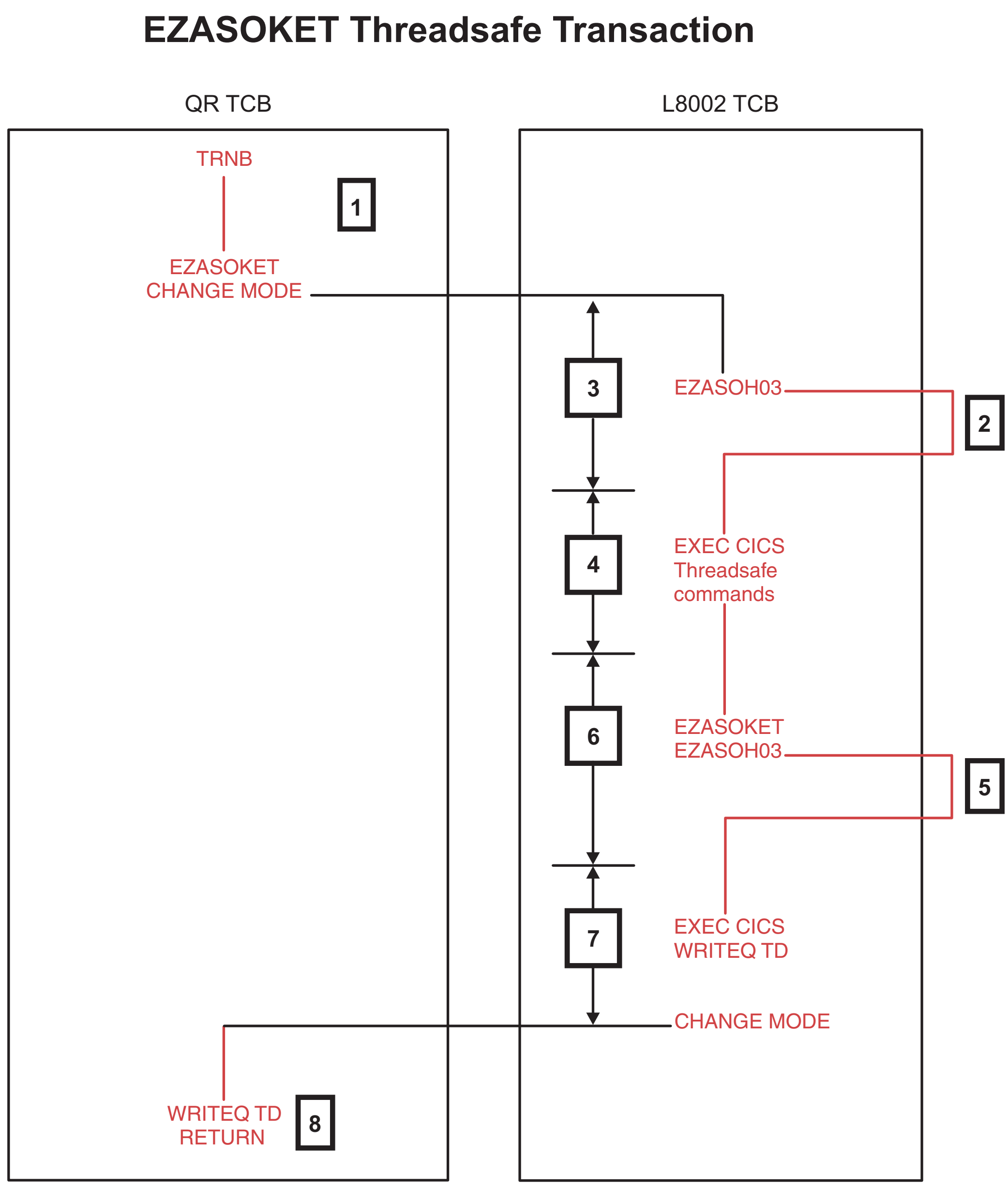


Figure 37. EZASOKET threadsafe transaction

1. Represents the task startup and the application until it issues the first EZASOKET call.
2. Actual time spent in Sockets Extended, processing the first EZASOKET call.
3. Time spent in the resource manager interface (RMI), processing the EZASOKET call
4. Threadsafe application code and EXEC CICS commands running.
5. Time spent in Sockets Extended, processing the second EZASOKET call.
6. Time spent in the RMI, processing the second request.
7. Final application code, which issues a non-threadsafe EXEC CICS WRITEQ TD command causing a change_mode back to the QR TCB.
8. Final task processing on the QR TCB.

CICS program list table (PLT)

You can allow automatic startup/shutdown of the CICS socket interface through updates to the PLT. This is achieved through placing the EZACIC20 module in the appropriate PLT.

To start the IP CICS socket interface automatically, make the following entry in PLTPI *after* the DFHDELIM entry:

```
*
* PLT USED TO SUPPORT IP CICS SOCKETS STARTUP
*
         DFHPLT TYPE=INITIAL,SUFFIX=SI
         DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
         DFHPLT TYPE=ENTRY,PROGRAM=EZACIC20
*
* Add other IP CICS Socket PLT startup programs here...
*
         DFHPLT TYPE=FINAL
         END
```

To shut down the IP CICS socket interface automatically (including all other IP CICS sockets enabled programs), make the following entry in the PLTSD *before* the DFHDELIM entry:

```
*
* PLT USED TO SUPPORT IP CICS SOCKETS SHUTDOWN
*
         DFHPLT TYPE=INITIAL,SUFFIX=SD
*
* Add other IP CICS Socket PLT shutdown programs here...
*
         DFHPLT TYPE=ENTRY,PROGRAM=EZACIC20
         DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
         DFHPLT TYPE=FINAL
         END
```

System recovery table

The system recovery table (SRT) contains a list of codes for abends that CICS intercepts. After intercepting one, CICS attempts to remain operational by causing the offending task to abend.

You can modify the default recovery action by writing your own recovery program. You do this using the XSRAB global user exit point within the system recovery program (SRP). For programming information about the XSRAB exit, see the *CICS Customization Guide*.

Note: Recovery is attempted only if a user task (not a system task) is in control at the time the abend occurs.

DFHSRT macroinstruction types

The following macroinstructions can be coded in a system recovery table:

- DFHSRT TYPE=INITIAL establishes the control section.
- DFHSRT TYPE=SYSTEM or DFHSRT TYPE=USER specifies the abend codes that are to be handled.
- DFHSRT TYPE=FINAL concludes the SRT. For details about the TYPE=FINAL macroinstruction, see the *CICS Resource Definition Guide*.

Control section: The DFHSRT TYPE=INITIAL macroinstruction generates the system recovery table control section.

```
►►──DFHSRT──TYPE=INITIAL──┬─────────────────┬──────────────────────────────►◄
                          └─,──SUFFIX=──xx──┘
```

For general information about TYPE=INITIAL macroinstructions, including the use of the SUFFIX operand, see the *CICS Resource Definition Guide*.

Abend codes: The DFHSRT TYPE=SYSTEM and DFHSRT TYPE=USER macroinstructions indicate the type of abend codes to be intercepted.

```
►►──DFHSRT──TYPE=──┬─SYSTEM─┬──,──ABCODE=──(codes)──┬───────────────────────────┬──►◄
                   └─USER───┘                       │              ┌─NO──┐      │
                                                    └─,──RECOVER=──┴─YES─┴──────┘
```

SYSTEM
: The abend code is an operating system abend code corresponding to an MVS S*xxx* abend code. The abend code must be three hexadecimal digits (*xxx*) representing the MVS system abend code S*xxx*.

USER
: The abend code is a user (including CICS) abend code corresponding to an MVS U*nnnn* abend code. The abend code must be a decimal number (*nnnn*) representing the user part of the MVS abend code U*nnnn*. This is usually the same number as the CICS message that is issued before CICS tries to terminate abnormally (see *CICS Messages and Codes*).

ABCODE=(*codes***)**
: ABCODE includes the abend code (or codes) to be intercepted. If you specify a single abend code, parentheses are not required. To specify multiple abend codes, separate the codes with commas.

RECOVER
: Specifies whether codes are to be added or removed from the SRT. Code YES to add the specified codes to the SRT. Code NO to remove the specified codes from the SRT.

CICS intercepts the following abend codes automatically and tries to recover:

```
001,002,013,020,025,026,030,032,033,034,035,
036,037,03A,03B,03D,0F3,100,113,137,213,214,
237,283,285,313,314,337,400,413,437,513,514,
613,614,637,713,714,737,813,837,913,A13,A14,
B13,B14,B37,D23,D37,E37
```

Abend code 0F3 covers various machine check conditions. It also covers the Alternate Processor Retry condition that can occur only when running on a multiprocessor. CICS-supplied recovery code attempts to recover from instruction-failure machine checks on the assumption that they are not permanent. It also attempts to recover from Alternate Processor Retry conditions.

CICS tries to recover from the standard abend codes above if you code the system recovery table simply as follows. There is no need to list the standard codes individually.

```
DFHSRT  TYPE=INITIAL
DFHSRT  TYPE=FINAL
END
```

If you want CICS to handle other errors, you can code the SRT as follows:

```
DFHSRT  TYPE=INITIAL
DFHSRT  TYPE=SYSTEM,or USER,
        ABCODE=(user or system codes),
        RECOVER=YES
DFHSRT  TYPE=FINAL
END
```

If you do not want CICS to try to recover after one or more of the above standard abend codes occurs, specify the codes with RECOVER=NO (or without the RECOVER parameter).

Note: Recovery is attempted only if a user task (not a system task) is in control at the time the abend occurs.

DFHSRT example

Following is an example of the coding required to generate a SRT:

```
DFHSRT TYPE=INITIAL,            *
       SUFFIX=K1
DFHSRT TYPE=SYSTEM,             *
       ABCODE=777,              *
       RECOVER=YES
DFHSRT TYPE=USER,
       ABCODE=(888,999),        *
       RECOVER=YES
DFHSRT TYPE=USER,               *
       ABCODE=020
DFHSRT TYPE=FINAL
END
```

Security considerations

The following transactions should be added to your xCICSTRN RACF® class:

EZAC Configure sockets interface.

EZAO Enable sockets interface.

EZAP Disable socket interface started by the EZAO, STOP, and YES transactions.

CSKL Listener. Also, any user defined transactions that execute EZACIC02.

The EZAC and EZAO transactions are designed to be run with a terminal. If you want a user to administer the IP CICS sockets configuration then you must grant the user authorization to the EZAC transaction. If you want a user to manually start and stop the IP CICS socket interface then you must grant the user authorization to the EZAO and EZAP transactions . If you want a user to manually start and stop the listener then you must grant the user authorization to the EZAO and CSKL (and any user defined transaction defined to execute EZACIC02) transactions.

For terminal tasks where a user has not signed on, the user ID is the CICS user ID associated with the terminal and is either:

- The default CICS user ID as specified on the CICS parameter DFLTUSER coded in the CICS System Initialization Table, SIT.
- A preset security user ID specified on the terminal definition.

The IP CICS socket interface can be started and shutdown by placing EZACIC20 in the PLT; therefore, an entry must be placed in your PLT RACF class to allow this action. User ID's that are used to start the IP CICS socket interface include those defined with the PLTPIUSR SIT macro should be allowed USE access to the resource class where the IP CICS sockets transactions are defined. The CICS region user ID must also be authorized to be the surrogate of the user ID specified on the PLTPIUSR parameter.

User ID's used to manage the starting and stopping of the CICS socket interface (EZAO), the listener (CSKL or user defined transactions executing EZACIC02) and user application programs linking to the IP CICS domain name server module, EZACICxx should at least be granted UPDATE access to the EXITPROGRAM resource.

There are three WLM Group Name listener configuration options. When these are specified, the listener registers and deregister the listeners group names with WLM. The CICS address space user ID requires read access to the BPX.WLMSERVER profile if that profile is defined and one of the WLM Group Name configuration options is specified.

For more information about RACF security management in the CICS environment, see the *CICS RACF Security Guide*.

TCP/IP services — Modifying data sets

To run CICS TCP/IP, you need to make entries in the *hlq*.PROFILE.TCPIP configuration data set. [6]

The *hlq*.PROFILE.TCPIP data set

You define the CICS region to TCP/IP on z/OS in the *hlq*.PROFILE.TCPIP data set (described in *z/OS Communications Server: IP Configuration Reference* and *z/OS Communications Server: IP Configuration Guide*). In it, you must provide entries for the CICS region in the PORT statement, as shown in Figure 38 on page 50.

The format for the PORT statement is:

```
port_number TCP CICS_jobname
```

Write an entry for each port that you want to reserve for an application. Figure 38 on page 50 shows two entries, allocating port number 3000 for SERVA, and port number 3001 for SERVB. SERVA and SERVB are the job names of our CICS regions.

These two entries reserve port 3000 for exclusive use by SERVA and port 3001 for exclusive use by SERVB. The listener transactions for SERVA and SERVB should be bound to ports 3000 and 3001 respectively. Other applications that want to access TCP/IP on z/OS 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.

6. Note that in this document, the abbreviation *hlq* stands for high level qualifier. This qualifier is installation dependent.

```
;
; hlq.PROFILE.TCPIP
; ==================
;
; This is a sample configuration file for the TCPIP address space.
; For more information about this file, see "Configuring the TCPIP
; Address Space" and "Configuring the Telnet Server" in the
; Customization and Administration Manual.
       ..........
       ..........
; ----------------------------------------------------------------------
; Reserve PORTs for the following servers.
;
; NOTE:  A port that is not reserved in this list can be used by
;        any user.  If you have TCP/IP hosts in your network that
;        reserve ports in the range 1-1023 for privileged
;        applications, you should reserve them here to prevent users
;        from using them.
PORT
       ..........
       ..........
  3000 TCP SERVA           ; CICS Port for SERVA               1
  3001 TCP SERVB           ; CICS Port for SERVB
```

Figure 38. Definition of the hlq.TCP/IP profile

Two different CICS listeners running on the same host can share a port. See the discussion on port descriptions in *z/OS Communications Server: IP Configuration Reference* for more information about ports.

The *hlq*.TCPIP.DATA data set

For CICS TCP/IP, 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 Services address space.

You need it when you initialize CICS TCP/IP (see Chapter 4, "Managing IP CICS sockets," on page 103). 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 specifies the name of the started procedure which was
; used to start the TCP/IP address space.    TCPIP is the default.
;
TCPIPJOBNAME TCPV3
       ..........
       ..........
       ..........
```

Figure 39. The TCPIPJOBNAME parameter in the hlq.TCPIP.DATA data set

z/OS UNIX Systems Services — adding a UNIX system services segment

The user ID associated with the CICS/TS region where z/OS IP CICS Sockets is used requires a z/OS UNIX System Services segment. See the information in *z/OS Security Server RACF Security Administrator's Guide* about defining groups and users, user profiles, and the OMVS segment in user profiles for more details about specifying a segment.

Configuring the CICS TCP/IP environment

The Configuration File contains information about the CICS sockets environment. The file is organized by two types of objects—CICS instances and listeners within those instances. The creation of this data set is done in three stages:

1. Create the empty data set using VSAM IDCAMS (Access Method Services).
2. Initialize the data set using the program generated by the EZACICD macro. The first two steps are described in "JCL for the configuration macro" on page 66.
3. Add to or modify the data set using the configuration transaction EZAC. This step is described in "Customizing the configuration data set" on page 70.[7]

Building the configuration data set with EZACICD

The configuration macro (EZACICD) is used to build the configuration data set. This data set can then be incorporated into CICS using resource definition online (RDO) and can be modified using the configuration transactions (see "Configuration transaction (EZAC)" on page 70). The macro is keyword driven; the TYPE keyword controlling the specific function request. The data set contains one record for each instance of CICS that it supports, and one record for each listener. The following is an example of the macros required to create a configuration file for two instances of the CICS socket interface listeners each:

7. The EZAC transaction is modeled after the CEDA transaction used by CICS Resource Definition Online (RDO).

```
EZACICD TYPE=INITIAL,   Start of macro assembly input        X
      FILNAME=EZACICDF, DD name for configuration file       X
      PRGNAME=EZACICDF  Name of batch program to run
EZACICD TYPE=CICS,      CICS record definition               X
      APPLID=CICSPROD,  APPLID of CICS region not using OTE  X
      TCPADDR=TCPIP,    Job/Step name for TCP/IP             X
      PLTSDI=YES,       PLT shutdown method is immediately   X
      NTASKS=20,        Number of subtasks                   X
      DPRTY=0,          Subtask dispatch priority difference X
      CACHMIN=15,       Minimum refresh time for cache       X
      CACHMAX=30,       Maximum refresh time for cache       X
      CACHRES=10,       Maximum number of resident resolvers X
      ERRORTD=CSMT,     Transient data queue for error msgs  X
      TCBLIM=0,         Open API TCB Limit                   X
      OTE=NO,           Use Open Transaction Environment     X
      TRACE=NO,         Trace CICS Sockets                   X
      APPLDAT=YES,      Register Application Data            X
      SMSGSUP=NO,       STARTED Messages Suppressed?         X
      TERMLIM=100       Subtask Termination Limit
EZACICD TYPE=CICS,      CICS record definition               X
      APPLID=CICSPRDB,  APPLID of CICS region using OTE      X
      TCPADDR=TCPIP,    Job/Step name for TCP/IP             X
      PLTSDI=NO,        PLT shutdown method is deferred      X
      CACHMIN=15,       Minimum refresh time for cache       X
      CACHMAX=30,       Maximum refresh time for cache       X
      CACHRES=10,       Maximum number of resident resolvers X
      ERRORTD=CSMT,     Transient data queue for error msgs  X
      TCBLIM=12,        Open API TCB Limit                   X
      OTE=YES,          Use Open Transaction Environment     X
      TRACE=NO,         Trace CICS Sockets                   X
      APPLDAT=NO,       No Application Data                  X
      SMSGSUP=NO        STARTED Messages Suppressed?
EZACICD TYPE=LISTENER,  Listener record definition           X
      FORMAT=STANDARD,  Standard Listener                    X
      APPLID=CICSPROD,  Applid of CICS region                X
      TRANID=CSKL,      Transaction name for Listener        X
      PORT=3010,        Port number for Listener             X
      IMMED=YES,        Listener starts up at initialization? X
      BACKLOG=20,       Backlog value for Listener           X
      NUMSOCK=50,       # of sockets supported by Listener   X
      MINMSGL=4,        Minimum input message length         X
      ACCTIME=30,       Timeout value for Accept             X
      GIVTIME=30,       Timeout value for Givesocket         X
      REATIME=30,       Timeout value for Read               X
```

Figure 40. EZACICFG configuration file (Part 1 of 3)

```
|             RTYTIME=10,       Wait 10 seconds for TCP to come back  X
|             LAPPLD=YES,       Register Application Data             X
              TRANTRN=YES,      Is TRANUSR=YES conditional?           X
              TRANUSR=YES,      Translate user data?                  X
              SECEXIT=EZACICSE, Name of security exit program         X
              WLMGN1=WLMGRP01,  WLM group name 1                      X
              WLMGN2=WLMGRP02,  WLM group name 2                      X
              WLMGN3=WLMGRP03   WLM group name 3
        EZACICD TYPE=LISTENER,  Listener record definition            X
              FORMAT=ENHANCED,  Enhanced Listener                     X
              APPLID=CICSPROD,  Applid of CICS region                 X
              TRANID=CSKM,      Transaction name for Listener         X
              PORT=3011,        Port number for Listener              X
              IMMED=YES,        Listener starts up at initialization? X
              BACKLOG=20,       Backlog value for Listener            X
              NUMSOCK=50,       # of sockets supported by Listener    X
              ACCTIME=30,       Timeout value for Accept              X
              GIVTIME=30,       Timeout value for Givesocket          X
              REATIME=30,       Timeout value for Read                X
|             RTYTIME=20,       Wait 20 seconds for TCP to come back  X
|             LAPPLD=INHERIT,   Inherit interface setting             X
              CSTRAN=TRN1,      Name of child IPv4 server transaction X
              CSSTTYP=KC,       Child server startup type             X
              CSDELAY=000000,   Child server delay interval           X
              MSGLEN=0,         Length of input message               X
              PEEKDAT=NO,       Peek option                           X
              MSGFORM=ASCII,    Output message format                 X
              SECEXIT=EZACICSE, Name of security exit program         X
              WLMGN1=WLMGRP04,  WLM group name 1                      X
              WLMGN2=WLMGRP05,  WLM group name 2                      X
              WLMGN3=WLMGRP06   WLM group name 3
        EZACICD TYPE=LISTENER,  Listener record definition            X
              FORMAT=STANDARD,  Standard listener                     X
              APPLID=CICSPRDB,  Applid of CICS region                 X
              TRANID=CS6L,      Transaction name for listener         X
              PORT=3012,        Port number for listener              X
              AF=INET6,         Listener Address Family               X
              IMMED=YES,        Listener starts up at initialization? X
              BACKLOG=20,       Backlog value for listener            X
              NUMSOCK=50,       # of sockets supported by listener    X
              MINMSGL=4,        Minimum input message length          X
              ACCTIME=30,       Timeout value for Accept              X
              GIVTIME=30,       Timeout value for Givesocket          X
              REATIME=30,       Timeout value for Read                X
|             RTYTIME=0,        Listener will end when TCP ends       X
|             LAPPLD=NO,        No Application Data                   X
              TRANTRN=YES,      Is TRANUSR=YES conditional?           X
              TRANUSR=YES,      Translate user data?                  X
              SECEXIT=EZACICSE, Name of security exit program         X
              WLMGN1=WLMGRP01,  WLM group name 1                      X
```

Figure 40. EZACICFG configuration file (Part 2 of 3)

```
       WLMGN2=WLMGRP02,  WLM group name 2                      X
       WLMGN3=WLMGRP03   WLM group name 3
 EZACICD TYPE=LISTENER,  Listener record definition            X
       FORMAT=ENHANCED,  Enhanced listener                     X
       APPLID=CICSPRDB,  Applid of CICS region                 X
       TRANID=CS6M,      Transaction name for listener         X
       PORT=3013,        Port number for listener              X
       AF=INET6,         Listener Address Family               X
       IMMED=YES,        Listener starts up at initialization? X
       BACKLOG=20,       Backlog value for listener            X
       NUMSOCK=50,       # of sockets supported by listener    X
       ACCTIME=30,       Timeout value for Accept              X
       GIVTIME=30,       Timeout value for Givesocket          X
       REATIME=30,       Timeout value for Read                X
|      RTYTIME=0,        Listener will end when TCP ends       X
|      LAPPLD=INHERIT,   Inherit interface setting             X
       CSTRAN=TRN6,      Name of IPv6 child server transaction X
       CSSTTYP=KC,       Child server startup type             X
       CSDELAY=000000,   Child server delay interval           X
       MSGLEN=0,         Length of input message               X
       PEEKDAT=NO,       Peek option                           X
       MSGFORM=ASCII,    Output message format                 X
       SECEXIT=EZACICSE, Name of security exit program         X
       WLMGN1=WLMGRP04,  WLM group name 1                      X
       WLMGN2=WLMGRP05,  WLM group name 2                      X
       WLMGN3=WLMGRP06   WLM group name 3
 EZACICD TYPE=FINAL      End of assembly input
```

Figure 40. EZACICFG configuration file (Part 3 of 3)

TYPE parameter

The TYPE parameter controls the function requests. It can have the following values:

Value Meaning

INITIAL
Initialize the generation environment. This value should be used only once per generation and it should be in the first invocation of the macro. For subparameters, see "TYPE=INITIAL."

CICS Identify a CICS object. This value corresponds to a specific instance of CICS. Specifying this value creates a configuration record. For subparameters, see "TYPE=CICS" on page 55.

LISTENER
Identify a listener object. This value creates a listener record. For subparameters, see "TYPE=LISTENER" on page 59.

FINAL
Indicates the end of the generation. There are no subparameters.

TYPE=INITIAL: When TYPE=INITIAL is specified, the following parameters apply:

Value Meaning

PRGNAME
The name of the generated initialization program. The default value is EZACICDF.

FILNAME
The DDNAME used for the Configuration File in the execution of the initialization program. The default value is EZACICDF.

TYPE=CICS: When TYPE=CICS is specified, the following parameters apply:

Value Meaning

APPLDAT
Indicates whether the IP CICS socket interface automatically registers application data that is unique to IP CICS sockets TCP connections. All socket-enabled CICS programs are affected. Listener programs are affected based on the LAPPLD configuration option. See the listener's LAPPLD configuration option for information about configuring listeners to register application data. Possible values for the APPLDAT option are YES and NO; NO is the default when the APPLDAT parameter is not specified. Specify the value APPLDAT=YES to automatically apply application data to the TCP connection when the following socket commands are invoked:

- Before LISTEN or listen()
- Before GIVESOCKET for the IBM listener
- After TAKESOCKET or takesocket()
- After CONNECT or connect()

The IBM listener's optional security exit can override this setting for each accepted connection that is to be given to a child server. Overriding the setting enables application data that is specific to the child server to be registered against the accepted connections. For more information about using the security exit to register application data, see Chapter 6, "Application programming guide," on page 123 and application data in *z/OS Communications Server: IP Configuration Reference*. For more information about programming applications, see application data in *z/OS Communications Server: IP Configuration Reference*. The associated application data is made available on the Netstat ALL/-A, ALLConn/-a and COnn/-c reports, in the SMF 119 TCP connection termination records, and through the network management interface (NMI) on the GetTCPListeners and GetConnectionDetail poll requests. The Netstat and NMI interfaces support new filters for selecting sockets based on wildcard comparisons of the application data. This support can assist in locating application sockets during problem determination and can aid capacity planning and accounting applications to correlate TCP/IP SMF resource records with other applications records. It is the responsibility of the using applications to document the content, format, and meaning of the associated data.

APPLID
The APPLID of the CICS address space in which this instance of CICS/sockets is to run. This field is mandatory.

CACHMAX
The maximum refresh time for the Domain Name Server cache in minutes. This value depends on the stability of your network, that is, the time you would expect a domain name to have the same Internet address. Higher values improve performance but increase the risk of getting an incorrect (expired) address when resolving a name. The value must be greater than CACHMIN. The default value is 30.

CACHMIN
The minimum refresh time for the Domain Name Server cache in minutes. This value depends on the stability of your network, that is, the time you

would expect a domain name to have the same Internet address. Higher values improve performance but increase the risk of getting an incorrect (expired) address when resolving a name. The value must be less than CACHMAX. The default value is 15.

CACHRES
The maximum number of concurrent resolvers desired. If the number of concurrent resolvers is equal to or greater than this value, refresh of cache records does not happen unless their age is greater than the CACHMAX value. The default value is 10.

DPRTY
The difference between the dispatching priority of the subtasks and the attaching CICS task. Use this parameter to balance the CPU demand between CICS and the socket interface subtasks. Specifying a nonzero value causes the subtasks to be dispatched at a lower priority than CICS. Use the default value of 0 unless tuning data indicates that CICS is CPU-constrained. This value should be specified as 0 or not specified when OTE=YES is specified because the pool of reusable MVS subtasks is not needed. If DPRTY is specified as a nonzero value and OTE=YES, DPTRY is forced to 0.

ERRORTD
The name of a Transient Data destination to which error messages are written. The default value is CSMT. A check is made when the IP CICS socket interface is initialized to determine whether the transient data destination is defined to CICS. If the destination is not defined, the interface sends its messages to CSMT.

NTASKS
The number of reusable MVS subtasks that are allocated for this execution. This number should approximate the highest number of concurrent CICS transactions using the TCP/sockets interface, excluding listeners. The default value is 20. This value should be specified as 0 or not specifed when OTE=YES is specified because the pool of reusable MVS subtasks is not needed. If NTASKS is specified as a nonzero value and OTE=YES, NTASKS is forced to 0.

OTE The value for OTE is YES or NO (the default). A value of YES causes the IP CICS sockets task-related user exit to execute using the CICS Open Transaction Environment.

Note: OTE is supported on CICS/TS V2R2M0 and later. If OTE=YES is specified on a pre-CICS/TS V2R2M0 system, the IP CICS socket interface fails initialization.

When OTE=YES is specified, CICS/TS switches all EZASOKET calls and all IP CICS C socket functions from the QR TCB to an L8 TCB. IP CICS sockets applications must be coded using threadsafe programming practices as defined by CICS, and must be defined to CICS as threadsafe. A value of NO causes IP CICS sockets to continue executing EZASOKET calls on an MVS subtask managed by the IP CICS sockets interface. If OTE=YES, the values of NTASKS, DPRTY and TERMLIM are forced to 0 (if specified).

Table 4 on page 57 shows the relationships between the configuration options affected by OTE.

Table 4. Configuration options affected by OTE

OTE	TCBLIM	NTASKS	DPRTY	TERMLIM
YES	0 then • No IP CICS sockets applications are subject to TCBLIM • IP CICS sockets applications are subject to MAXOPENTCBS	If specified, forced to 0	If specified, forced to 0	If specified, forced to 0
YES	TCBLIM= MAXOPENTCBS As MAXOPENTCBS takes precedence over TCBLIM, IP CICS sockets applications are suspended by CICS/TS.	If specified, forced to 0	If specified, forced to 0	If specified, forced to 0
YES	1-MAXOPENTCBS	If specified, forced to 0	If specified, forced to 0	If specified, forced to 0
	Not numeric, then MNOTE 12			
NO	0	Using MVS subtasks	Using MVS subtasks	Using MVS subtasks
NO	1-MAXOPENTCBS, forced to 0	Using MVS subtasks	Using MVS subtasks	Using MVS subtasks
If neither YES or NO, then MNOTE 12				

PLTSDI
: The IP CICS sockets program load table (PLT) shutdown immediate configuration option. When IP CICS sockets is being shutdown using the EZACIC20 PLT program, then the PLTSDI parameter specifies whether the interface should shutdown immediately. The values are NO and YES. The default, if not specified, is NO. The value NO specifies a deferred shutdown. The value YES specifies an immediate shutdown. If the PLTSDI parameter is not specified then a deferred shutdown is performed. A deferred shutdown enables all IP CICS sockets tasks to end gracefully. An immediate shutdown directs all IP CICS sockets tasks to be immediately terminated.

SMSGSUP
: The value for SMSGSUP is either YES or NO (the default). A value of YES causes messages EZY1318E, EZY1325I, and EZY1330I to be suppressed. A value of NO allows these messages to be issued. If OTE=YES and when SMSGSUP is specified as YES then no TCB switch from the open API TCB to the QR TCB occurs for the above messages.

For detailed information on CICS sockets messages, see Appendix D, "CICS sockets messages," on page 417.

TCBLIM
: Specifies the maximum number of open API (L8) TCBs that can be used by the IP CICS socket interface to support socket calls, which, in turn, limits the maximum number of concurrently supported socket calls.

Note: TCBLIM is supported on CICS/TS V2R2M0 and later. If OTE=YES is specified on a pre-CICS/TS V2R2M0 system then the IP CICS socket interface fails initialization.

The CICS MAXOPENTCBS system initialization parameter controls the total number of open API, L8, TCBs that the CICS region can have in operation at any one time. It is relevant when CICS is connected to DB2 Version 6 or later, when open API TCBs are used to run threads into DB2, and when open API TCBs are used to support sockets extended calls into TCP/IP. In the open transaction environment, TCBLIM controls how many open API TCB's can be used by the IP CICS sockets task-related user exit to support socket calls into TCP/IP. The listener is not subjected to this limitation; however, it is subject to MAXOPENTCBS. This allows listeners to be started prohibiting a possible denial of service. If MAXOPENTCBS is reached then no more open API TCBs are available in the CICS region and the IP CICS sockets task-related user exit cannot obtain an open TCB for its use. The default value for TCBLIM is 0. If this value is set to zero and OTE=YES, then the IP CICS socket interface uses the entire open API (L8) pool. This value should be set high enough to accommodate the number of concurrently active child server tasks and the number of concurrently active outbound clients. TCBLIM can be set from 0 to the value specified by CICS's MAXOPENTCBS. If OTE=NO and TCBLIM>0, TCBLIM is forced to 0.

A check is made when the IP CICS socket interface is initialized to determine if TCBLIM>MAXOPENTCBS. If so then TCBLIM is dynamically set to the value specified by MAXOPENTCBS and message EZY1355I is issued and the interface continues to initialize. Use the EZAC configuration transaction to update the configuration to reflect this change or adjust the offending TYPE=CICS,TCBLIM entry in your configuration macro.

Use the EZAO Operator transaction to inquire on the current IP CICS socket interface levels and also to dynamically alter the value specified by TCBLIM. When TCBLIM is reached, message EZY1356E is issued. Message EZY1360I is issued once the TCBLIM condition is relieved. See Table 4 on page 57 for more information.

TCPADDR
The name of the z/OS Communication Server TCP/IP address space.

TERMLIM
During a quiescent termination of the CICS sockets interface, the termination program posts unused reusable subtasks (see NTASKS) for termination. TERMLIM specifies the maximum number of these posts that can be issued in a single second. Too low of a TERMLIM value can cause termination to take a long time to complete. Too high of a TERMLIM value can cause the CICS region to ABEND due to storage shortage. The default is 100. A value of 0 causes the default value of 100 to be used. This value should be specified as zero or not specified when OTE=YES is specified as the pool of reusable MVS subtasks are not needed. If TERMLIM is specified as a nonzero value and OTE=YES, TERMLIM is forced to zero.

TRACE
The value for TRACE is either YES (the default) or NO. A value of NO will direct the TRUE and the listener to not generate CICS AP trace records even if CICS trace is active. The value of YES will direct the TRUE and the listener to generate CICS AP trace records which also requires that CICS Trace be active. Trace records are generated only if CICS tracing is active and TRACE=YES. See the *CICS Transaction Server for z/OS CICS Supplied*

Transactions publication for guidance on enabling and disabling the CICS trace. See the *CICS Transaction Server for z/OS CICS Operations and Utilities Guide* for guidance printing the CICS trace. Use the EZAO,START|STOP,TRAce to dynamically enable or disable tracing. Suppressing the generation of trace records after IP CICS sockets application programs are tested and debugged or for normal operations can improve performance.

TYPE=LISTENER: When TYPE=LISTENER is specified the following parameters apply:

ACCTIME
The time in seconds this listener waits for a connection request before checking for a CICS/sockets shutdown or CICS shutdown. The default value is 60. A value of 0 results in the listener continuously checking for a connection request without waiting. Setting this to a high value reduces the resources used to support the listener on a lightly loaded system and consequently lengthens shutdown processing. Conversely, setting this to a low value increases resources used to support the listener but facilitate shutdown processing.

AF Determines if the listener being defined supports IPv6 partners and be able to give an IPv6 socket descriptor to an IPv6 child server program. YES indicates that the listener gives an IPv6 socket to the child server program. NO, the default, indicates that the listener gives an IPv4 socket to the child server program. You must ensure that the child server program performing the TAKESOCKET command must match the domain of the socket being given by the listener.

APPLID
The APPLID value of the CICS object for which this listener is being defined. If this is omitted, the APPLID from the previous TYPE=CICS macro is used.

BACKLOG
The number of unaccepted connections that can be queued to this listener. The default value is 20.

Note: The BACKLOG value specified on the LISTEN call cannot be greater than the value configured by the SOMAXCONN statement in the stack's TCP/IP profile (default=10); no error is returned if a greater BACKLOG value is requested. If you want a larger backlog, update the SOMAXCONN statement. See *z/OS Communications Server: IP Configuration Reference* for details.

CSDELAY
This parameter is specific to the enhanced version of the listener and is applicable only if CSSTTYPE is IC. It specifies the delay interval to be used on the EXEC CICS START command, in the form hhmmss (hours/minutes/seconds).

CSSTTYPE
This parameter is specific to the enhanced version of the listener and specifies the default start method for the child server task. This can be overridden by the security/transaction exit. Possible values are IC, KC, and TD.

IC Indicates that the child server task is started using EXEC CICS START with the value specified by CSDELAY (or an overriding value from the security/transaction exit) as the delay interval.

KC Indicates that the child server task is started using EXEC CICS START with no delay interval. This is the default.

TD Indicates that the child server task is started using the EXEC CICS WRITEQ TD command, which uses transient data to trigger the child server task. If OTE=YES, the listener incurs a TCB switch from an open API TCB to the QR TCB when starting the specified child server transaction.

CSTRANID
This parameter is specific to the enhanced version of the listener and specifies the default child server transaction that the listener starts. This can be overridden by the security/transaction exit. The child server transaction is verified to be defined to CICS and enabled when the listener is started by the EZAO Operator transaction.

FORMAT
The default value of STANDARD indicates that this is the original CICS listener that requires the client to send the standard header. The value of ENHANCED indicates that this is the enhanced CICS listener that does not expect the standard header from the client.

GETTID
The GETTID parameter is provided for the CICS listener that communicates with clients using SSL/TLS (Secure Socket Layer/Transport Layer Security) services available with the Application Transparent Transport Layer Security (AT-TLS) function provided by the TCP/IP stack. Specifically, it allows the listener to receive the user ID that is associated in the system's security product (such as RACF), with the connecting client's SSL certificate. This allows the listener to pass this user ID to the security exit where it can be accepted or overridden.

The GETTID values have the following meaning for the listener:

NO The listener does not request the client's certificate or user ID. This is the default action for GETTID.

YES The listener accepts the connection and asks for the client's certificate and user ID if available. If available, the address and the length of the client's certificate are sent to the security exit COMMAREA (if the security exit is specified) to signify that the client's certificate exists along with any received user ID. This allows the security exit to examine the contents. If the user ID is not extracted (either the client certificate does not exist or the client certificate does not contain a user ID), the security exit COMMAREA USERID field contains binary zeros.

GETTID values of YES should only be specified if the following is true:

- AT-TLS is currently enabled by the TCP/IP stack with the TTLS parameter specified on the TCPCONFIG TCP/IP profile statement.
- AT-TLS policy is in effect for connections processed by this listener, and the TTLSEnvironmentAction or TTLSConnectionAction statement associated with the listener must specify the HandshakeRole as ServerWithClientAuth. The level of client authentication for a connection is determined by the TTLSEnvironmentAdvancedParms statement ClientAuthType parameter.

If GETTID is YES then the listener attempts to obtain that user ID. If a user ID is successfully obtained and the start type is task control (KC) or interval control (IC), the listener uses that to initialize the user ID of the child server. The security exit can override it. If there is no security exit or the security exit chooses not to override it, that is the user ID of the child server task unless the start type is transient data (TD).

Note: The user ID under which the listener executes must have CICS RACF surrogate authority to any user ID that it uses to initialize the child server. See the *CICS RACF Security Guide* for details.

See Application Transparent Transport Layer Security (AT-TLS) topic of the *z/OS Communications Server: IP Configuration Guide* for more information.

GIVTIME

The time in seconds this listener waits for a response to a GIVESOCKET. If this time expires, the listener assumes that either the server transaction did not start or the TAKESOCKET failed. At this time, the listener sends the client a message indicating the server failed to start and close the socket (connection). If this parameter is not specified, the ACCTIME value is used.

IMMED

Specify YES or NO. YES indicates this listener is to be started when the interface starts. No indicates this listener is to be started independently using the EZAO transaction. The default is YES.

LAPPLD

This optional configuration option indicates whether the IP CICS socket interface automatically registers IP CICS sockets-unique application data for the listener's connection being defined. Both the IBM listener and user written listeners are affected. When defined for the IBM listener then it additionally registers application data against the accepted connections to be given to a child server. Only the listener being defined is affected. The possible values for LAPPLD are YES, NO, or INHERIT (the default). If the LAPPLD option is not specified or specified as INHERIT, then the option inherits the value specified by the APPLDAT configuration option. Alternatively, when LAPPLD is specified as YES or NO, then the option overrides the value specified by the APPLDAT configuration option. When the value of LAPPLD=NO is specified or it inherits the APPLDAT=NO specification, then no application data is automatically registered for the listener being defined. When LAPPLD=YES or it inherits the APPLDAT=YES specification then application data is automatically registered against a socket when the following socket commands are successfully invoked:

- Before LISTEN or listen()
- Before GIVESOCKET for the IBM listener
- After TAKESOCKET or takesocket()
- After CONNECT or connect()

The IBM listener's optional security exit can override this setting for each accepted connection that is to be given to a child server. Overriding the setting enables application data that is specific to the child server to be registered against the accepted connections to be given. For more information about programming applications, see Chapter 6, "Application programming guide," on page 123 and application data in *z/OS Communications Server: IP Configuration Reference*. For more information about programming applications, see application data in *z/OS Communications Server: IP Configuration Reference*. The associated application data is made available on the Netstat ALL/-A, ALLConn/-a and COnn/-c reports, in the SMF 119 TCP connection termination records and through the network management interface (NMI) on the GetTCPListeners and GetConnectionDetail poll requests. The Netstat and NMI interfaces support new filters for selecting sockets based on wildcard comparisons of the application data. This support can assist in locating application sockets during problem determination and can aid capacity planning and accounting applications to correlate TCP/IP SMF resource records with other applications records. It is the responsibility of the using applications to document the content, format, and meaning of the associated data.

Result: Listener configurations defined before V1R9 is set to the value NO.

MINMSGL
This parameter is specific to the standard version of the listener. The minimum length of the Transaction Initial Message from the client to the listener. The default value is 4. The listener continues to read on the connection until this length of data has been received. FASTRD handles blocking.

MSGFORM
This parameter is specific to the enhanced version of the listener and indicates whether an error message returned to the client should be in ASCII or EBCDIC. ASCII is the default. MSGFORM is displayed as MSGFORMat on the EZAC screens.

MSGLEN
This parameter is specific to the enhanced version of the listener and specifies the length of the data to be received from the client. The valid range is 0 to 999. If the value is 0, the listener does not read in any data from the client.

NUMSOCK
The number of sockets supported by this listener. One socket is the listening socket. The others are used to pass connections to the servers using the GIVESOCKET call so, in effect, one less than this number is the maximum number of concurrent GIVESOCKET requests that can be active. The default value is 50.

The number of CICS transactions must be less than what is specified on the MAXFILEPROC parameter on the BPXPRMxx parmlib member. For more detail on setting the MAXFILEPROC parameter, see *z/OS UNIX System Services Planning*.

PEEKDAT
This parameter is specific to the enhanced version of the listener and applies only if MSGLEN is not 0. A value of NO indicates that the listener performs a normal read of the client data. The child server application accesses this data in the *data area-2* portion of the transaction input message (TIM). A value of YES indicates that the listener reads the data using the peek option; the data remains queued in TCP/IP and the child server applications actually read it in rather than accessing it through the TIM.

PORT The port number this listener uses for accepting connections. This parameter is mandatory. The ports can be shared. See *z/OS Communications Server: IP Configuration Reference* for more information on port sharing.

REATIME
The time in seconds this listener waits for a response to a RECV request. If this time expires, the listener assumes that the client has failed and terminates the connection by closing the socket. If this parameter is not specified, checking for read timeout is not performed.

RTYTIME
This optional configuration option specifies the length of time, in seconds, that the listener waits after a TCP/IP stack outage occurs before it attempts to connect or reconnect. The value 0 specifies that the listener cleans up any resources and then the listener ends. A value greater than 0 and less than 15 results in a RTYTIME value of 15 seconds; the listener task is delayed 15 seconds before it attempts to connect or reconnect. The stack that it tries to connect to is the stack specified by the listener's IP CICS socket interface TCPADDR configuration option. If the connection fails, then the listener task is delayed for the length of time specified by the RTYTIME parameter. After this interval lapses, the listener attempts to connect to its stack. The listener continues to attempt to connect to the stack until either it succeeds or is terminated by the operator. Valid values are in the range 0 - 999. The default setting is 15 seconds. Table 5 shows a summary of the listener's action based on the combination of the RTYTIME value and the state of the listener's TCP stack.

Table 5. Listener's action based on RTYTIME and stack state

Listener	RTYTIME	TCP down	TCP up
Initially started	0	Listener ends	Listener initializes
	>0	Listener waits	
Previously active	0	Listener ends	
	>0	Listener waits	

SECEXIT
The name of the user written security exit used by this listener. The default is no security exit. The listener uses the EXEC CICS LINK command to give control to the security exit. If OTE=YES then it should be expected that the security exit program is defined to CICS as threadsafe, implying it is coded to threadsafe standards. A flag which indicates that the IP CICS socket interface is using CICS's Open Transaction Environment is passed to the security exit. This flag enables the security exit to decide which child server transaction to use and if it should possibly limit its use of non-threadsafe resources or commands. See "Writing your own security/transaction link module for the listener" on page 143 for a thorough discussion on the data passed to the exit. See "Threadsafe considerations for IP CICS sockets applications" on page 148 for more information about coding threadsafe programs. A check is made to ensure the specified security exit program is defined to CICS and enabled for use when the listener is started by the EZAO Operator transaction.

TRANID
The transaction name for this listener. The default is CSKL.

TRANTRN
This parameter is specific to the standard version of the listener. Specify YES or NO. YES indicates that the translation of the user data is based on

the character format of the transaction code. That is, with YES specified for TRANTRN, the user data is translated if and only if TRANUSR is YES and the transaction code is not uppercase EBCDIC. If NO specified for TRANTRN, the user data is translated if and only if TRANUSR is YES. The default value for TRANTRN is YES. See Table 6 for more information.

Note: Regardless of how TRANTRN is specified, translation of the transaction code occurs if and only if the first character is not uppercase EBCDIC.

TRANUSR

This parameter is specific to the standard version of the listener. Specify YES or NO. NO indicates that the user data from the Transaction Initial Message should not be translated from ASCII to EBCDIC. YES indicates that the user data can be translated depending on TRANTRN and whether the transaction code is uppercase EBCDIC. The default value for TRANUSR is YES. See Table 6 for more information.

Note: Previous implementations functioned as if TRANTRN and TRANUSR were both set to YES. Normally, data on the Internet is ASCII and should be translated. The exceptions are data coming from an EBCDIC client or binary data in the user fields. In those cases, you should set these values accordingly. If you are operating in a mixed environment, use of multiple listeners on multiple ports is recommended.

Table 6 shows how the listener handles translation with different combinations of TRANTRN, TRANSUSR, and character format of the transaction code.

Table 6. Conditions for translation of tranid and user data

TRANTRN	TRANUSR	Tranid format	Translate tranid?	Translate user data?
YES	YES	EBCDIC	NO	NO
YES	NO	EBCDIC	NO	NO
NO	YES	EBCDIC	NO	YES
NO	NO	EBCDIC	NO	NO
YES	YES	ASCII	YES	YES
YES	NO	ASCII	YES	NO
NO	YES	ASCII	YES	YES
NO	NO	ASCII	YES	NO

USERID

The 8-character user ID under which the listener runs. If this parameter is not specified, then the listener task obtains the user ID from either the CICS PLT user ID (if the listener is started via the CICS PLT) or the ID of the user that invoked the EZAO transaction (if the listener is started using the EZAO transaction). If this parameter is specified, then any user that starts the listener (the PLT user if the listener is started using the PLT) must have surrogate security access to this user ID. This user ID has to be permitted to any resources the listener accesses such as child server transactions and programs. See the *CICS RACF Security Guide* for details.

| The value specified for the user ID's FILEPROCMAX parameter should be
| configured appropriately. If the number of sockets that the listener creates

exceeds FILEPROCMAX value on the listener's user ID, then the listener stops accepting new sockets until the number of active sockets is equal to or less than the FILEPROCMAX value. For more information about the FILEPROCMAX specification, see the documentation provided for the SAF product in use on your system. If you are using RACF, see *z/OS Security Server RACF Security Administrator's Guide*..

WLMGN1
The group name this listener uses to participate in workload connection balancing. The group name is used to register the CICS listener with Workload Manager (WLM) so that a BIND-based Domain Name System (DNS) can be used to balance requests across multiple MVS hosts in a sysplex. DNS/WLM continues to support CICS listeners desiring to participate in work load balancing for IPv4 clients. IPv6-enabled listeners can participate in work load balancing for their IPv4 and IPv6 clients.

Note: BIND4 does not support IPv6; therefore, this option does not provide IPv6 balancing.

IPv6 clients should use unique hostnames and you should enable DNS entries to allow unique host names to exist in different DNS zones. This enables an IPv6 client to get an AAAA address to use when connecting to an IPv6 enabled listener. IPv6 enabled clients wanting to participate in work load balancing should continue to get the IPv4 address of the participating listener from the DNS/WLM server and then convert the IPv4 address to an IPv4-mapped IPv6 address. Use this address to connect to the IPv6 enabled listener. Note that this is not a true IPv6 connection as DNS/WLM does not give an IPv6 address. Clients that want to connect to the server over an IPv6 network should use an IPv6 address.

The group name can be 1 – 12 characters in length. The name is padded to the right with blanks to meet the 18-character name requirement by the Workload Manager.

The default is no registration.

When a you specify a group name, the listener registers and deregisters the listeners group names with WLM. The CICS address space user ID requires read access to the BPX.WLMSERVER profile if that profile is defined and one of the WLM group name configuration options is specified.

See *z/OS Communications Server: IP Configuration Reference* for information about connection balancing and BIND-based DNS.

Tip: The automated domain name registration (ADNR) application cannot provide WLM-based load balancing; however, you can configure it to provide round-robin connection balancing as supported by the BIND 9 name server. See the information in *z/OS Communications Server: IP Configuration Guide* for more about load balancing using an external load balancer and one or more load balancing agents. See automated domain name registration information in *z/OS Communications Server: IP Configuration Guide* for more details about dynamically updating name servers with information about sysplex resources in near real time.

WLMGN2
See WLMGN1 for information.

WLMGN3
See WLMGN1 for information.

JCL for the configuration macro

The configuration macro is used as part of a job stream to create and initialize the configuration file. The job stream consists of IDCAMS steps to create the file, the assembly of the initialization module generated by the configuration macro, linking of the initialization module, and execution of the initialization module that initializes the file.

Figure 41 on page 67 illustrates a job stream used to define a configuration file. See *hlq*.SEZAINST(EZACICFG) for a sample job stream.

```
//*********************************************************//
//* THE FOLLOWING JOB DEFINES AND THEN LOADS THE VSAM      *//
//* FILE USED FOR CICS/TCP CONFIGURATION. THE JOBSTREAM    *//
//* CONSISTS OF THE FOLLOWING STEPS.                       *//
//* 1). DELETE A CONFIGURATION FILE IF ONE EXISTS          *//
//* 2). DEFINE THE CONFIGURATION FILE TO VSAM              *//
//* 3). ASSEMBLE THE INITIALIZATION PROGRAM                *//
//* 4). LINK THE INITIALIZATION PROGRAM                    *//
//* 5). EXECUTE THE INITIALIZATION PROGRAM TO LOAD THE     *//
//* FILE                                                   *//
//*********************************************************//
//CONFIG JOB MSGLEVEL=(1,1)
//*
//* THIS STEP DELETES AN OLD COPY OF THE FILE
//* IF ONE IS THERE.
//*
//DEL    EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
   DELETE -
      CICS.TCP.CONFIG -
      PURGE -
      ERASE
//*
//* THIS STEP DEFINES THE NEW FILE
//*
//DEFILE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  DEFINE CLUSTER (NAME(CICS.TCP.CONFIG) VOLUMES(CICSVOL) -
      CYL(1 1) -
      IMBED -
      RECORDSIZE(150 150) FREESPACE(0 15) -
      INDEXED -
      SHAREOPTIONS(2,3)) -
      DATA ( -
        NAME(CICS.TCP.CONFIG.DATA) -
        KEYS (16 0) ) -
      INDEX ( -
        NAME(CICS.TCP.CONFIG.INDEX) )
/*
//*
//* THIS STEP ASSEMBLES THE INITIALIZATION PROGRAM
//*
//PRGDEF  EXEC PGM=ASMA90,PARM='OBJECT,TERM',REGION=1024K
//SYSLIB    DD DISP=SHR,DSNAME=SYS1.MACLIB
//          DD DISP=SHR,DSNAME=TCPIP.SEZACMAC
//SYSUT1    DD UNIT=SYSDA,SPACE=(CYL,(5,1))
//SYSUT2    DD UNIT=SYSDA,SPACE=(CYL,(2,1))
//SYSUT3    DD UNIT=SYSDA,SPACE=(CYL,(2,1))
//SYSPUNCH  DD DISP=SHR,DSNAME=NULLFILE
//SYSLIN    DD DSNAME=&&OBJSET,DISP=(MOD,PASS),UNIT=SYSDA,

//             SPACE=(400,(500,50)),
//             DCB=(RECFM=FB,BLKSIZE=400,LRECL=80)
//SYSTERM   DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//SYSIN     DD *
```

Figure 41. Example of JCL to define a configuration file (Part 1 of 4)

```
EZACICD TYPE=INITIAL,   Start of macro assembly input          X
      FILNAME=EZACICDF, DD name for configuration file         X
      PRGNAME=EZACICDF  Name of batch program to run
EZACICD TYPE=CICS,      CICS record definition                 X
      APPLID=CICSPROD,  APPLID of CICS region not using OTE    X
      TCPADDR=TCPIP,    Job/Step name for TCP/IP               X
      NTASKS=20,        Number of subtasks                     X
      DPRTY=0,          Subtask dispatch priority difference   X
      CACHMIN=15,       Minimum refresh time for cache         X
      CACHMAX=30,       Maximum refresh time for cache         X
      CACHRES=10,       Maximum number of resident resolvers   X
      ERRORTD=CSMT,     Transient data queue for error msgs    X
      TCBLIM=0,         Open API TCB Limit                     X
      OTE=NO,           Open Transaction Environment           X
      TRACE=NO,         No CICS Trace records                  X
      SMSGSUP=NO        STARTED Messages Suppressed?
EZACICD TYPE=CICS,      CICS record definition                 X
      APPLID=CICSPRDB,  APPLID of CICS region using OTE        X
      TCPADDR=TCPIP,    Job/Step name for TCP/IP               X
      CACHMIN=15,       Minimum refresh time for cache         X
      CACHMAX=30,       Maximum refresh time for cache         X
      CACHRES=10,       Maximum number of resident resolvers   X
      ERRORTD=CSMT,     Transient data queue for error msgs    X
      TCBLIM=12,        Open API TCB Limit                     X
      OTE=YES,          Open Transaction Environment           X
      TRACE=NO,         No CICS Trace records                  X
      SMSGSUP=NO        STARTED Messages Suppressed?
EZACICD TYPE=LISTENER macro,  Listener record definition             X
      FORMAT=STANDARD,  Standard listener                      X
      APPLID=CICSPROD,  Applid of CICS region                  X
      TRANID=CSKL,      Transaction name for listener          X
      PORT=3010,        Port number for listener               X
      IMMED=YES,        Listener starts up at initialization?  X
      BACKLOG=20,       Backlog value for listener             X
      NUMSOCK=50,       # of sockets supported by listener     X
      MINMSGL=4,        Minimum input message length           X
      ACCTIME=30,       Timeout value for Accept               X
      GIVTIME=30,       Timeout value for Givesocket           X
      REATIME=30,       Timeout value for Read                 X
      TRANTRN=YES,      Is TRANUSR=YES conditional?            X
      TRANUSR=YES,      Translate user data?                   X
      SECEXIT=EZACICSE, Name of security exit program          X
      WLMGN1=WLMGRP01,  WLM group name 1                       X
      WLMGN2=WLMGRP02,  WLM group name 2                       X
      WLMGN3=WLMGRP03   WLM group name 3
EZACICD TYPE=LISTENER macro,  Listener record definition             X
      FORMAT=ENHANCED,  Enhanced listener                      X
      APPLID=CICSPROD,  Applid of CICS region                  X
      TRANID=CSKM,      Transaction name for listener          X
      PORT=3011,        Port number for listener               X
      IMMED=YES,        Listener starts up at initialization?  X
      BACKLOG=20,       Backlog value for listener             X
      NUMSOCK=50,       # of sockets supported by listener     X
      ACCTIME=30,       Timeout value for Accept               X
      GIVTIME=30,       Timeout value for Givesocket           X
      REATIME=30,       Timeout value for Read                 X
      CSTRAN=TRN1,      Name of child IPv4 server transaction  X
      CSSTTYP=KC,       Child server startup type              X
      CSDELAY=000000,   Child server delay interval            X
      MSGLEN=0,         Length of input message                X
```

Figure 41. Example of JCL to define a configuration file (Part 2 of 4)

```
      PEEKDAT=NO,       Peek option                            X
      MSGFORM=ASCII,    Output message format                  X
      SECEXIT=EZACICSE, Name of security exit program          X
      WLMGN1=WLMGRP04,  WLM group name 1                       X
      WLMGN2=WLMGRP05,  WLM group name 2                       X
      WLMGN3=WLMGRP06   WLM group name 3
EZACICD TYPE=LISTENER macro,  Listener record definition             X
      FORMAT=STANDARD,  Standard listener                      X
      APPLID=CICSPRDB,  Applid of CICS region                  X
      TRANID=CS6L,      Transaction name for listener          X
      PORT=3012,        Port number for listener               X
      AF=INET6,         Listener Address Family                X
      IMMED=YES,        Listener starts up at initialization?  X
      BACKLOG=20,       Backlog value for listener             X
      NUMSOCK=50,       # of sockets supported by listener     X
      MINMSGL=4,        Minimum input message length           X
      ACCTIME=30,       Timeout value for Accept               X
      GIVTIME=30,       Timeout value for Givesocket           X
      REATIME=30,       Timeout value for Read                 X
      TRANTRN=YES,      Is TRANUSR=YES conditional?            X
      TRANUSR=YES,      Translate user data?                   X
      SECEXIT=EZACICSE, Name of security exit program          X
      WLMGN1=WLMGRP01,  WLM group name 1                       X
      WLMGN2=WLMGRP02,  WLM group name 2                       X
      WLMGN3=WLMGRP03   WLM group name 3
EZACICD TYPE=LISTENER,  Listener record definition             X
      FORMAT=ENHANCED,  Enhanced listener                      X
      APPLID=CICSPRDB,  Applid of CICS region                  X
      TRANID=CS6M,      Transaction name for listener          X
      PORT=3013,        Port number for listener               X
      AF=INET6,         Listener Address Family                X
      IMMED=YES,        Listener starts up at initialization?  X
      BACKLOG=20,       Backlog value for listener             X
      NUMSOCK=50,       # of sockets supported by listener     X
      ACCTIME=30,       Timeout value for Accept               X
      GIVTIME=30,       Timeout value for Givesocket           X
      REATIME=30,       Timeout value for Read                 X
      CSTRAN=TRN6,      Name of child IPv6 server transaction  X
      CSSTTYP=KC,       Child server startup type              X
      CSDELAY=000000,   Child server delay interval            X
      MSGLEN=0,         Length of input message                X
      PEEKDAT=NO,       Peek option                            X
      MSGFORM=ASCII,    Output message format                  X
      SECEXIT=EZACICSE, Name of security exit program          X
      WLMGN1=WLMGRP04,  WLM group name 1                       X
      WLMGN2=WLMGRP05,  WLM group name 2                       X
      WLMGN3=WLMGRP06   WLM group name 3
EZACICD TYPE=FINAL      End of assembly input
```

Figure 41. Example of JCL to define a configuration file (Part 3 of 4)

```
/*
//*
//* THIS STEP LINKS THE INITIALIZATION PROGRAM
//*
//LINK   EXEC PGM=IEWL,PARM='LIST,MAP,XREF',
//            REGION=512K,COND=(4,LT)
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD SPACE=(CYL,(5,1)),DISP=(NEW,PASS),UNIT=SYSDA
//SYSLMOD  DD DSNAME=&&LOADSET(EZACICDF),DISP=(MOD,PASS),UNIT=SYSDA,

//            SPACE=(TRK,(1,1,1)),
//            DCB=(DSORG=PO,RECFM=U,BLKSIZE=32760)
//SYSLIN   DD DSNAME=&&OBJSET,DISP=(MOD,PASS)

  NAME EZACICDF(R)
//*
//* THIS STEP EXECUTES THE INITIALIZATION PROGRAM
//*
//FILELOAD EXEC PGM=EZACICDF,COND=(4,LT)
//STEPLIB    DD DSN=&&LOADSET,DISP=(MOD,PASS)

//EZACICDF   DD DSNAME=hlq.EZACONFG,DISP=OLD
```

Figure 41. Example of JCL to define a configuration file (Part 4 of 4)

Customizing the configuration data set

There is a CICS object for each CICS that uses the TCP/IP socket interface and is controlled by the configuration file. The CICS object is identified by the APPLID of the CICS it references.

There is a listener object for each listener defined for a CICS. It is possible that a CICS does not have a listener, but this is not common practice. A CICS can have multiple listeners that are either multiple instances of the supplied listener with different specifications, multiple user-written listeners, or some combination.

Configuration transaction (EZAC)

The EZAC transaction is a panel-driven interface that lets you add, delete, or modify the configuration file. The following table lists and describes the functions supported by the EZAC transaction.

Modifying data sets: You can use the EZAC transaction to modify the configuration data set while CICS is running.

Table 7. Functions supported by the EZAC transaction

Command	Object	Function
ALTER	CICS/listener	Modifies the attributes of an existing resource definition
CONVERT	Listener	Converts listener from the standard listener that requires the standard header to the enhanced listener that does not require the header.
COPY	CICS/listener	• CICS - Copies the CICS object and its associated listeners to create another CICS object. COPY fails if the new CICS object already exists. • Listener - Copies the listener object to create another listener object. COPY fails if the new listener object already exists.
DEFINE	CICS/listener	Creates a new resource definition
DELETE	CICS/listener	• CICS - Deletes the CICS object and all of its associated listeners. • Listener - Deletes the listener object.
DISPLAY	CICS/listener	Shows the parameters specified for the CICS/listener object.
RENAME	CICS/listener	Performs a COPY followed by a DELETE of the original object.

If you enter EZAC, the following screen is displayed:

```
EZAC,                                                        APPLID = ........

Enter One of the Following

ALTer
CONvert
COPy
DEFine
DELete
DISplay
REName




PF 3 END                                                           12 CNCL
```

Figure 42. EZAC initial screen

ALTER function: The ALTER function is used to change CICS objects or their listener objects. If you specify ALter on the EZAC Initial Screen or enter EZAC,ALT on a blank screen, the following screen is displayed:

```
EZAC,ALTer,                                                   APPLID = ........

Enter One of the Following

CICS
LISTENER






















PF 3 END                                                            12 CNCL
```

Figure 43. EZAC,ALTER screen

Note: You can skip this screen by entering either EZAC,ALTER,CICS or EZAC,ALTER,LISTENER.

ALTER,CICS: For alteration of a CICS object, the following screen is displayed:

```
EZAC,ALTer,CICS                                               APPLID = ........

Enter all fields




APPLID       ===> ........             APPLID of CICS System


















PF 3 END                                                            12 CNCL
```

Figure 44. EZAC,ALTER,CICS screen

After the APPLID is entered, the following screen is displayed:

```
EZAC,ALTer,CICS                                              APPLID = ........

Overtype to Enter

APPLID       ===> ........             APPLID of CICS System
TCPADDR      ===> ........             Name of TCP Address Space
NTASKS       ===> ...                  Number of Reusable Tasks
DPRTY        ===> ...                  DPRTY Value for ATTACH
CACHMIN      ===> ...                  Minimum Refresh Time for Cache
CACHMAX      ===> ...                  Maximum Refresh Time for Cache
CACHRES      ===> ...                  Maximum Number of Resolvers
ERRORTD      ===> ....                 TD Queue for Error Messages
SMSGSUP      ===> ...                  Suppress Task Started Messages
TERMLIM      ===> ...                  Subtask Termination Limit
TRACE        ===> ...                  Trace CICS Sockets
OTE          ===> ...                  Open Transaction Environment
TCBLIM       ===> .....                Number of open API TCBs
PLTSDI       ===> ...                  CICS PLT Shutdown Immediate
APPLDAT      ===> ...                  Register Application Data

Press ENTER or PF3 to exit


PF 3 END                                                           12 CNCL
```

Figure 45. EZAC,ALTER,CICS detail screen

The system requests a confirmation of the values displayed. After the changes are confirmed, the changed values are in effect for the next initialization of the CICS sockets interface.

ALTER,LISTENER: For alteration of a listener, the following screen is displayed:

```
EZAC,ALTer,LISTENER                                          APPLID = ........

Enter all fields



APPLID       ===> ........             APPLID of CICS System

TRANID       ===> ....                 Transaction Name of listener










PF 3 END                                                           12 CNCL
```

Figure 46. EZAC,ALTER,LISTENER screen

If you are altering a standard listener, the first screen shows the attributes of the standard listener:

```
EZAC,ALTer,LISTENER (standard listener.  screen 1 of 2)      APPLID = ........

Overtype to Enter

APPLID       ===> ........             APPLID of CICS System
TRANID       ===> ....                 Transaction Name of listener
PORT         ===> .....                Port Number of listener
AF           ===> .....                Listener Address Family
IMMEDIATE    ===> ...                  Immediate Startup   Yes|No
BACKLOG      ===> ...                  Backlog Value for listener
NUMSOCK      ===> ...                  Number of Sockets in listener
ACCTIME      ===> ...                  Timeout Value for ACCEPT
GIVTIME      ===> ...                  Timeout Value for GIVESOCKET
REATIME      ===> ...                  Timeout Value for READ
RTYTIME      ===> ...                  Stack Connection Retry Time
LAPPLD       ===> ...                  Register Application Data




Verify parameters, press PF8 to go to screen 2
                   or ENTER if finished making changes

PF 3 END                         8 NEXT                          12 CNCL
```

Figure 47. EZAC,ALTER,LISTENER detail screen 1- Standard listener

Pressing PF8 displays the screen used to manage the unique attributes of the standard listener

```
EZAC,ALTer,LISTENER (standard listener.  screen 2 of 2)      APPLID = ........

Overtype to Enter


MINMSGL      ===> ...                  Minimum Message Length
TRANTRN      ===> ...                  Translate TRNID     Yes|No
TRANUSR      ===> ...                  Translate User Data Yes|No
SECEXIT      ===> ........             Name of Security Exit
GETTID       ===> ...                  Get TTLS ID  (YES|NO)
USERID       ===> ........             Listeners User ID
WLM group 1  ===> ............         Workload Manager Group Name 1
WLM group 2  ===> ............         Workload Manager Group Name 2
WLM group 3  ===> ............         Workload Manager Group Name 3






Verify parameters, press PF7 to go back to screen 1
                   or ENTER if finished making changes

PF 3 END           7 PREV                                        12 CNCL
```

Figure 48. EZAC,ALTER,LISTENER detail screen 2- Standard listener

Pressing PF7 displays the screen used to manage the common attributes of the standard listener.

If altering an enhanced listener, then the first screen shows the attributes of the enhanced listener.

```
EZAC,ALTer,LISTENER (enhanced listener.  screen 1 of 2)      APPLID = ........

Overtype to Enter

APPLID       ===> ........            APPLID of CICS System
TRANID       ===> ....                Transaction Name of listener
PORT         ===> .....               Port Number of listener
AF           ===> .....               Listener Address Family
IMMEDIATE    ===> ...                 Immediate Startup   Yes|No
BACKLOG      ===> ...                 Backlog Value for listener
NUMSOCK      ===> ...                 Number of Sockets in listener
ACCTIME      ===> ...                 Timeout Value for ACCEPT
GIVTIME      ===> ...                 Timeout Value for GIVESOCKET
REATIME      ===> ...                 Timeout Value for READ
RTYTIME      ===> ...                 Stack Connection Retry Time
LAPPLD       ===> ...                 Register Application Data




Verify parameters, press PF8 to go to screen 2


PF 3 END                           8 NEXT                          12 CNCL
```

Figure 49. EZAC,ALTER,LISTENER detail screen 1- Enhanced listener

Pressing PF8 displays the screen used to manage the unique attributes of the enhanced listener.

```
EZAC,ALTer,LISTENER (enhanced listener.  screen 2 of 2)      APPLID = ........

Overtype to Enter

CSTRANid     ===> ....                Child Server Transaction Name
CSSTTYPe     ===> ..                  Startup Method (KC|IC|TD)
CSDELAY      ===> ......              Delay Interval (hhmmss)
MSGLENgth    ===> ...                 Message Length (0-999)
PEEKDATa     ===> ...                 Enter Y|N
MSGFORMat    ===> ......              Enter ASCII|EBCDIC
USEREXIT     ===> ........            Name of User/Security exit
GETTID       ===> ...                 Get TTLS ID  (YES|NO)
USERID       ===> ........            Listeners User ID
WLM group 1  ===> ............        Workload Manager Group Name 1
WLM group 2  ===> ............        Workload Manager Group Name 2
WLM group 3  ===> ............        Workload Manager Group Name 3




Verify parameters, press PF7 to go back to screen 1
                   or ENTER if finished making changes

PF 3 END          7 PREV                                           12 CNCL
```

Figure 50. EZAC,ALTER,LISTENER detail screen 2- Enhanced listener

Pressing PF7 displays the screen used to manage the common attributes of the enhanced listener.

The system requests a confirmation of the values displayed. After the changes are confirmed, the changed values is in effect for the next initialization of the CICS sockets interface.

CONVERT function: The CONVERT function is used to convert between standard and enhanced versions of the listener. If you specify CONvert on the EZAC Initial Screen or enter EZAC,CON on a blank screen, the following screen is displayed:

```
EZAC,CONvert,LISTENER                                        APPLID = ........

Enter all fields



APPLID       ===> ........             APPLID of CICS System

TRANID       ===> ....                 Transaction Name of listener

Format       ===> STANDARD             Enter STANDARD|ENHANCED









PF 3 END                                                           12 CNCL
```

Figure 51. EZAC,CONVERT,LISTENER screen

After the names and format type are entered, one of the following two screens is displayed. The first screen is displayed for the standard version:

If converting to a standard listener, then the first screen shows the attributes of the standard listener.

```
EZAC,CONvert,LISTENER (standard listener.  screen 1 of 2)    APPLID = ........

Overtype to Enter

APPLID       ===> ........             APPLID of CICS System
TRANID       ===> ....                 Transaction Name of listener
PORT         ===> .....                Port Number of listener
AF           ===> .....                Listener Address Family
IMMEDIATE    ===> ...                  Immediate Startup   Yes|No
BACKLOG      ===> ...                  Backlog Value for listener
NUMSOCK      ===> ...                  Number of Sockets in listener
ACCTIME      ===> ...                  Timeout Value for ACCEPT
GIVTIME      ===> ...                  Timeout Value for GIVESOCKET
REATIME      ===> ...                  Timeout Value for READ
RTYTIME      ===> ...                  Stack Connection Retry Time
LAPPLD       ===> ...                  Register Application Data




Verify parameters, press PF8 to go to screen 2


PF 3 END                          8 NEXT                           12 CNCL
```

Figure 52. EZAC,CONVERT,LISTENER detail screen 1- Standard listener

Pressing PF8 displays the screen used to manage the unique attributes of the standard listener.

```
EZAC,CONvert,LISTENER (standard listener.  screen 2 of 2)    APPLID = ........

Overtype to Enter


MINMSGL      ===> ...                Minimum Message Length
TRANTRN      ===> ...                Translate TRNID     Yes|No
TRANUSR      ===> ...                Translate User Data Yes|No
SECEXIT      ===> ........           Name of Security Exit
GETTID       ===> ...                Get TTLS ID  (YES|NO)
USERID       ===> ........           Listeners User ID
WLM group 1  ===> ............       Workload Manager Group Name 1
WLM group 2  ===> ............       Workload Manager Group Name 2
WLM group 3  ===> ............       Workload Manager Group Name 3







Verify parameters, press PF7 to go back to screen 1
                   or ENTER if finished making changes

PF 3 END           7 PREV                                 12 CNCL
```

Figure 53. EZAC,CONVERT,LISTENER detail screen 2- Standard listener

Pressing PF7 displays the screen used to manage the common attributes of the standard listener.

If converting to an enhanced listener, the first screen shows the attributes of the enhanced listener.

```
EZAC,CONvert,LISTENER (enhanced listener.  screen 1 of 2)    APPLID = ........

Overtype to Enter

APPLID       ===> ........           APPLID of CICS System
TRANID       ===> ....               Transaction Name of listener
PORT         ===> .....              Port Number of listener
AF           ===> .....              Listener Address Family
IMMEDIATE    ===> ...                Immediate Startup   Yes|No
BACKLOG      ===> ...                Backlog Value for listener
NUMSOCK      ===> ...                Number of Sockets in listener
ACCTIME      ===> ...                Timeout Value for ACCEPT
GIVTIME      ===> ...                Timeout Value for GIVESOCKET
REATIME      ===> ...                Timeout Value for READ
RTYTIME      ===> ...                Stack Connection Retry Time
LAPPLD       ===> ...                Register Application Data




Verify parameters, press PF8 to go to screen 2


PF 3 END                        8 NEXT                    12 CNCL
```

Figure 54. EZAC,CONVERT,LISTENER detail screen 1- Enhanced listener

Pressing PF8 displays the screen used to manage the unique attributes of the enhanced listener

```
EZAC,CONvert,LISTENER (enhanced listener.  screen 2 of 2)    APPLID = ........

Overtype to Enter

CSTRANid     ===> ....                Child Server Transaction Name
CSSTTYPe     ===> ..                  Startup Method (KC|IC|TD)
CSDELAY      ===> ......              Delay Interval (hhmmss)
MSGLENgth    ===> ...                 Message Length (0-999)
PEEKDATa     ===> ...                 Enter Y|N
MSGFORMat    ===> ......              Enter ASCII|EBCDIC
USEREXIT     ===> ........            Name of User/Security exit
GETTID       ===> ...                 Get TTLS ID  (YES|NO)
USERID       ===> ........            Listeners User ID
WLM group 1  ===> ............        Workload Manager Group Name 1
WLM group 2  ===> ............        Workload Manager Group Name 2
WLM group 3  ===> ............        Workload Manager Group Name 3


Verify parameters, press PF7 to go back to screen 1
                   or ENTER if finished making changes

PF 3 END           7 PREV                                     12 CNCL
```

Figure 55. EZAC,CONVERT,LISTENER detail screen 2- Enhanced listener

Pressing PF7 displays the screen used to manage the common attributes of the enhanced listener.

The system requests a confirmation of the values displayed. After the changes are confirmed, the changed values are in effect for the next initialization of the CICS sockets interface.

COPY function: The COPY function is used to copy an object into a new object. If you specify COPy on the EZAC Initial Screen or enter EZAC,COP on a blank screen, the following screen is displayed:

```
EZAC,COPy,                                                   APPLID = ........

Enter One of the Following

CICS
LISTENER




PF 3 END                                                      12 CNCL
```

Figure 56. EZAC,COPY screen

Note: You can skip this screen by entering either EZAC,COPY,CICS or EZAC,COPY,LISTENER.

COPY,CICS: If you specify CICS on the previous screen, the following screen is displayed:

```
EZAC,COPy,CICS                                              APPLID = ........

Enter all fields



SCICS        ===> ........            APPLID of Source CICS

TCICS        ===> ........            APPLID of Target CICS














PF 3 END                                                          12 CNCL
```

Figure 57. EZAC,COPY,CICS screen

After the APPLIDs of the source CICS object and the target CICS object are entered, confirmation is requested. When confirmation is entered, the copy is performed.

COPY,LISTENER: If you specify COPY,LISTENER, the following screen is displayed:

```
EZAC,COPy,LISTENER                                          APPLID = ........

Enter all fields



SCICS        ===> ........            APPLID of Source CICS

SLISTENER    ===> ....                Name of Source listener



TCICS        ===> ........            APPLID of Target CICS

TLISTENER    ===> ....                Name of Target listener







PF 3 END                                                          12 CNCL
```

Figure 58. EZAC,COPY,LISTENER screen

After the APPLIDs of the source and target CICS objects and the names of the source and target listeners are entered, confirmation is requested. When the confirmation is entered, the copy is performed.

DEFINE function: The DEFINE function is used to create CICS objects and their listener objects. If you specify DEFine on the EZAC Initial Screen or enter EZAC,DEF on a blank screen, the following screen is displayed:

```
EZAC,DEFine,                                                APPLID = ........

Enter One of the Following

CICS
LISTENER










PF 3 END                                                          12 CNCL
```

Figure 59. EZAC,DEFINE screen

Note: You can skip this screen by entering either EZAC,DEFINE,CICS or EZAC,DEFINE,LISTENER.

DEFINE,CICS: For definition of a CICS object, the following screen is displayed:

```
EZAC,DEFine,CICS                                            APPLID = ........

Enter all fields



APPLID       ===> ........              APPLID of CICS System










PF 3 END                                                          12 CNCL
```

Figure 60. EZAC,DEFINE,CICS screen

After the APPLID is entered, the following screen is displayed.

```
EZAC,DEFine,CICS                                            APPLID = ........

Overtype to Enter

APPLID       ===> ........            APPLID of CICS System
TCPADDR      ===> ........            Name of TCP Address Space
NTASKS       ===> ...                 Number of Reusable Tasks
DPRTY        ===> ...                 DPRTY Value for ATTACH
CACHMIN      ===> ...                 Minimum Refresh Time for Cache
CACHMAX      ===> ...                 Maximum Refresh Time for Cache
CACHRES      ===> ...                 Maximum Number of Resolvers
ERRORTD      ===> ....                TD Queue for Error Messages
SMSGSUP      ===> ...                 Suppress Task Started Messages
TERMLIM      ===> ...                 Subtask Termination Limit
TRACE        ===> ...                 Trace CICS Sockets
OTE          ===> ...                 Open Transaction Environment
TCBLIM       ===> .....               Number of open API TCBs
PLTSDI       ===> ...                 CICS PLT Shutdown Immediate
APPLDAT      ===> ...                 Register Application Data

Press ENTER or PF3 to exit


PF 3 END                                                          12 CNCL
```

Figure 61. EZAC,DEFINE,CICS detail screen

After the definition is entered, confirmation is requested. When confirmation is entered, the object is created on the configuration file.

DEFINE,LISTENER: For definition of a listener, the following screen is displayed:

```
EZAC,DEFine,LISTENER                                        APPLID = ........

Enter all fields



APPLID       ===> ........            APPLID of CICS System

TRANID       ===> ....                Transaction Name of listener

Format       ===> ........            Enter STANDARD|ENHANCED










PF 3 END                                                          12 CNCL
```

Figure 62. EZAC,DEFINE,LISTENER screen

If defining a standard listener, the first screen shows the attributes of the standard listener.

```
EZAC,DEFine,LISTENER (standard listener.  screen 1 of 2)     APPLID = ........

Overtype to Enter

APPLID       ===> ........              APPLID of CICS System
TRANID       ===> ....                  Transaction Name of listener
PORT         ===> .....                 Port Number of listener
AF           ===> .....                 Listener Address Family
IMMEDIATE    ===> ...                   Immediate Startup   Yes|No
BACKLOG      ===> ...                   Backlog Value for listener
NUMSOCK      ===> ...                   Number of Sockets in listener
ACCTIME      ===> ...                   Timeout Value for ACCEPT
GIVTIME      ===> ...                   Timeout Value for GIVESOCKET
REATIME      ===> ...                   Timeout Value for READ
RTYTIME      ===> ...                   Stack Connection Retry Time
LAPPLD       ===> ...                   Register Application Data




Verify parameters, press PF8 to go to screen 2


PF 3 END                          8 NEXT                          12 CNCL
```

Figure 63. EZAC,DEFINE,LISTENER detail screen 1- Standard listener

Pressing PF8 displays the screen used to manage the unique attributes of the standard listener.

```
EZAC,DEFine,LISTENER (standard listener.  screen 2 of 2)     APPLID = ........

Overtype to Enter


MINMSGL      ===> ...                   Minimum Message Length
TRANTRN      ===> ...                   Translate TRNID     Yes|No
TRANUSR      ===> ...                   Translate User Data Yes|No
SECEXIT      ===> ........              Name of Security Exit
GETTID       ===> ...                   Get TTLS ID  (YES|NO)
USERID       ===> ........              Listeners User ID
WLM group 1  ===> ............          Workload Manager Group Name 1
WLM group 2  ===> ............          Workload Manager Group Name 2
WLM group 3  ===> ............          Workload Manager Group Name 3






Verify parameters, press PF7 to go back to screen 1
                   or ENTER if finished making changes

PF 3 END           7 PREV                                         12 CNCL
```

Figure 64. EZAC,DEFINE,LISTENER detail screen 2- Standard listener

Pressing PF7 displays the screen used to manage the common attributes of the standard listener.

If defining an enhanced listener, the first screen shows the attributes of the enhanced listener.

```
EZAC,DEFine,LISTENER (enhanced listener.  screen 1 of 2)     APPLID = ........

Overtype to Enter

APPLID       ===> ........            APPLID of CICS System
TRANID       ===> ....                Transaction Name of listener
PORT         ===> .....               Port Number of listener
AF           ===> .....               Listener Address Family
IMMEDIATE    ===> ...                 Immediate Startup   Yes|No
BACKLOG      ===> ...                 Backlog Value for listener
NUMSOCK      ===> ...                 Number of Sockets in listener
ACCTIME      ===> ...                 Timeout Value for ACCEPT
GIVTIME      ===> ...                 Timeout Value for GIVESOCKET
REATIME      ===> ...                 Timeout Value for READ
RTYTIME      ===> ...                 Stack Connection Retry Time
LAPPLD       ===> ...                 Register Application Data




Verify parameters, press PF8 to go to screen 2


PF 3 END                         8 NEXT                          12 CNCL
```

Figure 65. EZAC,DEFINE,LISTENER detail screen 1- Enhanced listener

Pressing PF8 displays the screen used to manage the unique attributes of the enhanced listener

```
EZAC,DEFine,LISTENER (enhanced listener.  screen 2 of 2)     APPLID = ........

Overtype to Enter

CSTRANid     ===> ....                Child Server Transaction Name
CSSTTYPe     ===> ..                  Startup Method (KC|IC|TD)
CSDELAY      ===> ......              Delay Interval (hhmmss)
MSGLENgth    ===> ...                 Message Length (0-999)
PEEKDATa     ===> ...                 Enter Y|N
MSGFORMat    ===> ......              Enter ASCII|EBCDIC
USEREXIT     ===> ........            Name of User/Security exit
GETTID       ===> ...                 Get TTLS ID  (YES|NO)
USERID       ===> ........            Listeners User ID
WLM group 1  ===> ............        Workload Manager Group Name 1
WLM group 2  ===> ............        Workload Manager Group Name 2
WLM group 3  ===> ............        Workload Manager Group Name 3




Verify parameters, press PF7 to go back to screen 1
                    or ENTER if finished making changes

PF 3 END          7 PREV                                         12 CNCL
```

Figure 66. EZAC,DEFINE,LISTENER detail screen 2- Enhanced listener

Pressing PF7 displays the screen used to manage the common attributes of the enhanced listener.

After the definition is entered, confirmation is requested. When confirmation is entered, the object is created on the configuration file.

DELETE function: The DELETE function is used to delete a CICS object or a listener object. Deleting a CICS object deletes all listener objects within that CICS object. If you specify DELete on the EZAC initial screen or enter EZAC,DEL on a blank screen, the following screen is displayed:

```
EZAC,DELete,                                            APPLID = ........

Enter One of the Following

CICS
LISTENER












PF 3 END                                                     12 CNCL
```

Figure 67. EZAC,DELETE screen

DELETE,CICS: If you specify DELETE,CICS, the following screen is displayed:

```
EZAC,DELete,CICS                                        APPLID = ........

Enter all fields



APPLID       ===> ........            APPLID of CICS System












PF 3 END                                                     12 CNCL
```

Figure 68. EZAC,DELETE,CICS screen

After the APPLID is entered, confirmation is requested. When the confirmation is entered, the CICS object is deleted.

DELETE,LISTENER: If you specify DELETE,LISTENER, the following screen is displayed:

```
EZAC,DELete,LISTENER                                          APPLID = ........

Enter all fields



APPLID       ===> ........            APPLID of CICS System

TRANID       ===> ....                Transaction Name of listener











PF 3 END                                                            12 CNCL
```

Figure 69. EZAC,DELETE,LISTENER screen

After the APPLID and listener name are entered, confirmation is requested. When confirmation is entered, the listener object is deleted

DISPLAY function: The DISPLAY function is used to display the specification of an object. If you specify DISplay on the initial EZAC screen or enter EZAC,DIS on a blank screen, the following screen is displayed:

```
EZAC,DISplay,                                                 APPLID = ........

Enter One of the Following

CICS
LISTENER

















PF 3 END                                                            12 CNCL
```

Figure 70. EZAC,DISPLAY screen

Note: You can skip this screen by entering either EZAC,DISPLAY,CICS or EZAC,DISPLAY,LISTENER.

DISPLAY,CICS: If you specify DISPLAY,CICS, the following screen is displayed:

```
EZAC,DISplay,CICS                                            APPLID = ........

Enter all fields



APPLID       ===> ........            APPLID of CICS System












PF 3 END                                                           12 CNCL
```

Figure 71. EZAC,DISPLAY,CICS screen

After the APPLID is entered, the following screen is displayed:

```
EZAC,DISplay,CICS                                            APPLID = ........



APPLID       ===> ........            APPLID of CICS System
TCPADDR      ===> ........            Name of TCP Address Space
NTASKS       ===> ...                 Number of Reusable Tasks
DPRTY        ===> ...                 DPRTY Value for ATTACH
CACHMIN      ===> ...                 Minimum Refresh Time for Cache
CACHMAX      ===> ...                 Maximum Refresh Time for Cache
CACHRES      ===> ...                 Maximum Number of Resolvers
ERRORTD      ===> ....                TD Queue for Error Messages
SMSGSUP      ===> ...                 Suppress Task Started Messages
TERMLIM      ===> ...                 Subtask Termination Limit
TRACE        ===> ...                 Trace CICS Sockets
OTE          ===> ...                 Open Transaction Environment
TCBLIM       ===> .....               Number of open API TCBs
PLTSDI       ===> ...                 CICS PLT Shutdown Immediate
APPLDAT      ===> ...                 Register Application Data

Press ENTER or PF3 to exit


PF 3 END                                                           12 CNCL
```

Figure 72. EZAC,DISPLAY,CICS detail screen

DISPLAY,LISTENER: If you specify DISPLAY,LISTENER, the following screen is displayed:

```
EZAC,DISplay,LISTENER                                       APPLID = ........

Enter all fields



APPLID       ===> ........            APPLID of CICS System

TRANID       ===> ....                Transaction Name of listener










PF 3 END                                                          12 CNCL
```

Figure 73. EZAC,DISPLAY,LISTENER screen

If displaying a standard listener, the first screen shows the attributes of the standard listener.

```
EZAC,DISplay,LISTENER (standard listener.  screen 1 of 2)    APPLID = ........



APPLID       ===> ........            APPLID of CICS System
TRANID       ===> ....                Transaction Name of listener
PORT         ===> .....               Port Number of listener
AF           ===> .....               Listener Address Family
IMMEDIATE    ===> ...                 Immediate Startup   Yes|No
BACKLOG      ===> ...                 Backlog Value for listener
NUMSOCK      ===> ...                 Number of Sockets in listener
ACCTIME      ===> ...                 Timeout Value for ACCEPT
GIVTIME      ===> ...                 Timeout Value for GIVESOCKET
REATIME      ===> ...                 Timeout Value for READ
RTYTIME      ===> ...                 Stack Connection Retry Time
LAPPLD       ===> ...                 Register Application Data




Verify parameters, press PF8 to go to screen 2


PF 3 END                        8 NEXT                            12 CNCL
```

Figure 74. EZAC,DISPLAY,LISTENER detail screen 1- Standard listener

Pressing PF8 displays the screen used to manage the unique attributes of the standard listener.

```
EZAC,DISplay,LISTENER (standard listener.  screen 2 of 2)    APPLID = ........



MINMSGL      ===> ...                  Minimum Message Length
TRANTRN      ===> ...                  Translate TRNID     Yes|No
TRANUSR      ===> ...                  Translate User Data Yes|No
SECEXIT      ===> ........             Name of Security Exit
GETTID       ===> ...                  Get TTLS ID  (YES|NO)
USERID       ===> ........             Listeners User ID
WLM group 1  ===> ............         Workload Manager Group Name 1
WLM group 2  ===> ............         Workload Manager Group Name 2
WLM group 3  ===> ............         Workload Manager Group Name 3






Verify parameters, press PF7 to go back to screen 1
Press ENTER or PF3 to exit

PF 3 END          7 PREV                                      12 CNCL
```

Figure 75. EZAC,DISPLAY,LISTENER detail screen 2- Standard listener

Pressing PF7 displays the screen used to manage the common attributes of the standard listener.

If displaying an enhanced listener, the first screen shows the attributes of the enhanced listener.

```
EZAC,DISplay,LISTENER (enhanced listener.  screen 1 of 2)    APPLID = ........



APPLID       ===> ........             APPLID of CICS System
TRANID       ===> ....                 Transaction Name of listener
PORT         ===> .....                Port Number of listener
AF           ===> .....                Listener Address Family
IMMEDIATE    ===> ...                  Immediate Startup   Yes|No
BACKLOG      ===> ...                  Backlog Value for listener
NUMSOCK      ===> ...                  Number of Sockets in listener
ACCTIME      ===> ...                  Timeout Value for ACCEPT
GIVTIME      ===> ...                  Timeout Value for GIVESOCKET
REATIME      ===> ...                  Timeout Value for READ
RTYTIME      ===> ...                  Stack Connection Retry Time
LAPPLD       ===> ...                  Register Application Data




Verify parameters, press PF8 to go to screen 2


PF 3 END                        8 NEXT                        12 CNCL
```

Figure 76. EZAC,DISPLAY,LISTENER detail screen 1- Enhanced listener

Pressing PF8 displays the screen used to manage the unique attributes of the enhanced listener.

```
EZAC,DISplay,LISTENER (enhanced listener.  screen 2 of 2)    APPLID = ........



CSTRANid     ===> ....                 Child Server Transaction Name
CSSTTYPe     ===> ..                   Startup Method (KC|IC|TD)
CSDELAY      ===> ......               Delay Interval (hhmmss)
MSGLENgth    ===> ...                  Message Length (0-999)
PEEKDATa     ===> ...                  Enter Y|N
MSGFORMat    ===> ......               Enter ASCII|EBCDIC
USEREXIT     ===> ........             Name of User/Security exit
GETTID       ===> ...                  Get TTLS ID  (YES|NO)
USERID       ===> ........             Listeners User ID
WLM group 1  ===> ............         Workload Manager Group Name 1
WLM group 2  ===> ............         Workload Manager Group Name 2
WLM group 3  ===> ............         Workload Manager Group Name 3




Verify parameters, press PF7 to go back to screen 1
 Press ENTER or PF3 to exit

PF 3 END          7 PREV                                      12 CNCL
```

Figure 77. EZAC,DISPLAY,LISTENER detail screen 2- Enhanced listener

RENAME function: The RENAME function is used to rename a CICS or listener object. It consists of a COPY followed by a DELETE of the source object. For a CICS object, the object and all of its associated listeners are renamed. For a listener object, only that listener is renamed.

If you specify REName on the initial EZAC screen or enter EZAC,REN on a blank screen, the following screen is displayed:

```
EZAC,REName,                                                 APPLID = ........

Enter One of the Following

CICS
LISTENER



















PF 3 END                                                      12 CNCL
```

Figure 78. EZAC,RENAME screen

Note: You can skip this screen by entering either EZAC,RENAME,CICS or EZAC,RENAME,LISTENER.

RENAME,CICS: If you specify CICS on the previous screen, the following screen is displayed:

```
EZAC,REName,CICS                                              APPLID = ........

Enter all fields



SCICS       ===> ........            APPLID of Source CICS

TCICS       ===> ........            APPLID of Target CICS









PF 3 END                                                            12 CNCL
```

Figure 79. EZAC,RENAME,CICS screen

After the APPLIDs of the source CICS object and the target CICS object are entered, confirmation is requested. When confirmation is entered, the rename is performed.

RENAME,LISTENER: If you specify RENAME,LISTENER, the following screen is displayed:

```
EZAC,REName,LISTENER                                          APPLID = ........

Enter all fields



SCICS       ===> ........            APPLID of Source CICS

SLISTENER   ===> ....                Name of Source listener


TCICS       ===> ........            APPLID of Target CICS

TLISTENER   ===> ....                Name of Target listener





PF 3 END                                                            12 CNCL
```

Figure 80. EZAC,RENAME,LISTENER screen

After the APPLIDs of the source and target CICS objects and the names of the source and target listeners are entered, confirmation is requested. When the confirmation is entered, the rename is performed.

UNIX Systems Services environment effects on IP CICS sockets

The UNIX Systems Services provides controls on the number of sockets that can be opened concurrently by a single process (in a CICS region). You can use this to limit the number of socket descriptors that a process can have, thereby limiting the amount of CICS and system resources a single process can use at one time.

Two specifications affect this limit:

- The MAXFILEPROC parameter of the BPXPRMxx parmlib member, which specifies a default limit for any process in the system
- FILEPROCMAX specification in the OMVS segment of the SAF profile for the CICS region's userid, which overrides the default; NOFILEPROCMAX can also be specified, which removes this limit

For more information on how MAXFILEPROC affects tuning applications, see *z/OS UNIX System Services Planning*. The z/OS configuration tool, called Managed System Infrastructure (msys), contains additional information about the impacts of the UNIX MAXFILEPROC parameter settings.

For more information about the FILEPROCMAX specification, see the documentation provided for the SAF product in use on your system. If using RACF, this can be found in the *z/OS Security Server RACF Security Administrator's Guide*

Chapter 3. Configuring the CICS Domain Name System cache

The Domain Name System (DNS) is like a telephone book that contains a person's name, address, and telephone number. The name server maps a host name to an IP address, or an IP address to a host name. For each host, the name server can contain IP addresses, nicknames, mailing information, and available well-known services (for example, SMTP, FTP, or Telnet).

Translating host names into IP addresses is just one way of using the DNS. Other types of information related to hosts can also be stored and queried. The different possible types of information are defined through input data to the name server in the resource records.

Although the CICS DNS cache function is optional, it is useful in a highly active CICS client environment. It combines the GETHOSTBYNAME() call supported in CICS sockets and a cache that saves results from the GETHOSTBYNAME() for future reference. If your system receives repeated requests for the same set of domain names, using the DNS improves performance significantly. Your threadsafe program is switched to the QR TCB if you have specified that IP CICS sockets should use the Open Transaction Environment and you link to the Domain Name Service module, EZACIC25. Instead of using this service module to resolve a host name to an address, consider using a caching-only BIND 9 name server on a local system.

If the server intends to use WLM connection balancing, the client should not cache DNS names. Connection balancing relies on up-to-date information about the current capacity of hosts in the sysplex. If DNS names are retrieved from a cache instead of the DNS/WLM name server, connections are made without regard for current host capacity, This degrades the effectiveness of connection balancing. Choosing not to cache names can mean more IP traffic, which in some cases can outweigh the benefits of connection balancing.

See *z/OS Communications Server: IP Configuration Reference* for information about caching issues.

Recommendations for CICS DNS Caching and DNS/WLM support: The following recommendations apply when configuring CICS DNS Caching:

- DNS Caching does not support the caching of IPv6 addresses as the gethostbyname() function is not IPv6 enabled.
- If you require improved performance for Domain Name Server lookups for both IPv4 and IPv6 resources, consider configuring a caching-only BIND 9 name server on the local system. This has the following benefits:
  - After a hostname is resolved, it is cached locally, allowing all other applications running in the system to retrieve this information without incurring the overhead of network communications.
  - A caching domain name server honors the time to live (TTL) value that indicates when a resource record's information should expire.
  - BIND 9 supports caching of both IPv4 and IPv6 resources.
- DNS Caching continues to support the caching of an IPv4 address. You can also start using a DNS BIND 9 caching-only server for both IPv4 and IPv6 name resolution. In this case, IPv6 clients should use unique hostnames and you

should enable DNS entries to allow unique host names to exist in different DNS zones. This enables an IPv6 client to get an AAAA address to use when connecting to an IPv6 enabled listener.

- DNS/WLM continues to support CICS listeners wanting to participate in work load balancing. IPv6 enabled listeners are still able to participate in work load balancing for their IPv4 clients and IPv6 clients.
- DNS/WLM is not possible when using IPv6 addresses because DNS/WLM is supported only on the BIND 4.9.3 server, and BIND 4.9.3 does not support AAAA records.
- The IPv6 client is not able to get an IPv6 address back from DNS/WLM for the IPv6 listener to which they are trying to connect. The IP address from DNS/WLM must be turned into an IPv4-mapped IPv6 address for the IPv6 enabled listener.
- If you want to support IPv6 clients and DNS/WLM (for IPv4 clients), set up a caching-only BIND 9 name server to support both IPv4 and IPv6 addresses and keep your BIND 4.9.3 name server in the sysplex for DNS/WLM support. Have your IPv6-enabled client get the IPv4 address from the DNS/WLM server and then convert the IPv4 address to an IPv4-mapped IPv6 address. Use this address to connect to the IPv6-enabled listener. This is not a true IPv6 connection because DNS/WLM do not give an IPv6 address. Clients that want to connect to the server over an IPv6 network should use an IPv6 address.

Function components

The function consists of three parts.

- A VSAM file which is used for the cache.

  Note: The CICS DATATABLE option can be used with the cache.
- A macro, EZACICR, which is used to initialize the cache file.
- A CICS application program, EZACIC25, which is invoked by the CICS application in place of the GETHOSTBYNAME socket call.

VSAM cache file

The cache file is a VSAM KSDS (Key Sequenced Data Set) with a key of the host name padded to the right with binary zeros. The cache records contain a compressed version of the hostent structure returned by the name server plus a time of last refresh field. When a record is retrieved, EZACIC25 determines if it is usable based on the difference between the current time and the time of last refresh.

EZACICR macro

The EZACICR macro builds an initialization module for the cache file, because the cache file must start with at least one record to permit updates by the EZACIC25 module. To optimize performance, you can preload dummy records for the host names which you expect to be used frequently. This results in a more compact file and minimizes the I/O required to use the cache. If you do not specify at least one dummy record, the macro builds a single record of binary zeros. See “Step 1: Create the initialization module” on page 96.

EZACIC25 module

This module is a normal CICS application program which is invoked by an EXEC CICS LINK command. The COMMAREA passes information between the invoking CICS program and the DNS Module. If domain name resolves successfully, EZACIC25 obtains storage from CICS and builds a hostent structure in that storage. When finished with the hostent structure, release this storage using the EXEC CICS FREEMAIN command.

The EZACIC25 module uses four configuration parameters plus the information passed by the invoking application to manage the cache. These configuration parameters are as follows:

Error destination - ERRORTD
: The Transient Data destination to which error messages are sent.

Minimum refresh time - CACHMIN
: The minimum time in minutes between refreshes of a cache record. If a cache record is younger than this time, it is used. This value is set to 15 minutes.

Maximum refresh time - CACHMAX
: The maximum time in minutes between refreshes of a cache record. If a cache record is older than this time, it is refreshed. This value is set to 30 minutes.

Maximum resolver requests - CACHRES
: The maximum number of concurrent requests to the resolver. It is set at 10. See “How the DNS cache handles requests.”

If the transaction program is executing in the Open Transaction Environment, expect a TCB switch to occur for each call to EZACIC25.

How the DNS cache handles requests

When a request is received where cache retrieval is specified, the following takes place:

1. Attempt to retrieve this entry from the cache. If unsuccessful, issue the GETHOSTBYNAME call unless request specifies cache only.
2. If cache retrieval is successful, calculate the age of the record. This is the difference between the current time and the time this record was created or refreshed.
   - If the age is not greater than minimum cache refresh, use the cache information and build the Hostent structure for the requestor. Then return to the requestor.
   - If the age is greater than the maximum cache refresh, issue the GETHOSTBYNAME call and refresh the cache record with the results.
   - If the age is between the minimum and maximum cache refresh values, do the following:
     a. Calculate the difference between the maximum and minimum cache refresh times and divide it by the maximum number of concurrent resolver requests. The result is called the time increment.
     b. Multiply the time increment by the number of currently active resolver requests. Add this time to the minimum refresh time giving the adjusted refresh time.
     c. If the age of the record is less than the adjusted refresh time, use the cache record.
     d. If the age of the record is greater than the adjusted refresh time, issue the GETHOSTBYNAME call and refresh the cache record with the results.

- If the GETHOSTBYNAME is issued and is successful, the cache is updated and the update time for the entry is changed to the current time.

Using the DNS cache

There are three steps to using the DNS cache.

1. Create the initialization module, which in turn defines and initializes the file and the EZACIC25 module. See "Step 1: Create the initialization module."
2. Define the cache files to CICS. See "Step 2: Define the cache file to CICS" on page 99.
3. Use EZACIC25 to replace GETHOSTBYNAME calls in CICS application modules. See "Step 3: Execute EZACIC25" on page 100.

Step 1: Create the initialization module

The initialization module is created using the EZACICR macro. A minimum of two invocations of the macro are coded and assembled and the assembly produces the module. An example follows:

```
EZACICR TYPE=INITIAL
EZACICR TYPE=FINAL
```

This produces an initialization module which creates one record of binary zeros. If you want to preload the file with dummy records for frequently referenced domain names, it resembles the following:

```
EZACICR TYPE=INITIAL
EZACICR TYPE=RECORD,NAME=HOSTA
EZACICR TYPE=RECORD,NAME=HOSTB
EZACICR TYPE=RECORD,NAME=HOSTC
EZACICR TYPE=FINAL
```

where HOSTA, HOSTB, AND HOSTC are the host names you want in the dummy records. The names can be specified in any order.

The specifications for the EZACICR macro are as follows:

Operand **Meaning**

TYPE There are three acceptable values:

Value	**Meaning**
INITIAL	Indicates the beginning of the generation input. This value should only appear once and should be the first entry in the input stream.
RECORD	Indicates a dummy record the user wants to generate. There can be from 0 to 4096 dummy records generated and each of them must have a unique name. Generating dummy records for frequently used host names improves the performance of the cache file. A TYPE=INITIAL must precede a TYPE=RECORD statement.
FINAL	Indicates the end of the generation input. This value should only appear once and should be the last entry in the input stream. A TYPE=INITIAL must precede a TYPE=FINAL.

AVGREC The length of the average cache record. This value is specified on the TYPE=INITIAL macro and has a default value of 500. It is recommend that you use the default value until you have adequate statistics to determine a better value. This parameter is the same as the first subparameter in the RECORDSIZE parameter of the IDCAMS DEFINE statement. Accurate definition of this parameter along with use of dummy records minimizes control interval and control area splits in the cache file.

NAME
Specifies the host name for a dummy record. The name must be from 1 to 255 bytes long. The NAME operand is required for TYPE=RECORD entries.

The macro can be used in conjunction with IDCAMS to define and load the file. Figure 81 on page 98 shows a sample job to define and initialize a cache file:

```
//*******************************************************//
//*   THE FOLLOWING JOB DEFINES AND THEN LOADS THE VSAM    *//
//*   FILE USED FOR THE CACHE.  THE DEFINITION CONSISTS OF *//
//*   TWO IDCAMS STEPS TO PERFORM THE VSAM DEFINITION      *//
//*   AND A STEP USING EZACICR TO BUILD THE FILE LOAD      *//
//*   PROGRAM. THE FINAL STEP EXECUTES THE FILE LOAD       *//
//*   PROGRAM TO CREATE THE FILE.                          *//
//*******************************************************//
//CACHEDEF  JOB  MSGLEVEL=(1,1)
//*
//* THIS STEP DELETES AN OLD COPY OF THE FILE
//* IF ONE IS THERE.
//*
//DEL    EXEC  PGM=IDCAMS
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
   DELETE -
      CICS.USER.CACHE -
      PURGE -
      ERASE
//*
//*  THIS STEP DEFINES THE NEW FILE
//*
//DEFILE EXEC  PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  DEFINE CLUSTER (NAME(CICS.USER.CACHE) VOLUMES(CICVOL) -
      CYL(1 1) -
      IMBED -
      RECORDSIZE(500 1000) FREESPACE(0 15) -
      INDEXED ) -
      DATA ( -
        NAME(CICS.USER.CACHE.DATA) -
        KEYS (255 0) ) -
      INDEX ( -
        NAME(CICS.USER.CACHE.INDEX) )
/*
//*
//*  THIS STEP DEFINES THE FILE LOAD PROGRAM
//*
//PRGDEF  EXEC PGM=ASMA90,PARM='OBJECT,TERM',REGION=1024K
//SYSLIB    DD DISP=SHR,DSNAME=SYS1.MACLIB
//            DD DISP=SHR,DSNAME=TCPV34.SEZACMAC
//SYSUT1    DD UNIT=SYSDA,SPACE=(CYL,(5,1))
//SYSUT2    DD UNIT=SYSDA,SPACE=(CYL,(2,1))
//SYSUT3    DD UNIT=SYSDA,SPACE=(CYL,(2,1))
//SYSPUNCH  DD DISP=SHR,DSNAME=NULLFILE
//SYSLIN    DD DSNAME=&&OBJSET,DISP=(MOD,PASS),UNIT=SYSDA,
//             SPACE=(400,(500,50)),
//             DCB=(RECFM=FB,BLKSIZE=400,LRECL=80)
//SYSTERM   DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
```

Figure 81. Example of defining and initializing a DNS cache file (Part 1 of 2)

```
//SYSIN     DD *
         EZACICR TYPE=INITIAL
         EZACICR TYPE=RECORD,NAME=RALVM12
         EZACICR TYPE=FINAL
/*
//LINK    EXEC PGM=IEWL,PARM='LIST,MAP,XREF',
//             REGION=512K,COND=(4,LT)
//SYSPRINT  DD SYSOUT=*
//SYSUT1    DD SPACE=(CYL,(5,1)),DISP=(NEW,PASS),UNIT=SYSDA
//SYSLMOD   DD DSNAME=&&LOADSET(GO),DISP=(MOD,PASS),UNIT=SYSDA,
//             SPACE=(TRK,(1,1,1)),
//             DCB=(DSORG=PO,RECFM=U,BLKSIZE=32760)
//SYSLIN    DD DSNAME=&&OBJSET,DISP=(OLD,DELETE)
//*
//*  THIS STEP EXECUTES THE FILE LOAD PROGRAM
//*
//LOAD EXEC PGM=*.LINK.SYSLMOD,COND=((4,LT,ASM),(4,LT,LINK))
//EZACICRF DD DSN=CICS.USER.CACHE,DISP=OLD
```

Figure 81. Example of defining and initializing a DNS cache file (Part 2 of 2)

After the cache file has been created, it has the following layout:

Field name **Description**

Host name
A 255-byte character field specifying the host name. This field is the key to the file.

Record type
A 1-byte binary field specifying the record type. The value is X’00000001’.

Last refresh time
An 8-byte packed field specifying the last refresh time. It is expressed in seconds because 0000 hours on January 1, 1990 and is derived by taking the ABSTIME value obtained from an EXEC CICS ASKTIME and subtracting the value for January 1, 1990.

Offset to alias pointer list
A halfword binary field specifying the offset in the record to DNSALASA.

Number of INET addresses
A halfword binary field specifying the number of INET addresses in DNSINETA.

INET addresses
One or more fullword binary fields specifying INET addresses returned from GETHOSTBYNAME().

Alias names
An array of variable length character fields specifying the alias names returned from the name server cache. These fields are delimited by a byte of binary zeros. Each of these fields have a maximum length of 255 bytes.

Step 2: Define the cache file to CICS

All CICS definitions required to add this function to a CICS system can be done using CICS RDO without disruption to the operation of the CICS system.

Use the following parameters with RDO FILE to define the cache file:

RDO keyword	Value
File	EZACACHE

Group	Name of group you are placing this function in.
DSName	Must agree with name defined in the IDCAMS step above (for example, CICS.USER.CACHE).
STRings	Maximum number of concurrent users.
Opentime	Startup
Disposition	Old
DAtabuffers	STRings value X 2
Indexbuffers	Number of records in index set.
Table	User
Maxnumrecs	Maximum number of destinations queried.
RECORDFormat	V

Use the following parameters with RDO PROGRAM to define the EZACIC25 module:

RDO keyword	Value
PROGram	EZACIC25
Group	Name of group you are placing this function in
Language	Assembler

Step 3: Execute EZACIC25

EZACIC25 replaces the GETHOSTBYNAME socket call. It is invoked by a EXEC CICS LINK COMMAREA(com-area) where com-area is defined as follows:

Field name **Description**

Return code
A fullword binary variable specifying the results of the function:

Value **Meaning**

-1 ERRNO value returned from GETHOSTBYNAME() call. Check ERRNO field.

0 Host name could not be resolved either within the cache or by use of the GETHOSTBYNAME call.

Note: In some instances, a 10214 errno is returned from the resolve, which can mean that the host name could not be resolved by use of the GETHOSTBYNAME call.

1 Host name was resolved using cache.

2 Host name was resolved using GETHOSTBYNAME call.

ERRNO
A fullword binary field specifying the ERRNO returned from the GETHOSTBYNAME call.

HOSTENT address
The address of the returned HOSTENT structure.

Command
A 4-byte character field specifying the requested operation.

Value **Meaning**

GHBN
GETHOSTBYNAME. This is the only function supported.

Namelen
A fullword binary variable specifying the actual length of the host name for the query.

Query_Type
A 1-byte character field specifying the type of query:

Value	Meaning
0	Attempt query using cache. If unsuccessful, attempt using GETHOSTBYNAME() call.
1	Attempt query using GETHOSTBYNAME() call. This forces a cache refresh for this entry.
2	Attempt query using cache only.

Note: If the cache contains a matching record, the contents of that record is returned regardless of its age.

Name
A 256-byte character variable specifying the host name for the query.

If the transaction program is executing in the Open Transaction Environment, a TCB switch occurs for each call to EZACIC25.

HOSTENT structure

The returned HOSTENT structure is shown in Figure 82.

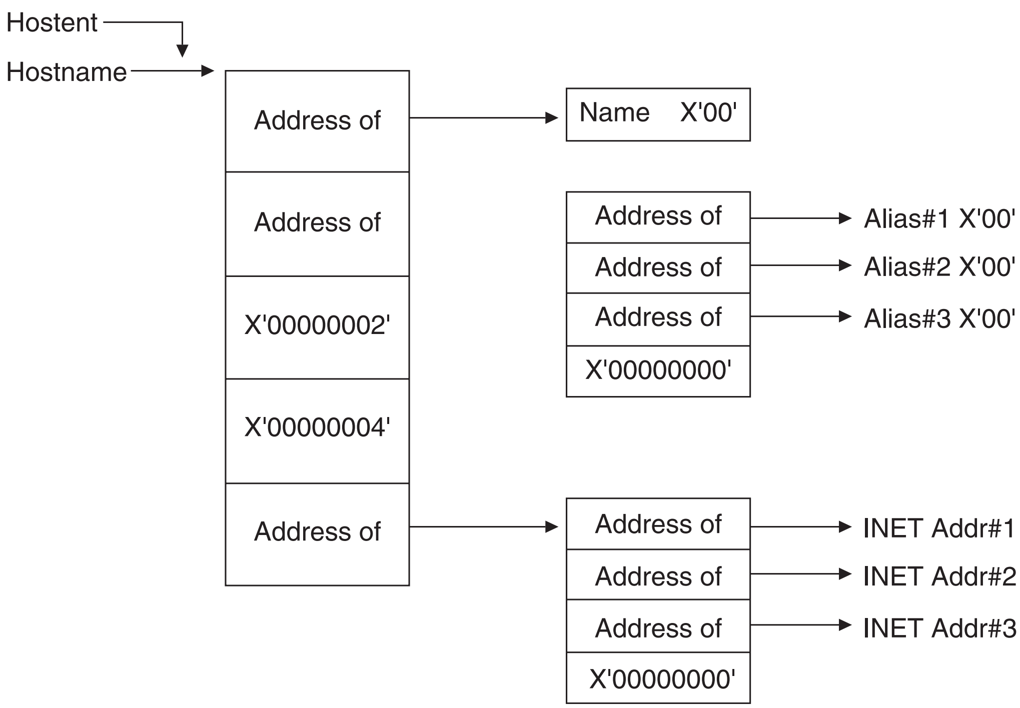


Figure 82. The DNS HOSTENT

Chapter 4. Managing IP CICS sockets

Use the CICS TCP/IP interface to:

- Customize your system so that CICS TCP/IP starts and stops automatically. See "Starting and stopping CICS automatically."
- Manually start and stop CICS TCP/IP after CICS has been initialized. An operator can also query and change specific CICS TCP/IP interface attributes after CICS has been initialized. See "IP CICS socket interface management" on page 104.
- Start and stop CICS TCP/IP from a CICS application program. See "Starting/stopping CICS TCP/IP with program link" on page 116.

Starting and stopping CICS automatically

Modify the CICS Program List Table (PLT) to start and stop the CICS socket interface automatically.

- Startup (PLTPI)

  To start the IP CICS socket interface automatically, make the following entry in PLTPI *after* the DFHDELIM entry:

```
*
* PLT USED TO SUPPORT IP CICS SOCKETS STARTUP
*
         DFHPLT TYPE=INITIAL,SUFFIX=SI
         DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
         DFHPLT TYPE=ENTRY,PROGRAM=EZACIC20
*
* Add other IP CICS Socket PLT startup programs here...
*
         DFHPLT TYPE=FINAL
         END
```

- Shutdown (PLTSD)

  To shut down the IP CICS socket interface automatically (including all other IP CICS sockets enabled programs), make the following entry in the PLTSD *before* the DFHDELIM entry:

```
*
* PLT USED TO SUPPORT IP CICS SOCKETS SHUTDOWN
*
         DFHPLT TYPE=INITIAL,SUFFIX=SD
*
* Add other IP CICS Socket PLT shutdown programs here...
*
         DFHPLT TYPE=ENTRY,PROGRAM=EZACIC20
         DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
         DFHPLT TYPE=FINAL
         END
```

Requirement: If the IP CICS socket interface is started in the PLT (started by invoking EZACIC20), the PLTPIUSR user ID also requires the UPDATE access to the EXITPROGRAM resource when CICS command security is active. Failure to have at least the UPDATE access to the EXITPROGRAM resource causes the IP CICS socket interface and listener to not start when starting or not stop when stopping. Message EZY1350E is issued, and the IP CICS socket interface does not start.

IP CICS socket interface management

Use the EZAO operator transaction to start CICS TCP/IP manually. You should run the EZAO transaction on the CICS region where you want the intended action to occur.

This operational transaction has the following functions:

Interface Startup
: Starts the interface in a CICS address space and starts all listeners that are identified for immediate start.

  Requirement: The EZAO transaction must be running on the CICS where you want to start the CICS sockets interface. You cannot start a CICS socket interface from a different CICS.

Interface Shutdown
: Stops the interface in a CICS address space.

Listener Startup
: Starts a listener in a CICS address space.

Listener Shutdown
: Stops a listener in a CICS address space.

Set Interface
: Alters some attributes of the IP CICS socket interface and listener.

Query Interface
: Shows the current value of some attributes of the IP CICS socket interface and listener.

Trace startup
: Starts CICS tracing for the CICS socket interface in a CICS address space.

Trace shutdown
: Stops CICS tracing for the CICS socket interface in a CICS address space.

When you enter EZAO, the following screen is displayed:

```
EZAO                                                          APPLID = ........

Enter one of the following


SET
INQUIRE
START
STOP
















PF 3 END                                                            12 CNCL
```

Figure 83. EZAO initial screen

INQUIRE function

Use the INQUIRE function to query certain IP CICS socket interface and listener attributes. Use the EZAO,SET command to dynamically change any values. The INQUIRE function can be abbreviated as INQ. Use the EZAO,INQUIRE command to query certain values. If you enter INQ in the screen shown in Figure 83 or enter the EZAO,INQ command on a blank screen, the following screen is displayed:

```
EZAO,INQUIRE                                                  APPLID = ........

Enter one of the following


CICS         ===> ...                Enter Yes|No
LISTENER     ===> ...                Enter Yes|No


















PF 3 END                                                            12 CNCL
```

Figure 84. EZAO INQUIRE screen

If you enter INQUIRE CICS, the following screen is displayed:

```
EZAO,INQUIRE,CICS                                          APPLID = ........



TRACE        ===> ...                    Trace CICS Sockets
MAXOPENTCBS  ===> .....                  CICS open API, L8, TCB Limit
ACTOPENTCBS  ===> .....                  Active CICS open API, L8, TCBs
TCBLIM       ===> .....                  Open API TCB Limit
ACTTCBS      ===> .....                  Number of Active open API TCBs
QUEUEDEPTH   ===> .....                  Number of Suspended Tasks
SUSPENDHWM   ===> .....                  Suspended Tasks HWM
APPLDAT      ===> ...                    Register Application Data













PF 3 END                                                      12 CNCL
```

Figure 85. EZAO INQUIRE CICS screen

This screen displays the following information:

- TRACE is the current IP CICS sockets CICS tracing flag.
- MAXOPENTCBS is the CICS limit of open API TCBs.
- ACTOPENTCBS is the current number of open API TCBs in use across all CICS.
- TCBLIM is the IP CICS sockets-imposed TCB limit.
- ACTTCBS is the current number of open API TCBs in use by IP CICS sockets.
- QUEUEDEPTH is the current number of CICS tasks suspended as the result of TCB limit (TCBLIM).
- SUSPENDHWM is the high-water mark of CICS tasks suspended as the result of TCB limit (TCBLIM).
- APPLDAT indicates whether the IP CICS socket interface automatically registers socket application data.

If you enter INQUIRE LISTENER, the following screen is displayed where you can choose from a list of active listeners:

Figure 86. EZAO INQUIRE LISTENER selection screen

```
EZAO,INQUIRE,LISTENER                                       APPLID = ........

 Choose a listener transaction:

 Sel  Tran Task#   Type     Day Date     Time     Message
  _   .... ....... ........ ... mm/dd/yy hh:mm:ss .............................
  _   .... ....... ........ ... mm/dd/yy hh:mm:ss .............................
  _   .... ....... ........ ... mm/dd/yy hh:mm:ss .............................
  _   .... ....... ........ ... mm/dd/yy hh:mm:ss .............................
  _   .... ....... ........ ... mm/dd/yy hh:mm:ss .............................
  _   .... ....... ........ ... mm/dd/yy hh:mm:ss .............................
  _   .... ....... ........ ... mm/dd/yy hh:mm:ss .............................
  _   .... ....... ........ ... mm/dd/yy hh:mm:ss .............................
  _   .... ....... ........ ... mm/dd/yy hh:mm:ss .............................
  _   .... ....... ........ ... mm/dd/yy hh:mm:ss .............................
  _   .... ....... ........ ... mm/dd/yy hh:mm:ss .............................
  _   .... ....... ........ ... mm/dd/yy hh:mm:ss .............................
  _   .... ....... ........ ... mm/dd/yy hh:mm:ss .............................
  _   .... ....... ........ ... mm/dd/yy hh:mm:ss .............................



 PF 3 END  7 DOWN  8 UP  9 TOP  10 BOTTOM  12 CNCL  ENTER SELECT
```

If you select a listener transaction, the following screen is displayed:

Figure 87. EZAO INQUIRE LISTENER screen

```
EZAO,INQUIRE,LISTENER(....)                                 APPLID = ........



 LAPPLD       ===> ...                Register Application Data



















 PF 3 END                                                           12 CNCL
```

The LAPPLD entry indicates whether the IP CICS socket interface automatically registers socket application data for the listener.

SET function

Use the SET function to dynamically change certain attributes of the IP CICS socket interface and listener. Changes made in this way are not reflected in the configuration options contained in the EZACONFG dataset. Use the EZAO,INQUIRE command to query some values. If you enter SET in the screen shown in Figure 83 on page 105 or if you enter EZAO,SET on a blank screen, the following screen is displayed:

```
EZAO,SET                                                       APPLID = ........

Enter one of the following


CICS         ===> ...                 Enter Yes|No
LISTENER     ===> ...                 Enter Yes|No
















PF 3 END                                                             12 CNCL
```

Figure 88. EZAO SET screen

If you enter SET CICS, the following screen is displayed:

```
EZAO,SET,CICS                                                  APPLID = ........

Overtype to Enter

TRACE        ===> ...                 Trace CICS Sockets
TCBLIM       ===> .....               Open API TCB Limit
APPLDAT      ===> ...                 Register Application Data
















PF 3 END                                                             12 CNCL
```

Figure 89. EZAO SET CICS screen

This screen displays the following information:

- TRACE is the current IP CICS sockets CICS tracing flag. Specify YES or NO to dynamically enable or disable IP CICS sockets CICS tracing.
- TCBLIM is the current IP CICS sockets-imposed TCB limit. Specify a value in the range 0 to the value specified by the MAXOPENTCBS option to dynamically change the TCB limiting factor.

- APPLDAT is the current IP CICS socket interface socket application data registration flag. Specify YES or NO to dynamically enable or disable the registration of socket application data.

If you enter SET LISTENER, the following screen is displayed where you can choose from a list of active listeners:

Figure 90. EZAO SET LISTENER selection screen

```
EZAO,SET,LISTENER                                             APPLID = ........

Choose a listener transaction:

Sel  Tran Task#   Type     Day Date     Time    Message
 _   .... ....... ........ ... mm/dd/yy hh:mm:ss ..............................
 _   .... ....... ........ ... mm/dd/yy hh:mm:ss ..............................
 _   .... ....... ........ ... mm/dd/yy hh:mm:ss ..............................
 _   .... ....... ........ ... mm/dd/yy hh:mm:ss ..............................
 _   .... ....... ........ ... mm/dd/yy hh:mm:ss ..............................
 _   .... ....... ........ ... mm/dd/yy hh:mm:ss ..............................
 _   .... ....... ........ ... mm/dd/yy hh:mm:ss ..............................
 _   .... ....... ........ ... mm/dd/yy hh:mm:ss ..............................
 _   .... ....... ........ ... mm/dd/yy hh:mm:ss ..............................
 _   .... ....... ........ ... mm/dd/yy hh:mm:ss ..............................
 _   .... ....... ........ ... mm/dd/yy hh:mm:ss ..............................
 _   .... ....... ........ ... mm/dd/yy hh:mm:ss ..............................
 _   .... ....... ........ ... mm/dd/yy hh:mm:ss ..............................
 _   .... ....... ........ ... mm/dd/yy hh:mm:ss ..............................



PF 3 END  7 DOWN  8 UP  9 TOP  10 BOTTOM  12 CNCL  ENTER SELECT
```

If you select a listener transaction, the following screen is displayed:

Figure 91. EZAO SET LISTENER screen

```
EZAO,SET,LISTENER(....)                                       APPLID = ........

Overtype to Enter

LAPPLD       ===> .......             Register Application Data




















PF 3 END                                                            12 CNCL
```

The LAPPLD entry indicates whether the IP CICS socket interface registers socket application data for the listener.

START function

The START function starts the CICS socket interface or a listener within the interface. When the interface is started, all listeners marked for immediate start are also started. The START function also enables CICS tracing for the CICS socket interface and the listener.

If you type `STA` on the current screen or type `EZAO STA` on a blank screen, the following screen is displayed:

```
EZAO,START                                                   APPLID = ........

Enter one of the following


CICS         ===> ...                    Enter Yes|No
LISTENER     ===> ...                    Enter Yes|No
TRACE        ===> ...                    Enter Yes|No














PF 3 END                                                           12 CNCL
```

Figure 92. EZAO START screen

START CICS

If you type START CICS, the following screen is displayed:

```
EZAO,START,CICS                                          APPLID = ........



APPLID=      ===> ........             APPLID of CICS














CICS socket interface Startup Complete


PF 3 END                                                      12 CNCL
```

Figure 93. EZAO START CICS response screen

START LISTENER

If you type START LISTENER, the following screen is displayed:

```
EZAO,START,LISTENER                                      APPLID = ........



APPLID=      ===> ........             APPLID of CICS
LISTENER     ===> ....                 Enter Name of listener

















PF 3 END                                                      12 CNCL
```

Figure 94. EZAO START LISTENER screen

After you type the listener name, the listener starts. The following screen is displayed, and the results appear in the message area:

```
EZAO,START,LISTENER(CSKL)                                    APPLID = ........



APPLID=      ===> ........            APPLID of CICS
LISTENER     ===> ....                Enter Name of listener














CICS socket interface listener CSKL is Started


PF 3 END                                                          12 CNCL
```

Figure 95. EZAO START LISTENER result screen

START TRACE

If you type START TRACE, the following screen is displayed:

```
EZAO,START,TRACE                                             APPLID = ........



APPLID=      ===> ........            APPLID of CICS















CICS/SOCKETS CICS TRACING IS ENABLED


PF 3 END                                                          12 CNCL
```

Figure 96. EZAO START TRACE screen

Issue the EZAO,START,TRACE command on the CICS region where APPLID matches the IP CICS socket interface and where CICS tracing is to be started.

STOP function

The STOP function is used to stop the CICS socket interface or a listener within the interface. If the interface is stopped, all listeners are stopped before the interface is stopped. The STOP function also disables CICS tracing for the CICS socket interface and the listener. If you type STO in the screen shown in Figure 83 on page 105 page 105 or enter EZAO STO on a blank screen, the following screen is displayed:

```
EZAO,STOP                                                APPLID = ........

Enter one of the following


CICS         ===> ...                 Enter Yes|No
LISTENER     ===> ...                 Enter Yes|No
TRACE        ===> ...                 Enter Yes|No






















PF 3 END                                                       12 CNCL
```

Figure 97. EZAO STOP screen

STOP CICS

If you specify STOP CICS, the following screen is displayed:

```
EZAO,STOP,CICS                                           APPLID = ........




APPLID=      ===> ........            APPLID of CICS
IMMEDIATE    ===> ...                 Enter Yes|No
























PF 3 END                                                       12 CNCL
```

Figure 98. EZAO STOP CICS screen

The following options are available to stop CICS TCP/IP:

IMMEDIATE=NO
: Used this option in most cases because it gracefully terminates the interface. This option has the following effects on applications using this API:
  - If no other socket applications are active or suspended, the listener transaction (CSKL) quiesces after a maximum wait of 3 minutes.

- If active or suspended sockets applications exist, the listener allows them to continue processing. When all of these tasks are complete, the listener terminates.
- This option denies access to this API for all new CICS tasks. Tasks that start after CICS TCP/IP has been stopped END with the CICS abend code AEY9.

IMMEDIATE=YES
This option is reserved for unusual situations and abruptly terminates the interface. It has the following effect on applications using this API:
- Purges the master server (listener) CSKL.
- Denies access to the API for all CICS tasks. Tasks that have successfully called the API previously abend with the AETA abend code on the next socket call. New tasks that have started are denied by the AEY9 abend code.

After you choose an option, the stop is attempted. The screen is displayed again, and the results appear in the message area.

STOP LISTENER

If you specify STOP LISTENER, the following screen is displayed:

```
EZAO,STOP,LISTENER                                          APPLID = ........



APPLID=      ===> ........             APPLID of CICS
LISTENER     ===> ....                 Enter Name of listener
















PF 3 END                                                          12 CNCL
```

Figure 99. EZAO STOP LISTENER screen

When you input the listener named, that listener is stopped. The screen is displayed again, and the results appear in the message area.

STOP TRACE

If you specify STOP TRACE, the following screen is displayed:

```
EZAO,STOP,TRACE                                             APPLID = ........



APPLID=      ===> ........            APPLID of CICS

















CICS/SOCKETS CICS TRACING IS DISABLED


PF 3 END                                                          12 CNCL
```

Figure 100. EZAO STOP TRACE screen

Issue the EZAO,STOP,TRACE command on the CICS region where APPLID matches the IP CICS socket interface and where CICS tracing is to be stopped.

Abbreviating the EZAO transaction parameters

It is possible to abbreviate the parameters of the EZAO transaction, but a minimum of three characters must be specified. This capability allows the command to be issued using minimal keystrokes. The following list of commands shows the abbreviated parameters:

EZAO,STArt,CICs
: Starts the interface

EZAO,STOp,CICs
: Stops the interface

EZAO,STArt,LIStener
: Starts a listener

EZAO,STOp,LIStener
: Stops a listener

EZAO,STArt,TRAce
: Enables CICS tracing

EZAO,STOp,TRAce
: Disables CICS tracing

Notes:

1. The values in uppercase characters are the minimal acceptable value for parameters.
2. You can use spaces instead of commas as a parameter delimiter. This is shown in the following example:

   ```
   EZAO STArt CICs
   ```

   This is the same as the following:

   ```
   EZAO,STArt,CICs
   ```

Starting/stopping CICS TCP/IP with program link

Issue an EXEC CICS LINK to program EZACIC20 to start or stop the CICS socket interface. Include the following steps in the LINKing program:

1. Define the COMMAREA for EZACIC20 by including the following instruction in your DFHEISTG definition:
   ```
   EZACICA AREA=P20,TYPE=CSECT
   ```
   The length of the area is equated to P20PARML, and the name of the structure is P20PARMS.
2. Initialize the COMMAREA values as follows:

   P20TYPE

   I Initialization

   T Immediate termination

   D Deferred termination

   Q Query the PLT shutdown immediate configuration option

   P20OBJ

   C CICS sockets interface

   L Listener

   P20LIST
   Name of listener (if this is listener initialization or termination)
3. Issue the EXEC CICS LINK to program EZACIC20. EZACIC20 does not return until the function is complete.
4. Check the P20RET field for the response from EZACIC20.

EZACIC20 can issue the following user abend codes:

- Abend code E20L is issued if the CICS socket interface is not in startup or termination and no COMMAREA was provided.
- Abend code E20T is issued if CICS is not active or if you run the EZACIC20 program at the wrong PLT phase. See “CICS program list table (PLT)” on page 46 for more information about setting CICS TCP sockets to automatically startup or shutdown by using updates to the PLT.

Chapter 5. Writing your own listener

The IP CICS socket interface provides a structure that supports multiple listeners. These listeners can be multiple copies of the IBM-supplied listener, user-written listeners, or a combination of the two. You can also run without a listener.

For each listener (IBM-supplied or user-written), there are certain basic requirements that enable the interface to manage the listeners correctly, particularly during initialization and termination. They are:

- Each listener instance must have a unique transaction name, even if you are running multiple copies of the same listener.
- Each listener should have an entry in the CICS sockets configuration data set. Even if you do not use automatic initiation for your listener, the lack of an entry would prevent correct termination processing and could prevent CICS from completing a normal shutdown.

For information on the IBM-supplied listener, see "The IBM listener" on page 134.

Prerequisites

Some installations can require a customized, user-written listener. Writing your own listener has the following prerequisites:

1. Determine what capability is required that is not supplied by the IBM-supplied listener. Is this capability a part of the listener or a part of the server?
2. Knowledge of the CICS-Assembler environment is required.
3. Knowledge of multi-threading applications is required. A listener must be able to perform multiple functions concurrently to achieve good performance.
4. Knowledge of the CICS socket interface is required.
5. Knowledge of how to use compare and swap logic for serially updating shared resources.

Using IBM's environmental support

A user-written listener can use the environmental support supplied and used by the IBM-supplied listener. To employ this support, the user-written listener must do the following in addition to the requirements described above:

- The user-written listener must be written in Assembler.
- The RDO definitions for the listener transaction and program should be identical to those for the IBM-supplied listener with the exception of the transaction/program names. Reference the program definition for the IBM-supplied listener, EZACIC02, in SEZAINST(EZACICCT).

```
DEFINE PROGRAM(EZACIC02)
DESCRIPTION(IBM LISTENER)
GROUP(SOCKETS) CEDF(YES) DATALOCATION(ANY) EXECKEY(CICS)
RELOAD(NO) RESIDENT(YES) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(NORMAL)
CONCURRENCY(THREADSAFE)
```

Figure 101. Program Definition for listener EZACIC02

- In the program, define an input area for the configuration file records. If you are going to read the configuration file using MOVE mode, you can define the area by making the following entry in your DFHEISTG area:

```
EZACICA AREA=CFG,TYPE=CSECT
```

If you are going to read the configuration file using LOCATE mode you can define a DSECT for the area as follows:

```
EZACICA AREA=CFG,TYPE=DSECT
```

In either case, the length of the area is represented by the EQUATE label CFGLEN. The name of the area/DSECT is CFG0000.
- In the program, define a DSECT for mapping the Global Work Area (GWA). This is done by issuing the following macro:

```
EZACICA AREA=GWA,TYPE=DSECT
```

The name of the DSECT is GWA0000.
- In the program, define a DSECT for mapping the Task Interface Element (TIE). This is done by issuing the following macro:

```
EZACICA AREA=TIE,TYPE=DSECT
```

The name of the DSECT is TIE0000.
- In the program define a DSECT for mapping the listener Control Area (LCA). This is done by issuing the following macro:

```
EZACICA AREA=LCA,TYPE=DSECT
```

The name of the DSECT is LCA0000.
- Obtain address of the GWA. This can be done using the following CICS command:

```
EXEC  CICS EXTRACT EXIT PROGRAM(EZACIC01) GASET(ptr) GALEN(len)
```

where *ptr* is a register and *len* is a halfword binary variable. The address of the GWA is returned in *ptr* and the length of the GWA is returned in len. Use of the Extract Exit command requires UPDATE access to the EXITPROGRAM resource. Failure to have at least the UPDATE access to the EXITPROGRAM resource causes the IP CICS socket interface and listener to either not start when starting or not stop when stopping.

Guideline: As of CICS/TS 2.3, the EXEC CICS EXTRACT command is not threadsafe. If the interface is using the CICS Open Transaction Environment, you should issue this command with other non-threadsafe commands to prevent excessive TCB switching.
- Read the configuration file during initialization of the listener. The configuration file is identified as EZACONFG in the CICS Configuration file. The record key for the user-written listener is as follows:
  - APPLID

    An 8-byte character field set to the APPLID value for this CICS. This value can be obtained from the field GWACAPPL in the GWA or by using the following CICS command:

    ```
    EXEC  CICS ASSIGN APPLID(applid)
    ```

    where *applid* is an 8-byte character field.
  - Record Type

    A 1-byte character field set to the record type. It must have the value L.

– Reserved Field

A 3-byte hex field set to binary zeros.

– Transaction

A 4-byte character field containing the transaction name for this listener. It can be obtained from the EIBTRNID field in the Execute Interface Block.

The configuration record provides the information entered by either the EZACICD configuration macro or the EZAC Configuration transaction. The user-written listener can use this information selectively, but it is highly recommended as it contains the values specified for PORT, BACKLOG, and NUMSOCK. See Chapter 2, “Setting up and configuring CICS TCP/IP,” on page 23 for more information about the configuration data set with EZACICD TYPE parameter subsection.

For shared files: If the user-written listener reads the configuration file, it must first issue an EXEC CICS SET command to enable and open the file. When the file operation is complete, the user-written listener must issue an EXEC CICS SET command to disable and close the file. Failure to do so results in file errors in certain shared-file situations.

Requirement: Use of the EXEC CICS ENABLE command requires UPDATE access to EXITPROGRAM resources. Failure to have at least the UPDATE access to the EXITPROGRAM resource causes the IP CICS socket interface and listener to either not start when starting or not stop when stopping.

- The user-written listener should locate its listener Control Area (LCA). The LCAs are located contiguously in storage with the first one pointed to by the GWALCAAD field in the GWA. The correct LCA has the transaction name of the listener in the field LCATRAN.
- The user-written listener should set the LCASTAT field to a value specified by LCASTATP so that the IP CICS socket interface is aware that the listener is active. Otherwise, the IP CICS sockets listener termination logic bypasses the posting of the listeners termination ECB.
- The user-written listener should monitor either the LCASTAT field in the LCA or the GWATSTAT field in the GWA for shutdown status. If either field shows an immediate shutdown in progress, the user-written listener should terminate by issuing the EXEC CICS RETURN command and allow the interface to clean up any socket connections. If either field shows a deferred termination in progress, the user-written listener should do the following:
  1. Accept any pending connections, and close the passive (listen) socket.
  2. Complete the processing of any sockets involved in transaction initiation (that is, processing the GIVESOCKET command). When processing is complete, close these sockets.
  3. When all sockets are closed, issue the EXEC CICS RETURN command.
- The user-written listener should avoid socket calls which imply blocks dependent on external events such as ACCEPT or READ. These calls should be preceded by a single SELECTEX call that waits on the ECB LCATECB in the LCA. This ECB is posted when an immediate termination is detected, and its posting causes the SELECTEX to complete with a RETCODE of 0 and an ERRNO of 0. The program should check the ECB when the SELECTEX completes in this way as this is identical to the way SELECTEX completes when a timeout happens. The ECB can be checked by looking for a X’40’ in the first byte (post bit).

This SELECTEX should also specify a timeout value. This provides the listener with a way to periodically check for a deferred termination request. Without this, CICS sockets Deferred Termination or CICS Deferred Termination cannot complete.

- The user-written listener should use a non-reusable subtask. Issue the INITAPI command or an INITAPIX command with the letter *L* in the last byte of the subtask name. The user-written listener implements the termination and detach logic in the same way that the IBM-supplied listener does.
- The user-written listener should update LCASTAT with one of the following:

```
LCASTAT DS X Status of this listener
LCASTAT0 EQU B'00000000' Listener not in operation
LCASTATI EQU B'00000001' Listener in initialization
LCASTATS EQU B'00000010' Listener in SELECT
LCASTATP EQU B'00000100' Listener processing
LCASTATE EQU B'00001000' Listener had initialization error
LCASTATC EQU B'00010000' Immediate termination in progress
LCASTATD EQU B'00100000' Deferred termination in progress
LCASTATA EQU B'01000000' Listener is active
LCASTATR EQU B'10000000' Listener is CICS delayed retry
```

Rule: If IP CICS sockets is configured to use CICS's Open Transaction Environment, then ensure that you serially update the LCASTAT vaue. The Listener Control Area (LCA) is part of the global work area (GWA), and is considered to be a shared resource. An appropriate value to move into LCASTAT would be LCASTATP (B'00000100') when the user-written listener starts. This value enables the CICS socket logic to correctly post the LCATECB during both deferred and immediate termination.

- User-written listener programs can use the LCASTAT2A status flag to determine whether this listener should register application data. The user-written listener should update LCASTAT2 with one of the following:

```
LCASTAT2 DS X Listener status byte 2
LCASTAT2C EQU B'00000001' Listener can now connect to TCP
LCASTAT2A EQU B'00000010' Register Application Data
LCASTAT2H EQU B'00000100' LAPPLD inherits APPLDAT
LCASTAT2S EQU B'00100000' This is a STANDARD listener
LCASTAT2E EQU B'01000000' This is an ENHANCED listener
LCASTAT26 EQU B'10000000' Listeners AF is AF_INET6
```

WLM registration and unregistration for sysplex connection optimization

If you are writing your own listeners, an interface to the WLM registration and unregistration module, EZACIC12 is available and can be used for registration and unregistration. The registration and unregistration should be done at the same times the IBM listener does it. It is important to deregister for any termination situation because the Workload Manager does not detect the termination of a listener (it does detect CICS termination) and the Domain Name Server could continue to respond to gethostbyname () requests within the address of this listener.

The interface to EZACIC12 is through the EXEC CICS LINK command. The linking program (listener) builds a COMMAREA for EZACIC12. COMMAREA is described below and, for assembler use, issuing the macro EZACICA TYPE={CSECT|DSECT},AREA=P12 provides a storage definition or DSECT for the area.

The format of the COMMAREA for EZACIC12 is as follows:

Field name
: **Description**

P12CONFG
: A 4-byte field containing the address of the Configuration Record for this listener.

P12REGST
: A one byte field output from WLM Registration. A one byte field input for WLM unregistration.

  The same value output from Registrations should be input for the associated unregistration. The byte represents the registration status of up to three WLM groups. Each bit within the byte represents a WLM group registration.

  B'00000000'
  : No WLM groups registered.

  B'00000001'
  : WLM group 1 registered.

  B'00000010'
  : WLM group 2 registered.

  B'00000100'
  : WLM group 3 registered.

P12TYPE
: A 1-byte character field containing the request code for EZACIC12.

  C'R' Registration.

  C'D' Deregistration.

P12HOST
: A 24-character field containing the host name for EZACIC12. It is the Domain Name of the host that the listener is executing on as obtained by the gethostname() socket call. EZACIC12 pads it to the right with blanks to meet the WLM requirement.

  Guideline: The EZACIC12 program is defined to CICS as threadsafe indicating that programs linking to it can take advantage of staying on an open API TCB.

Tip: The automated domain name registration application (ADNR) cannot provide WLM-based load balancing; however, it can be configured to provide round robin connection balancing as supported by the BIND 9 domain name server (DNS). See *z/OS Communications Server: IP Configuration Guide* for more information about load balancing using an external load balancer and one or more load balancing agents. See the information in *z/OS Communications Server: IP Configuration Guide* for more about load balancing using an external load balancer and one or more load balancing agents. See automated domain name registration information in *z/OS Communications Server: IP Configuration Guide* for more details about dynamically updating name servers with information about sysplex resources in near real time.

Chapter 6. Application programming guide

This topic describes how to write applications that use the IP CICS sockets API. It describes typical sequences of calls for client, concurrent server (with associated child server processes), and iterative server programs. The contents of the topic are:

- The following setups for writing CICS TCP/IP applications are available:
  - Concurrent server (the supplied listener transaction) and child server processes run under CICS TCP/IP.
  - The same as 1 but with a user-written concurrent server.
  - An iterative server running under CICS TCP/IP.
  - A client application running under CICS TCP/IP.
- Socket addresses
- MVS address spaces
- GETCLIENTID, GIVESOCKET, and TAKESOCKET commands
- The listener program
- CICS Open Transaction Environment considerations
- Application Transparent Transport Layer Security (AT-TLS)

Chapter 7, “C language application programming,” on page 157 describes the C language calls that can be used with CICS.

Chapter 8, “Sockets extended API,” on page 223 provides reference information on the Sockets Extended API for COBOL, PL/I, and Assembler language. The Sockets Extended API is the recommended interface for new application development.

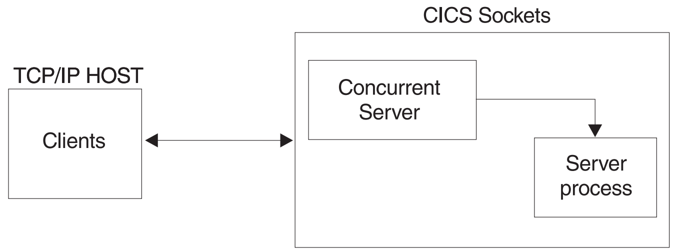
Note: Appendix A, “Original COBOL application programming interface (EZACICAL),” on page 367 provides reference information on the EZACICAL API for COBOL and assembler language. This interface was made available in a prior release of TCP/IP Services and is being retained in the current release for compatibility. For the best results, however, use the Sockets Extended API whenever possible. It is described in Chapter 8, “Sockets extended API,” on page 223.

Writing CICS TCP/IP applications

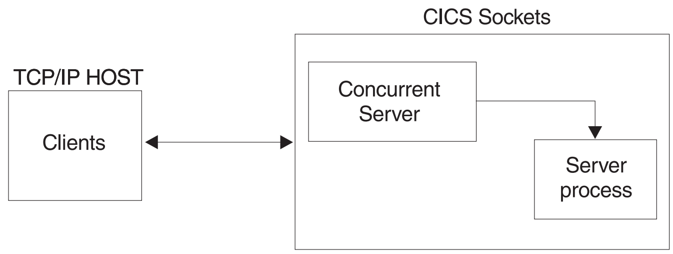
Chapter 1, “Introduction to CICS TCP/IP,” on page 1 describes the basics of TCP/IP client/server systems and the two types of server: iterative and concurrent. This topic considers in detail four TCP/IP setups in which CICS TCP/IP applications are used in various parts of the client/server system.

The setups are:

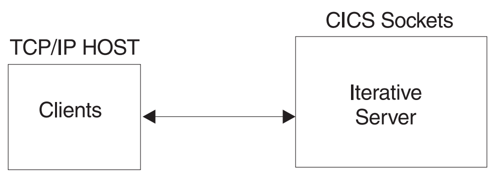
- **The client-listener-child server application set**. The concurrent server and child server processes run under CICS TCP/IP. The concurrent server is the supplied listener transaction. The client might be running TCP/IP under one of the various UNIX operating systems such as AIX.



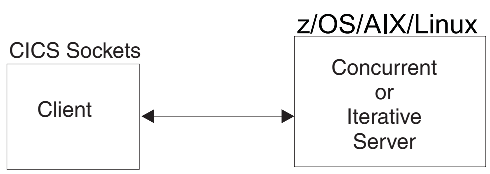
- **Writing your own concurrent server**. This is the same setup as the first except that a user-written concurrent server is being used instead of the IBM listener.



- **The iterative server CICS TCP/IP application**. This setup is designed to process one socket at a time.



- **The client CICS TCP/IP application**. In this setup, the CICS application is the client and the server is the remote TCP/IP process.



For details of how the CICS TCP/IP calls should be specified, see Chapter 7, "C language application programming," on page 157, Chapter 8, "Sockets extended API," on page 223, and Appendix A, "Original COBOL application programming interface (EZACICAL)," on page 367.

1. The client-listener-child-server application set

Figure 102 on page 125 shows the sequence of CICS commands and socket calls involved in this setup. CICS commands are prefixed by EXEC CICS; all other numbered items in the figure are CICS TCP/IP calls.

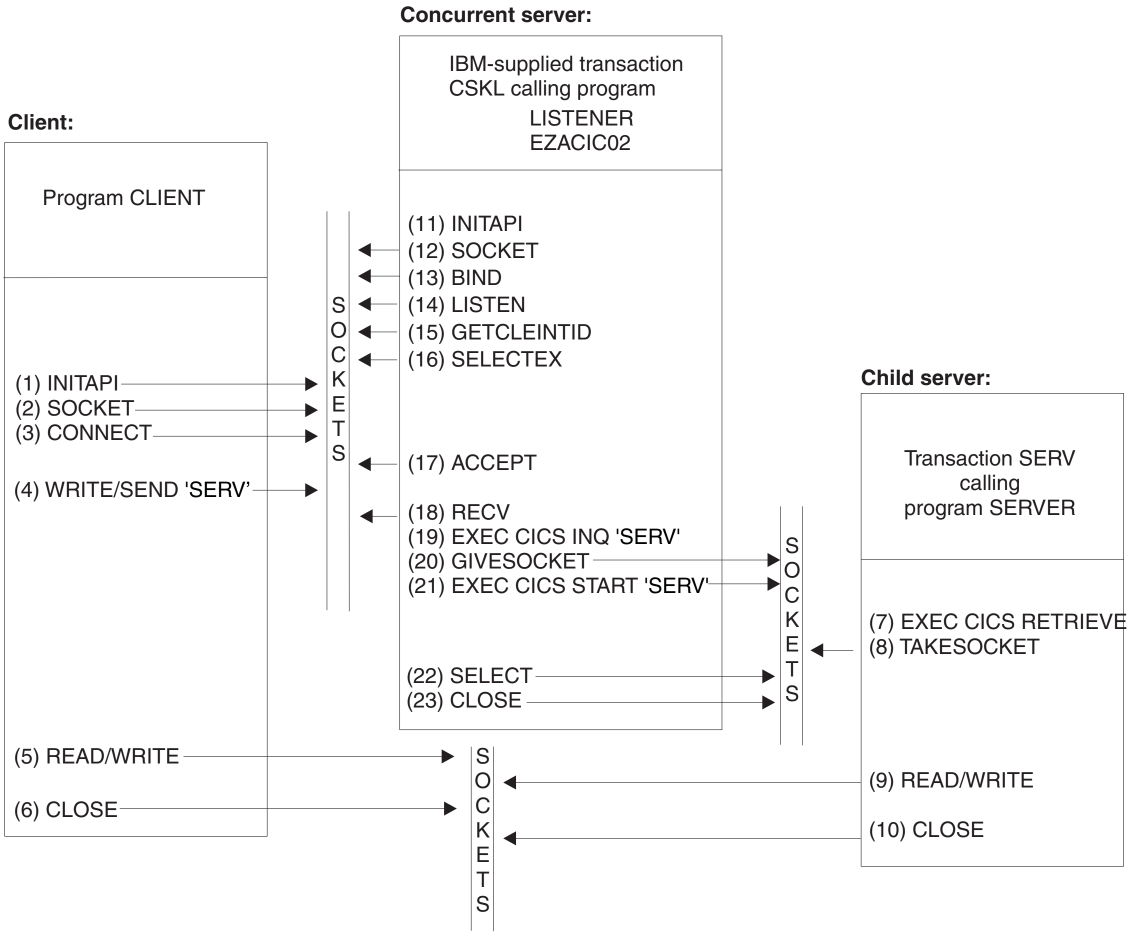


Figure 102. The sequence of sockets calls

Client call sequence

Table 8 explains the functions of each of the calls listed in Figure 102.

Table 8. Calls for the client application

(1) INITAPI	Connect the CICS application to the TCP/IP interface. (This call is only used by applications written in Sockets Extended or the EZACICAL interface). Use the MAXSOC parameter on the Sockets Extended INITAPI or the MAX-SOCK parameter on the EZACICAL interface to specify the maximum number of sockets to be used by the application.

Table 8. Calls for the client application (continued)

(2) SOCKET	This obtains a socket. You define a socket with three parameters: • The domain, or addressing family • The type of socket • The protocol For CICS TCP/IP, the domain can only be one of the TCP/IP Internet domains, either AF_INET (2) for IPv4 or AF_INET6 (19) for IPv6. The type can be SOCK_STREAM (1) for stream sockets (TCP) or SOCK_DGRAM (2) for datagram sockets (UDP). The protocol can be either TCP or UDP. Passing 0 for the protocol selects the default protocol. If successful, the SOCKET call returns a socket descriptor, S, which is always a small integer. Notice that the socket obtained is not yet attached to any local or destination address.
(3) CONNECT	Client applications use this to establish a connection with a remote server. You must define the local socket S to be used in this connection and the address and port number of the remote socket. The system supplies the local address, so on successful return from CONNECT, the socket is completely defined, and is associated with a TCP connection (if stream) or UDP connection (if datagram).
(4) WRITE	This sends the first message to the listener. The message contains the CICS transaction code as its first 4 bytes of data. You must also specify the buffer address and length of the data to be sent.
(5) READ/WRITE	These calls continue the conversation with the server until it is complete.
(6) CLOSE	This closes a specified socket and so ends the connection. The socket resources are released for other applications.

Listener call sequence

The listener transaction CSKL is provided as part of CICS TCP/IP. These are the calls issued by the CICS listener. Your client and server call sequences must be prepared to work with this sequence. These calls are documented in “2. Writing your own concurrent server” on page 127, where the listener calls in Figure 102 are explained.

Child server call sequence

Table 9 explains the functions of each of the calls listed in Figure 102 on page 125.

Table 9. Calls for the server application

(7) EXEC CICS RETRIEVE	This retrieves the data passed by the EXEC CICS START command in the concurrent server program. This data includes the socket descriptor and the concurrent server client ID as well as optional additional data from the client.
(8) TAKESOCKET	This acquires the newly created socket from the concurrent server. The TAKESOCKET parameters must specify the socket descriptor to be acquired and the client ID of the concurrent server. This information was obtained by the EXEC CICS RETRIEVE command. **Note:** If TAKESOCKET is the first call, it issues an implicit INITAPI with default values.
(9) READ/WRITE	The conversation with the client continues until complete.
(10) CLOSE	Terminates the connection and releases the socket resources when finished.

2. Writing your own concurrent server

The overall setup is the same as the first scenario, but your concurrent server application performs many of the functions performed by the listener. Obviously, the client and child server applications have the same functions.

Concurrent server call sequence

Table 10 explains the functions of each of the steps listed in Figure 102 on page 125.

Table 10. Calls for the concurrent server application

(11) INITAPI	Connects the application to TCP/IP, as in Table 8.
(12) SOCKET	This obtains a socket, as in Table 8.
(13) BIND	After a socket has been obtained, a concurrent server uses this call to attach itself to a specific port at a specific address so that the clients can connect to it. The socket descriptor and a local address and port number are passed as arguments. On successful return of the BIND call, the socket is *bound* to a port at the local address, but not (yet) to any remote address.
(14) LISTEN	After binding an address to a socket, a concurrent server uses the LISTEN call to indicate its readiness to accept connections from clients. LISTEN tells TCP/IP that all incoming connection requests should be held in a queue until the concurrent server can deal with them. The BACKLOG parameter in this call sets the maximum queue size.
(15) GETCLIENTID	This command returns the identifiers (MVS address space name and subtask name) by which the concurrent server is known by TCP/IP. This information is needed by the EXEC CICS START call.
(16) SELECTEX	The SELECTEX call monitors activity on a set of sockets. In this case, it is used to interrogate the queue (created by the LISTEN call) for connections. It returns when an incoming CONNECT call is received or when LCATECB was posted because immediate termination was detected, or else times out after an interval specified by one of the SELECTEX parameters.
(17) ACCEPT	The concurrent server uses this call to accept the first incoming connection request in the queue. ACCEPT obtains a new socket descriptor with the same properties as the original. The original socket remains available to accept more connection requests. The new socket is associated with the client that initiated the connection.
(18) RECV	A RECV is not issued if the FORMAT parameter is ENHANCED and MSGLENTH is 0. If FORMAT is ENHANCED, MSGLENTH is not 0, and PEEKDATA is YES, the listener peeks the number of bytes specified by MSGLENTH. If FORMAT is STANDARD, the listener processes the client data as in earlier releases.
(19) CICS INQ	This checks that the SERV transaction is defined to CICS (else the TRANSIDERR exceptional condition is raised), and, if so, that its status is ENABLED. If either check fails, the listener does not attempt to start the SERV transaction.
(20) GIVESOCKET	This makes the socket obtained by the ACCEPT call available to a child server program.
(21) CICS START	This initiates the CICS transaction for the child server application and passes the ID of the concurrent server, obtained with GETCLIENTID, to the server. For example, in "Listener output format" on page 136, the parameters `LSTN-NAME` and `LSTN-SUBNAME` define the listener.

Table 10. Calls for the concurrent server application (continued)

(22) SELECTEX [8]	Again, the SELECTEX call is used to monitor TCP/IP activity. This time, SELECTEX returns when the child server issues a TAKESOCKET call.
(23) CLOSE	This releases the new socket to avoid conflicts with the child server.

Passing sockets

In CICS, a socket belongs to a CICS task. Therefore, sockets can be passed between programs within the same task by passing the descriptor number. However, passing a socket between CICS tasks does require a GIVESOCKET/TAKESOCKET sequence of calls.

3. The iterative server CICS TCP/IP application

Figure 103 shows the sequence of socket calls involved in a simple client-iterative server setup.

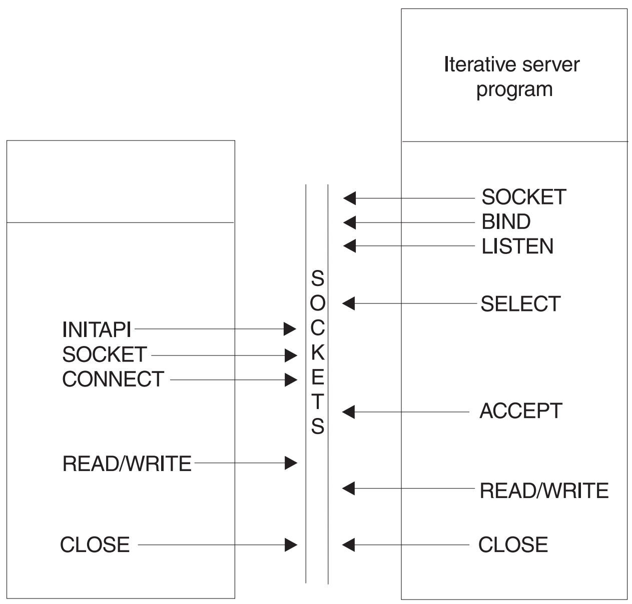


Figure 103. Sequence of socket calls with an iterative server

The setup with an iterative server is much simpler than the previous cases with concurrent servers.

Iterative server use of sockets

The iterative server need only obtain 2 socket descriptors. The iterative server makes the following calls:

1. As with the concurrent servers, SOCKET, BIND, and LISTEN calls are made to inform TCP/IP that the server is ready for incoming requests, and is listening on socket 0.
2. The SELECT call then returns when a connection request is received. This prompts the issuing of an ACCEPT call.

8. This SELECTEX is the same as the SELECTEX call in Step 16. They are shown as two calls to clarify the functions being performed.

3. The ACCEPT call obtains a new socket (1). Socket 1 is used to handle the transaction. After this completed, socket 1 closes.
4. Control returns to the SELECT call, which then waits for the next connection request.

The disadvantage of an iterative server is that it remains blocked for the duration of a transaction, as described in Chapter 1, “Introduction to CICS TCP/IP,” on page 1.

4. The client CICS TCP/IP application

Figure 104 shows the sequence of calls in a CICS client-remote server setup. The calls are similar to the previous examples.

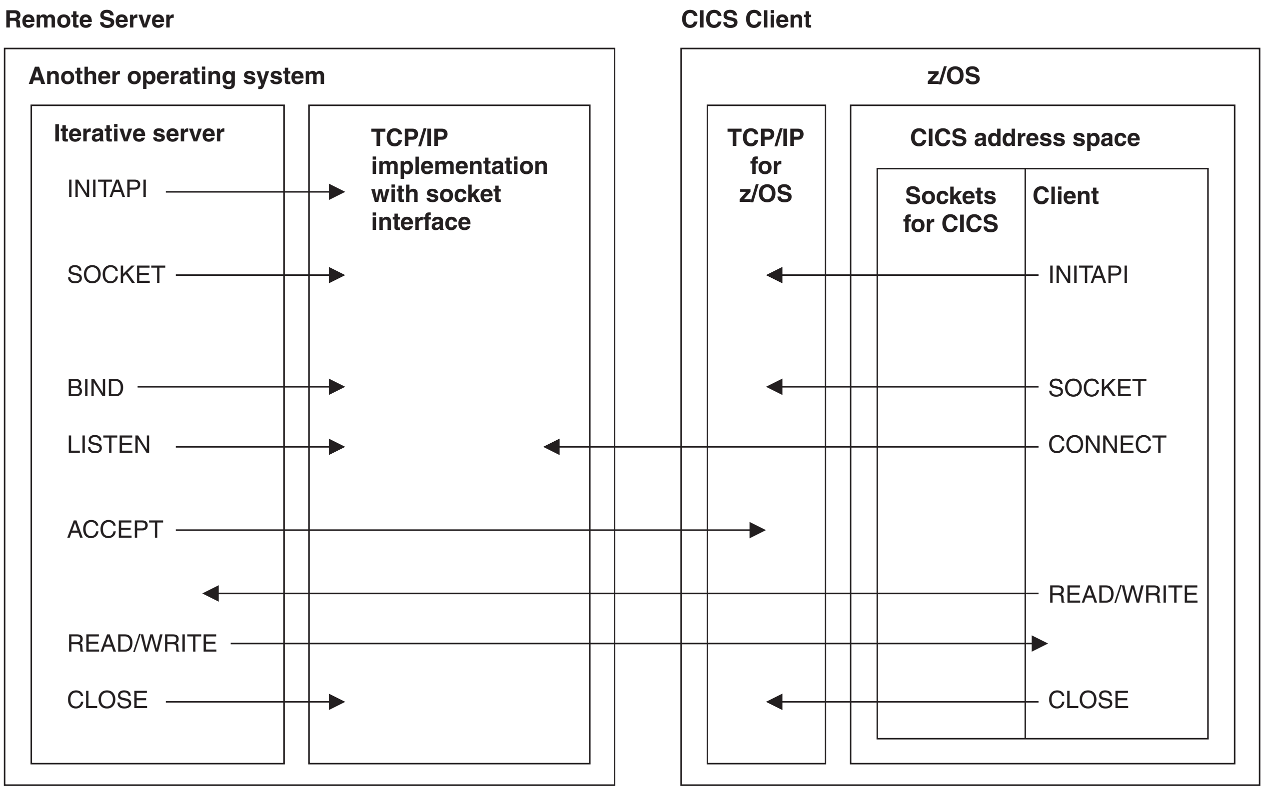


Figure 104. Sequence of socket calls between a CICS client and a remote iterative server

Figure 104 shows that the server can be on any processor and can run under any operating system, provided that the combined software-hardware configuration supports a TCP/IP server.

For simplicity, the figure shows an iterative server. A concurrent server would need a child server in the remote processor and an adjustment to the calls according to the model in Figure 102 on page 125.

A CICS server issues a READ call to read the client’s first message, which contains the CICS transaction name of the required child server. When the server is in a non-CICS system, application design must specify how the first message from the CICS client indicates the service required (in Figure 104, the first message is sent by a WRITE call).

If the server is a concurrent server, this indication is typically the name of the child server. If the server is iterative, as in Figure 104, and all client calls require the same service, this indication might not be necessary.

Socket addresses

Socket addresses are defined by specifying the address family and the address of the socket in the Internet. In CICS TCP/IP, the address is specified by the IP address and port number of the socket.

Address family (domain)

CICS TCP/IP supports the AF_INET and AF_INET6 TCP/IP addressing family (or domain, as it is called in the UNIX system). This is the Internet domain, denoted by AF_INET or AF_INET6 in C. Many of the socket calls require you to define the domain as one of their parameters.

A socket address is defined by the IP address of the socket and the port number allocated to the socket.

IP addresses

IP addresses are allocated to each TCP/IP services address on a TCP/IP Internet. Each address is a unique 32-bit (an IPv4 Internet Address) or a unique 128-bit (an IPv6 Internet Address) quantity defining the host's network and the particular host. A host can have more than one IP address if it is connected to more than one network (a so-called multihomed host).

Ports

A host can maintain several TCP/IP connections at one time. One or more applications using TCP/IP on the same host are identified by a port number. The port number is an additional qualifier used by the system software to get data to the correct application. Port numbers are 16-bit integers; some numbers are reserved for particular applications and are called well-known ports (for example, 23 is for TELNET).

Address structures

The address structure depends on the IP addressing family. An IPv4 socket address in an IP addressing family is comprised of the following four fields:

Address family
Set to AF_INET in C, or to a decimal 2 in other languages.

Port Port used by the application, in network byte order (which is explained in "Network byte order" on page 132).

IPv4 address
The IPv4 address of the network interface used by the application. It is also in network byte order.

Character array
Should always be set to all zeros.

An IPv6 socket address in an IP addressing family is comprised of the following five fields:

Address family
Set to AF_INET6 in C or to a decimal 19 in other languages.

Port Port used by the application, in network byte order (which is explained in "Network byte order" on page 132).

Flow Information
Four bytes in binary format indicating traffic class and flow label. This field is currently not implemented.

IPv6 address
The IPv6 address of the network interface used by the application. It is in network byte order.

Scope ID
Used to specify link scope for an IPv6 address as a interface index. If specified, and the destination is not link local, the socket call fails.

For COBOL, PL/I, and assembler language programs

The address structure of an IPv4 Internet socket address should be defined as follows:

Parameter	Assembler	COBOL	PL/I
IPv4 NAME STRUCTURE:			
FAMILY	H	PIC 9(4) BINARY	FIXED BIN(15)
PORT	H	PIC 9(4) BINARY	FIXED BIN(15)
ADDRESS	F	PIC 9(8) BINARY	FIXED BIN(31)
ZEROS	XL8	PIC X(8)	CHAR(8)

The address structure of an IPv6 Internet socket address should be defined as follows:

Parameter	Assembler	COBOL	PL/I
IPv6 NAME STRUCTURE:			
FAMILY	H	PIC 9(4) BINARY	FIXED BIN(15)
PORT	H	PIC 9(4) BINARY	FIXED BIN(15)
FLOWINFO	F	PIC 9(8) BINARY	FIXED BIN(31)
ADDRESS	XL16	two PIC 9(16) BINARY	CHAR(16)
SCOPE ID	F	PIC 9(8) BINARY	FIXED BIN(31)

For C programs

The structure of an IPv4 Internet socket address is defined by the *sockaddr_in* structure, which is found in the IN.H header file. The structure of an IPv6 Internet socket address structure is defined by the *sockaddr_in6* structure, which is found in the IN.H header file. The format of these structures is shown in Table 19 on page 160.

MVS address spaces

Figure 105 on page 132 shows the relationship between TCP/IP and CICS in terms of MVS address spaces.

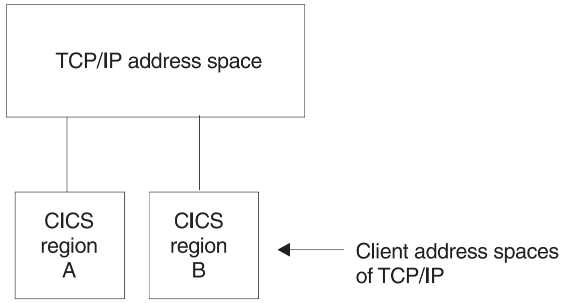


Figure 105. MVS address spaces

Within each CICS region, server and client processes are allocated subtask numbers. TCP/IP treats each CICS region together with its application programs as a client application. Because of this, the address space and subtask of each CICS TCP/IP application is called its CLIENTID. This applies to CICS TCP/IP servers as well as to clients.

A single task can support up to 65535 sockets. However, the maximum number of sockets that the TCP/IP address space is capable of supporting is determined by the value of MAXSOCKETS. Therefore, using multiple tasks, a single CICS region can support a number of sockets up to the setting of MAXSOCKETS, which has a maximum possible value of 16 777 215.

MAXFILEPROC limits the number of sockets per process. Because CICS is considered a process, MAXFILEPROC can limit the number of files allocated for the CICS region. Ensure that MAXFILEPROC is set to accommodate the total number of sockets used by all tasks running in the region.

The structure of `CLIENTID` is shown in Table 11. With CICS TCP/IP, the domain is always AF_INET, so the name (that is, address space) and subtask are the items of interest.

Table 11. CLIENTID structures

<table>
<tr><th>C structure</th><th>COBOL structure</th></tr>
<tr><td><pre>struct clientid {
    int   domain;
    char   name[8];
    char   subtaskname[8];
    char   reserved[20];
};</pre></td><td><pre>CLIENTID STRUCTURE:
     01 CLIENTID.
       02 DOMAIN PIC 9(8) BINARY.
       02 NAME PIC X(8).
       02 TASK PIC X(8).
       02 RESERVED PIC X(20).</pre></td></tr>
</table>

Network byte order

Ports and addresses are specified using the TCP/IP network byte ordering convention, which is known as big endian.

In a big endian system, the most significant byte comes first. By contrast, in a little endian system, the least significant byte comes first. MVS uses the big endian convention; because this is the same as the network convention, CICS TCP/IP applications do not need to use any conversion routines, such as `htonl`, `htons`, `ntohl`, and `ntohs`.

Note: The socket interface does not handle differences in data byte ordering within application data. Sockets application writers must handle these differences themselves.

GETCLIENTID, GIVESOCKET, and TAKESOCKET

The socket calls GETCLIENTID, GIVESOCKET, and TAKESOCKET are unique to the IBM implementation of the socket interface. In CICS TCP/IP, they are used with the EXEC CICS START and EXEC CICS RETRIEVE commands to make a socket available to a new process. This is shown in Figure 106.

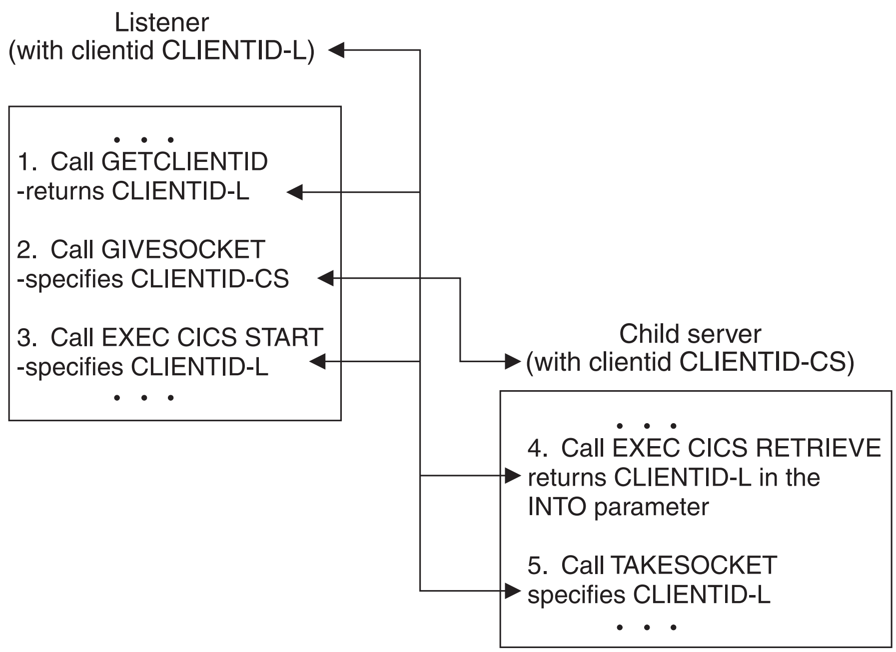


Figure 106. Transfer of CLIENTID information

Figure 106 shows the calls used to make a listener socket available to a child server process. It shows the following steps:

1. The listener calls GETCLIENTID. This returns the listener's own `CLIENTID` (`CLIENTID-L`), which comprises the MVS address space name and subtask identifier of the listener. The listener transaction needs access to its own `CLIENTID` for step 3.
2. The listener calls GIVESOCKET, specifying a socket descriptor and the `CLIENTID` of the child server.

   If the listener and child server processes are in the same CICS region (and so in the same address space), the MVS address space identifier in `CLIENTID` can be set to blanks. This means that the listener's address space is also the child's address space.

   If the listener and child server processes are in different CICS regions, enter the new address space and subtask.

   In the `CLIENTID` structure, the supplied listener sets the address space name and subtask identifier to blanks. This makes the socket available to a TAKESOCKET command from any task in the same MVS image, but only the child server receives the socket descriptor number, so the exposure is minimal. For total integrity, the subtask identifier of the child server should be entered.

3. The listener performs an EXEC CICS START. In the `FROM` parameter, the `CLIENTID-L`, obtained by the previous GETCLIENTID, is specified. The listener is telling the new child server where to retrieve its socket from in step 5.
4. The child server performs an EXEC CICS RETRIEVE. In the `INTO` parameter, `CLIENTID-L` is retrieved.
5. The child server calls TAKESOCKET, specifying `CLIENTID-L` as the process from which it wants to take a socket.

The IBM listener

In a CICS system based on SNA terminals, the CICS terminal management modules perform the functions of a concurrent server. Because the TCP/IP interface does not use CICS terminal management, CICS TCP/IP provides these functions in the form of a CICS application transaction, the listener. The CICS transaction ID of the IBM distributed listener is CSKL. This transaction is defined at installation to execute the EZACIC02 program and is to be further referenced as the listener. This transaction ID can be configured to a transaction ID suitable for the user's requirements through the use of the EZACICD macro or the EZAC CICS transaction and the accompanying RDO transaction definition.

The listener performs the following functions:

- It issues appropriate TCP/IP calls to listen on the port specified in the configuration file and waits for incoming connection requests issued by clients. The port number must be reserved in the *hlq*.TCPIP.PROFILE to the CICS region using the TCP/IP CICS sockets interface.
- It registers and deregisters with WLM for load balancing in a sysplex environment.
  - WLM registration is performed immediately after the listener socket is activated. It is performed by invoking EZACIC12, which checks the Configuration File record for the presence of WLM Group Names and performs registration for those groups specified.
  - WLM deregistration is performed for any of the following conditions:
    - Request of a listener Quiesce, by either an EZAO STOP or a CEMT PERFORM SHUTDOWN command. In this case, deregistration is done when the listening socket is closed.
    - Request for an Immediate Shutdown using an EZAO STOP. In this case, deregistration is done when the listener detects the request.
    - Abnormal termination of the listener:
      - Fatal error related to the listening socket.
      - Abend of the subtask.
      - CICS immediate termination.
      - CICS Abend.

      In these cases, deregistration is done when the listener detects the error.
- When an incoming connection request arrives, the listener accepts it and obtains a new socket to pass to the CICS child server application program.
- The standard listener starts the CICS child server transaction based on information in the first message on the new connection. The format of this information is given in "Listener input format" on page 135. For the enhanced listener, it starts the CICS child server transaction based on information in the TCP/IP CICS configuration file, EZACONFG.

- It waits for the child server transaction to take the new socket and then issues the close call. When this occurs, the receiving application assumes ownership of the socket and the listener has no more interest in it.

The listener program is written so that some of this activity goes on in parallel. For example, while the program is waiting for a new server to accept a new socket, it listens for more incoming connections. The program can be in the process of starting 49 child servers simultaneously. The starting process begins when the listener accepts the connection and ends when the listener closes the socket it has given to the child server.

Listener input format

The standard listener requires the following input format from the client in its first transmission. The client should then wait for a response before sending any subsequent transmissions. Input can be in uppercase or lowercase. The commas are required.

Note: Because the listener cannot distinguish between a comma used as a delimiter in the listener's initial message and a comma that is part of the client-in-data format, the client-in-data format should not contain a comma. In text such as x'2C' in ASCII data or such as '6B' in EBCDIC data, the single quote can be interpreted as a comma.

```
                                            ┌─kc─────────────────┐
►►──tran──,─┬──────────────────┬─,──┼────────────────────┼──────────────────────►◄
            └─client-in-data─┘    ├─ic─┬───────────────┬─┤
                                    │    └─,──hhmmss─┘ │
                                    └─td─────────────────┘
```

tran
The CICS transaction ID (in uppercase) that the listener is going to start. This field can be one to four characters.

client-in-data
Optional. Application data, used by the optional security exit [9] or the server transaction. The maximum length of this field is a 40-byte character (35 bytes, plus one byte filler and 4 bytes for startup type).

/ic/td/kc
Optional. The startup type that can be either `KC` for CICS task control, `IC` for CICS interval control or `TD` for CICS transient data. These can also be entered in lowercase (`kc,ic,` or `td`). If this field is left blank, startup is immediate using CICS task control (`KC`). `KC` or `kc` can be specified to indicate that the child server task is started using EXEC CICS START with no delay interval. This is the same as specifying IC,000000.

hhmmss
Optional. Hours, minutes, and seconds for interval time if the transaction is started using interval control. All six digits must be given.

Note: TD ignores the timefield.

9. See “Writing your own security/transaction link module for the listener” on page 143

Examples

The following are examples of client input and the listener processing that results from them. The data fields referenced can be found in "Listener output format."

Note: Parameters are separated by commas.

Example	Listener response
`TRN1,userdataishere`	It starts the CICS transaction TRN1 using task control, and passes to it the data `userdataishere` in the field CLIENT-IN-DATA.
`TRN2,,IC,000003`	It starts the CICS transaction TRN2 using interval control, without user data. There is a 3-second delay between the initiation request from the listener and the transaction startup in CICS.
`TRN3,userdataishere,TD`	It writes a message to the transient data queue named `TRN3` in the format described by the structure TCPSOCKET-PARM, described in "Listener output format." The data contained in `userdataishere` is passed to the field CLIENT-IN-DATA. This queue must be an intrapartition queue with trigger-level set to 1. It causes the initiation of transaction TRN3 if it is not already active. This transaction should be written to read the transient data queue and process requests until the queue is empty. This mechanism is provided for those server transactions that are used very frequently and for which the overhead of initiating a separate CICS transaction for each server request could be a performance concern.
`TRN3,,TD`	It causes data to be placed on transient data queue TRN3, which in turn causes the start or continued processing of the CICS transaction TRN3, as described in the TRN3 previous example. There is no user data passed.
`TRN4`	It starts the CICS transaction TRN4 using task control. There is no user data passed to the new transaction.

Listener output format

There are two different formats for the listener output; one for child server tasks started through a standard listener and one for child server tasks started through the enhanced listener.

Recommendations: The listener output format now supports an IPv6 socket address structure for both the standard and the enhanced listener. The size of the standard listener output format has increased. Child server programs should consider the following:

- A child server transaction program, using the EXEC CICS RETRIEVE function to get the data passed to it by the listener, should expand the storage it has allocated to contain the IPv6 socket address structure. The LENGTH specified on the EXEC CICS RETRIEVE function should reflect the amount of storage allocated to contain the listener output format. The LENGERR flag is raised if the LENGTH is smaller than the amount of data sent. Coding a HANDLE condition allows you to contain this.
- A child server transaction program, using the EXEC CICS READQ TD function to get the data placed on a CICS Transient Data Queue by the listener, should expand the storage it has allocated to contain the IPv6 socket address structure.

The LENGTH specified on the EXEC CICS READQ TD function should reflect the amount of storage allocated to contain the listener output format.

Table 12 shows the format of the listener output data area passed to the child server through a standard listener.

Table 12. Listener output format - Standard listener

Description	Offset	Format	Value
Socket descriptor being given to the child subtask	0	Fullword binary	Socket number to be specified on the TAKESOCKET command by the child subtask
MVS address space identifier	+4	8-byte character	Name of the listener's address space
TCP/IP task identifier	+12	8-byte character	The listener's task identifier
Data area	+20	35-byte character	Either the CLIENT-IN-DATA from the listener (if FORMAT is STANDARD) or the first 35 bytes data that was read by the listener (if FORMAT is ENHANCED)
OTE	+55	1-byte character	Indicates that the IP CICS socket interface is using CICS Open Transaction Environment. **1** Using OTE **0** Using MVS subtasks
Filler	+55	1-byte character	Unused byte for fullword alignment
Socket address structure	+56	28 bytes	
Addressing family	+56	Halfword binary	Is 2 to indicate AF_INET or 19 to indicate AF_INET6
IPv4 portion of the socket address structure	+58	26 bytes	See the next three fields
Port number	+58	Halfword binary	The client's port number
32-bit IPv4 address	+60	Fullword binary	The IPv4 address of the client's host
Unused portion	+64	8 bytes	Reserved
	+72	12 bytes	For alignment with the IPv6 socket address structure
IPv6 portion of the socket address structure	+58	26 bytes	See the next four fields
Port number	+58	Halfword binary	The client's port number
Flow Information	+60	Fullword binary	Indicates traffic class and flow label
128-bit IPv6 address	+64	16 bytes	The IPv6 address of the client's host
Scope ID	+80	Fullword binary	Indicates link scope
Reserved	+84	17 fullwords	Reserved for future use

For a standard listener, the following COBOL definition is used:

```
01  TCPSOCKET-PARM.
    05  GIVE-TAKE-SOCKET    PIC 9(8) COMP.
    05  LSTN-NAME           PIC X(8).
    05  LSTN-SUBNAME        PIC X(8).
    05  CLIENT-IN-DATA      PIC X(35).
    05  OTE                 PIC X(1).
    05  SOCKADDR-IN-PARM.
        10 SOCK-FAMILY                 PIC 9(4) BINARY.
        10 SOCK-DATA                   PIC X(26).
        10 SOCK-SIN REDEFINES SOCK-DATA.
           15 SOCK-SIN-PORT            PIC 9(4) BINARY.
           15 SOCK-SIN-ADDR            PIC 9(8) BINARY.
           15 FILLER                   PIC X(8).
           15 FILLER                   PIC X(12).
        10 SOCK-SIN6 REDEFINES SOCK-DATA.
           15 SOCK-SIN6-PORT           PIC 9(4) BINARY.
           15 SOCK-SIN6-FLOWINFO       PIC 9(8) BINARY.
           15 SOCK-SIN6-ADDR.
              20 FILLER                PIC 9(16) BINARY.
              20 FILLER                PIC 9(16) BINARY.
          15 SOCK-SIN6-SCOPEID         PIC 9(8) BINARY.
    05 FILLER                          PIC X(68).
```

Figure 107. Example of COBOL layout of the listener output format - Standard listener

```
DCL 1 TCPSOCKET_PARM,
      2 GIVE_TAKE_SOCKET          FIXED BIN(31),
      2 LSTN_NAME                 CHAR(8),
      2 LSTN_SUBNAME              CHAR(8),
      2 CLIENT_IN_DATA            CHAR(35),
      2 OTE                       CHAR(1),
      2 FILLER_1                  CHAR(1),
      2 SOCK_FAMILY               FIXED BIN(15),
      2 SOCK_SIN_PORT             FIXED BIN(15),
      2 SOCK_SIN_ADDR             FIXED BIN(31),
      2 SOCK_SIN_RESERVED         CHAR(8),
      2 SOCK_SIN_FILLER           CHAR(12),
      2 FILLER_68                 CHAR(68);
```

Figure 108. Example of PL/I layout of the listener output format - Standard listener with an IPv4 socket address structure

```
DCL 1 TCPSOCKET_PARM,
       2 GIVE_TAKE_SOCKET          FIXED BIN(31),
       2 LSTN_NAME                 CHAR(8),
       2 LSTN_SUBNAME              CHAR(8),
       2 CLIENT_IN_DATA            CHAR(35),
       2 OTE                       CHAR(1),
       2 SOCK_FAMILY               FIXED BIN(15),
       2 SOCK_SIN6_PORT            FIXED BIN(15),
       2 SOCK_SIN6_FLOWINFO        FIXED BIN(31),
       2 SOCK_SIN6_ADDR            CHAR(16),
       2 SOCK_SIN6_SCOPEID         FIXED BIN(31),
       2 FILLER_68                 CHAR(68);
```

Figure 109. Example of PL/I layout of the listener output format - Standard listener with an IPv6 socket address structure

```
TCPSOCKET_PARM DS 0C
GIVE_TAKE_SOCKET DS F
LSTN_NAME DS   CL8
LSTN_SUBNAME DS CL8
CLIENT_IN_DATA DS CL35
OTE      DS    CL1
SOCKADDR DS    0F
SOCK_FAMILY DS H
SOCK_DATA DS   0C
SOCK#LEN EQU   *-SOCKADDR
         ORG   SOCK_DATA
SOCK_SIN DS    0C
SOCK_SIN_PORT DS H
SOCK_SIN_ADDR DS CL4
         DS    CL8
         DS    20F
SOCK_SIN#LEN EQU *-SOCK_SIN
       ORG  SOCK_DATA
SOCK_SIN6 DS 0C
SOCK_SIN6_PORT DS H
SOCK_SIN6_FLOWINFO DS CL4
SOCK_SIN6_ADDR DS CL16
SOCK_SIN6_SCOPE_ID DS CL4
SOCK_SIN6#LEN EQU *-SOCK_SIN6
         ORG
         DS    CL68
```

Figure 110. Example of Assembler layout of the listener output format - Standard listener supporting both an IPv4 and an IPv6 socket address structure

```
struct sock_tim {
    unsigned long   give_take_socket;
             char   listen_name[8];
             char   listen_taskid[8];
             char   client_in_data[35];
             char   ote[1];
             union {
              struct sockaddr_in sin;
              struct sockaddr_in6 sin6;
             } sockaddr_in_parm;
             char   reserved2[68];
}
```

Figure 111. Example of C structure of the listener output format - Standard listener supporting both an IPv4 and an IPv6 socket address structure

Table 13 on page 140 shows the format of the listener output data area passed to the child server through the enhanced listener.

Note: With the enhanced listener, no CLIENT-IN-DATA is extracted from the initial client data. The child server program must either read the initial client data itself (if PEEKDATA is YES) or obtain it from DATA-AREA-2 (if PEEKDATA is NO). If a listener is converted from a standard listener to an enhanced listener, its corresponding child server applications must be changed to handle the larger transaction initial message (TIM) by specifying a large enough length on the EXEC CICS RETRIEVE command or on the EXEC CICS READQ TD command. Otherwise, the command fails with a LENGERR response and the child server task could abend.

Table 13. Listener output format - Enhanced listener

Description	Offset	Format	Value
Socket descriptor being given to the child subtask	0	Fullword binary	Socket number to be specified on the TAKESOCKET command by the child subtask
MVS address space identifier	+4	8-byte character	Name of the listener's address space
TCP/IP task identifier	+12	8-byte character	The listener's task identifier
Data area	+20	35-byte character	Either the CLIENT-IN-DATA from listener (if FORMAT is STANDARD) or the first 35 bytes of data read by the listener (if FORMAT is ENHANCED)
OTE	+55	1-byte character	Indicates that the IP CICS socket interface is using CICS's Open Transaction Environment. **1** Using OTE **0** Using MVS subtasks
Socket address structure	+56	28 bytes	
Addressing family	+56	Halfword binary	Is 2 to indicate AF_INET or 19 to indicate AF_INET6
IPv4 portion of the socket address structure	+58	26 bytes	See the next three fields
Port number	+58	Halfword binary	The client's port number
32-bit IPv4 address	+60	Fullword binary	The IPv4 address of the client's host
Unused portion	+64	8 bytes	Reserved
	+72	12 bytes	For alignment with the IPv6 socket address structure
IPv6 portion of the socket address structure	+58	26 bytes	See the next four fields
Port number	+58	Halfword binary	The client's port number
Flow Information	+60	Fullword binary	Indicates traffic class and flow label
128-bit IPv6 address	+64	16 bytes	The IPv6 address of the client's host
Scope ID	+80	Fullword binary	Indicates link scope
Reserved	+84	17 fullwords	Reserved for future use
Data length	+152	Halfword binary	The length of the data received from the client. If the PEEKDATA option was configured, Data length is zero with no data in Data area-2.
Data area - 2	+154	Length determined by the previous field	The data received from the client starting at position 1

For the enhanced listener, the following COBOL definition is used:

```
01  TCPSOCKET-PARM.
    05  GIVE-TAKE-SOCKET    PIC 9(8) COMP.
    05  LSTN-NAME           PIC X(8).
    05  LSTN-SUBNAME        PIC X(8).
    05  CLIENT-IN-DATA      PIC X(35).
    05  OTE                 PIC X(1).
    05  SOCKADDR-IN-PARM.
        10 SOCK-SIN REDEFINES SOCK-DATA.
           15 SOCK-SIN-PORT           PIC 9(4) BINARY.
           15 SOCK-SIN-ADDR           PIC 9(8) BINARY.
           15 FILLER                  PIC X(8).
           15 FILLER                  PIC X(12).
        10 SOCK-SIN6 REDEFINES SOCK-DATA.
           15 SOCK-SIN6-PORT          PIC 9(4) BINARY.
           15 SOCK-SIN6-FLOWINFO      PIC 9(8) BINARY.
           15 SOCK-SIN6-ADDR.
              20 FILLER               PIC 9(16) BINARY.
              20 FILLER               PIC 9(16) BINARY.
           15 SOCK-SIN6-SCOPEID       PIC 9(8) BINARY.
    05 FILLER                         PIC X(68).
    05 CLIENT-IN-DATA-LENGTH          PIC 9(4) BINARY.
    05 CLIENT-IN-DATA-2               PIC X(xxx).
```

Figure 112. Example of COBOL layout of the listener output format - Enhanced listener

The value of xxx is at least equal to the largest MSGLENgth parameter for the listeners that can start this application.

```
DCL 1 TCPSOCKET_PARM,
      2 GIVE_TAKE_SOCKET           FIXED BIN(31),
      2 LSTN_NAME                  CHAR(8),
      2 LSTN_SUBNAME               CHAR(8),
      2 CLIENT_IN_DATA             CHAR(35),
      2 OTE                        CHAR(1),
      2 SOCK_FAMILY                FIXED BIN(15),
      2 SOCK_SIN_PORT              FIXED BIN(15),
      2 SOCK_SIN_ADDR              FIXED BIN(31),
      2 SOCK_SIN_RESERVED          CHAR(8),
      2 SOCK_SIN_FILLER            CHAR(12),
      2 FILLER_68                  CHAR(68),
      2 CLIENT_IN_DATA_LENGTH      FIXED BIN(15),
      2 CLIENT_IN_DATA_2           CHAR(xxx);
```

Figure 113. Example of PL/I layout of the listener output format - Enhanced listener with an IPv4 socket address structure

The value of xxx is at least equal to the largest MSGLENgth parameter for the listeners that can start this application.

```
DCL 1 TCPSOCKET_PARM,
      2 GIVE_TAKE_SOCKET           FIXED BIN(31),
      2 LSTN_NAME                  CHAR(8),
      2 LSTN_SUBNAME               CHAR(8),
      2 CLIENT_IN_DATA             CHAR(35),
      2 OTE                        CHAR(1),
      2 SOCK_FAMILY                FIXED BIN(15),
      2 SOCK_SIN6_PORT             FIXED BIN(15),
      2 SOCK_SIN6_FLOWINFO         FIXED BIN(31),
      2 SOCK_SIN6_ADDR             CHAR(16),
      2 SOCK_SIN6_SCOPEID          FIXED BIN(31),
      2 FILLER_68                  CHAR(68),
      2 CLIENT_IN_DATA_LENGTH      FIXED BIN(15),
      2 CLIENT_IN_DATA_2           CHAR(xxx);
```

Figure 114. Example of PL/I layout of the listener output format - Enhanced listener with an IPv6 socket address structure

The value of xxx is at least equal to the largest MSGLENgth parameter for the listeners that can start this application.

```
TCPSOCKET_PARM DS 0C
GIVE_TAKE_SOCKET DS F
LSTN_NAME DS   CL8
LSTN_SUBNAME DS CL8
CLIENT_IN_DATA DS CL35
OTE      DS    CL1
SOCKADDR DS    0F
SOCK_FAMILY DS H
SOCK_DATA DS   0C
SOCK#LEN EQU   *-SOCKADDR
         ORG   SOCK_DATA
SOCK_SIN DS    0C
SOCK_SIN_PORT DS H
SOCK_SIN_ADDR DS CL4
         DS    CL8
         DS    20F
SOCK_SIN#LEN EQU *-SOCK_SIN
          ORG  SOCK_DATA
SOCK_SIN6 DS 0C
SOCK_SIN6_PORT DS H
SOCK_SIN6_FLOWINFO DS CL4
SOCK_SIN6_ADDR DS CL16
SOCK_SIN6_SCOPE_ID DS CL4
SOCK_SIN6#LEN EQU *-SOCK_SIN6
         ORG
         DS    CL68
CLIENT_IN_DATA_LENGTH DS H
CLIENT_IN_DATA_2 DS 0CL
```

Figure 115. Example of assembler layout of the listener output format - Enhanced listener supporting both an IPv4 and an IPv6 socket address structure

```
struct sock_tim {
    unsigned long   give_take_socket;
             char   listen_name[8];
             char   listen_taskid[8];
             char   client_in_data[35];
             char   ote[1];
             union {
               struct sockaddr_in sin;
               struct sockaddr_in6 sin6;
             } sockaddr_in_parm;
             char   reserved2[68];
             short  client_in_data_length;
             char   client_in_data_2[xxx];
}
```

Figure 116. Example of C structure of the listener output format - Enhanced listener supporting both an IPv4 and an IPv6 socket address structure

The value of xxx is at least equal to the largest MSGLENgth parameter for the listeners that can start this application.

Writing your own security/transaction link module for the listener

The listener process provides an exit point for those users who want to write and include a module that performs the following:

- Check to indicate whether the expanded security/transaction input format is used
- Security check before a CICS transaction is initiated

The exit point is implemented so that if a module is not provided, all valid transactions are initiated.

If you write a security/transaction module, you can name it anything you want, as long as you define it in the configuration data set. (In previous releases, you needed to name the module EZACICSE; you can still use that module name. You can write this program in COBOL, PL/I, or assembler language and must provide an appropriate CICS program definition.

Note: Specify the name of the security/transaction module in the SECEXIT field in Alter or Define. If you do not name the module, CICS assumes you do not have one. See Figure 63 on page 82 for more information.

Just before the child server task creation process, the listener invokes the security/transaction module by a conditional CICS LINK passing a COMMAREA. The listener passes a data area to the module that contains information for the module to use for security checking and a 1-byte switch. Your security/transaction module should perform a security check and set the switch accordingly. Included in this data is the OTE indicator which indicates when the IP CICS socket interface is using CICS's open transaction environment. The security exit should follow threadsafe programming practices to ensure that CICS continues to execute the listener on an open API TCB.

When the security/transaction module returns, the listener checks the state of the switch and initiates the transaction if the switch indicates security clearance. The module can perform any function that is valid in the CICS environment. Excessive processing, however, could cause performance degradation.

A field is supplied to indicate if the expanded security/transaction input format is used. If used, fields also exist for the listener's IP address and port number, a data length field, and a second data area (up to MSGLENTH in length). Table 14 shows the data area used by the security/transaction module.

Table 14. Security/transaction exit data

Description	**Offset**	**Format**	**Value**
CICS transaction identifier	0	4-byte character	CICS transaction requested by the client or supplied by the CSTRANID parameter.
Data area	+4	35-byte character	If the FORMAT parameter value is STANDARD, then this contains the 35-byte application data that was extracted from the client's initial data. Otherwise, it contains up to the first 35 bytes of data sent by the client (The MSGLENTH value determines the limit).
Security/transaction exit data level	+39	1-byte character	Indicates whether or not this data area is in the expanded format: **1** Expanded format (the area in green is included) **0** Not expanded (the area in green is not included)
OTE indicator	+40	1-byte character	Indicates whether the IP CICS socket interface is using CICS's open transaction environment. **1** Using OTE **0** Using MVS subtasks
TTLS indicator	+41	1-byte character	Indicates whether this connection is secured using AT-TLS. **1** This connection is secured using AT-TLS **0** This connection is not secured using AT-TLS
Register Application Data	+42	1-byte character	Indicates that application data is registered against the accepted connection to be given. This flag has the value 1 when either the LAPPLD value is yes or the LAPPLD parameter inherited the APPLDAT=YES specification. **1** Application data is registered **0** Application data is not registered
Reserved	+43	1-byte character	Reserved for IBM use.

Table 14. Security/transaction exit data (continued)

Description	Offset	Format	Value
Action	+44	2-byte character	Method of starting the task: **IC** Interval control **KC** Task control **TD** Transient data
Interval control time	+46	6-byte character	Interval requested for IC start. Has the form *hhmmss*.
Address family	+52	Halfword binary	Network address family. The value contains a 2 to indicate AF_INET and a 19 to indicate AF_INET6.
Client's port	+54	Halfword binary	The number of the requestor's port.
Client's IPv4 address	+56	Fullword binary	The IPv4 address of the requestor's host.
Switch	+60	1-byte character	**1** Permit the transaction **Not 1** Prohibit the transaction
Switch-2	+61	1-byte character	**1** Listener sends message to the client **Not 1** Security/transaction exit sends message to client
Terminal identification	+62	4-byte character	Return binary zeroes if no terminal is to be associated with the new task. Otherwise, return the CICS terminal ID to be associated with the new task.
Socket descriptor	+66	Halfword binary	Current socket descriptor.

Table 14. Security/transaction exit data (continued)

Description	Offset	Format	Value
User ID	+68	8-byte character	A user ID can be returned so that it is associated with the new task. This is mutually exclusive from terminal ID. • If the GETTID value is YES in the listener definition and the listener is able to obtain the user ID that is associated with the connection client's certificate, then this field is initialized using that user ID. Otherwise, it is initialized as binary zeroes. The security exit can use that user ID to identify the client. • If the security exit permits the transaction and does not overwrite this field, then the child server task inherits this user ID (unless the start type is TD). • If the security exit overwrites this field with nulls or blanks, then the child server inherits the listener task's user ID (unless the start type is TD). • If the security exit overwrites this field with another user ID, then the child server task inherits that user ID (unless the start type is TD). The user ID under which the listener executes must have RACF surrogate authority to use any user ID that can be specified by this field. See the *CICS RACF Security Guide* for details.
Listener's IPv4 address	+76	Fullword binary	The local IPv4 address associated with this new TCP/IP connection.
Listener's port	+80	Halfword binary	The listener's port number.
Listener's IPv6 address	+82	16 bytes binary	The local IPv6 address associated with this new TCP/IP connection.
Listener's scope ID	+98	Fullword binary	The scope ID of the listener's IPv6 address.
Client's IPv6 address	+102	16 bytes binary	The IPv6 address of the requestor's host.
Client's scope ID	+118	Fullword binary	The scope ID of the listener's IPv6 address.
Client's certificate length	+122	Halfword binary	Indicates whether the client's certificate exists.

Table 14. Security/transaction exit data (continued)

Description	Offset	Format	Value
Client's certificate address	+124	Fullword binary	The address of the client's certificate.
Reserved	+128	34 bytes	Reserved for future use.
Data length	+162	Halfword binary	The length of the data received from the client.
Data area - 2	+164	Length determined by the previous field	The data received from the client starting at position 1. If this is the enhanced listener, the first 35 bytes are the same as Data Area-1.

Notes:

1. The security/user exit can change the value of the following fields:
   - CICS transaction identifier
   - Data area
   - Action
   - Register Application Data
   - Interval control time
   - Address family
   - Client's port
   - Client's IPv4 address
   - Switch
   - Terminal identification (output only)
   - User ID
   - Client's IPv6 address
   - Client's Scope ID
   - Data length
   - Data area -2
2. Although the security exit can alter the contents of the Data area, Data length, and Data area -2 fields when PEEK=YES, the changed values are not reflected to the child server in the listener input data. The child server must read the data itself if the listener is configured with PEEK=YES.

Use the EZACICSX assembler macro contained in the *hlq*.SEZACMAC dataset to format the security/user exit COMMAREA pass by the listener.

Table 15 illustrates the listener configuration in contrast with the connected clients address family and indicates the contents of the IPv4 and IPv6 IP address fields presented to the security/transaction exit.

Table 15. Listener configuration presented to security/transaction exit

Listeners AF configuration	Connected client's AF	Exits address family	Exits client's IPv4 address	Exits client's IPv6 address	Exits listener's IPv4 address	Exits listener's IPv6 address
not specified	AF_INET	AF_INET	IPv4 addr	zeros	IPv4 addr	zeros
AF_INET	AF_INET	AF_INET	IPv4 addr	zeros	IPv4 addr	zeros

Table 15. Listener configuration presented to security/transaction exit (continued)

Listeners AF configuration	Connected client's AF	Exits address family	Exits client's IPv4 address	Exits client's IPv6 address	Exits listener's IPv4 address	Exits listener's IPv6 address
AF_INET6	AF_INET	AF_INET6	zeros	IPv4 mapped IPv6 addr	zeros	IPv4 mapped IPv6 addr
AF_INET6	AF_INET6	AF_INET6	zeros	IPv6 addr	zeros	IPv6 addr

Threadsafe considerations for IP CICS sockets applications

This topic describes how to enable IP CICS sockets applications to exploit the Open Transaction Environment (OTE) through threadsafe programming.

The IP CICS socket interface includes the IP CICS sockets task-related user exit, EZACIC01, which is invoked when an application program makes an EZASOKET request. This includes the following programs:

- EZASOKET
- EZACICSO
- EZACICAL
- using any of the IP CICS C sockets functions that are provided through EZACIC17 (Programs using IP CICS sockets functions that are provided though EZACIC07 are not considered threadsafe due to not being re-entrant.)

The IP CICS socket interface manages the process of transferring to TCP/IP and returning control to the application program when EZASOKET processing is complete.

When the IP CICS sockets configuration option is specified as OTE=NO, then the IP CICS sockets task-related user exit operates as a quasi-reentrant task-related user exit program. It runs on the CICS main TCB (the QR TCB) and uses its own MVS subtask TCB to process the EZASOKET request. However, when the IP CICS sockets configuration option is specified as OTE=YES, then the IP CICS socket interface exploits the Open Transaction Environment (OTE) to enable the IP CICS sockets task-related user exit to invoke and return from TCP/IP without switching TCBs. In the OTE, the IP CICS sockets task-related user exit operates as a threadsafe and open API task-related user exit program; it is automatically enabled using the OPENAPI option on the ENABLE PROGRAM command during connection processing. This enables it to receive control on an open L8 mode TCB.

In the OTE, if the user application program that invoked the task-related user exit conforms to threadsafe coding conventions and is defined to CICS as threadsafe, it can also run on the L8 TCB. Before its first EZASOKET request, the application program runs on the CICS main TCB, the QR TCB. When it makes an EZASOKET request and invokes the task-related user exit, control passes to the L8 TCB, and IP CICS sockets processing is carried out. On return from TCP/IP, if the application program is threadsafe, it continues to run on the L8 TCB.

When the correct conditions are met, the use of open TCBs for IP CICS sockets applications decreases usage of the QR TCB, and avoids TCB switching. An ideal IP CICS sockets application program for the open transaction environment is a threadsafe program, containing only threadsafe EXEC CICS commands, and using only threadsafe user exit programs. An application like this moves to an L8 TCB when it makes its first EZASOKET request, and then continues to run on an L8 TCB through any amount of IP CICS sockets requests and application code, requiring no TCB switching. This situation produces a significant performance improvement where an application program issues multiple EZASOKET calls. The gains are also significant when making a DB2 request because the DB2 task-related user exit also operates as threadsafe and exploits the open transaction environment. If the application program does not issue many EZASOKET calls, the performance benefits might not be as significant.

If the execution of a user application involves any actions that are not threadsafe, CICS switches back to the QR TCB. Such actions are non-threadsafe CICS requests issued by the program, the use of non-threadsafe task-related user exits, and the involvement of non-threadsafe global user exits. Switching back and forth between the open TCB and the QR TCB is detrimental to the application's performance.

Requirements: In order to gain the performance benefits of the OTE for IP CICS sockets applications, you must meet the following conditions:
- IP CICS sockets must be configured to use the Open Transaction Environment with the OTE=YES configuration option.
- The system initialization parameter FORCEQR must be set to NO. FORCEQR forces programs defined as threadsafe to run on the QR TCB; it can be set to YES as a temporary measure while problems connected with threadsafe-defined programs are investigated and resolved. FORCEQR applies to all programs defined as threadsafe that are not invoked as task-related user exits, global user exits, or user-replaceable modules.
- The IP CICS sockets application must have threadsafe application logic (that is, the native language code in between the EXEC CICS commands must be threadsafe), use only threadsafe EXEC CICS commands, and be defined to CICS as threadsafe. Only code that has been identified as threadsafe is permitted to execute on open TCBs. If your IP CICS sockets application is not defined as threadsafe, or if it uses EXEC CICS commands that are not threadsafe, TCB switching occurs and some or all of the performance benefits of OTE exploitation are lost. If your IP CICS sockets application is defined as threadsafe and it contains non-threadsafe code between the EXEC CICS commands, unpredictable results can occur.
- Any global user exits on the execution path used by the application must be coded to threadsafe standards and defined to CICS as threadsafe.
- Any other task-related user exits used by the application must be defined to CICS as threadsafe or enabled as OPENAPI.

See the *CICS Application Programming Guide* for information about how to make application programs and user exit programs threadsafe. By defining a program to CICS as threadsafe, you are specifying that only the application logic is threadsafe, not that all the EXEC CICS commands included in the program are threadsafe. CICS can ensure that EXEC CICS commands are processed safely by switching to the QR TCB for those commands not yet converted that must be quasi-reentrant. To permit your program to run on an open TCB, CICS requires you to verify that your application logic is threadsafe.

The EXEC CICS commands that are threadsafe, and do not involve TCB switching, are indicated in the command syntax diagrams in the appendices of *CICS System Programming Reference*.

If a user application program in the open transaction environment is not threadsafe, the IP CICS sockets task-related user exit still runs on an L8 TCB, but the application program runs on the QR TCB throughout the task. Every time the program makes an EZASOKET request, CICS switches from the QR TCB to the L8 TCB and back again, so the performance benefits of the open transaction environment are negated.

Table 16 shows what happens when application programs with different concurrency attributes invoke the IP CICS sockets task-related user exit.

Table 16. Different concurrency attributes for IP CICS sockets task-related user exits

Program's concurrency attribute	IP CICS sockets task-related user exit's operation	Effect
QUASIRENT or THREADSAFE	Quasi-reentrant when OTE=NO	Application program and task-related user exit run under the CICS QR TCB. The task-related user exit manages its own TCBs, switching to and from them for each EZASOKET request.
QUASIRENT	Threadsafe and open API (when OTE=YES)	Application program runs under the CICS QR TCB. Task-related user exit runs under an L8 TCB, and EZASOKET calls are executed under the L8 TCB. CICS switches to and from the CICS QR and the L8 TCB for each EZASOKET call.
THREADSAFE	Threadsafe and open API (when OTE=YES)	OTE exploitation. Task-related user exit runs under an open API, L8 TCB, and EZASOKET calls are executed under the open API, L8, TCB. The application program also runs on the open API, L8, TCB when control is returned to it. No TCB switches are needed until the task terminates, or the program issues a non-threadsafe CICS command, which forces a switch back to the QR TCB for CICS to ensure resource integrity.

If you define a program with CONCURRENCY(THREADSAFE), then all routines that are statically or dynamically called from that program (for example, COBOL routines) must also be coded to threadsafe standards.

When an EXEC CICS LINK command is used to link from one program to another, the program link stack level is incremented. However, a routine that is statically called, or dynamically called, does not involve passing through the CICS command level interface, and does not cause the program link stack level to be incremented. With COBOL routines, for a static call, a simple branch and link is used when an address is resolved by the Linkage Editor. For a dynamic call, although there is a program definition involved, this is required only so Language Environment can load the program. After the load, a simple branch and link is executed. When a routine is called by either of these methods, CICS does not regard this as a change of program. The program that called the routine is still considered to be executing, and the program definition for that program is still considered to be the current one.

If the program definition for the calling program states CONCURRENCY(THREADSAFE), then the called routine must also comply with this specification. Programs with the CONCURRENCY(THREADSAFE) attribute remain on an open API TCB until they return from a EZASOKET call, and this is not appropriate for a program that is not threadsafe. For example, consider the situation where the initial program of a transaction, program A, issues a dynamic call to program B, which is a COBOL routine. Because the CICS command level interface was not involved, CICS is unaware of the call to program B, and considers the current program to be program A. Program B further issues a EZASOKET call. On return from the EZASOKET call, CICS needs to determine whether the program can remain on the open API TCB, or whether the program must switch back to the QR TCB to ensure threadsafe processing. To do this, CICS examines the CONCURRENCY attribute of what it considers to be the current program, which is program A. If program A is defined as CONCURRENCY(THREADSAFE), then CICS allows processing to continue on the open API TCB. In fact program B is executing, so if processing is to continue safely, program B must be coded to threadsafe standards.

In summary, to gain the performance benefits of the open transaction environment:

1. IP CICS sockets must be configured to use the open transaction environment by the use of the OTE=YES configuration option.
2. FORCEQR must be set to NO.
3. The IP CICS sockets application must have threadsafe application logic (that is, the native language code in between the EXEC CICS commands must be threadsafe), use only threadsafe EXEC CICS commands, and be defined to CICS as threadsafe. If the application program is not defined as threadsafe, and so must operate on the CICS QR TCB, TCB switching occurs for every EZASOKET request, even if the task-related user exit is running on an open TCB. If the application program is defined as threadsafe but uses non-threadsafe EXEC CICS commands, TCB switching occurs for every non-threadsafe EXEC CICS commands.
4. The IP CICS sockets application must use only threadsafe task-related user exits and global user exits. If any non-threadsafe exits are used, this forces a switch back to the QR TCB. If application programs are defined to CICS as CONCURRENCY(THREADSAFE) and they contain non-threadsafe code, unpredictable results can occur.

How CICS selects an L8 mode TCB

The CICS dispatcher manages the pool of L8 mode TCBs up to the limit set by the MAXOPENTCBS system initialization parameter. At any one time, the pool can consist of some TCBs that are allocated to tasks, and others that are free. For example, if the maximum number of L8 mode TCBs is set to 10, at a particular time the pool can consist of 5 TCBs, not all of which are allocated to running tasks. The CICS dispatcher attaches a new TCB when it cannot find a free TCB that is suitable. The process of allocating an L8 mode TCB is summarized in the following steps:

1. If the transaction already has an L8 mode TCB allocated, it is used.
2. If there is a free L8 mode TCB for the current subspace, it is allocated and used.
3. If the number of open TCBs is less than the MAXOPENTCBS limit, a new L8 mode TCB is created, and associated with the task's subspace.
4. If the number of open TCBs is at the MAXOPENTCBS limit, but there is a free L8 mode TCB with the wrong subspace, then the CICS dispatcher destroys it and creates a new one for the required subspace. This technique avoids suspending the task until the number of TCBs is less than the pool limit, and is called stealing. This action is recorded in the CICS dispatcher TCB mode statistics under the count of **TCB steals**.
5. If the number of open TCBs is at the MAXOPENTCBS limit and there is no free open TCB to steal, the task is suspended (with an OPENPOOL wait) until one becomes free, or the MAXOPENTCBS limit is increased.

The various events that can occur during the TCB allocation process are recorded in the dispatcher TCB pool statistics, and these are reported by the DFH0STAT statistics program.

Data conversion routines

CICS uses the EBCDIC data format, whereas TCP/IP networks use ASCII. When moving data between CICS and the TCP/IP network, your application programs must initiate the necessary data conversion. Sockets for CICS programs can use routines provided by TCP/IP Services for:

- Converting data from EBCDIC to ASCII and back (when sending and receiving data to and from the TCP/IP network) with the SEND, SENDMSG, SENDTO, READ, READV, RECV, RECVFROM, RECVMSG, WRITE, and WRITEV calls.
- Converting between bit arrays and character strings when using the SELECT or SELECTEX call.

For details of these routines, see EZACIC04, EZACIC05, and EZACIC06, EZACIC14, and EZACIC15 in Chapter 8, "Sockets extended API," on page 223.

Application Transparent Transport Layer Security

Before reading this topic, first read the Application Transparent Transport Layer Security (AT-TLS) topic of the *z/OS Communications Server: IP Configuration Guide*.

The z/OS Communications Server TCP/IP stack provides Application Transparent Transport Layer Security (AT-TLS). This allows socket applications that use the TCP protocol to transparently use the Secure Socket Layer protocol (TLS/SSL) to communicate with partners in the network. IP CICS sockets enabled applications can take advantage of this support. This requires the following:

- The TCP/IP stack must support AT-TLS. This can be determined by the TTLS parameter on the TCPCONFIG statement.
- An AT-TLS Policy configuration that matches identifiers of the CICS applications that use it. Examples of identifiers that can be used are whether the application is a listener or client, the IP addresses, and the ports that are used for communication. Note that for CICS applications, the AT-TLS identity associated with the AT-TLS environment is always the user ID of the CICS region. This is the case even if individual CICS transactions are running under their own identity.
- SSL key rings and certificates must be created for these applications. For CICS applications using SSL, the user ID that is associated with the keyring is that of

the CICS region. See the *z/OS Communications Server: IP Configuration Guide* for the RACF commands necessary for creating SSL keyrings and certificates. See the *z/OS Security Server RACF Security Administrator's Guide* for more information on setting up and managing digital certificates.

- For policy level or application level (such as GETTID) support that requires mapping SSL Certificates to RACF user IDs see the *z/OS Communications Server: IP Configuration Guide* for more information.

Careful consideration must be given for IP CICS sockets-enabled applications that act as clients connecting outbound because the AT-TLS policy might not be specific enough to restrict individual CICS users from logging on to and invoking these clients. Additional CICS security controls such as transaction security and resource security can be considered in order to limit users' access to remote hosts. See "Example of outbound AT-TLS support" on page 154 for more information.

If a CICS listener is AT-TLS enabled but the client does not use SSL, there is a mismatch; AT-TLS receives unencrypted data when it is expecting encrypted data. In this case, AT-TLS resets the connection. See the Application Transparent Transport Layer Security (AT-TLS) topic in the *z/OS Communications Server: IP Configuration Guide* for information regarding defining keyrings, client certificates, mapping them to user IDs, permitting users access to keyrings, and other AT-TLS details.

When taking advantage of AT-TLS support, CICS application programmers and TCP/IP administrators must work together to provide the required support. This can also require communication with RACF administrators.

Example of inbound AT-TLS support

No inbound AT-TLS support is needed for listener port 3010, inbound AT-TLS support needed for listener port 3011.

Table 17. Inbound AT-TLS support

AT-TLS Definitions	CICS listener Parameters

<pre>TTLSRule   CSKLrule
 {
LocalPortRange   3010
Direction   Inbound
TTLSGroupActionRef   NOTTLSGR
}
TTLSGroupAction   NOTTLSGR
{
TTLSEnabled   OFF
}
 	
TTLSRule   CSKMrule
{
LocalPortRange   3011
Direction   Inbound
TTLSGroupActionRef   TTLSGRP1
TTLSEnvironmentActionRef TTLSENV1
}
TTLSEnvironmentAction   TTLSENV1
{
HandshakeRole   ServerWithClientAuth
EnvironmentUserInstance 1
TTLSEnvironmentAdvancedParmsRef TTLSADV1
}
TTLSEnvironmentAdvancedParms   TTLSADV1
{
ClientAuthType   SAFcheck
}
TTLSGroupAction   TTLSGRP1
{
TTLSEnabled   ON
}</pre>
	<pre>TRANID ===> CSKL
	PORT ===> 03010
	GETTID ===> NO
	TRANID ===> CSKM
	PORT ===> 03011
	GETTID ===> YES</pre>

Example of outbound AT-TLS support

No outbound AT-TLS support is needed for remote port 3010, outbound AT-TLS support needed for remote port 3011

Table 18. Outbound AT-TLS support

AT-TLS Definitions

<pre>TTLSRule   ClientRule1
 {
RemotePortRange   3010
Userid   CICS1
Direction   Outbound
TTLSGroupActionRef   NOTTLSGR
}
TTLSGroupAction   NOTTLSGR
{
TTLSEnabled   OFF
}
 	
TTLSRule   ClientRule2
{
RemotePortRange   3011
Direction   Outbound
TTLSGroupActionRef   TTLSGRP2
TTLSEnvironmentActionRef TTLSENV2
}
TTLSEnvironmentAction   TTLSENV2
{
HandshakeRole   Client
EnvironmentUserInstance 1
}
TTLSGroupAction   TTLSGRP2
{
TTLSEnabled   ON
}</pre>

Chapter 7. C language application programming

This topic describes the C language API provided by CICS TCP/IP and contain the following topics:

- "C socket library" lists the required header files and explains how to make them available to your programs.
- "C socket compilation" on page 158 shows how to compile a C socket program that contains calls to sockets for CICS.
- "Structures used in socket calls" on page 160 lists data structures used in C language socket calls.
- "The ERRNO variable" on page 163 describes the use of a global variable used by the socket system to report errors.
- "C socket calls" on page 163 describes the syntax and semantics of the socket calls and explains what they do and how they work together in the context of an application.

C socket library

To use the socket routines described in this topic, you must include these header files:

```
fnctl.h            manifest.h (non-reentrant programs only)
if.h               cmanifes.h (reentrant programs only)
in.h               ezacichd.h (non-reentrant programs only)
inet.h             errno.h    (reentrant programs only)
ioctl.h            netdb.h
bsdtypes.h         socket.h
rtrouteh.h         uio.h
ezbztlsc.h (if using IOCTL calls related to AT-TLS)
```

The files are in the SEZACMAC data set, which must be concatenated to the SYSLIB DD in the compilation JCL (as described in Step **2** of "C socket compilation" on page 158). These files carry a .h extension in this text to distinguish them as header files.

In the IBM implementation, you must include either manifest.h (if the program is non-reentrant) or cmanifes.h (if the program is reentrant) to remap function long names to eight-character names. To reference manifest.h or cmanifes.h, you need to include one of the following statements as the first #include at the beginning of each program:

```
Non-reentrant programs:
#include <manifest.h>

Reentrant programs:
#include <cmanifes.h>
```

Include the following definition to expose the required IPv6 structures, macros and definitions in the header files above:

```
#define __CICS_IPV6
```

Include the following definition to expose structures, macros and definitions in the TCP C header files previously listed:

```
#define __CICS_SOCKETS
```

C socket compilation

To compile a C socket program that contains calls to CICS TCP/IP, you must change the standard procedure for C socket compilation provided with CICS. The CICS sample compile procedures can be found in SDFHSAMP. You should also tailor them to the version CICS and C Compiler you have installed on your system. Figure 117 on page 159 shows a sample job for the compilation of a C socket program that contains calls to CICS TCP/IP. It includes the following modifications:

- **1** The prototyping statement is required for CICS.
- **2** In the C step (running the C socket compiler) you must concatenate the SEZACMAC data set to the SYSLIB DD.
- **3** In the PLKED step you must concatenate the *hlq*.SEZARNT1 data set to the SYSLIB DD if and only if the program is to be compiled as reentrant (that is, with the RENT option).

  Requirement: Ensure that the system administrator has performed the actions listed for Program Reentrancy in the *Restrictions for Using MVS TCP/IP API with z/OS Unix* topic in the *z/OS XL C/C++ Programming Guide*.
- **4** In the LKED step you must concatenate the SEZATCP and SEZACMTX data sets to the SYSLIB DD.
- **5** Also in the LKED step, you must add an INCLUDE for either module EZACIC07 (if the program is non-reentrant) or module EZACIC17 (if the program is reentrant).

Notes:

1. Furthermore, regarding Step 5 above, sockets for CICS application programs must include either EZACIC07 (if the program is non-reentrant) or EZACIC17 (if the program is reentrant) instead of CMIUCSOC, which is included in most C programs.
2. You must specify the compiler option of NORENT (non-reentrant) when including the module EZACIC07 and <ezacichd.h>.
3. You must specify the compiler option of RENT (reentrant) when including the module EZACIC17 and <errno.h>.
4. For more information about compiling and linking, see *z/OS XL C/C++ User's Guide* and *z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference*.
5. The IP CICS C sockets API does not support C++ programs.

```
//CICSRS1C JOB (999,POK),'CICSRS1',NOTIFY=CICSRS1,
//    CLASS=A,MSGCLASS=T,TIME=1439,
//    REGION=5000K,MSGLEVEL=(1,1)
//DFHEITDL PROC SUFFIX=1$,
//         INDEX='CICS410',
//         INDEX2='CICS410',
//CPARM='DEFINE(MVS)',      1
           ..........
//TRN      EXEC PGM=DFHEDP&SUFFIX,
//         REGION=&REG
           ..........
//*
//C        EXEC PGM=EDCCOMP,REGION=&REG,
//         COND=(7,LT,TRN),
//         PARM=(,'&CPARM')
//STEPLIB  DD DSN=&VSCCHD..&CVER..SEDCLINK,DISP=SHR
//         DD DSN=&COMHD..&COMVER..SIBMLINK,DISP=SHR
//         DD DSN=&VSCCHD..&CVER..SEDCCOMP,DISP=SHR
//SYSMSGS  DD DSN=&VSCCHD..&CVER..SEDCMSGS(EDCMSGE),DISP=SHR
//SYSLIB   DD DSN=&VSCCHD..&CVER..SEDCHDRS,DISP=SHR
//         DD DSN=&INDEX..SDFHC370,DISP=SHR
//         DD DSN=&INDEX..SDFHMAC,DISP=SHR
//         DD DSN=hlq.SEZACMAC,DISP=SHR  2
//SYSLIN   DD DSN=&&LOAD,DISP=(,PASS),
//            UNIT=&WORK,SPACE=&WRKSPC,DCB=&DCB80
//SYSPRINT DD SYSOUT=&OUTC
//SYSCPRT  DD SYSOUT=&OUTC
//SYSTERM  DD DUMMY
//SYSUT1   DD DSN=&&SYSUT1,DISP=(,PASS),
//            UNIT=&WORK,SPACE=&WRKSPC,DCB=&DCB80
           ..........
//SYSUT10  DD DUMMY
//SYSIN    DD DSN=*.TRN.SYSPUNCH,DISP=(OLD,DELETE)
//*
//COPYLINK EXEC PGM=IEBGENER,COND=((7,LT,C),(7,LT,TRN))
           ..........
//*
//PLKED    EXEC PGM=EDCPRLK,COND=((7,LT,C),(7,LT,TRN)),  3
//         REGION=&REG,PARM='&PPARM'
//SYSLIB   DD DSN=hlq.SEZARNT1 (reentrant programs only)
           ..........
//*
//LKED     EXEC PGM=IEWL,REGION=&REG,
//         PARM='&LNKPARM',
//         COND=((7,LT,C),(7,LT,PLKED),(7,LT,TRN))
//SYSLIB   DD DSN=&INDEX2..SDFHLOAD,DISP=SHR
//         DD DSN=&VSCCHD..&CVER..SEDCBASE,DISP=SHR
//         DD DSN=&COMHD..&COMVER..SIBMBASE,DISP=SHR
//         DD DSN=hlq.SEZATCP,DISP=SHR  4
//         DD DSN=hlq.SEZACMTX,DISP=SHR  4
//SYSLIN   DD DSN=*.PLKED.SYSMOD,DISP=(OLD,DELETE)
//         DD DSN=*.COPYLINK.SYSUT2,DISP=(OLD,DELETE)
//         DD DDNAME=SYSIN
//SYSLMOD  DD DSN=CICSRS2.CICS410.PGMLIB,DISP=SHR
//*RESLIB   DD DSN=&IMSIND..RESLIB,DISP=SHR
//SYSUT1   DD DSN=&&SYSUT1L,DISP=(,PASS),
//            UNIT=&WORK,SPACE=&WRKSPC,DCB=&DCB80
```

Figure 117. Modified JCL for C socket compilation (Part 1 of 2)

```
//SYSPRINT DD SYSOUT=&OUTC
// PEND
//APPLPROG EXEC DFHEITDL
//TRN.SYSIN  DD DISP=SHR,DSN=CICSRS1.JCL.DATA(SICUCCLD)
//LKED.SYSIN DD *
INCLUDE SYSLIB(EZACIC07) (non-reentrant programs only)  5
INCLUDE SYSLIB(EZACIC17) (reentrant programs only)  5
  NAME SICUCCLD(R)
/*
```

Figure 117. Modified JCL for C socket compilation (Part 2 of 2)

Structures used in socket calls

The parameter lists for some C language socket calls include a pointer to a data structure defined by a C structure. The structures are defined in the header files in.h,, socket.h, and if.h. Table 19 shows the C structure calls.

Table 19. C structures

C structure	Format

clientid	`struct clientid {`
	`   int       domain;`
	`   char      name[8];`
	`   char      subtaskname[8];`
	`   char      reserved[20];`
	`};`

ifconf
 	
Used in the ioctl() call only
	`struct   ifconf {`
	`    int  ifc_len;`
	`    union {`
	`    caddr_t   ifcu_buf;`
	`    struct    ifreq *ifcu_req;`
	`    } ifc_ifcu;`
	`};`

ifreq
 	
Used in the ioctl() call only
	`struct   ifreq {`
	`#define  IFNAMSIZ  16`
	`    char    ifr_name[IFNAMSIZ];`
	`    union {`
	`    struct  sockaddr ifru_addr;`
	`    struct  sockaddr ifru_dstaddr;`
	`    struct  sockaddr ifru_broadaddr;`
	`    short   ifru_flags;`
	`    int     ifru_metric;`
	`    caddr_t ifru_data;`
	`    } ifr_ifru;`
	`};`

NetConfHdr
 	
Used in the ioctl() call only
	`struct HomeIf {`
	`struct in6_addr HomeIfAddress;`
	`};`
	`struct NetConfHdr {`
	`       char     NchEyeCatcher[4];`
	`       uint32_t NchIOCTL;`
	`       int32_t  NchBufferLength;`
	`       union {`
	`         struct HomeIf * __ptr32 NchIfHome;`
	`         struct GRT6RtEntry * __ptr32`
	`              NchGRT6RtEntry;`
	`         } NchBufferPtr;`
	`         int32_t  NchNumEntryRet;`
	`};`

Table 19. C structures (continued)

<table>
<tr><th>C structure</th><th>Format</th></tr>
<tr><td>If_NameIndex

Used in the if_freenameindex(), if_indextoname(), if_nameindex(), and if_nametoindex() calls</td><td><pre>struct if_nameindex {
unsigned int if_index;
char * if_name;
};</pre></td></tr>
<tr><td>linger

Used in the getsockopt() and setsockopt() calls only</td><td><pre>struct linger {
    int   l_onoff;
    int   l_linger;
};</pre></td></tr>
<tr><td>ip_mreq

Used in the setsockopt() call only</td><td><pre>struct ip_mreq {
            struct in_addr imr_multiaddr;
            struct in_addr imr_interface;
};</pre></td></tr>
<tr><td>ipv6_mreq

Used in the setsockopt() call only</td><td><pre>struct ipv6_mreq {
      struct in6_addr ipv6mr_multiaddr;
      unsigned int ipv6mr_interface;
};</pre></td></tr>
<tr><td>sockaddr_in</td><td><pre>struct in_addr
{
        unsigned long s_addr;
};
struct sockaddr_in {
    short   sin_family;
    ushort sin_port;
    struct in_addr sin_addr;
    char   sin_zero[8];
};</pre></td></tr>
<tr><td>sockaddr_in6</td><td><pre>struct in6_addr {
      union {
        uint8_t _S6_u8[16];
        uint32_t _S6_u32[4];
      } _S6_un;
    };
    struct sockaddr_in6 {
     uint8_t   sin6_len;
     sa_family_t   sin6_family;
     in_port_t   sin6_port;
     uint32_t   sin6_flowinfo;
     struct in6_addr sin6_addr;
     uint32_t   sin6_scope_id;
};</pre></td></tr>
</table>

Table 19. C structures (continued)

C structure	Format

addrinfo
 	
Use in the getaddrinfo()
 	
and freeaddrinfo() calls
	struct addrinfo {
	int ai_flags;
	int ai_family;
	int ai_socktype;
	int ai_protocol;
	socklen_t ai_addrlen;
	char *ai_canonname;
	struct sockaddr *ai_addr;
	struct addrinfo *ai_next;
	};

timeval
 	
Used in the select() call only
	struct timeval {
	long tv_sec;
	long tv_usec;
	};

ip_mreq_source
 	
Used in the setsockopt() call only
	struct ip_mreq_source {
	struct in_addr imr_multiaddr;
	struct in_addr imr_sourceaddr;
	struct in_addr imr_interface;
	};

group_req
 	
Used in the setsockopt() call only
	struct group_req {
	uint32_t gr_interface;
	uint32_t __gr_01;
	struct sockaddr_storage gr_group;
	};

group_source_req
 	
Used in the setsockopt() call only
	struct group_source_req {
	uint32_t gsr_interface;
	uint32_t __gsr_01;
	struct sockaddr_storage gsr_group;
	struct sockaddr_storage gsr_source;
	};

SetApplData Used in the SIOCSAPPLDATA ioctl() call	#define SetAD_eye1 "SETAPPLD" #define SETADVER 1 struct { char SetAD_eye1[8]; short SetAD_ver; short SetAD_len; char SetAD_rsv[4]; #ifndef _LP64 int SetAD_ptrHW; #endif SetADcontainer *SetAD_ptr; } SetApplData;
SetADcontainer Used in the SIOCSAPPLDATA ioctl() call	#define SETADEYE2 "APPLDATA" typedef struct { char SetAD_eye2[8]; char SetAD_buffer[40]; } SetADcontainer;

The ERRNO variable

The global variable *errno* is used by the socket system calls to report errors. If a socket call results in an error, the call returns a negative value, and an error value is set in *errno*. To be able to access these values, you must add one of the following include statements:

```
Non-reentrant programs:
#include <ezacichd.h>

Reentrant programs:
#include <errno.h>
```

Notes:

1. Do not use tcperror().
2. A copy of EZACICHD.H can be found in dataset *hlq*.SEZAINST.

C socket calls

This topic contains guidance for each C socket call supported by CICS TCP/IP.

For syntax, parameters, and other reference information for each C socket call, see *z/OS Communications Server: IP Programmer's Guide and Reference*.

accept()

A server issues the accept() call to accept a connection request from a client. The call uses a socket already created with a socket() call and marked by a listen() call.

An accept() call

1. Accepts the first connection on its queue of pending connections.
2. Creates a new socket with the same properties as the socket used in the call.
3. Returns the new socket descriptor to the server.

The new socket cannot be used to accept new connections, but is used by the client for application purposes. The server issues a givesocket() call and a CICS START command to enable a child server to communicate with the client for application purposes. The original socket remains available to the server to accept more connection requests.

The accept() call optionally saves the connection requester's address for use by the server.

Notes:

1. 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 ioctl().
2. accept() calls are the only way to screen clients. The application cannot predetermine clients from which it accepts connections, but it can close a connection immediately after discovering the identity of the client.
3. The select() call checks a socket for incoming connection requests.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <bsdtypes.h>
#include <in.h>
#include <socket.h>
int accept(int s, struct sockaddr *name, int *namelen)
```

Parameters

s
: The *s* parameter is a stream socket descriptor that has already been created with the socket() call. It is usually bound to an address with the bind() call. The listen() call marks the socket as one that accepts connections and allocates a queue to hold pending connection requests. The listen() call allows the caller to place an upper boundary on the size of the queue.

name
: The pointer to a *sockaddr* structure into which the address of a client requesting a connection is placed on completion of the accept() call. If the server application does not need the client address, set the *name* parameter to the NULL pointer before making the accept() call.

The format of the name buffer is expected to be *sockaddr_in*, for an IPv4 socket address, or *sockaddr_in6*, for an IPv6 socket address, as defined in the header file in.h. The format of the structure is shown in Table 19 on page 160.

Use the following fields to define the IPv4 socket address structure for the socket that is to be accepted:

sin_family
: Field must be set to AF_INET.

sin_port
: Field contains the client's port number.

in_addr.sin_addr
: Field contains the 32-bit IPv4 Internet address, in network byte order, of the client's host machine.

sin_zero
: Field is not used and is set to all zeros.

Use the following fields to define the IPv6 socket address structure for the socket that is to be accepted:

sin6_family
: Field must be set to AF_INET6.

sin6_port
: Field contains the client's port number.

sin6_flowinfo
: Field contains the traffic class and flow label. The value of this field is undefined.

in6_addr.sin6_addr
: Field contains the 128-bit IPv6 Internet address, in network byte order, of the client's host machine.

sin6_scope_id
: Field identifies a set of interfaces as appropriate for the scope of the address carried in the *in6_addr.sin6_addr* field. For a link scope

in6_addr.sin6_addr, *sin6_scope_id* contains the link index for the *in6_addr.sin6_addr*. For all other address scopes, *sin6_scope_id* is undefined.

namelen
: The size, in bytes, of the buffer pointed to by *name*. For an IPv4 socket address, the *namelen* parameter should contain a decimal 16. For an IPv6 socket address, the *namelen* parameter should contain a decimal 28.

Return values

A nonnegative socket descriptor indicates success; the value −1 indicates an error. To determine which error occurred, check the *errno* global variable, which is set to a return code. Possible codes include:

EBADF
: The *s* parameter is not a valid socket descriptor.

EFAULT
: Using *name* and *namelen* results in an attempt to copy the address into a portion of the caller's address space into which information cannot be written.

EINVAL
: Listen() was not called for socket *s*.

ENOBUFS
: Insufficient buffer space is available to create the new socket.

EOPNOTSUPP
: The *s* parameter is not of type SOCK_STREAM.

EWOULDBLOCK
: The socket *s* is in nonblocking mode, and no connections are in the queue.

bind()

The bind() call binds a unique local port to an existing socket. Note that, on successful completion of a socket() call, the new socket descriptor does not have an associated port.

The bind() call can specify the required port or let the system choose. A listener application should always bind to the same well-known port, so that clients can know which port to use.

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 an effect similar to the application specifying an explicit IP address on the bind() function. For more information, see *z/OS Communications Server: IP Configuration Reference*.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <bsdtypes.h>
#include <socket.h>
#include <in.h>
int bind(int s, struct sockaddr *name, int namelen)
```

Parameters

s
: The socket descriptor returned by a previous socket() call.

name

The pointer to a socket address structure that contains the name that is to be bound to *s*. The format of the *name* buffer is expected to be *sockaddr_in* for an IPv4 socket address or *sockaddr_in6* for an IPv6 socket address, as defined in the header file in.h. The format of the structure is shown in Table 19 on page 160.

Use the following fields to specify the IPv4 socket address structure for the socket that is to be bound:

sin_family
: Field must be set to AF_INET.

sin_port
: Field is set to the port to which the application must bind. It must be specified in network byte order. If *sin_port* is set to 0, the caller expects the system to assign an available port. The application can call getsockname() to discover the port number assigned.

in_addr.sin_addr
: Field is set to an IPv4 IP address and must be specified in network byte order. On hosts with more than one network interface (called multihomed hosts), you can select the interface to which it is to bind. Subsequently, only TCP connection requests from this interface are routed to the application.

If you set this field to the constant INADDR_ANY, as defined in in.h, the socket is bound to all network interfaces on the host. By leaving the address unspecified with INADDR_ANY, the server can accept all TCP connection requests made for its port, regardless of the network interface on which the requests arrived. Set INADDR_ANY for servers that offer a service to multiple networks.

sin_zero
: Field is not used and must be set to all zeros.

Use the following fields to specify the IPv6 socket address structure for the socket that is to be bound:

sin6_family
: Field must be set to AF_INET6.

sin6_port
: Field is set to the port to which the application must bind. It must be specified in network byte order. If *sin6_port* is set to 0, the caller expects the system to assign an available port. The application can call getsockname() to discover the port number assigned.

sin6_flowinfo
: Field is used to specify the traffic class and flow label. This field must be set to zero.

in6_addr.sin6_addr
: Field is set to an IPv6 address and must be specified in network byte order. On hosts with more than one network interface (called multihomed hosts), you can select the interface to which it is to

bind. Subsequently, only TCP connection requests from this interface are routed to the application.

If you set this field to the constant *in6addr_any*, as defined in in.h, the socket is bound to all network interfaces on the host. By leaving the address unspecified with *in6addr_any*, the server can accept all TCP connection requests made for its port, regardless of the network interface on which the requests arrived. Set *in6addr_any* for servers that offer a service to multiple networks.

sin6_scope_id
Field is used to identify a set of interfaces as appropriate for the scope of the address carried in the *in6_addr.sin6_addr* field. A value of zero indicates the *sin6_scope_id* field does not identify the set of interfaces to be used, and might be specified for any address types and scopes. For a link scope *in6_addr.sin6_addr* field, *sin6_scope_id* might specify a link index which identifies a set of interfaces. For all other address scopes, *sin6_scope_id* must be set to zero.

namelen
The size, in bytes, of the buffer pointed to by *name*. For an IPv4 socket address, the *namelen* parameter should contain a decimal 16. For an IPv6 socket address, the *namelen* parameter should contain a decimal 28.

Return values

The value 0 indicates success; the value −1 indicates an error. To determine which error occurred, check the *errno* global variable, which is set to a return code. Possible codes include:

EADDRINUSE
The address is already in use. See the SO_REUSEADDR option described in “getsockopt(), setsockopt()” on page 187 for more information.

The address is in a timed wait because a LINGER delay from a previous close or another process is using the address. This error also occurs if the port specified in the bind call has been configured as RESERVED on a port reservation statement in the TCP/IP profile.

If you want to reuse the same address, use the SO_REUSEADDR parameter in setsockopt(). If you do not want to reuse the same address, use a different address or port in the socket address structure. If the port has been configured as RESERVED, then the port is unavailable for bind.

EADDRNOTAVAIL
The address specified is not valid on this host. For example, the IP address does not specify a valid network interface.

EAFNOSUPPORT
The address family is not supported (it is not AF_INET or AF_INET6).

EBADF
The *s* parameter is not a valid socket descriptor.

EFAULT
Using *name* and *namelen* results in an attempt to copy the address into a nonwritable portion of the caller’s address space.

EINVAL
The socket is already bound to an address. An example is trying to bind a name to a socket that is in the connected state. This value is also returned if *namelen* is not the expected length.

close()

A close() call shuts down a socket and frees all resources allocated to the socket. If the socket refers to an open TCP connection, the connection is closed. If a stream socket is closed when input data is queued, the TCP connection is reset rather than being cleanly closed.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <socket.h>
int close(int s)
```

Parameter

s The descriptor of the socket to be closed.

Return values

The value 0 indicates success; the value −1 indicates an error. To determine which error occurred, check the *errno* global variable, which is set to a return code. Possible codes include:

EBADF
The *s* parameter is not a valid socket descriptor.

connect()

A connect() call attempts to establish a connection between a local socket and a remote socket. For a stream socket, the call performs two tasks. First, it completes the binding necessary for a stream socket in case it has not been previously bound by a bind() call. Second, it attempts to make a connection to another socket.

The connect() call on a stream socket is used by a client application to establish a connection to a server. To be able to accept a connection with an accept() call, the server must have a passive open pending, which means it must have successfully called bind() and listen() before the client issues connect().

If the socket is in blocking mode, the connect() call blocks the caller until the connection is set up, or until an error is received. If the socket is in nonblocking mode and no errors occurred, the return codes indicate that the connection can be initiated. The caller can test the completion of the connection setup by calling select() and testing for the ability to write to the socket.

Stream sockets can call connect() one time only.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <bsdtypes.h>
#include <socket.h>
#include <in.h>
int connect(int s, struct sockaddr *name, int namelen)
```

Parameters

s The socket descriptor of the socket that is going to be used as the local endpoint of the connection.

name The pointer to a socket address structure that contains the destination socket address to which a connection is requested.

The format of the name buffer is expected to be *sockaddr_in* for an IPv4 socket address or *sockaddr_in6* for an IPv6 socket address, as defined in the header file in.h. The format of the structure is shown in Table 19 on page 160.

Use the following fields to specify the IPv4 socket address structure for the socket that is to be bound:

sin_family
: Field must be set to AF_INET.

sin_port
: Field is set to the port to which the server is bound. It must be specified in network byte order.

in_addr.sin_addr
: Field is set to the 32-bit IPv4 Internet address of the server's host machine in network byte order.

sin_zero
: Field is not used and must be set to all zeros.

Use the following fields to specify the IPv6 socket address structure for the socket that is to be bound:

sin6_family
: Field must be set to AF_INET6.

sin6_port
: Field is set to the port to which the server is bound. It must be specified in network byte order.

sin6_flowinfo
: Field is used to specify the traffic class and flow label. This field must be set to zero.

in6_addr.sin6_addr
: Field is set to the 128-bit IPv6 Internet address of the server's host machine in network byte order.

sin6_scope_id
: Field is used to identify a set of interfaces as appropriate for the scope of the address carried in the *in6_addr.sin6_addr* field. A value of zero indicates the *sin6_scope_id* field does not identify the set of interfaces to be used, and might be specified for any address types and scopes. For a link scope *in6_addr.sin6_addr*, *sin6_scope_id* might specify a link index which identifies a set of interfaces. For all other address scopes, *sin6_scope_id* must be set to zero.

namelen
: The size of the socket address pointed to by *name* in bytes. For an IPv4 socket address the *namelen* parameter should contain a decimal 16 and for an IPv6 socket address the *namelen* parameter should contain a decimal 28.

Return values

The value 0 indicates success; the value –1 indicates an error. To determine which error occurred, check the *errno* global variable, which is set to a return code. Possible codes include:

EADDRNOTAVAIL
The calling host cannot reach the specified destination.

EAFNOSUPPORT
The address family is not supported.

EALREADY
The socket *s* is marked nonblocking, and a previous connection attempt has not completed.

EBADF
The *s* parameter is not a valid socket descriptor.

ECONNREFUSED
The connection request was rejected by the destination host.

EFAULT
Using *name* and *namelen* results in an attempt to copy the address into a portion of the caller's address space to which data cannot be written.

EINPROGRESS
The socket *s* is marked nonblocking, and the connection cannot be completed immediately. The EINPROGRESS value does not indicate an error condition.

EINVAL
The *namelen* parameter is not a valid length.

EISCONN
The socket *s* is already connected.

ENETUNREACH
The network cannot be reached from this host.

ETIMEDOUT
The connection establishment timed out before a connection was made.

fcntl()

The fcntl() call controls whether a socket is in blocking or nonblocking mode.

The blocking or nonblocking mode of a socket affects the operation of certain commands. In blocking mode, a call waits for certain events until they happen. When this happens, the operating system suspends the program until the event occurs.

In similar situations with nonblocking calls, the call returns an error return code and the program continues.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <socket.h>
#include <bsdtypes.h>
#include <fcntl.h>
signed int fcntl(int s, int cmd, int arg)
```

Parameters

s The socket descriptor.
cmd The command to perform. Set *cmd* to one of the following:

F_SETFL
: This command sets the status flags of socket *s*. One flag, FNDELAY, can be set.

  Setting the FNDELAY flag marks *s* as being in nonblocking mode. If data is not present on calls that can block, such as recvfrom(), the call returns −1, and errno is set to EWOULDBLOCK.

F_GETFL
: This command gets the status flags of socket *s*. One flag, FNDELAY, can be queried.

  The FNDELAY flag marks *s* as being in nonblocking mode. If data is not present on calls that can block, such as recvfrom(), the call returns with −1, and errno is set to EWOULDBLOCK.

arg Set to FNDELAY if using F_SETFL. Ignored otherwise.

Return values

For the F_GETFL command, the return value is a bit mask that is comprised of the flag settings. For the F_SETFL command, the value 0 indicates success; the value −1 indicates an error. To determine which error occurred, check the *errno* global variable, which is set to a return code. Possible codes include:

EBADF
: The *s* parameter is not a valid socket descriptor.

EINVAL
: The *arg* parameter is not a valid flag.

freeaddrinfo()

The freeaddrinfo() call receives an input addrinfo structure pointer and releases that storage (plus any other chained addrinfo structures and related storage) back into the general storage pool, thereby making the getaddrinfo() call thread-safe.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <socket.h>
#include <in.h>
#include <netdb.h>

void freeaddrinfo(struct addrinfo *ai)
```

Parameters

ai A pointer to an addrinfo structure returned by the getaddrinfo() *res* function variable.

Return values

The value 0 indicates success; the value -1 indicates an error. To determine which error occurred, check the errno global variable, which is set to a return code. Possible codes include:

EAI_AGAIN
: The resolver address space has not been started. The request can be retried later.

EAI_FAIL
: An unrecoverable error has occurred.

gai_strerror()

The gai_strerror() function returns a pointer to a text string describing the error value returned by a failure return from either the getaddrinfo() or getnameinfo() function. If the *ecode* is not one of the EAI_xxx values from the <netdb.h> then gai_strerror() returns a pointer to a string indicating an unknown error. Subsequent calls to gai_strerror() overwrites the buffer that contains the text string.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <netdb.h>

const char *gai_strerror(int ecode)
```

Parameters

ecode The errno value returned by the getaddrinfo() or getnameinfo() functions.

Return values

When successful, gai_strerror() returns a pointer to a string describing the error. Upon failure, gai_strerror() returns NULL and set *errno* to the following:

ENOMEN
Insufficient memory to allocate buffer for text string describing the error.

getaddrinfo()

The getaddrinfo() call translates 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. This information is used to open a socket with which to address the specified service or to send a datagram to the specified service.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <socket.h>
#include <in.h>
#include <netdb.h>

int getaddrinfo(const char *nodename, const char *servname,
                cons struct addrinfo *hints,
                struct addrinfo **res)
```

Parameters

nodename
Maximum storage of 256 bytes that contains the null terminated host name being queried. If the AI_NUMERICHOST flag is specified in the storage pointed to by the *hints* parameter, *nodename* should contain the queried host IP address in presentation form.

| You can append scope information to the host name, using the format *nodename%scope information*. The combined length of the value specified must still fit within 256 bytes, and must still be null terminated. For information about using scope information about getaddrinfo() processing, see *z/OS Communications Server: IPv6 Network and Application Design Guide* .

servname
Maximum storage of 33 bytes that contains the null terminated service

name being queried. If the AI_NUMERICSERV flag is specified in the storage pointed to by the *hints* parameter, *servname* should contain the queried port number in presentation form.

hints Contains the address of an *addrinfo* structure that contains input values that might direct the operation by providing options and by limiting the returned information to a specific socket type, address family, and protocol. If the *hints* parameter is 0, then the information returned is as if it referred to a structure that contains the value 0 for the *ai_flags*, *ai_socktype*, and *ai_protocol fields*, and AF_UNSPEC for the *ai_family* field.

The addrinfo structure has the following fields:

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

AI_PASSIVE

Specifies how to fill in the *ai_addr* pointed to by the returned *res*.

If this flag is specified, the returned address information is 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 *nodename* parameter is null, the IP address portion of the socket address structure pointed to by the returned *res* is set to INADDR_ANY, for an IPv4 address, or to the IPv6 unspecified address (in6addr_any).

If this flag is not set, the returned address information is 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 *nodename* parameter is not specified, the *ai_addr* pointed to by the returned *res* is set to the loopback address.

This flag is ignored if the *nodename* parameter is specified.

AI_CANONNAMEOK
If this flag is specified and the *nodename* parameter is specified, the getaddrinfo() call attempts to determine the canonical name corresponding to the *nodename* parameter.

AI_NUMERICHOST
If this flag is specified, the *nodename* parameter must be a numeric host address in presentation form. Otherwise, an error of host not found [EAI_NONAME] is returned.

AI_NUMERICSERV
If this flag is specified, the *servname* parameter must be a numeric port in presentation form. Otherwise, an error [EAI_NONAME] is returned.

AI_V4MAPPED
If this flag is specified with the *ai_family* field using the value of AF_INET6, or the value of

AF_UNSPEC when IPv6 is supported on the system, the caller accepts 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 *ai_family* field does not have the value of AF_INET6, or the *ai_family* field contains AF_UNSPEC but IPv6 is not supported on the system, then this flag is ignored.

AI_ALL
If the *ai_family* 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 accepts all addresses: IPv6 and IPv4-mapped IPv6 addresses. If the *ai_family* field has a value of AF_UNSPEC when the system supports IPv6 and AI_ALL is set, the caller accepts both IPv6 and IPv4 addresses. A query is first made for IPv6 addresses and if successful, the IPv6 addresses are returned. Another query is then made for IPv4 addresses, and any IPv4 addresses found are returned as IPv4-mapped IPv6 addresses (if AI_V4MAPPED is also specified) or as IPv4 addresses (if AI_V4MAPPED is not specified). If the *ai_family* field does not have the value of AF_INET6, or does not have the value of AF_UNSPEC when the system supports IPv6, then this flag is ignored.

AI_ADDRCONFIG
If this flag is specified, then a query on the name in *nodename* occurs if the resolver determines that one of the following is true:

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

ai_family
Used to limit the returned information to a specific address family. The value of AF_UNSPEC means that the caller accepts 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.

ai_socktype
Used to limit the returned information to a specific socket type. A value of 0 means that the caller accepts any socket type. If a specific socket type is not given (for example, a value of 0), information about all supported socket types are returned.

The following are the acceptable socket types:

Type Name	Decimal Value	Description
SOCK_STREAM	1	for stream socket

Type Name	Decimal Value	Description
SOCK_DGRAM	2	for datagram socket
SOCK_RAW	3	for raw-protocol interface

Any other socket type fails with a return code of EAI_SOCKTYPE. Note that although SOCK_RAW is accepted, it is only valid when *servname* is numeric (for example, servname=23). A lookup for a service name never occurs in the appropriate services file (for example, *hlq*.ETC.SERVICES) using any protocol value other than SOCK_STREAM or SOCK_DGRAM. If *ai_protocol* is not 0 and *ai_socktype* is 0, the only acceptable input values for *ai_protocol* are IPPROTO_TCP and IPPROTO_UDP; otherwise, the getaddrinfo() function fails with a return code of EAI_BADFLAGS. If *ai_socktype* and *ai_protocol* are both specified as 0, getaddrinfo() proceeds as follows:

- If *servname* is null, or if *servname* is numeric, any returned *addrinfo* structures default to a specification of *ai_socktype* as SOCK_STREAM.
- If *servname* is specified as a service name, for example *servname*=FTP, the getaddrinfo() call searches the appropriate services file (for example, *hlq*.ETC.SERVICES) twice. The first search uses SOCK_STREAM as the protocol, and the second search uses SOCK_DGRAM as the protocol. No default socket type provision exists in this case.

If both *ai_socktype* and *ai_protocol* are specified as nonzero, then they should be compatible, regardless of the value specified by the *servname* parameter. In this context, compatibility means one of the following:

- *ai_socktype*=SOCK_STREAM and *ai_protocol*=IPPROTO_TCP
- *ai_socktype*=SOCK_DGRAM and *ai_protocol*=IPPROTO_UDP
- *ai_socktype* is specified as SOCK_RAW. In this case, *ai_protocol* can be anything.

ai_protocol
Used to limit the returned information to a specific protocol. A value of 0 means that the caller accepts any protocol.

The following are the acceptable protocols:

Protocol Name	Decimal Value	Description
IPPROTO_TCP	6	TCP
IPPROTO_UDP	17	user datagram

If *ai_protocol* and *ai_socktype* are both specified as 0, getaddrinfo() proceeds as follows:

- If *servname* is null, or if *servname* is numeric, then any returned addrinfos default to a specification of *ai_socktype* as SOCK_STREAM.

- If *servname* is specified as a service name (for example, *servname*=FTP), getaddrinfo() searches the appropriate services file (for example, *hlq*.ETC.SERVICES) twice. The first search uses SOCK_STREAM as the protocol, and the second search uses SOCK_DGRAM as the protocol. No default socket type provision exists in this case.

If both *ai_socktype* and *ai_protocol* are specified as nonzero then they should be compatible, regardless of the value specified by servname. In this context, compatibility means one of the following:
- *ai_socktype*=SOCK_STREAM and *ai_protocol*=IPPROTO_TCP
- *ai_socktype*=SOCK_DGRAM and *ai_protocol*=IPPROTO_UDP
- *ai_socktype*=SOCK_RAW. In this case, *ai_protocol* can be anything.

If the lookup for the value specified in *servname* fails [that is, the service name does not appear in the appropriate services file (for example, *hlq*.ETC.SERVICES) using the input protocol], the getaddrinfo() call fails with return code of EAI_SERVICE.

ai_addrlen
: On input, this field must be 0.

ai_canonname
: On input, this field must be 0.

ai_addr
: On input, this field must be 0.

ai_next
: On input, this field must be 0.

res
: On a successful return this field contains a pointer to an *addrinfo* structure. This pointer is also used as input to the freeaddrinfo() call, which must be used to free storage obtained by this call. The structures returned by getaddrinfo() are a tasks's serially reusable storage area. They should not be used or referenced between MVS tasks. The storage is freed when a freeaddrinfo() is issued or when the task terminates. The freeaddrinfo() call receives an input addrinfo structure pointer and releases that storage (plus any other chained addrinfo structures and related storage) back into the general storage pool, thereby making the getaddrinfo() call thread-safe.

The address information structure contains the following fields:

ai_flags
: Not used as output.

ai_family
: The value returned in this field can be used as the *domain* argument on the socket() call to create a socket suitable for use with the returned socket address pointed to by *ai_addr*.

ai_socktype
: The value returned in this field can be used as the *type* argument on the socket() call to create a socket suitable for use with the returned address socket pointed to by *ai_addr*.

ai_protocol
: The value returned in this field can be used as the *protocol* argument on the socket() call to create a socket suitable for use with the returned socket address pointed to by *ai_addr*.

ai_addrlen
: The length of the socket address structure pointed to by the *ai_addr* field. The value returned in this field can be

used as the arguments for the connect() or bind() call with this socket type, according to the AI_PASSIVE flag.

ai_canonname
A pointer to the canonical name for the value specified by *nodename*. If the *nodename* argument is specified, and if the AI_CANONNAMEOK flag was specified by the *hints* parameter, the *ai_canonname* field in the first returned address information structure contains the address of storage that contains the canonical name corresponding to the input *nodename* parameter. If the canonical name is not available, the *ai_canonname* field refers to the *nodename* parameter or a string with the same contents.

ai_addr
The address of the returned socket address structure. The value returned in this field can be used as the arguments for the connect() or bind() call with this socket type, according to the AI_PASSIVE flag.

ai_next
Contains the address of the next address information structure on the list, or zeros if it is the last structure on the list.

Return values

The value 0 indicates success; the value -1 indicates an error. To determine which error occurred, check the *errno* global variable, which is set to a return code. Possible codes include:

EAI_AGAIN
The name specified by the *nodename* parameter could be not be resolved within the configured time interval, or the resolver address space has not been started. The request can be retried later.

EAI_BADFLAGS
The flags parameter had a value that is incorrect.

EAI_BADFLAGS
The flags parameter had a value that is incorrect.

EAI_FAMILY
The family parameter has a value that is incorrect.

EAI_MEMORY
Memory allocation failure occurred trying to acquire an addrinfo structure.

EAI_NONAME
The name does not resolve for the specified parameters. At least one of the *nodename* or *servname* parameters must be specified. Or the requested nodename parameter is valid but does not have a record at the name server.

EAI_SERVICE
The service passed was not recognized for the specified socket type.

EAI_SOCKTYPE
The intended socket type was not recognized.

getclientid()

A getclientid() call returns the identifier by which the calling application is known to the TCP/IP address space. Do not be confused by the term *client* in the name of this call; the call always returns the ID of the calling process, be it client or server.

For example, in CICS TCP/IP, this call is issued by the IBM listener; the identifier returned in that case is that of the listener (a server). This identifier is used in the givesocket() and takesocket() calls.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <bsdtypes.h>
#include <socket.h>
int getclientid(int domain, struct clientid *clientid)
```

Parameters

domain The domain must be set to AF_INET when requesting client data from an IPv4 stack and it must be set to AF_INET6 when requesting client data from an IPv6 stack.

clientid Points to a clientid structure to be provided.

- *domain* Domain associated with the program executing this call. Contains either AF_INET (a decimal 2) or AF_INET6 (a decimal 19).
- *name* Address space name associated with the program executing this call.
- *subtaskname* Subtask name associated with the program executing this call.
- *reserved* Binary zeros.

Return values

The value 0 indicates success; the value −1 indicates an error. To determine which error occurred, check the *errno* global variable, which is set to a return code. Possible codes include:

EFAULT
Using the *clientid* parameter as specified results in an attempt to access storage outside the caller’s address space, or storage not modifiable by the caller.

EPFNOSUPPORT
Domain is not AF_INET or AF_INET6.

gethostbyaddr()

The gethostbyaddr() call tries to resolve the IP address to a host name. The 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 using local host tables.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <netdb.h>
 struct hostent *gethostbyaddr(char *addr, int addrlen, int domain)
```

Parameters

addr The pointer to an unsigned long value that contains the address of the host.

addrlen
The size of *addr* in bytes.

domain The address domain supported (AF_INET).

Return values

The gethostbyaddr() call returns a pointer to a hostent structure for the host address specified on the call. For more information about the hostent structure, see Figure 128 on page 250. A null pointer is returned if the gethostbyaddr() call fails.

There are no errno values for gethostbyaddr().

gethostbyname()

The gethostbyname() call tries to resolve the host name to an IP address. The 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 using local host tables.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <netdb.h>
 struct hostent *gethostbyname(char *name)
```

Parameters

name The name of the host being queried. The name has a maximum length of 255 characters.

Return values

The gethostbyname() call returns a pointer to a hostent structure for the host name specified on the call. For more information about the hostent structure, see Figure 130 on page 252. A null pointer is returned if the gethostbyname() call fails.

There are no errno values for gethostbyname().

A new part called EZACIC17 has been created. EZACIC17 is like EZACIC07 except it uses the internal C errno function. Also, a new header file called cmanifes.h has been created to remap EZACIC17's long function names into unique 8-character names.

EZACIC07 and EZACIC17 now support the gethostbyaddr() and gethostbyname() functions.

gethostid()

The gethostid() call gets the unique 32-bit identifier for the current host in network byte order. This value is the default home IP address.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <socket.h>

unsigned long gethostid()
```

Parameters

None.

Return values

The gethostid() call returns the 32-bit identifier of the current host, which should be unique across all hosts.

gethostname()

The gethostname() call returns the name of the host processor on which the program is running.

Note: The host name returned is the host name that the TCPIP stack learned at startup from the TCPIP.DATA file that was found.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <socket.h>

int gethostname(char *name, int namelen)
```

Parameters

name
: The character array to be completed with the host name. The name that is returned is NULL-terminated unless truncated to the size of the name array.

namelen
: The length of the*name* value. The minimum length of the *name* field is 1 character. The maximum length of the *name* field is 24 characters.

Return values

The value 0 indicates success; the value −1 indicates an error. To determine what error has occurred, check the *errno* global variable, which is set to a return code. Possible codes are:

EFAULT
: The *name* parameter specified an address outside the caller's address space.

getipv4sourcefilter()

Obtains a list of the IPv4 source addresses that comprise the source filter, along with the current mode on a given interface and a multicast group for a socket. The source filter can either include or exclude the set of source addresses, depending on the filter mode (MCAST_INCLUDE or MCAST_EXCLUDE).

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <camifes.h> (reentrant programs only)
#include <netinet.h>
int getipv4sourcefilter(int s,
                        struct in_addr interface,
                        struct in_addr group,
                        uint32_t *fmode,  uint32_t *numsrc,
                        struct in_addr *slist)
```

Parameters

s The socket descriptor.

interface
The local IP address of the interface.

group The IP multicast address of the group.

fmode A pointer to an integer that contains the filter mode on a successful return. The value of the filter mode can be MCAST_INCLUDE or MCAST_EXCLUDE.

numsrc
As an input parameter, a pointer to the number of source addresses that can fit in the array specified by the *slist* parameter. As an output parameter, a pointer to the total number of source addresses in the filter.

slist A pointer to an array of IP addresses that is either included or excluded, depending on the filter mode. If the *numsrc* value was 0 on input, a NULL pointer can be supplied.

If the application does not know the size of the source list before, it can make a reasonable guess (for example, 0). When the process completes, the *numsrc* value is larger, the operation can be repeated with a larger buffer.

On return, the *numsrc* value is always updated to be the total number of sources in the filter. The *slist* value specifies as many source addresses as fit, up to the minimum array size that was specified by the *numsrc* value and the total number of sources in the filter.

Return values

When successful, the value 0 is returned. When an error has occurred, the value -1 is returned and the errno value is one of the following:

EBADF
The *s* parameter value is not a valid socket descriptor.

EINVAL
The *interface* or *group* parameter value is not a valid IPv4 address, or the socket *s* has already requested multicast setsockopt options. For more information, see the *z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference*.

EPROTOTYPE
The socket protocol type is not correct.

EADDRNOTAVAIL
The tuple consisting of socket, interface, and multicast group values does not exist, or the specified interface address is incorrect for this host, or the specified interface address is not multicast capable.

ENOMEM
Insufficient storage is available to supply the array.

getnameinfo()

The getnameinfo() call returns the node name and service location of a socket address that is specified in the call.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <socket.h>
#include <in.h>
#include <netdb.h>

int getnameinfo(const struct sockaddr *sa, socklen_t salen,
                char *host, socklen_t hostlen,
                char *serv, socklen_t servlen,
                int flags)
```

Parameters

sa The pointer to a socket address structure that is expected to be either *sockaddr_in* for an IPv4 socket address or *sockaddr_in6* for an IPv6 socket address, as defined in the header file in.h. Table 19 on page 160 shows the format of the structure.

The following fields are used to specify the IPv4 socket address structure to be translated.

- The *sin_family* field must be set to AF_INET.
- The *sin_port* field is set to a port number, in network byte order.
- The *in_addr.sin_addr* field is set to an IPv4 address and must be specified in network byte order.
- The *sin_zero* field is not used and must be set to all zeros.

The following fields are used to specify the IPv6 socket address structure to be translated.

- The *sin6_family* field must be set to AF_INET6.
- The *sin6_port* field is set to the a port number, in network byte order.
- The *sin6_flowinfo* field is used to specify the traffic class and flow label. This field is currently not implemented.
- The *in6_addr.sin6_addr* field is set to an IPv6 address and must be specified in network byte order.
- The *sin6_scope_id* field is used to specify the link scope for an IPv6 address as an interface index. The resolver ignores the *sin6_scope_id* field, unless the input IPv6 address is a link-local address and the *host* parameter is also specified.

salen The size, in bytes, of the buffer pointed to by *sa*. For an IPv4 socket address, the *salen* parameter should contain a decimal 16, and for an IPv6 socket address, the *salen* parameter should contain a decimal 28.

host On input, storage capable of holding the returned resolved host name. The host name can be a maximum of 255 bytes for a null terminated string, for the input socket address. If inadequate storage is specified to contain the resolved host name, then the resolver returns the host name up to the storage amount specified and truncation might occur. If the host name cannot be located, the numeric form of the host address is returned instead of its name. However, if the NI_NAMEREQD option is specified and no host name is located, an error is returned.

If the specified IPv6 address is a link-local address, and the *sin6_scope_id* interface index is a non-zero value, scope information is appended to the resolved host name using the format *host%scope information*. The scope information can be either the numeric form of the interface index, or the interface name associated with the interface index.

Use the NI_NUMERICSCOPE option to select which form should be returned. The combined host name and scope information is always a null-terminated string that is no more than 256 bytes in length. For more information about scope information and getnameinfo() processing, see *z/OS Communications Server: IPv6 Network and Application Design Guide* .

This is an optional field, but if this field value is not 0, you must also specify the *hostlen* parameter. Specify both the *service* and *servlen* parameters or both the *host* and *hostlen* parameters. An error occurs if both are omitted.

hostlen A field that contains the length of the host storage used to contain the resolved host name. The *hostlen* parameter value must be equal to or greater than the length of the longest host name or of the host name and scope information combination, plus one for the null termination character, to be returned. The getnameinfo() call returns the host name, or host name and scope information, up to the length specified by the *hostlen* parameter. If the *hostlen* parameter is 0 on input, then the resolved host name is not returned.

This is an optional field, but if the field value is not 0, you must also specify the *host* parameter. Specify both the *service* and *servlen* parameters or both the *host* and *hostlen* parameters. An error occurs if both are omitted.

serv On input, storage capable of holding the returned resolved service name, which can be a maximum of 33 bytes for a null terminated string, for the input socket address. If inadequate storage is specified to contain the resolved service name, the resolver returns the service name up to the storage specified and truncation might occur. If the service name cannot be located, or if NI_NUMERICSERV was specified in the *flags* parameter, then the numeric form of the service address is returned instead of its name.

This is an optional field, but if the value is not 0, then you must also specify the *servlen* parameter. Specify both the *service* and *servlen* parameters or both the *host* and *hostlen* parameters. An error occurs if both are omitted.

servlen A field that contains the length of the storage used to contain the returned resolved service name (specified by the *serv* parameter). The *servlen* parameter must be equal to or greater than the length of the longest service name to be returned, plus one for the null termination character. The getnameinfo() call returns the service name up to the length specified by the *servlen* parameter value. If the *servlen* value is 0 on input, the service name information is not returned.

This is an optional field, but if the value is not 0, you must also specify the *serv* parameter. Specify both the *service* and *servlen* parameters or both the *host* and *hostlen* parameters. An error occurs if both are omitted.

flags The parameter can be set to 0 or one of the following:

NI_NOFQDN
: Return the NAME portion of the fully qualified domain name.

NI_NUMERICHOST
: Return only the numeric form of host's address.

NI_NAMEREQD
: Return an error if the host's name cannot be located.

NI_NUMERICSERV
: Return only the numeric form of the service address.

NI_DGRAM
: Indicates that the service is a datagram service. The default behavior is to assume that the service is a stream service.

NI_NUMERICSCOPE
: Return only the numeric form of the *sin6_scope_id* interface index, if applicable.

Return values

The value 0 indicates success; the value -1 indicates an error. To determine which error occurred, check the *errno* global variable, which is set to a return code. Possible codes include:

EAI_AGAIN
: The host address specified could not be resolved within the configured time interval, or the resolver address space has not been started. The request can be retried later.

EAI_BADFLAGS
: The flags parameter had an incorrect value.

EAI_FAIL
: An unrecoverable error has occurred.

EAI_FAMILY
: The address family was not recognized, or the address length was incorrect for the specified family.

EAI_MEMORY
: A memory allocation failure occurred.

EAI_NONAME
: The hostname does not resolve for the supplied parameters. NI_NAMEREQD is set and the hostname cannot be located, or both *nodename* and *servname* were null. Or the requested address is valid but does not have a record at the name server.

getpeername()

The getpeername() call returns the name of the peer connected to a specified socket.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <socket.h>
#include <bsdtypes.h>
int getpeername(int s, struct sockaddr *name, int *namelen)
```

Parameters

s
: The socket descriptor.

name
: A pointer to a structure that contains the IP address of the connected socket that is filled by getpeername() before it returns. The exact format of *name* is determined by the domain in which communication occurs.

  The following fields are used to define the IPv4 socket address structure for the remote socket that is connected to the local socket specified in field *s*.

  - The *sin_family* field is set to AF_INET.
  - The *sin_port* field contains the connection peer's port number.
  - The *in_addr.sin_addr* field contains the 32-bit IPv4 Internet address, in network byte order, of the connection peer's host machine.
  - The *sin_zero* field is not used and is set to all zeros.

  The following fields are used to define the IPv6 socket address structure for the remote socket that is connected to the local socket specified in field *s*.

  - The *sin6_family* field is set to AF_INET6.
  - The *sin6_port* field contains the connection peer's port number.
  - The *sin6_flowinfo* field contains the traffic class and flow label. The value of this field is undefined.
  - The *in6_addr.sin6_addr* field contains the 128-bit IPv6 Internet address, in network byte order, of the connection peer's host machine.
  - The *sin6_scope_id* field identifies a set of interfaces as appropriate for the scope of the address carried in the *in6_addr.sin6_addr* field. For a link scope *in6_addr.sin6_addr*, *sin6_scope_id* contains the link index for the *in6_addr.sin6_addr*. For all other address scopes, *sin6_scope_id* is undefined.

namelen
: A pointer to the structure that contains the size of the address structure pointed to by *name* in bytes. For an IPv4 socket address the *namelen* parameter should contain a decimal 16 and for an IPv6 socket address the *namelen* parameter should contain a decimal 28.

Return values

The value 0 indicates success; the value −1 indicates an error. To determine which error occurred, check the *errno* global variable, which is set to a return code. Possible codes include:

EBADF
: The *s* parameter is not a valid socket descriptor.

EFAULT
: Using the *name* and *namelen* parameters as specified results in an attempt to access storage outside of the caller's address space.

ENOTCONN
: The socket is not in the connected state.

getsockname()

A getsockname() call returns the current name for socket *s* in the *sockaddr* structure pointed to by the *name* parameter. It returns the address of the socket that has been bound. If the socket is not bound to an address, the call returns with family set, and the rest of the structure set to zero. For example, an unbound IPv4 socket causes the name to point to a *sockaddr_in* structure with the *sin_ family* field set to AF_INET and all other fields set to zero. An unbound IPv6 socket causes the name to point to a sockaddr_in6 structure with the sin6_family field set to AF_INET6 and all other fields set to zero.

Stream sockets are not assigned a name until after a successful call to either bind(), connect(), or accept().

The getsockname() call is often used to discover the port assigned to a socket after the socket has been implicitly bound to a port. For example, an application can call connect() without previously calling bind(). In this case, the connect() call completes the binding necessary by assigning a port to the socket. This assignment can be discovered with a call to getsockname().

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <socket.h>
#include <bsdtypes.h>
#include <in.h>

int getsockname(int s, struct sockaddr *name, int *namelen)
```

Parameters

s The socket descriptor.

name The address of the buffer into which getsockname() copies the name of *s*.

The following fields are used to define the IPv4 socket address structure returned by the call.

- The *sin_family* field is set to AF_INET.
- The *sin_port* field contains the port number bound to this socket. If the socket is not bound, 0 is returned.
- The *in_addr.sin_addr* field contains the 32-bit IPv4 Internet address, in network byte order, of the local host machine. If the socket is not bound, the address is INADDR_ANY.
- The *sin_zero* field is not used and is set to all zeros.

The following fields are used to define the IPv6 socket address structure returned by the call.

- The *sin6_family* field is set to AF_INET6.
- The *sin6_port* field contains the port number bound to this socket. If the socket is not bound, 0 is returned.
- The *sin6_flowinfo* field contains the traffic class and flow label. The value of this field is undefined.
- The *in6_addr.sin6_addr* field contains the 128-bit IPv6 Internet address, in network byte order, of the local host machine. If the socket is not bound, the address is the IPv6 unspecified address (in6addr_any).
- The *sin6_scope_id* field identifies a set of interfaces as appropriate for the scope of the address carried in the *in6_addr.sin6_addr* field. For a link

scope *in6_addr.sin6_addr*, *sin6_scope_id* contains the link index for the *in6_addr.sin6_addr*. For all other address scopes, *sin6_scope_id* is undefined.

namelen
: Must initially point to an integer that contains the size in bytes of the storage pointed to by *name*. Upon return, that integer contains the size of the data returned in the storage pointed to by *name*. For an IPv4 socket address the *namelen* parameter contains a decimal 16 and for an IPv6 socket address the *namelen* parameter contains a decimal 28.

Return values

The value 0 indicates success; the value −1 indicates an error. To determine which error occurred, check the *errno* global variable, which is set to a return code. Possible codes include:

EBADF
: The *s* parameter is not a valid socket descriptor.

EFAULT
: Using the *name* and *namelen* parameters as specified results in an attempt to access storage outside of the caller’s address space.

getsockopt(), setsockopt()

The getsockopt() call gets options associated with a socket; setsockopt() sets the options.

The following options are recognized at the IPPROTO_IP level:
- Joining a multicast group
- Leaving a multicast group or leaving all sources for a given multicast group
- Setting the multicast interface
- Setting the IP time-to-live of outgoing multicast datagrams
- Looping back multicast datagrams
- Joining a source-specific multicast group
- Leaving a source-specific multicast group
- Blocking data from a given source to a given multicast group
- Unblocking a previously blocked source for a given multicast group

The following options are recognized at the IPPROTO_IPV6 level:
- Joining a multicast group
- Leaving a multicast group
- Setting the multicast interface
- Setting multicast hop limit
- Looping back multicast datagrams
- Setting unicast hop limit
- Restricting sockets to AF_INET6 sockets

The following options are recognized at the IPPROTO_IP and IPPROTO_IPV6 level:
- Joining an IPv4 or IPv6 multicast group
- Leaving an IPv4 or IPv6 multicast group or leaving all sources for a given IPv4 or IPv6 multicast group

- Joining an IPv4 or IPv6 source-specific multicast group
- Leaving an IPv4 or IPv6 source-specific multicast group
- Blocking IPv4 or IPv6 data from a given source to a given multicast group
- Unblocking an IPv4 or IPv6 previously blocked source for a given multicast group

The following options are recognized at the socket level:
- Broadcasting messages (IPv4 UDP socket only)
- Toggling the TCP keep-alive mechanism for a stream socket
- Lingering on close if data is present
- Receiving of out-of-band data
- Local address reuse

The following option is recognized at the TCP level (IPPROTO_TCP):
- Disable sending small data amounts until acknowledgment (Nagle algorithm)

As well as checking current options, getsockopt() can return pending errors and the type of socket.

Format

The format for getsockopt() is as follows:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <socket.h>
#include <bsdtypes.h>

int getsockopt(int s, int level, int optname, char *optval, int *optlen)
```

The format for setsockopt() is as follows:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <socket.h>
#include <bsdtypes.h>

int setsockopt(int s, int level, int optname, char *optval, int optlen)
```

Note: The above code sample is for getsockopt(). The setsockopt() call requires the same parameters and declarations, except that:
- The socket function name changes; getsockopt() becomes setsockopt().
- `int *optlen` should be replaced by `int optlen` (without the asterisk).

Parameters

s
The socket descriptor.

level
When manipulating socket options, you must specify the level at which the option resides and the name of the option. To manipulate options at the socket level, the *level* parameter must be set to SOL_SOCKET as defined in *socket.h*. For TCP_NODELAY at the TCP level, the level parameter must be set to IPPROTO_TCP. To manipulate other TCP level options or options at any other level, such as the IP level, supply the appropriate protocol number for the protocol controlling the option. Currently, only the IPPROTO_IP, IPPROTO_IPV6, IPPROTO_TCP, and SOL_SOCKET levels are supported.

optname
: The name of a specified socket option. The options that are available with CICS TCP/IP are shown in "Possible entries for optname."

optval **and** *optlen*
: For getsockopt(), the *optval* and *optlen* parameters are used to return data used by the particular form of the call. The *optval* parameter points to a buffer that is to receive the data requested by the get command. The *optlen* parameter points to the size of the buffer pointed to by the *optval* parameter. It must be initially set to the size of the buffer before calling getsockopt(). On return it is set to the actual size of the data returned.

  For setsockopt(), the *optval* and *optlen* parameters are used to pass data used by the particular set command. The *optval* parameter points to a buffer that contains the data needed by the set command. The *optval* parameter is optional and can be set to the NULL pointer, if data is not needed by the command. The *optlen* parameter must be set to the size of the data pointed to by *optval*.

  For both calls, all of the socket level options except SO_LINGER expect *optval* to point to an integer and *optlen* to be set to the size of an integer. When the integer is nonzero, the option is enabled. When it is zero, the option is disabled. The SO_LINGER option expects *optval* to point to a *linger* structure as defined in *socket.h*.

  This structure is defined in the following example:

```
#include <manifest.h>
struct  linger
{
       int     l_onoff;               /* option on/off */
       int     l_linger;              /* linger time */
};
```

  The *l_onoff* field is set to zero if the SO_LINGER option is being disabled. A nonzero value enables the option. The *l_linger* field specifies the amount of time to linger on close. The units of *l_linger* are seconds.

Possible entries for optname

The following options are recognized at the IPPROTO_IP level:

Option **Description**

IP_ADD_MEMBERSHIP
: Enables an application to join a multicast group on a specific interface. An interface must be specified with this option. Only applications that want to receive multicast datagrams need to join multicast groups. This is an IPv4 only socket option.

  For setsockopt(), set the *optval* value to the structure as defined in in.h. The ip_mreq structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 interface address.

  This option cannot be specified with the getsockopt() call.

IP_ADD_SOURCE_MEMBERSHIP
: Enables an application to join a multicast group on a specific interface and a specific source address. An interface and a source address must be specified with this option. Only applications that want to receive multicast datagrams need to join source multicast groups. This socket option applies only to IPv4.

For the setsockopt() function, set the *optval* value to the ip_mreq_source structure as defined in the in.h header. The ip_mreq_source structure contains the following:

- 4-byte IPv4 multicast address
- 4-byte IPv4 source address
- 4-byte IPv4 interface address

This option cannot be specified with the getsockopt() function.

IP_BLOCK_SOURCE
Enables an application to block multicast packets that have a source address that matches the given IPv4 source address. An interface and a source address must be specified with this option. The specified multicast group must be joined previously. This socket option applies only to IPv4.

For the setsockopt() function, set the *optval* value to the ip_mreq_source structure as defined in the in.h header. The ip_mreq_source structure contains the following:

- 4-byte IPv4 multicast address
- 4-byte IPv4 source address
- 4-byte IPv4 interface address

This option cannot be specified with the getsockopt() function.

IP_DROP_MEMBERSHIP
Enables an application to exit a multicast group or to exit a multicast group and drop all sources. This is an IPv4-only socket option.

For the setsockopt() function, set the *optval* value to the ip_mreq structure as defined in the in.h header. The ip_mreq structure contains the following:

- 4-byte IPv4 multicast address
- 4-byte IPv4 interface address

This option cannot be specified with the getsockopt() function.

IP_DROP_SOURCE_MEMBERSHIP
Enables an application to exit a source multicast group. This socket option applies only to IPv4.

For the setsockopt() function, set the optval value to the ip_mreq_source structure as defined in the in.h header. The ip_mreq_source structure contains the following:

- 4-byte IPv4 multicast address
- 4-byte IPv4 source address
- 4-byte IPv4 interface address

This option cannot be specified with the getsockopt() function.

IP_MULTICAST_IF
Sets or obtains the IPv4 interface address used for sending outbound multicast datagrams from the socket application. This is an IPv4-only socket option.

Note: Multicast datagrams can be transmitted only on one interface at a time.

For setsockopt(), set *optval* to an IPv4 interface address.

For getsockopt(), *optval* contains an IPv4 interface address.

IP_MULTICAST_TTL
Sets or obtains 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. This is an IPv4-only socket option.

For setsockopt(), set *optval* to a value in the range of x'00'–x'ff' specifying the time-to-live. *optval* is a 1 byte field.

For getsockopt(), *optval* contains a value in the range from x'00'–x'ff', indicating time-to-live. *optval* is a one byte field.

IP_MULTICAST_LOOP
Controls or determines if a copy of multicast datagrams is looped back for multicast datagrams sent to a group to which the sending host itself belongs. The default is to loop the datagrams back. This is an IPv4-only socket option.

For setsockopt(), set *optval* to 1 to enable and set to 0 to disable.

For getsockopt(), *optval* contains a 1 when enabled and contains a 0 when disabled.

| **IP_UNBLOCK_SOURCE**
| Enables an application to unblock a previously blocked source for a given IPv4 source multicast group. An interface and a source address must be specified with this option. This socket option applies only to IPv4.

| For the setsockopt() function, set the optval value to the ip_mreq_source structure as defined in the in.h header. The ip_mreq_source structure contains the following:

- 4-byte IPv4 multicast address
- 4-byte IPv4 source address
- 4-byte IPv4 interface address

| This option cannot be specified with the getsockopt() function.

The following options are recognized at the IPPROTO_IPV6 level:

Option **Description**

IPV6_JOIN_GROUP
Controls the reception of multicast packets and specifies that the socket join a multicast group. This is an IPv6-only socket option.

For setsockopt(), set *optval* to the ipv6_mreq structure as defined in in.h. The ipv6_mreq structure contains a 16-byte IPv6 multicast address followed by a 4-byte IPv6 interface index number. If the interface number is 0, the stack chooses the local interface.

This cannot be specified with getsockopt().

IPV6_LEAVE_GROUP
Controls the reception of multicast packets and specify that the socket leave a multicast group. This is an IPv6-only socket option.

For setsockopt(), set *optval* to the ipv6_mreq structure as defined in in.h. The ipv6_mreq structure contains a 16-byte IPv6 multicast address followed by a 4-byte IPv6 interface index number. If the interface number is 0, then the stack chooses the local interface.

This cannot be specified with getsockopt().

IPV6_MULTICAST_HOPS
Sets or obtains the hop limit used for outgoing multicast packets. This is an IPv6-only socket option.

For setsockopt(), set *optval* to a value in the range of 0 to 255, specifying the multicast hops. If *optval* is not specified or is set to 0, the default is 1 hop. If *optval* is set to a -1, the stack default hop is used.

Rule: An application must be APF authorized to enable it to set the hop limit value above the system defined hop limit value. The CICS application cannot execute as APF authorized.

For getsockopt(), *optval* contains a value in the range from 0–255, indicating the number of multicast hops.

IPV6_MULTICAST_IF
Sets or obtains the index of the IPv6 interface used for sending outbound multicast datagrams from the socket application. This is an IPv6 only socket option.

For setsockopt(), set *optval* to a value that contains an IPv6 interface index.

For getsockopt(), *optval* contains an IPv6 interface index.

IPV6_MULTICAST_LOOP
Controls or determines whether a multicast datagram is looped back on the outgoing interface by the IP layer for local delivery when datagrams are sent to a group to which the sending host itself belongs. The default is to loop multicast datagrams back. This is an IPv6-only socket option.

For setsockopt(), set *optval* to 1 to enable and set to 0 to disable.

For getsockopt(), *optval* contains a 1 when enabled and contains a 0 when disabled.

IPV6_UNICAST_HOPS
Sets or obtains the hop limit used for outgoing unicast IPv6 packets. This is an IPv6 only socket option.

For setsockopt(), set *optval* to a value in the range of 0–255, specifying the unicast hops. If *optval* is not specified or is set to 0, the default is 1 hop. If *optval* is set to a -1, the stack default hop is used.

Rule: An application must be APF authorized to enable it to set the hop limit value above the system defined hop limit value. The CICS application cannot execute as APF authorized.

For getsockopt(), *optval* contains a value in the range from 0–255 indicating the number of unicast hops.

IPV6_V6ONLY
Sets or determines 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. This is an IPv6-only socket option.

For setsockopt(), set *optval* to 1 to enable and set to 0 to disable.

For getsockopt(), *optval* contains a 1 when enabled and contains a 0 when disabled.

The following options are recognized at the IPPROTO_IP and IPPROTO_IPV6 level:

Option **Description**

MCAST_BLOCK_SOURCE
Enables an application to block multicast packets that have a source address that matches the given source address. An interface index and a source address must be specified with this option. The specified multicast group must have been joined previously.

For the setsockopt() function, set the optval value to the group_source_req structure as defined in the in.h header. The group_source_req structure contains the following:

- 4-byte interface index number
- Socket address structure of the multicast address
- Socket address structure of the source address

This option cannot be specified with the getsockopt() function.

MCAST_JOIN_GROUP
Enables an application to join a multicast group on a specific interface. An interface index must be specified with this option. The stack chooses a default interface if the interface index 0 is specified. Only applications that want to receive multicast datagrams need to join multicast groups.

For the setsockopt() function, set the optval value to the group_req structure as defined in the in.h header. The group_req structure contains the following:

- 4-byte interface index number
- Socket address structure of the multicast address

This option cannot be specified with the getsockopt() function.

Sets the IPv4 or IPv6 multicast address and the local interface index. Use the setsockopt() function and specify the address of the group_req structure that controls the address and the interface index. The application can join multiple multicast groups on a single socket and can also join the same group on multiple interfaces on the same socket. However, there is a maximum limit of 20 groups per single UDP socket and there is a maximum limit of 256 groups per single RAW socket. The stack chooses a default multicast interface if the interface index 0 is passed. The format of the group_req structure is in the in.h header.

MCAST_JOIN_SOURCE_GROUP
Enables an application to join a multicast group on a specific interface and a source address. An interface index and the source address must be specified with this option. The stack chooses a

default interface if the interface index 0 is specified. Only applications that want to receive multicast datagrams need to join source multicast groups.

For the setsockopt() function, set the optval value to the group_source_req structure as defined in the in.h header. The group_source_req structure contains the following:

- 4-byte interface index number
- Socket address structure of the multicast address
- Socket address structure of the source address

This option cannot be specified with the getsockopt() function.

MCAST_LEAVE_GROUP
Enables an application to exit a multicast group or to exit a multicast group and drop all sources.

For the setsockopt() function, set the optval value to the group_req structure as defined in the in.h header. The group_req structure contains the following:

- 4-byte interface index number
- Socket address structure of the multicast address

This option cannot be specified with the getsockopt() function.

MCAST_LEAVE_SOURCE_GROUP
Enables an application to exit a source multicast group on a specific interface and a source address.

For the setsockopt() function, set the optval value to the group_source_req structure as defined in the in.h header. The group_source_req structure contains the following:

- 4-byte interface index number
- Socket address structure of the multicast address
- Socket address structure of the source address

This option cannot be specified with the getsockopt() function.

MCAST_UNBLOCK_SOURCE
Enables an application to unblock a previously blocked source for a given multicast group. An interface index and a source address must be specified with this option.

For the setsockopt() function, set the optval value to the group_source_req structure as defined in the in.h header. The group_source_req structure contains the following:

- 4-byte interface index number
- Socket address structure of the multicast address
- Socket address structure of the source address

This option cannot be specified with the getsockopt() function.

The following options are recognized at the TCP level:

TCP_KEEPALIVE
For setsockopt, the TCP_KEEPALIVE socket option specifies a socket-specific timer value which remains in effect until specified

by SETSOCKOPT or until the socket is closed. Valid values are in the range 0 - 2 147 460 seconds; if a value greater than the allowed range is specified, 2 147 460 seconds is used. For the getsockopt call, the TCP_KEEPALIVE socket option returns the specific timer value in seconds in effect for the given socket, or 0 if TCP_KEEPALIVE timing is not active. See *z/OS Communications Server: IP Programmer's Guide and Reference* for more information about the socket option parameters.

TCP_NODELAY
For setsockopt, toggles the use of the Nagle algorithm (RFC 896) for all data sent over the socket. Under most circumstances, TCP sends data when it is presented. However, when outstanding data has not yet been acknowledged, TCP gathers small amounts of output to be sent in a single packet after an acknowledgment is received. For interactive applications, such as ones that send a stream of mouse events which receive no replies, this gathering of output can cause significant delays. For these types of applications, disabling the Nagle algorithm improves response time. When the Nagle algorithm is disabled, TCP can send small amounts of data before the acknowledgment for previously sent data is received.

For getsockopt, returns the setting of the Nagle algorithm for the socket. When optval is 0, the Nagle algorithm is enabled and TCP waits to send small packets of data until the acknowledgment for the previous data is received. When optval is not 0, the Nagle algorithm is disabled and TCP can send small packets of data before the acknowledgment for previously sent data is received.

The following options are recognized at the socket level:

SO_BROADCAST
Toggles the ability to broadcast messages. If this option is enabled, it allows the application to send broadcast messages over *s*, if the interface specified in the destination supports the broadcasting of packets. This option has no meaning for stream sockets.

SO_ERROR
This cannot be specified with setsockopt(). It returns any pending error on the socket and clears the error status. It can be used to check for asynchronous errors on connected datagram sockets or for other asynchronous errors (errors that are not returned explicitly by one of the socket calls).

SO_KEEPALIVE
Sets or determines whether the keepalive mechanism periodically sends a packet on an otherwise idle connection for a stream socket. The default is disabled. When activated, the keepalive mechanism periodically sends a packet on an otherwise idle connection. If the remote TCP does not respond to the packet or to retransmissions of the packet, the connection is terminated with the error ETIMEDOUT.

SO_LINGER
Lingers on close if data is present. When this option is enabled and there is unsent data present when close() is called, the calling application is blocked during the close() call until the data is transmitted or the connection has timed out. If this option is disabled, the TCP/IP address space waits to try to send the data. Although the data transfer is usually successful, it cannot be

guaranteed, because the TCP/IP address space waits a finite amount of time trying to send the data. The close() call returns without blocking the caller.

Note: If you set a 0 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.

SO_OOBINLINE
Toggles reception of out-of-band data. When this option is enabled, it causes out-of-band data to be placed in the normal data input queue as it is received, making it available to recvfrom() without having to specify the MSG_OOB flag in the call. When this option is disabled, it causes out-of-band data to be placed in the priority data input queue as it is received, making it available to recvfrom(), and only by specifying the MSG_OOB flag in that call.

SO_REUSEADDR
Toggles local address reuse. When enabled, this option allows local addresses that are already in use to be bound. This alters the normal algorithm used in the bind() call. Normally, the system checks at connect time to ensure that the local address and port do not have the same foreign address and port. The error EADDRINUSE is returned if the association already exists. If you require multiple servers to bind to the same port and listen on INADDR_ANY or the IPv6 unspecified address (in6addr_any), see to the SHAREPORT option on the PORT statement in TCPIP.PROFILE.

SO_SNDBUF
Applies to getsockopt() only. Returns the size of the data portion of the TCP/IP send buffer in *optval*. The size of the data portion of the send buffer is protocol-specific, based on the DATABUFFERPOOLSIZE statement in the PROFILE.TCPIP data set. The value is adjusted to allow for protocol header information.

SO_TYPE
This is for getsockopt() only. This option returns the type of the socket. On return, the integer pointed to by *optval* is set to SOCK_STREAM or SOCK_DGRAM.

Return values

The value 0 indicates success; the value −1 indicates an error. To determine which error occurred, check the *errno* global variable, which is set to a return code. Possible codes include:

EBADF
The *s* parameter is not a valid socket descriptor.

EFAULT
Using *optval* and *optlen* parameters results in an attempt to access storage outside the caller's address space.

ENOPROTOOPT
The *optname* parameter is unrecognized, or the *level* parameter is not SOL_SOCKET.

getsourcefilter()

Obtains a list of the IPv4 or IPv6 source addresses that comprise the source filter, along with the current mode on a given interface and a multicast group for a socket. The source filter can either include or exclude the set of source addresses, depending on the filter mode (MCAST_INCLUDE or MCAST_EXCLUDE).

Format

This call has the following format:

```
 #include <manifest.h> (non-reentrant programs only)
#include <cmanfies.h> (reentrant programs only)
#include <netinet/in.h>
int getsourcefilter(int s, uint32_t interface,
struct sockaddr *group, socklen_t grouplen,
uint32_t *fmode, uint32_t *numsrc,
struct sockaddr_storage *slist);
```

Parameters

s The socket descriptor.

interface
The interface index of the interface.

group A pointer to either a sockaddr_in structure for IPv4 addresses or a sockaddr_in6 structure for IPv6 addresses that holds the IP multicast address of the group.

grouplen
The length of the sockaddr_in or sockaddr_in6 structure.

fmode A pointer to an integer that contains the filter mode on a successful return. The value of the filter mode can be either MCAST_INCLUDE or MCAST_EXCLUDE.

numsrc
On input, a pointer to the number of source addresses that can fit in the array specified by the *slist* parameter. On output, a pointer to the total number of source addresses in the filter.

slist A pointer to an array of IP addresses that is either included or excluded, depending on the filter mode. If a *numsrc* value 0 was specified on input, you can specify a NULL pointer.

On return, the *numsrc* value is always updated to be the total number of sources in the filter; the *slist* pointer points to an array that holds as many source addresses as fit, which is the minimum of the array size specified by the input *numsrc* value and the total number of sources in the filter.

If the application is not aware of the size of the source list before processing, it can make a reasonable guess (for example, 0). When the process completes, if the *numsrc* is large, the operation can be repeated with a large buffer.

Return values

When successful, the value 0 is returned. When an error has occurred, the value -1 is returned and the errno value is one of the following:

EBADF
The *s* parameter value is not a valid socket descriptor.

EAFNOSUPPORT
The address family of the sockaddr value is not AF_INET or AF_INET6.

EPROTOTYPE
The socket protocol type is not correct.

EADDRNOTAVAIL
The tuple consisting of socket, interface, and multicast group values does not exist, or the specified interface address is not multicast capable.

EINVAL
The socket address family of an input parameter is not correct or the socket specified by the *s* parameter already requested multicast setsockopt options. For more information, see the *z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference*.

ENOMEM
Insufficient storage is available to supply the array.

ENXIO
The interface index specified by the *interface* parameter does not exist.

givesocket()

The givesocket() call tells TCP/IP to make a specified socket available to a takesocket() call issued by another program. Any connected stream socket can be given. Typically, givesocket() is used by a parent server that obtains sockets by means of accept() and gives them to child servers that handle one socket at a time.

To pass a socket, the parent server first calls givesocket(), passing the name of the child server's address space.

The parent server then uses the EXEC CICS START command to start the child server. The START command uses the FROM data to pass the socket descriptor and the parent's client ID that were previously returned by the socket() and getclientid() calls respectively.

The child server calls takesocket(), specifying the parent's client ID and socket descriptor.

Having issued a givesocket() and started the child server that is to take the socket, the concurrent server uses select() to test the socket for an exception condition. When select() reports that an exceptional condition is pending, the concurrent server calls close() to free the socket. If the concurrent server closes the socket before a pending exception condition is indicated, the TCP connection is immediately reset, and the child server's takesocket() call is unsuccessful.

When a program has issued a givesocket() call for a socket, it cannot issue any further calls for that socket, except close().

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <bsdtypes.h>
#include <socket.h>

int  givesocket(int s, struct clientid *clientid)
```

Parameters

s
The descriptor of a socket to be given to another application.

clientid
A pointer to a clientid structure specifying the target program to whom the socket is to be given. You should fill the structure as follows:

domain
Set to either AF_INET (a decimal 2) or AF_INET6 (a decimal 19).

Rule: An AF_INET socket can be given only to an AF_INET takesocket(). An AF_INET6 socket can be given only to an AF_INET6 takesocket(). EBADF is set if the domain does not match.

name
This is the child server's address space name, left-justified and padded with blanks. The child server can run in the same address space as the parent server. In this case, the field is set to the parent server's address space.

subtaskname
Blanks.

reserved
Binary zeros.

Return Values

The value 0 indicates success; the value −1 indicates an error. To determine which error occurred, check the *errno* global variable, which is set to a return code. Possible codes include:

EBADF
The *s* parameter is not a valid socket descriptor, the socket has already been given, or the socket domain is not AF_INET or AF_INET6.

EBUSY
listen() has been called for the socket.

EFAULT
Using the *clientid* parameter as specified results in an attempt to access storage outside the caller's address space.

EINVAL
The *clientid* parameter does not specify a valid client identifier.

ENOTCONN
The socket is not connected.

EOPNOTSUPP
The socket type is not SOCK_STREAM.

if_freenameindex()

The if_freenameindex() function is used to release the array storage obtained by the if_nameindex() function.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanfies.h> (reentrant programs only)
#include <if.h>

void if_freenameindex(struct if_nameindex *ptr)
```

Parameters

ptr A pointer that contains the address of the array of structures returned by the if_nameindex() function.

Return values

No return value is defined.

if_indextoname()

The if_indextoname() function returns an interface name when given an interface index.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanfies.h> (reentrant programs only)
#include <if.h>

char * if_indextoname(unsigned int ifindex, char *ifname)
```

Parameters

ifindex
Storage that contains an interface index.

ifname
A buffer that contain the name of the index value specified in the *ifindex* parameter.

Return values

Possible return values include:

EINVAL The *ifindex* parameter was zero, or the *ifname* parameter was NULL, or both.

ENOMEM Insufficient storage is available to obtain the information for the interface name.

ENXIO The ifindex does not yield an interface name.

if_nameindex()

The if_nameindex() function is used to obtain a list of interface names and their corresponding indices. The if_nameindex() function is not supported by IPv4-only stacks. However, if a mixture of IPv4-only and IPv4 and IPv6 stacks are active under CINET, CINET assigns a single interface index to the IPv4-only stack. This allows applications using IPv6 sockets to target an IPv4-only stack but does not allow the selection of a particular interface on an IPv4-only stack. Not all interfaces are returned in the output from if_nameindex(). VIPA interfaces are not returned. Interfaces that have never been activated are not returned.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanfies.h> (reentrant programs only)
#include <if.h>

struct if_nameindex * if_nameindex(void)
```

Parameters

There are no input parameters as the if_nameindex() function returns a pointer to an array of structures that contains information about each system interface. Check the if_nameindex structure in *if.h* for the format of the returned data.

Return values

When successful, if_nameindex() returns a pointer to an array of if_nameindex structures. Upon failure, if_nameindex() returns NULL and sets *errno* to the following:

ENOMEM Insufficient storage is available to supply the array.

if_nametoindex()

The if_nametoindex() function returns an interface index when given an interface name.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanfies.h> (reentrant programs only)
#include <if.h>

unsigned int if_nametoindex(const char * ifname)
```

Parameters

ifname
A pointer to null terminated storage that contains the interface name. If the interface specified by *ifname* does not exist then 0 is returned.

Return values

When successful, if_nametoindex() returns the interface index corresponding to the interface name *ifname*. Upon failure, if_nametoindex() returns zero and sets *errno* to one of the following:

EINVAL A parameter was not specified. The *ifname* parameter was NULL.

ENOMEM Insufficient storage is available to obtain the information for the interface name.

ENXIO The specified interface name provided in the *ifname* parameter does not exist.

inet_ntop()

Converts numeric IP addresses to their printable form.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanfies.h> (reentrant programs only)
#include <inet.h>

const char * inet_ntop(int af, const void *src, char *dst, socklen_t size)
```

Parameters

af The address family of the IP address being converted specified as AF_INET or AF_INET6.

src A pointer to the IP address, in network byte order, to be converted to presentable form.

dst A pointer to storage used to contain the converted IP address.

size The size of the IP address pointed to by the *src* parameter.

Return values

If successful, inet_ntop() returns a pointer to the buffer that contains the converted address.

If unsuccessful, inet_ntop() returns NULL and sets *errno* to one of the following values:

EAFNOSUPPORT
The address family specified in *af* is unsupported.

ENOSPC The destination buffer *size* is too small.

inet_pton()

Converts IP addresses from presentable text form to numeric form.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanfies.h> (reentrant programs only)
#include <inet.h>

int inet_pton(int af, const char *src, void *dst)
```

Parameters

af The address family of the IP address being converted, specified as AF_INET or AF_INET6.

src A pointer to the IP address, in presentable text form, to be converted to numeric form.

dst A pointer to storage used to contain the converted IP address. The converted address is in numeric form and network byte order.

Return values

If successful, inet_pton() returns 1 and stores the binary form of the Internet address in the buffer pointed to by *dst*.

If unsuccessful because the input buffer pointed to by *src* is not a valid string, inet_pton() returns 0.

If unsuccessful because the *af* argument is unknown, inet_pton() returns -1 and sets *errno* to the following value:

EAFNOSUPPORT
The address family specified in *af* is unsupported.

initapi()

The initapi() call connects your application to the TCP/IP interface.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
int initapi(int max_sock, char *subtaskid)
```

Parameters

max_sock
: The maximum number of sockets requested. This value cannot exceed 2000. The minimum value is 50.

subtaskid
: A unique 8-character ID, which should be the 4-byte packed EIBTASKN value in the EIB plus three character 0's and a unique displayable character.

  Using the letter L as the last character in the subtask parameter causes the tasking mechanism to assume that the CICS transaction is a listener. The task mechanism schedules the transaction using a non-reusable subtask by way of MVS attach processing when OTE=NO. This value has no effect when OTE=YES.

Return values

A positive value indicates success; a value of −1 indicates an error. To determine which error occurred, check the *errno* global variable, which is set to a return code.

ioctl()

The ioctl() call controls the operating characteristics of sockets. This call can issue a command to do any of the following:

- Set or clear nonblocking input and output for a socket.
- Get the number of immediately readable bytes for the socket.
- Query whether the current location in the data input is pointing to out-of-band data.
- Get the IPv6 home interface addresses.
- Get the network interface address.
- Get the network interface broadcast address.
- Get the network interface configuration.
- Get the network interface names and indices.
- Control Application Transparent Transport Layer Security (AT-TLS) for a connection

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <bsdtypes.h>
#include <ioctl.h>
#include <ezbztlsc.h>
#include <ezbyaplc.h>
#include <rtrouteh.h>
#include <if.h>

int ioctl(int s, unsigned long cmd, char *arg)
```

Parameters

s The socket descriptor.

cmd **and** *arg*

cmd is the command to perform; *arg* is a pointer to the data associated with *cmd*. The following are valid ioctl() commands:

FIONBIO
Sets or clears nonblocking input and output for a socket. *arg* is a pointer to an integer. If the integer is 0, the socket is in nonblocking mode. Otherwise, the socket is set for nonblocking input/output.

FIONREAD
Gets the number of immediately readable bytes for the socket. *arg* is a pointer to an integer. Sets the value of the integer to the number of immediately readable characters for the socket.

SIOCATMARK
Queries whether the current location in the data input is pointing to out-of-band data. The *arg* parameter is a pointer to an integer. The parameter sets the argument to 1 if the socket points to a mark in the data stream for out-of-band data. Otherwise, it sets the argument to 0.

SIOCGHOMEIF6
Get the IPv6 home interfaces. The *arg* parameter is a pointer to a NetConfHdr structure, as defined in ioctl.h. A pointer to a *HomeIf* structure that contains a list of home interfaces is returned in the *NetConfHdr* pointed to by the argument.

SIOCGIFADDR
Gets the network interface address. The *arg* parameter is a pointer to an *ifreq* structure, as defined in if.h. The interface address is returned in the argument.

SIOCGIFBRDADDR
Gets the network interface broadcast address. The *arg* parameter is a pointer to an *ifreq* structure, as defined in if.h. The interface broadcast address is returned in the argument.

SIOCGIFCONF
Gets the network interface configuration. The *arg* parameter is a pointer to an *ifconf* structure, as defined in if.h. The interface configuration is returned in the argument.

SIOCGIFDSTADDR
Gets the network interface destination address. The *arg* parameter is a pointer to an *ifreq* structure, as defined in if.h. The interface destination (point-to-point) address is returned in the argument.

| **SIOCSAPPLDATA**
| Enables an application to associate 40 bytes of user-specified
| application data with a TCP connection. Identifies socket endpoints
| in tools such as Netstat, SMF, or network management
| applications.

| **Requirement:** When you issue the SIOCSAPPLDATA ioctl()
| function, ensure that the arg parameter contains a SetApplData
| structure as defined by the EZBYAPLC header file in the
| SEZANMAC dataset. See *z/OS Communications Server: IP*

Programmer's Guide and Reference for more information about programming the SIOCSAPPLDATA IOCTL.

SetAD_buffer
The user-defined application data comprises 40 bytes of data that is used to identify the TCP connection with the IP CICS socket API sockets application. The application data can be displayed in the following ways:

- By requesting Netstat reports. The information is displayed conditionally using the modifier APPLDATA on the ALLC/-a and COnn /-c reports and unconditionally on the ALL/-A report. See the Netstat ALL/-A report, Netstat ALLConn/-a report, and Netstat COnn/-c report in *z/OS Communications Server: IP System Administrator's Commands* for more information about Netstat reports.
- In the SMF 119 TCP connection termination record. See *z/OS Communications Server: IP Configuration Reference* for more information about the application data written on the SMF 119 record.
- By network management applications. See *z/OS Communications Server: IP Programmer's Guide and Reference* for more information about application data.

Applications using this ioctl need to consider the following guidelines:

- The application is responsible for documenting the content, format, and meaning of the ApplData strings that it associates with sockets 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 EZA or EZB, are reserved for IBM use. IBM product identifiers begin with a letter in the range A - I.
- Printable EBCDIC characters should be used for the entire string to enable searching with Netstat filters.

  Tip: Separate application data elements with a blank for easier reading.

SIOCTTLSCTL
Controls Application Transparent Transport Layer Security (AT-TLS) for the connection. The *arg* parameter is a pointer to a TTLS_IOCTL structure, as defined in ezbztlsc.h. 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. For more usage information, see *z/OS Communications Server: IP Programmer's Guide and Reference* .

Return values

The value 0 indicates success; the value −1 indicates an error. To determine which error occurred, check the *errno* global variable, which is set to a return code. Possible codes include:

EBADF
: The *s* parameter is not a valid socket descriptor.

EINVAL
: The request is not correct or not supported.

listen()

The listen() call performs two tasks for a specified stream socket:

1. Completes the necessary binding if bind() has not been called for the socket.
2. Creates a connection request queue of a specified length to queue incoming connection requests.

The listen() call indicates a readiness to accept client connection requests. It transforms an active socket into a passive socket. A passive socket can never be used as an active socket to initiate connection requests.

Calling listen() is the third of four steps that a server performs to accept a connection. It is called after allocating a stream socket with socket(), and after binding a name to the socket with bind(). It must be called before calling accept() to accept a connection request from a client.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <socket.h>

int listen(int s, int backlog)
```

Parameters

s The socket descriptor.

backlog Defines the maximum length for the queue of pending connections.

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

Return values

The value 0 indicates success; the value −1 indicates an error. To determine which error occurred, check the *errno* global variable, which is set to a return code. Possible codes include:

EBADF
: The *s* parameter is not a valid socket descriptor.

EOPNOTSUPP
: The *s* parameter is not a socket descriptor that supports the listen() call.

read()

The read() call reads data on a specified connected socket.

Stream sockets act like streams of information with no boundaries separating data. For example, if applications A and B are connected with a stream socket and application A sends 1000 bytes, each call to this function can return one byte, or 10 bytes, or the entire 1000 bytes. Therefore, applications using stream sockets should place this call in a loop, which should repeat until all data has been received.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)

int read(int s, char *buf, int len)
```

Parameters

s The socket descriptor.

buf The pointer to the buffer that receives the data.

len The length in bytes of the buffer pointed to by the *buf* parameter.

Return values

If successful, the number of bytes copied into the buffer is returned. The value 0 indicates that the connection is closed. The value −1 indicates an error. To determine which error occurred, check the *errno* global variable, which is set to a return code. Possible codes include:

EBADF
s is not a valid socket descriptor.

EFAULT
Using the *buf* and *len* parameters results in an attempt to access storage outside the caller's address space.

EWOULDBLOCK
s is in nonblocking mode, and data is not available to read.

recv()

The recv() call receives data on a specified socket.

If a datagram packet is too long to fit in the supplied buffer, datagram sockets discard extra bytes. Stream sockets act like streams of information with no boundaries separating data. For example, if applications A and B are connected with a stream socket and application A sends 1000 bytes, each call to this function can return 1 byte, or 10 bytes, or up to 1000 bytes. Therefore, applications using stream sockets should place this call in a loop, calling this function until all data has been received.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <bsdtypes.h>
#include <socket.h>
int recvfrom(int s, char *buf, int len, int flags)
```

Parameters

s The socket descriptor.

buf The pointer to the buffer that receives the data.

len The length in bytes of the buffer pointed to by the *buf* parameter.

flags A parameter that can be set to 0 or MSG_PEEK.

MSG_OOB
Reads any out-of-band data on the socket.

MSG_PEEK
Peeks at the data present on the socket. The data is returned but not destroyed, so that a subsequent receive operation can recognize the same data.

Return values

If successful, the length of the message or datagram in bytes is returned. The value 0 indicates that the connection is closed. The value −1 indicates an error. To determine which error occurred, check the *errno* global variable, which is set to a return code. Possible codes include:

EBADF
s is not a valid socket descriptor.

EFAULT
Using the *buf* and *len* parameters results in an attempt to access storage outside the caller's address space.

EWOULDBLOCK
s is in nonblocking mode, and data is not available to read.

recvfrom()

The recvfrom() call receives data on a specified socket. The recvfrom() call applies to any datagram socket, whether connected or unconnected.

The call returns the length of the incoming message or data. If a datagram packet is too long to fit in the supplied buffer, datagram sockets discard extra bytes. Stream sockets act like streams of information with no boundaries separating data. For example, if applications A and B are connected with a stream socket and application A sends 1000 bytes, each call to this function can return 1 byte, or 10 bytes, or the entire 1000 bytes. Therefore, applications using stream sockets should place this call in a loop, calling this function until all data has been received.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <bsdtypes.h>
#include <socket.h>

int recvfrom(int s, char *buf, int len, int flags,
struct sockaddr *name, int *namelen)
```

Parameters

s The socket descriptor.

buf The pointer to the buffer that receives the data.

len The length in bytes of the buffer pointed to by the *buf* parameter.

flags A parameter that can be set to 0 or MSG_PEEK.

MSG_OOB
Reads any out-of-band data on the socket.

MSG_PEEK
Peeks at the data present on the socket. The data is returned but not destroyed, so that a subsequent receive operation can recognize the same data.

name A pointer to a *socket address* structure from which data is received. If *name* is a nonzero value, the source address is returned.

The following fields are used to define the IPv4 socket address structure of the socket that sent the data.

sin_family This field is set to AF_INET.

sin_port Contains the port number of the sending socket.

in_addr.sin_addr
Contains the 32-bit IPv4 Internet address, in network byte order, of the sending socket.

sin_zero This field is not used and is set to all zeros.

The following fields are used to define the IPv6 socket address structure of the socket that sent the data.

sin6_family This field is set to AF_INET6.

sin6_port Contains the port number bound of the sending socket.

sin6_flowinfo Contains the traffic class and flow label. The value of this field is undefined.

in6_addr.sin6_addr
Contains the 128-bit IPv6 Internet address, in network byte order, of the sending socket.

sin6_scope_id Identifies a set of interfaces as appropriate for the scope of the address carried in the *in6_addr.sin6_addr* field. For a link scope *in6_addr.sin6_addr*, *sin6_scope_id* contains the link index for the *in6_addr.sin6_addr*. For all other address scopes, *sin6_scope_id* is undefined.

namelen
A pointer to an integer that contains the size of *name* in bytes. For an IPv4 socket address, the *namelen* parameter contains a decimal 16. For an IPv6 socket address, the *namelen* parameter contains a decimal 28.

Return values

If successful, the length of the message or datagram in bytes is returned. The value 0 indicates that the connection is closed. The value −1 indicates an error. To determine which error occurred, check the *errno* global variable, which is set to a return code. Possible codes include:

EBADF
s is not a valid socket descriptor.

EFAULT
Using the *buf* and *len* parameters results in an attempt to access storage outside the caller's address space.

EWOULDBLOCK
s is in nonblocking mode, and data is not available to read.

select()

The select() call is useful in processes where multiple operations can occur, and 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 blocks on a read() call, only one socket could be read at a time. Setting the sockets nonblocking solves this problem, but requires 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 does not block.

Defining which sockets to test

The select() call monitors for read operations, write operations, and exception operations:

- When a socket is ready to read, either:
  - A buffer for the specified sockets contains input data. If input data is available for a given socket, a read operation on that socket does 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 does not block.
- When an exception condition has occurred on a specified socket, it is an indication that a takesocket() has occurred for that socket.

Each socket is represented by a bit in a bit string. The bit strings are contained in 32-bit fullwords, numbered from right-to-left. The right-most bit represents socket 0, the leftmost bit represents socket 31, and so on. Thus, if the process uses 32 (or less) sockets, the bit string is one word long; if the process uses up to 64 sockets, the bit string is two words long, etc. You define which sockets to test by turning on the corresponding bit in the bit string.

Read operations: Read operations include accept(), read(), recv(), or recvfrom() 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 READFDS to '1' before issuing the select() call. When the select() call returns, the corresponding bits in the READFDS indicate sockets ready for reading.

Write operations: A socket is selected for writing (ready to be written) when:

- TCP/IP can accept additional outgoing data.
- A connection request is received in response to an accept() call.
- The socket is marked nonblocking, and a connect() cannot be completed immediately. In this case, ERRNO contains a value of 36 (EINPROGRESS). This is not an error condition.

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. After 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 WRITEFDS bits representing those sockets to 1 before issuing the select() call. When the select() call returns, the corresponding bits in the WRITEFDS indicate sockets 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 returns the out-of-band data ahead of program data.

To test whether any of several sockets have an exception condition, set the EXCEPTFDS bits representing those sockets to 1. When the select() call returns, the corresponding bits in the EXCEPTFDS indicate sockets with exception conditions.

NFDS parameter: The select() call tests each bit in each string before returning results. For efficiency, the NFDS parameter can be used to specify the number of socket descriptors that need to be tested for any event type. The select() call tests only bits in the range 0 through the (NFDS-1) value.

TIMEOUT parameter: If the time specified in the TIMEOUT parameter elapses before any event is detected, the select() call returns, and RETCODE is set to 0.

Format: This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <socket.h>
#include <bsdtypes.h>
#include <bsdtime.h>

int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds,
struct timeval *timeout)
```

Parameters:

nfds The number of socket descriptors to check.

readfds The pointer to a bit mask of descriptors to check for reading.

writefds
The pointer to a bit mask of descriptors to check for writing.

exceptfds
The pointer to a bit mask of descriptors to be checked for exceptional pending conditions.

timeout
The pointer to the time to wait for the select() call to complete. If *timeout* is a NULL pointer, a zero-valued timeval structure is substituted in the call. The zero-valued timeval structure causes TCP/IP stacks to poll the sockets and return immediately to the caller.

Return values: A positive value represents the total number of ready sockets in all bit masks. The value 0 indicates an expired time limit. The three bit masks indicate status (with one bit for each socket). A 1-bit indicates that the respective socket is ready; a 0-bit indicates that the respective socket is not ready. You can use the macro FD_ISSET [10] with each socket to test its status.

The value −1 indicates an error. To determine which error occurred, check the *errno* global variable, which is set to a return code. Possible codes include:

EBADF
One of the bit masks specified an incorrect socket. FD_ZERO was probably not called to clear the bit mask before the sockets were set.

EFAULT
One of the bit masks pointed to a value outside the caller's address space.

EINVAL
One of the fields in the timeval structure is not correct.

send()

The send() call sends data on an already-connected socket.

The select() call can be used prior to issuing the send() call to determine when it is possible to send more data.

Stream sockets act like streams of information with no boundaries separating data. For example, if an application is required to send 1000 bytes, each call to this function can send 1 byte, or 10 bytes, or the entire 1000 bytes. Therefore, applications using stream sockets should place this call in a loop, calling this function until all data has been sent.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <bsdtypes.h>
#include <socket.h>

int send(int s, char *msg, int len, int flags)
```

Parameters

s
The socket descriptor.

msg
The pointer to the buffer that contains the message to transmit.

len
The length of the message pointed to by the *buf* parameter.

flags
The *flags* parameter is set by specifying one or more of the following flags. If more than one flag is specified, the logical OR operator (|) must be used to separate them.

MSG_OOB
Sends out-of-band data.

MSG_DONTROUTE
The SO_DONTROUTE option is turned on for the duration of the operation. This is usually used only by diagnostic or routing programs.

10. See *z/OS Communications Server: IP Programmer's Guide and Reference* for details.

Return values

A positive value represents the number of bytes sent. The value −1 indicates locally detected errors. When datagram sockets are specified, no indication of failure to deliver is implicit in a send() routine.

To determine which error occurred, check the *errno* global variable, which is set to a return code. Possible codes include:

EBADF
: *s* is not a valid socket descriptor.

EFAULT
: Using the *buf* and *len* parameters results in an attempt to access storage outside the caller's address space.

ENOBUFS
: Buffer space is not available to send the message.

EWOULDBLOCK
: *s* is in nonblocking mode and data is not available to read.

sendto()

The sendto() call sends data to the address specified in the call.

Stream sockets act like streams of information with no boundaries separating data. For example, if an application wishes to send 1000 bytes, each call to this function can send 1 byte, or 10 bytes, or the entire 1000 bytes. Therefore, applications using stream sockets should place this call in a loop, calling this function until all data has been sent.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <bsdtypes.h>
#include <socket.h>

int sendto(int s, char *msg, int len, int flags,
struct sockaddr *to, int tolen)
```

Parameters

s The socket descriptor.

msg The pointer to the buffer that contains the message to transmit.

len The length of the message in the buffer pointed to by the *msg* parameter.

flags A parameter that can be set to 0 or MSG_DONTROUTE.

MSG_DONTROUTE
: The SO_DONTROUTE option is turned on for the duration of the operation. This is usually used only by diagnostic or routing programs.

to The address of the target socket address structure.

The following fields are used to define the IPv4 socket address structure the data is sent to.

sin_family Must be set to AF_INET.

sin_port Set to the port number bound to the socket.

in_addr.sin_addr
Set to the 32-bit IPv4 Internet address in network byte order.

sin_zero This field is not used and must be set to all zeros.

The following fields are used to specify the IPv6 socket address structure the data is sent to.

sin6_family Must be set to AF_INET6.

sin6_port Set to the port number bound to the socket.

sin6_flowinfo Used to specify the traffic class and flow label. This field must be set to zero.

in6_addr.sin6_addr
Set to the 128-bit IPv6 Internet address in network byte order.

sin6_scope_id Used to identify a set of interfaces as appropriate for the scope of the address carried in the *in6_addr.sin6_addr* field. A value of zero indicates the *sin6_scope_id* does not identify the set of interfaces to be used, and might be specified for any address types and scopes. For a link scope *in6_addr.sin6_addr*, *sin6_scope_id* might specify a link index which identifies a set of interfaces. For all other address scopes, *sin6_scope_id* is undefined.

tolen The size of the structure pointed to by *to*. For an IPv4 socket address, the *tolen* parameter contains a decimal 16. For an IPv6 socket address, the *tolen* parameter contains a decimal 28.

Return values

If positive, indicates the number of bytes sent. The value −1 indicates an error. No indication of failure to deliver is implied in the return value of this call when used with datagram sockets.

To determine which error occurred, check the *errno* global variable, which is set to a return code. Possible codes include:

EBADF
s is not a valid socket descriptor.

EFAULT
Using the *buf* and *len* parameters results in an attempt to access storage outside the caller's address space.

EINVAL
tolen is not the size of a valid address for the specified address family.

EMSGSIZE
The message was too big to be sent as a single datagram. The default is large-envelope-size.

ENOBUFS
Buffer space is not available to send the message.

EWOULDBLOCK
s is in nonblocking mode, and data is not available to read.

setipv4sourcefilter()

Sets a list of the IPv4 source addresses that comprise the source filter, along with the current mode on a given interface and a multicast group for a socket. The source filter can either include or exclude the set of source addresses, depending on the filter mode (MCAST_INCLUDE or MCAST_EXCLUDE).

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanfies.h> (reentrant programs only)
#include <netinet/in.h>
int setipv4sourcefilter (int s, struct in_addr interface,
                         struct in_addr group, uint32_t fmode,
                         uint32_t numsrc, struct in_addr *slist);
```

Parameters

s The socket descriptor.

interface
: The local IP address of the interface.

group The IP multicast address of the group.

fmode An integer that contains the filter mode to be set. The value of the filter mode can be MCAST_INCLUDE or MCAST_EXCLUDE.

numsrc
: The number of source addresses in the *slist* array.

slist A pointer to an array of IP addresses that is either included or excluded, depending on the filter mode. If the *numsrc* value 0 was specified on input, you can specify a NULL pointer. A maximum of 64 IP addresses can be specified.

Return values

When successful, the value 0 is returned. When an error occurs, the value -1 is returned and the errno value is one of the following:

EBADF
: The *s* parameter value is not a valid socket descriptor

EINVAL
: The *interface* or *group* parameter value is not a valid IPv4 address, the specified *fmode* value is not valid, or the socket *s* has already requested multicast setsockopt options. For more information, see *z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference*.

EPROTOTYPE
: The socket protocol type is not correct.

ENOBUFS
: The number of source addresses exceeds the allowed limit.

ENOMEM
: Insufficient storage is available to supply the array.

EADDRNOTAVAIL
: The specified interface address is incorrect for this host, or the specified interface address is not multicast capable.

setsockopt()

See "getsockopt(), setsockopt()" on page 187.

setsourcefilter()

Sets a list of the IPv4 or IPv6 source addresses that comprise the source filter, along with the current mode on a given interface and a multicast group for a socket. The source filter can either include or exclude the set of source addresses, depending on the filter mode (MCAST_INCLUDE or MCAST_EXCLUDE).

Format

This call has the following format:

```
 #include <manifest.h> (non-reentrant programs only)
#include <cmanfies.h> (reentrant programs only)
#include <netinet/in.h>
int setsourcefilter(int s, uint32_t interface,
struct sockaddr *group, socklen_t grouplen,
uint32_t fmode, uint32_t numsrc,
struct sockaddr_storage *slist);
```

Parameters

s
: The socket descriptor.

interface
: The interface index of the interface.

group
: A pointer to either a *sockaddr_in* structure for IPv4 addresses or a *sockaddr_in6* structure for IPv6 addresses. The pointer holds the IP multicast address of the group.

grouplen
: The length of the *sockaddr_in* or *sockaddr_in6* structure.

fmode
: An integer that contains the filter mode to be set. The value of the filter mode can be either MCAST_INCLUDE or MCAST_EXCLUDE.

numsrc
: An integer that specifies the number of source addresses that are provided in the array that is pointed to by the *slist* parameter.

slist
: A pointer to an array of IP addresses that is either included or excluded, depending on the filter mode. If the *numsrc* value 0 was specified on input, you can specify a NULL pointer.

Return values

When successful, the value 0 is returned. When an error occurs, the value -1 is returned and the errno value is one of the following:

EBADF
: The *s* parameter value is not a valid socket descriptor.

EAFNOSUPPORT
: The address family of the input sockaddr value is not AF_INET or AF_INET6.

EINVAL
: The socket address family of an input parameter is not correct, the specified *fmode* value is not correct, or the socket specified by the *s* parameter already requested multicast setsockopt options. See *z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference* for more information.

ENOBUFS
: The number of source addresses exceeds the allowed limit.

EPROTOTYPE
: The socket protocol type is not correct.

ENOMEM
: Insufficient storage is available to supply the array.

ENXIO
: The specified interface index provided in the *interface* parameter does not exist.

shutdown()

The shutdown() call shuts down all or part of a duplex connection.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <socket.h>

int shutdown(int s, int how)
```

Parameters

s The socket descriptor.

how The *how* parameter can have a value of 0, 1, or 2, where:
- 0 ends communication from socket *s*.
- 1 ends communication to socket *s*.
- 2 ends communication both to and from socket *s*.

Return values

The value 0 indicates success; the value −1 indicates an error. To determine which error occurred, check the *errno* global variable, which is set to a return code. Possible codes include:

EBADF
: *s* is not a valid socket descriptor.

EINVAL
: The *how* parameter was not set to one of the valid values. Valid values are 0, 1, and 2.

socket()

The socket() call creates an endpoint for communication and returns a socket descriptor representing the endpoint. Different types of sockets provide different communication services.

SOCK_STREAM sockets model duplex byte streams. They provide reliable, flow-controlled connections between peer applications. Stream sockets are either active or passive. Active sockets are used by clients that initiate connection requests with connect(). By default, socket() creates active sockets. Passive sockets are used by servers to accept connection requests with the connect() call. An active socket is transformed into a passive socket by binding a name to the socket with the bind() call and by indicating a willingness to accept connections with the listen() call. After a socket is passive, it cannot be used to initiate connection requests.

SOCK_DGRAM supports datagrams (connectionless messages) of a fixed maximum length. Transmission reliability is not guaranteed. Datagrams can be corrupted, received out of order, lost, or delivered multiple times.

Sockets are deallocated with the close() call.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <bsdtypes.h>
#include <socket.h>

int socket(int domain, int type, int protocol)
```

Parameters

domain The *domain* parameter specifies a communication domain within which communication is to take place. This parameter selects the address family (format of addresses within a domain) that is used. The only families supported by CICS TCP/IP are AF_INET and AF_INET6, which are both the Internet domain. The AF_INET and AF_INET6 constant is defined in the socket.h header file.

type The *type* parameter specifies the type of socket created. These socket type constants are defined in the socket.h header file.

This must be set to either SOCK_STREAM or SOCK_DGRAM.

protocol
The *protocol* parameter specifies a particular protocol to be used with the socket. In most cases, a single protocol exists to support a particular type of socket in a particular addressing family. If the *protocol* parameter is set to 0, the system selects the default protocol number for the domain and socket type requested. Protocol numbers are found in the *hlq*.ETC.PROTO data set. The default *protocol* for stream sockets is TCP. The default *protocol* for datagram sockets is UDP.

Return values

A nonnegative socket descriptor indicates success. The value –1 indicates an error. To determine which error occurred, check the *errno* global variable, which is set to a return code. Possible codes include:

EPROTONOSUPPORT
The *protocol* is not supported in this *domain*, or this *protocol* is not supported for this socket *type*.

takesocket()

The takesocket() call acquires a socket from another program. The CICS listener passes the client ID and socket descriptor in the COMMAREA.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <bsdtypes.h>
#include <socket.h>

int takesocket(struct clientid *client_id, int hisdesc)
```

Parameters

clientid A pointer to the clientid of the application from which you are taking a socket.

domain
Sets the domain of the program giving the socket. Set as either AF_INET (a decimal 2) or AF_INET6 (a decimal 19).

Rule: An AF_INET socket can be taken only from an AF_INET givesocket(). An AF_INET6 socket can be taken only from an AF_INET6 givesocket(). EBADF is set if the domain does not match.

name
Set to the address space identifier of the program that gave the socket.

subtaskname
Set to the task identifier of the task that gave the socket.

reserved
Binary zeros.

hisdesc The descriptor of the socket to be taken.

Return values

A nonnegative socket descriptor is the descriptor of the socket to be used by this process. The value −1 indicates an error. To determine which error occurred, check the *errno* global variable, which is set to a return code. Possible codes include:

EACCES
The other application did not give the socket to your application.

EBADF
The *hisdesc* parameter does not specify a valid socket descriptor owned by the other application. The socket has already been taken.

EFAULT
Using the *clientid* parameter as specified results in an attempt to access storage outside the caller's address space.

EINVAL
The *clientid* parameter does not specify a valid client identifier.

EMFILE
The socket descriptor table is already full.

ENOBUFS
The operation cannot be performed because of the shortage of SCB or SKCB control blocks in the TCP/IP address space.

EPFNOSUPPORT
The domain field of the *clientid* parameter is not AF_INET or AF_INET6.

write()

This call writes data on a connected socket.

Stream sockets act like streams of information with no boundaries separating data. For example, if an application wishes to send 1000 bytes, each call to this function can send 1 byte or 10 bytes or the entire 1 000 bytes. Therefore, applications using stream sockets should place this call in a loop, calling this function until all data has been sent.

Format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <socket.h>

int write(int s, char *buf, int len)
```

Parameters

s The socket descriptor.

buf The pointer to the buffer holding the data to be written.

len The length in bytes of the buffer pointed to by the *buf* parameter.

Return values

If successful, the number of bytes written is returned. The value −1 indicates an error. To determine which error occurred, check the *errno* global variable, which is set to a return code. Possible codes include:

EBADF
s is not a valid socket descriptor.

EFAULT
Using the *buf* and *len* parameters results in an attempt to access storage outside the caller’s address space.

ENOBUFS
Buffer space is not available to send the message.

EWOULDBLOCK
s is in nonblocking mode and data is not available to write.

Address Testing Macros

This topic describes the macros that can be used to test for special IPv6 addresses.

```
#include <netinet/in.h>

int IN6_IS_ADDR_UNSPECIFIED (const struct in6_addr *)

int IN6_IS_ADDR_LOOPBACK (const struct in6_addr *)

int IN6_IS_ADDR_MULTICAST (const struct in6_addr *)

int IN6_IS_ADDR_LINKLOCAL (const struct in6_addr *)

int IN6_IS_ADDR_SITELOCAL (const struct in6_addr *)

int IN6_IS_ADDR_V4MAPPED (const struct in6_addr *)

int IN6_IS_ADDR_V4COMPAT (const struct in6_addr *)

int IN6_IS_ADDR_MC_NODELOCAL (const struct in6_addr *)

int IN6_IS_ADDR_MC_LINKLOCAL (const struct in6_addr *)

int IN6_IS_ADDR_MC_SITELOCAL (const struct in6_addr *)

int IN6_IS_ADDR_MC_ORGLOCAL (const struct in6_addr *)

int IN6_IS_ADDR_MC_GLOBAL (const struct in6_addr *)
```

IN6_IS_ADDR_UNSPECIFIED
Returns true if the address is the unspecified IPv6 address (in6addr_any). Otherwise, the macro returns false.

IN6_IS_ADDR_LOOPBACK
Returns true if the address is an IPv6 loopback address. Otherwise, the macro returns false.

IN6_IS_ADDR_MULTICAST
Returns true if the address is an IPv6 multicast address. Otherwise, the macro returns false.

IN6_IS_ADDR_LINKLOCAL
Returns true if the address is an IPv6 link local address. Otherwise, the macro returns false.

Returns true for local-use IPv6 unicast addresses.

Returns false for the IPv6 loopback address.

Does not return true for IPv6 multicast addresses of link-local scope.

IN6_IS_ADDR_SITELOCAL
Returns true if the address is an IPv6 site local address. Otherwise, the macro returns false.

Returns true for local-use IPv6 unicast addresses.

Does not return true for IPv6 multicast addresses of site-local scope.

IN6_IS_ADDR_V4MAPPED
Returns true if the address is an IPv4 mapped IPv6 address. Otherwise, the macro returns false.

IN6_IS_ADDR_V4COMPAT
Returns true if the address is an IPv4 compatible IPv6 address. Otherwise, the macro returns false.

IN6_IS_ADDR_MC_NODELOCAL
Used to test the scope of a multicast address and returns true if the address is a multicast address of the specified scope or false if the address is not a multicast address or not of the specified scope.

IN6_IS_ADDR_MC_LINKLOCAL
Used to test the scope of a multicast address and returns true if the address is a multicast address of the specified scope or false if the address is either not a multicast address or not of the specified scope.

IN6_IS_ADDR_MC_SITELOCAL
Used to test the scope of a multicast address and returns true if the address is a multicast address of the specified scope or false if the address is either not a multicast address or not of the specified scope.

IN6_IS_ADDR_MC_ORGLOCAL
Used to test the scope of a multicast address and returns true if the address is a multicast address of the specified scope or false if the address is either not a multicast address or not of the specified scope.

IN6_IS_ADDR_MC_GLOBAL
Used to test the scope of a multicast address and returns true if the address is a multicast address of the specified scope or false if the address is either not a multicast address or not of the specified scope.

Chapter 8. Sockets extended API

This topic contains information about the sockets extended application programming interface (API).

Environmental restrictions and programming requirements

The following environmental restrictions and programming requirements apply to the Callable Socket API:

- SRB mode

  This API can only be invoked in TCB mode (task mode).
- Cross-memory mode

  This API can only be invoked in a non-cross-memory environment (PASN=SASN=HASN).
- Functional Recovery Routine (FRR)

  Do not invoke this API with an FRR set. This causes system recovery routines to be bypassed and severely damage the system.
- Locks

  No locks should be held when issuing this call.
- INITAPI, INITAPIX, and TERMAPI calls

  The INITAPI, INITAPIX, and TERMAPI calls must be issued under the same task.
- Storage

  Storage acquired for the purpose of containing data returned from a socket call must be obtained in the same key as the application program status word (PSW) at the time of the socket call.
- Nested socket API calls

  You can not 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 an error.

CALL instruction API

This topic 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. Only one copy of an interface can exist in a single address space.
3. For a PL/I program, include the following statement before your first call instruction.

   ```
   DCL EZASOKET ENTRY OPTIONS(RETCODE,ASM,INTER) EXT;
   ```
4. The entry point for the CICS Sockets Extended module (EZASOKET) is within the EZACICAL module; therefore, EZACICAL should be included explicitly in

your link-editing JCL. If not included, you could experience problems, such as the CICS region waiting for the socket calls to complete.

See Figure 175 on page 368.

If you do not want to explicitly include EZACICAL in your link-edit JCL then you can use the EZACICSO CICS Sockets Extended module. The EZACICSO CICS Sockets Extended module is an ALIAS for EZASOKET that resides in the same entry point in EZACICAL as EZASOKET. You must also substitute any ″CALL EZASOKET″ invocations in your program with ″CALL EZACICSO″. This allows you to use the Binder's Automatic Library Call option (AUTOCALL) to build your load modules.

Note: SEZATCP load library data set needs to be included in the SYSLIB DD concatenation.

Understanding COBOL, assembler, and PL/I call formats

This API is invoked by calling the EZASOKET or EZACICSO 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.────────►◄
```

The following is the ′EZACICSO′ call format for the COBOL language programs.

```
►►──CALL 'EZACICSO' USING SOC-FUNCTION──parm1, parm2, ...──ERRNO RETCODE.────────►◄
```

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

parm*n* A variable number of parameters depending on the type of 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. Because DATAREG is used to access the application's working storage, applications using the assembler language format should not code DATAREG but should let it default to the CICS data register.

```
►►──CALL EZASOKET,(SOC-FUNCTION,──parm1, parm2, ...──ERRNO RETCODE),VL,MF=(E, PARMLIST)────────►◄
```

The following is the 'EZACICSO' call format for assembler language programs.

```
►►──CALL EZACICSO,(SOC-FUNCTION,──parm1, parm2, ...──ERRNO RETCODE),VL,MF=(E, PARMLIST)──────────►◄
```

PARMLIST
A remote parameter list defined in dynamic storage DFHEISTG. This list contains addresses of 30 parameters that can be referenced by all execute forms of the CALL.

Note: This form of CALL is necessary to meet the CICS requirement for quasi-reentrant programming.

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

parm *n*
A variable number of parameters depending on the type call.

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

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

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);──────────────────►◄
```

The following is the 'EZACICSO' call format for the PL/I language programs.

```
►►──CALL EZACICSO (SOC-FUNCTION──parm1, parm2, ...──ERRNO RETCODE);──────────────────►◄
```

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

parm*n* A variable number of parameters depending on the type call.

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

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

Converting parameter descriptions

The parameter descriptions in this topic 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 118 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(8) COMP                     FULLWORD BINARY VALUE
  PIC   X(n)                         CHARACTER FIELD OF N BYTES

PL/1 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 118. Storage definition statement examples

Error messages and return codes

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

For information about error codes that are returned by TCP/IP, see Appendix B. Return codes.

Code CALL instructions

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

ACCEPT

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

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

1. Accepts the first connection on a queue of pending connections.
2. Creates a new socket with the same properties as s, and returns its descriptor in RETCODE. The original sockets remain available to the calling program to accept more connection requests.

3. The address of the client is returned in NAME for use by subsequent server calls.

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 blocks 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 do 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:	PASN = HASN
Amode:	31-bit or 24-bit
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 119 on page 228 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  FLOW-INFO   PIC 9(8) BINARY.
        03  IP-ADDRESS.
            05  FILLER  PIC 9(16) BINARY.
            05  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 119. ACCEPT call instructions example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 226.

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 IPv4 socket address.

  FAMILY
  A halfword binary field specifying the addressing family. The call returns the decimal value of 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 IPv6 socket address.

  FAMILY
  A halfword binary field specifying the addressing family. The call returns the decimal value of 19 for AF_INET6.

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

  FLOW-INFO
  A fullword binary field specifying the traffic class and flow label. The 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 IP-ADDRESS field. For a link scope IP-ADDRESS, SCOPE-ID contains the link index for the IP-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 B. Return codes on page 397 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.

BIND

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

The BIND call can either specify the required port or let the system choose the port. A listener program should always bind to the same well-known port, so that clients know what socket address to use when attempting to connect.

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 macro. For more information, see *z/OS Communications Server: IP Configuration Reference*.

In the AF_INET or AF_INET6 domain, the BIND call for a stream socket can specify the networks from which it is willing to accept connection requests. The application can fully specify the network interface by setting the IP-ADDRESS field to the Internet address of a network interface. Alternatively, the application can use a *wildcard* to specify that it wants to receive connection requests from any network interface. This is done by setting the IP-ADDRESS field to the value of INADDR-ANY or IN6ADDR-ANY.

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
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 120 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  FLOW-INFO   PIC 9(8) BINARY.
        03  IP-ADDRESS.
            05  FILLER  PIC 9(16) BINARY.
            05  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 120. BIND call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 226.

Parameter values set by the application

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

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

NAME

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

  FAMILY
  A halfword binary field specifying the addressing family. The value is set to a 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: If PORT is set to 0 when the call is issued, the system assigns the port number for the socket. The application can call the GETSOCKNAME call after the BIND call to discover the assigned port number.

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

  RESERVED
  Specifies an eight-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 addressing family. The value is set to a 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: If PORT is set to 0 when the call is issued, the system assigns the port number for the socket. The application can call the GETSOCKNAME call after the BIND call to discover the assigned port number.

  FLOW-INFO
  A fullword binary field specifying the traffic class and flow label. This field must be set to zero.

  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 IP-ADDRESS field. A value of zero indicates the SCOPE-ID field does not identify the set of interfaces to be used, and can be specified for any address types and scopes. For a link scope IP-ADDRESS, SCOPE-ID can specify a link index which identifies a set of interfaces. For all other address scopes, SCOPE-ID must be set to zero.

Parameter values returned to the application

ERRNO
A fullword binary field. If RETCODE is negative, this field contains an error number. See Appendix B. Return codes on page 397, 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

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 338 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 might be incomplete. The SETSOCKET call can be used to set a linger condition, in which TCP/IP continues 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 326.
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:

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
Amode:	31-bit or 24-bit
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 121 on page 233 shows an example of CLOSE call instructions.

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


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

Figure 121. CLOSE call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 226.

Parameter values returned to 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 set by the application

ERRNO
A fullword binary field. If RETCODE is negative, this field contains an error number. See Appendix B. Return codes on page 397 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

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:

1. It completes the binding process for a stream socket if a BIND call has not been previously issued.
2. 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 RETCODE of 0 indicates that the connection was completed.
  - A nonzero RETCODE with an ERRNO of 36 (EINPROGRESS) indicates that the connection is not completed but because the socket is nonblocking, the CONNECT call returns normally.

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

The completion cannot be checked by issuing a second CONNECT. For more information, see "SELECT" on page 307.

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
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 122 on page 235 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  FLOW-INFO   PIC 9(8) BINARY.
        03  IP-ADDRESS.
            05  FILLER  PIC 9(16) BINARY.
            05  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 122. CONNECT call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 226.

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

- A 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 addressing family. The value must be a 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 hexadecimal.

  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 '8104050C' in hexadecimal.

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

- A 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 addressing family. The value must be a 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 hexadecimal.

  FLOW-INFO
  A fullword binary field specifying the traffic class and flow label. This field must be set to zero.

  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-hexadecimal notation, it is set to X'12AB00000000CD300123456789ABCDEF'.

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

Parameter values returned to the application

ERRNO
A fullword binary field. If RETCODE is negative, this field contains an error number. See Appendix B. Return codes on page 397 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

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 278 for another way to control a socket's blocking mode.

Values for Command which are supported by the UNIX Systems Services fcntl callable service is also be accepted. See the *z/OS UNIX System Services Programming: Assembler Callable Services Reference* for more information.

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
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 123 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 123. FCNTL call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 226.

Parameter values set by the application

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

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

COMMAND
A fullword binary number with the following values.

Value	Description
3	Query the blocking mode of the socket
4	Set the mode to blocking or nonblocking for the socket

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

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

Parameter values returned to the application

ERRNO
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B. Return codes on page 397 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

FREEADDRINFO frees all the address information structures returned by GETADDRINFO in the RES parameter. Figure 124 shows an example of FREEADDRINFO call instructions.

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
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 124 shows an example of FREEADDRINFO call instructions.

```
WORKING-STORAGE SECTION.
    01  SOC-FUNCTION    PIC X(16) VALUE IS 'FREEADDRINFO'.
    01  ADDRINFO        PIC 9(8) BINARY.
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.

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

Figure 124. FREEADDRINFO call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 226.

Parameter values set by the application

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

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

Parameter values returned to the application

ERRNO
A fullword binary field. If RETCODE is negative, ERRNO contains an error number. See Appendix B. Return codes on page 397 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

GETADDRINFO

GETADDRINFO translates the name of a service location (for example, a host name), 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. Figure 125 on page 240 shows an example of GETADDRINFO call instructions.

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
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 125 on page 240 shows an example of GETADDRINFO call instructions.

```
WORKING-STORAGE SECTION.
    01  SOC-FUNCTION    PIC X(16) VALUE IS 'GETADDRINFO'.
    01  NODE            PIC X(255).
    01  NODELEN         PIC 9(8) BINARY.
    01  SERVICE         PIC X(32).
    01  SERVLEN         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-NUMERICSERV  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.
    01  HINTS           USAGE IS POINTER.
    01  RES             USAGE IS POINTER.
    01  CANNLEN         PIC 9(8) BINARY.
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.

LINKAGE SECTION.
    01  HINTS-ADDRINFO.
        03  FLAGS           PIC 9(8) BINARY.
        03  AF              PIC 9(8) BINARY.
        03  SOCTYPE         PIC 9(8) BINARY.
        03  PROTO           PIC 9(8) BINARY.
        03  FILLER          PIC 9(8) BINARY.
        03  FILLER          PIC 9(8) BINARY.
        03  FILLER          PIC 9(8) BINARY.
        03  FILLER          PIC 9(8) BINARY.
    01  RES-ADDRINFO.
        03  FLAGS           PIC 9(8) BINARY.
        03  AF              PIC 9(8) BINARY.
        03  SOCTYPE         PIC 9(8) BINARY.
        03  PROTO           PIC 9(8) BINARY.
        03  NAMELEN         PIC 9(8) BINARY.
        03  CANONNAME       USAGE IS POINTER.
        03  NAME            USAGE IS POINTER.
        03  NEXT            USAGE IS POINTER.

PROCEDURE DIVISION.
        MOVE 'www.hostname.com' TO NODE.
        MOVE 16 TO HOSTLEN.
        MOVE 'TELNET' TO SERVICE.
        MOVE 6 TO SERVLEN.
        SET HINTS TO ADDRESS OF HINTS-ADDRINFO.
        CALL 'EZASOKET' USING SOC-FUNCTION
              NODE NODELEN SERVICE SERVLEN HINTS
              RES CANNLEN ERRNO RETCODE.
```

Figure 125. GETADDRINFO call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 226.

Parameter values set by the application

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

NODE
Storage maximum of 255 bytes that contains the host name being queried. If the AI-NUMERICHOST flag is specified in the storage pointed to by the HINTS operand, then NODE should contain the queried hosts IP address in presentation form. This is an optional field but if specified you must also code NODELEN.

Scope information can be appended to the host name, using the format *node%scope information*. The combined length of the value specified must still fit within 255 bytes. For information about using scope information on GETADDRINFO processing, see *z/OS Communications Server: IPv6 Network and Application Design Guide* .

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

SERVICE
Storage maximum of 32 bytes that contains the service name being queried. If the AI-NUMERICSERV flag is specified in the storage pointed to by the HINTS operand, then SERVICE should contain the queried port number in presentation form. This is an optional field but if specified you must also code SERVLEN.

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

HINTS
If the HINTS argument is specified, it contains the address of an addrinfo structure containing input values that can direct the operation by providing options and by limiting the returned information to a specific socket type, address family, and protocol. If the HINTS argument is not specified, the information returned is 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 to enable your assembler program to contain the assembler mappings for the ADDR_INFO structure.

The EZBREHST macro is stored in SYS1.MACLIB, and it defines the Resolver hostent, addrinfo mappings, and services return codes. The hostent (host entry) is the name of the structure returned by the Resolver's gethostbyaddr() and gethostbyname() calls.

This is an optional field. The address information structure has the following fields:

Field — **Description**

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

AI-PASSIVE (X'00000001') or a decimal value of 1
Specifies how to fill in the name pointed to by the returned RES parameter.

If this flag is specified, the returned address information can be used to bind a socket for accepting incoming connections for the specified service (for example, using the BIND call). If you use the BIND call and if the NODE argument is not specified, the IP address portion of the socket address structure pointed to by the returned RES parameter is set to INADDR_ANY for an IPv4 address or to the IPv6 unspecified address (in6addr_any).

If this flag is not set, the returned address information can be used for the CONNECT call (for a connection-mode protocol) or on a CONNECT, SENDTO, or SENDMSG call (for a connectionless protocol). If you use a CONNECT call and if the NODE argument is not specified, the NAME pointed to by the returned RES is set to the loopback address.

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, 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, 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 on the system, the caller accepts 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, then 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 accepts all addresses (IPv6 and IPv4-mapped IPv6 addresses). When the AF field has a value of AF_UNSPEC, and when the system supports IPv6 and AI-ALL is set, the caller accepts both IPv6 and IPv4 addresses. A query is first made for IPv6 addresses and if successful, the IPv6 addresses are returned. Another query is then made for IPv4 addresses, 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, then this flag is ignored.

AI-ADDRCONFIG (X'00000040') or a decimal value of 64
If this flag is specified, a query on the name in nodename occurs if the resolver determines that one of the following is true:

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

Tip: To perform the binary OR'ing of the flags above in a COBOL program, add the necessary COBOL statements as in the following example. 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-ADDRCONFG TO FLAGS.
```

AF
A fullword binary field. Used to limit the returned information to a specific address family. The value of AF_UNSPEC means that the caller accepts 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 accepts any socket type. If a specific socket type is not given (for example, a value of 0), information about all supported socket types is returned.

The following are the acceptable socket types:

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

Anything else fails with return code EAI_SOCKTYPE. Although SOCK_RAW is accepted, it is only valid when SERVICE is numeric (for example, SERVICE=23). A lookup for a

SERVICE name never occurs in the appropriate services file (for example, *hlq*.ETC.SERVICES) using any protocol value other than SOCK_STREAM or SOCK_DGRAM. If PROTO is nonzero and SOCKTYPE is zero, the only acceptable input values for PROTO are IPPROTO_TCP and IPPROTO_UDP. Otherwise, the GETADDRINFO call fails with a return code of EAI_BADFLAGS. If SOCTYPE and PROTO are both specified as zero, GETADDRINFO proceeds as follows:

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

If both SOCTYPE and PROTO 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 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 accepts any protocol.

The following are the acceptable protocols:

Protocol Name	Decimal Value	Description
IPPROTO_TCP	6	TCP
IPPROTO_UDP	17	user datagram

If PROTO and SOCTYPE are both specified as zero, GETADDRINFO proceeds as follows:

- If SERVICE is null, or if SERVICE is numeric, any returned addrinfos default to a specification of SOCTYPE as SOCK_STREAM.
- If SERVICE is specified as a service name (for example, SERVICE=FTP), the GETADDRINFO call searches the appropriate services file (for example, *hlq*.ETC.SERVICES) file twice. The first search uses SOCK_STREAM as the protocol, and

the second search uses 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 [that is, the service name does not appear in the appropriate services file (for example, *hlq*.ETC.SERVICES) using the input protocol], the GETADDRINFO call fails with a 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.

RES Initially a fullword binary field. On a successful return this field contains a pointer to an addrinfo structure. This pointer is also used as input to the FREEADDRINFO call, which must be used to free storage obtained by this call. The structures returned by GETADDRINFO are a task's serially reusable storage area. Do not use or reference these structures between MVS tasks. The storage is freed when a FREEADDRINFO call is issued or when the task terminates.

The address information structure contains the following fields:

Field	Description
FLAGS	A fullword binary field that is not used as output.
AF	A fullword binary field. The value returned in this field can be used as the AF argument on the

SOCKET call to create a socket suitable for use with the returned address NAME.

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

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

NAMELEN
A fullword binary field. The length of the NAME socket address structure. The value returned in this field can be used as the arguments for the CONNECT or BIND call with this socket type, 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, the CANONNAME field in the first returned address information structure contains the address of storage containing the canonical name corresponding to the input NODE argument. If the canonical name is not available, the CANONNAME field refers 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 can be used as the arguments for the CONNECT or BIND call with this socket type, according to the AI-PASSIVE flag.

NEXT
A fullword binary field. Contains the address of the next address

information structure on the list, or zeros 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.

Parameter values returned to the application

ERRNO
ERRNO A fullword binary field. If RETCODE is negative, ERRNO contains an error number. See Appendix B. Return codes on page 397 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

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 straightforward manner. If you are coding in COBOL, this structure might be difficult to interpret. You can use the subroutine EZACIC09 to simplify interpretation of the information returned by the GETADDRINFO calls.

GETCLIENTID

GETCLIENTID call returns the identifier by which the calling application is known to the TCP/IP address space in the calling program. The CLIENT parameter is used in the GIVESOCKET and TAKESOCKET calls. See "GIVESOCKET" on page 274 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:

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
Amode:	31-bit or 24-bit
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 126 on page 248 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 126. GETCLIENTID call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 226.

Parameter values set by the application

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

Parameter values returned to the application

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

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

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

TASK An 8-byte character 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 B. Return codes on page 397 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

GETHOSTBYADDR

The GETHOSTBYADDR call returns the domain name and alias name of a host whose Internet address is specified in the call. A given TCP/IP host can have multiple alias names and multiple host Internet addresses.

The address resolution 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 using local host tables.

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
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 127 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 127. GETHOSTBYADDR call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 226.

Parameter values set by the application

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

HOSTADDR
A fullword binary field set to the Internet address (specified in network byte order) of the host whose name is being sought. See Appendix B. Return codes on page 397 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:

Value	Description
0	Successful call
–1	An error occurred

GETHOSTBYADDR returns the HOSTENT structure shown in Figure 128 on page 250.

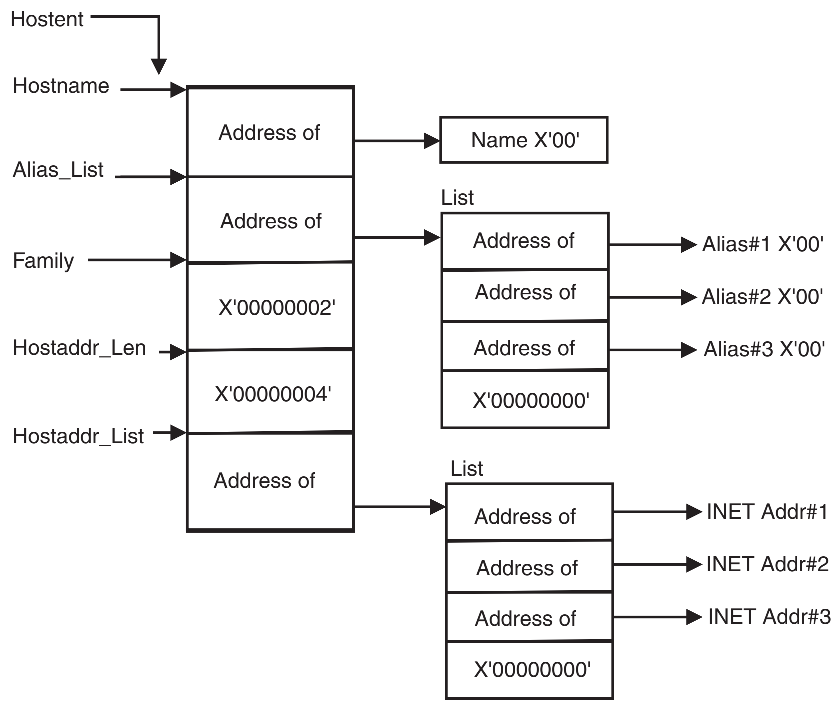


Figure 128. HOSTENT structure returned by the GETHOSTBYADDR call

This structure contains:

- The address of the host name that the call returns. 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 might be difficult to interpret. You can use the subroutine EZACIC08 to simplify interpretation of the information returned by the GETHOSTBYADDR and GETHOSTBYNAME calls. For more information about EZACIC08, see "EZACIC08" on page 356. If you are coding in assembler, this structure is defined in the EZBREHST macro.

GETHOSTBYNAME

The GETHOSTBYNAME call returns the alias name and the 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 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 using local host tables.

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
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 129 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 129. GETHOSTBYNAME call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 226.

Parameter values set by the application

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

NAMELEN
A value set to the length of the host name. The maximum is 255.

NAME
A character string, up to 255 characters, set to a host name. 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:

Value	Description
0	Successful call
−1	An error occurred

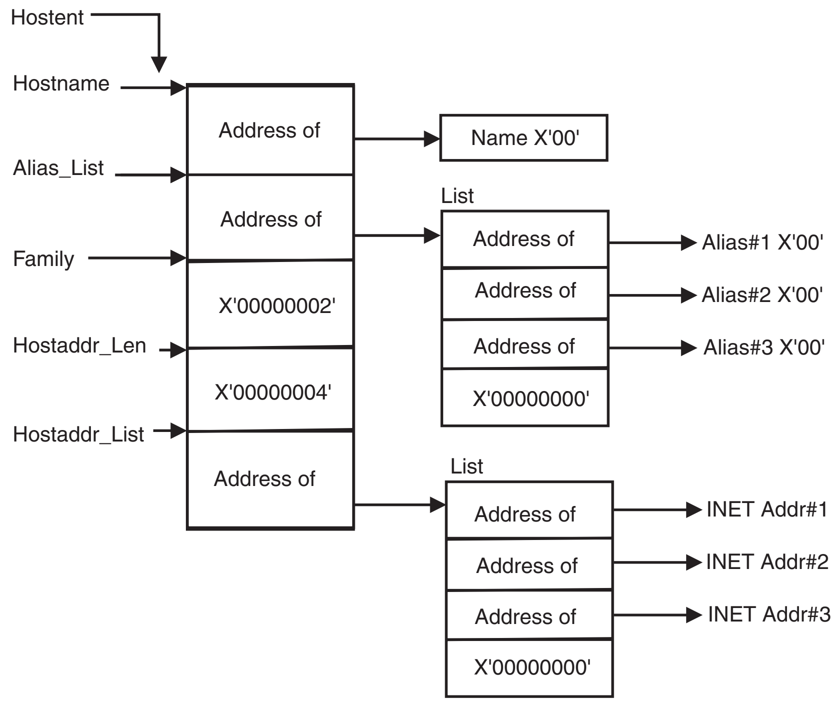


Figure 130. HOSTENT structure returned by the GETHOSTYBYNAME call

GETHOSTBYNAME returns the HOSTENT structure shown in Figure 130. This structure contains:

- The address of the host name that the call returns. 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 might be difficult to interpret. You can use the subroutine EZACIC08 to simplify interpretation of the information returned by the GETHOSTBYADDR and GETHOSTBYNAME calls. For more information about EZACIC08, see “EZACIC08” on page 356. If you are coding in assembler, this structure is defined in the EZBREHST macro.

GETHOSTID

The GETHOSTID call returns the 32-bit IPv4 Internet address for the current host.

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
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 131 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 131. GETHOSTID call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 226.

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

RETCODE
Returns a fullword binary field containing the 32-bit IPv4 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.

| The host name that is returned is the host name that the TCPIP stack learned at
| startup from the TCPIP.DATA file that was found.

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
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 132 shows an example of GETHOSTNAME call instructions.

```
WORKING-STORAGE SECTION.
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'GETHOSTNAME'.
    01  NAMELEN         PIC 9(8) BINARY.
    01  NAME            PIC X(24).
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.

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

Figure 132. GETHOSTNAME call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 226.

Parameter values set by the application

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

NAMELEN
A fullword binary field set to the length of the NAME field. The minimum length of the NAME field is 1 character. The maximum length of the NAME field is 255 characters.

Parameter values returned to the application

NAME
Indicates the receiving field for the host name. If the host name is shorter than the NAMELEN value, then the NAME field is filled with binary zeros after the host name. If the host name is longer than the NAMELEN value, then the name is truncated.

ERRNO
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B. Return codes on page 397 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

GETNAMEINFO

The GETNAMEINFO returns the node name and service location of a socket address that is specified in the call. On successful completion, GETNAMEINFO returns host name, host name length, service name, and service name length, if requested, in the buffers provided.

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
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 133 shows an example of GETNAMEINFO call instructions.

```
WORKING-STORAGE SECTION.
    01  SOC-FUNCTION    PIC X(16) VALUE IS 'GETNAMEINFO'.
    01  NAMELEN           PIC 9(8) BINARY.
    01  HOST              PIC X(255).
    01  HOSTLEN           PIC 9(8) BINARY.
    01  SERVICE           PIC X(32).
    01  SERVLEN           PIC 9(8) BINARY.
    01  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.
    01  NI-NUMERICSCOPE   PIC 9(8) BINARY VALUE 32.

* 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.

     MOVE 28 TO NAMELEN.
     MOVE 255 TO HOSTLEN.
     MOVE 32 TO SERVLEN.
     MOVE NI-NAMEREQD TO FLAGS.
     CALL 'EZASOKET' USING SOC-FUNCTION NAME NAMELEN HOST
           HOSTLEN SERVICE SERVLEN FLAGS ERRNO RETCODE.
```

Figure 133. GETNAMEINFO call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 226.

Parameter values set by the application

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

NAME

A socket address structure to be translated that has the following fields:

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

The IPv6 socket address structure specifies the following fields:

Field	Description
FAMILY	A halfword binary field specifying the IPv6 addressing family. For TCP/IP, the value is a decimal 19, indicating AF_INET6.
PORT	A halfword binary number specifying the port number.
FLOW-INFO	A fullword binary field specifying the traffic class and flow label. This field is not implemented.
IP-ADDRESS	A 16-byte binary field specifying the 128-bit IPv6 Internet address, in network byte order.
SCOPE-ID	A fullword binary field that specifies the link scope for an IPv6 address as an interface index. The resolver ignores the SCOPE-ID field, unless the address in the IP-ADDRESS field is a link-local address and the HOST parameter is also specified.

NAMELEN
A fullword binary field. The length of the socket address structure pointed to by the NAME argument.

HOST
On input, a storage area that is large enough to hold the returned resolved host name. The host name can be a maximum of 255 bytes, for the input socket address. If inadequate storage is specified to contain the resolved host name, then the resolver returns the host name value up to the storage amount specified and truncation can 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 this field is specified, you must also code the HOSTLEN parameter. Specify both the HOST and HOSTLEN parameters or both the SERVICE and SERVLEN parameters. An error occurs if both are omitted.

If the IP-ADDRESS value represents a link-local address, and the SCOPE-ID interface index is a nonzero value, scope information is appended to the resolved host name using the format *host%scope information*. The scope information can be either the numeric form of the SCOPE-ID interface index, or the interface name associated with the SCOPE-ID interface index.

Use the NI_NUMERICSCOPE option to select which form of scope information should be returned. The combined host name and scope information can be a maximum of 255 characters long. For more information about scope information and GETNAMEINFO processing, see the *z/OS Communications Server: IPv6 Network and Application Design Guide* for more information.

HOSTLEN
: An output parameter. A fullword binary field that contains the length of the host storage (HOST parameter) used to contain the resolved host name that is returned. The HOSTLEN value must be equal to or greater than the length of the longest host name, or the host name and scope information combination, to be returned. The GETNAMEINFO call returns the host name, or hostname and scope information combination, up to the length specified by the HOSTLEN parameter. On output, the HOSTLEN value contains the length of the returned resolved host name, or the host name and scope information combination. If the HOSTLEN value 0 is specified on input, then the resolved host name is not returned. This is an optional field, but if it is specified, you must also code the HOST parameter. Specify both the HOST and HOSTLEN parameters or both the SERVICE and SERVLEN parameters. An error occurs if both are omitted.

SERVICE
: On input, storage capable of holding the returned resolved service name, which can be a maximum of 32 bytes, for the input socket address. If inadequate storage is specified to contain the resolved service name, then the resolver returns the service name up to the storage specified and truncation can 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 specified, you must also code SERVLEN. Specify both the HOST and HOSTLEN parameters or both the SERVICE and SERVLEN parameters. An error occurs if both are omitted.

SERVLEN
: An output parameter. A fullword binary field. The length of the SERVICE storage used to contain the returned resolved service name. SERVLEN must be equal to or greater than the length of the longest service name to be returned. GETNAMEINFO returns the service name up to the length specified by SERVLEN. On output, SERVLEN contains the length of the returned resolved service name. If SERVLEN is 0 on input, then the service name information is not returned. This is an optional field but if specified you must also code SERVICE. Specify both the HOST and HOSTLEN parameters or both the SERVICE and SERVLEN parameters. An error occurs if both are omitted.

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:

Flag Name	Binary Value	Decimal Value	Description
'NI_NOFQDN'	X'00000001'	1	Return the NAME portion of the fully qualified domain name.
'NI_NUMERICHOST'	X'00000002'	2	Return only the numeric form of host's address.
'NI_NAMEREQD'	X'00000004'	4	Return an error if the host's name cannot be located.
'NI_NUMERICSERV'	X'00000008'	8	Return only the numeric form of the service address.
'NI_DGRAM'	X'00000010'	16	Indicates that the service is a datagram service. The default behavior is to assume that the service is a stream service.
'NI_NUMERICSCOPE'	X'00000020'	32	Return only the numeric form of the SCOPE-ID interface index, when applicable.

Parameter values returned to the application

ERRNO
A fullword binary field. If RETCODE is negative, ERRNO contains an error number. See Appendix B. Return codes on page 397 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

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:

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
Amode:	31-bit or 24-bit
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 134 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 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  FLOW-INFO   PIC 9(8) BINARY.
        03  IP-ADDRESS.
            05  FILLER  PIC 9(16) BINARY.
            05  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 134. GETPEERNAME call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 226.

Parameter values set by the application

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

S
A halfword binary number set to the socket descriptor of the local socket connected to the remote peer whose address is required.

Parameter values returned to the application

NAME

An IPv4 socket address structure to contain the peer name. The structure that is returned is the socket address structure for the remote socket that is 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 decimal value 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 eight-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 decimal value 19, indicating AF_INET6.

PORT A halfword binary field set to the connection peer's port number.

FLOW-INFO
A fullword binary field specifying the traffic class and flow label. The 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 that identifies a set of interfaces as appropriate for the scope of the address carried in the IP-ADDRESS field. For a link scope IP-ADDRESS, SCOPE-ID contains the link index for the IP-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 B. Return codes on page 397 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.

Because a stream socket is not assigned a name until after a successful call to either BIND, CONNECT, or ACCEPT, the GETSOCKNAME call can be used after an implicit bind to discover which port was assigned to the socket.

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
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 135 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  FLOW-INFO   PIC 9(8) BINARY.
        03  IP-ADDRESS.
            05  FILLER  PIC 9(16) BINARY.
            05  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 135. GETSOCKNAME call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 226.

Parameter values set by the application

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

S A halfword binary number set to the descriptor of a local socket whose address is required.

Parameter values returned to the application

NAME

Specifies the IPv4 socket address structure returned by the call.

FAMILY
A halfword binary field containing the addressing family. The call always returns the decimal value of 2, indicating AF_INET.

PORT A halfword binary field set to the port number bound to this socket. If the socket is not bound, zero is returned.

IP-ADDRESS
A fullword binary field set to the 32-bit IPv4 Internet address of the local host machine. If the socket is not bound, the address is the IPv6 unspecified address (in6addr_any).

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

Specifies the IPv6 socket address structure returned by the call.

FAMILY
A halfword binary field containing the addressing family. The call always returns the decimal value of 19, indicating AF_INET6.

PORT
A halfword binary field set to the port number bound to this socket. If the socket is not bound, zero is returned.

FLOW-INFO
A fullword binary field specifying the traffic class and flow label. The value of this field is undefined.

IP-ADDRESS
A 16-byte binary field set to the 128-bit IPv6 Internet address of the local host machine. If the socket is not bound, the address is IN6ADDR_ANY.

SCOPE-ID
A fullword binary field which identifies a set of interfaces as appropriate for the scope of the address carried in the IP-ADDRESS field. For a link scope IP-ADDRESS, SCOPE-ID contains the link index for the IP-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 B. Return codes on page 397 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

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:

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
Amode:	31-bit or 24-bit

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 136 shows an example of GETSOCKOPT call instructions.

```
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.

    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 136. GETSOCKOPT call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 226.

Parameter values set by the application

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

S
A halfword binary number specifying the socket descriptor for the socket requiring options.

OPTNAME
Input parameter. Set OPTNAME to the required option before you issue GETSOCKOPT. See the table below for a list of the options and their unique requirements. See Appendix C, “GETSOCKOPT/SETSOCKOPT command values,” on page 415 for the numeric values of OPTNAME.

Note: COBOL programs cannot contain field names with the underscore character. Fields representing the option name should contain dashes instead.

Parameter values returned to the application

OPTVAL
Output parameter. Contains the status of the specified option. See the table below for a list of the options and their unique requirements

OPTLEN
Output parameter. A fullword binary field containing the length of the data returned in OPTVAL. See the table below for how to determine the value of OPTLEN.

ERRNO
A fullword binary field. If RETCODE is negative, the field contains an

error number. See Appendix B, “Return codes,” on page 397 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.

Table 20. OPTNAME options for GETSOCKOPT and SETSOCKOPT

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
IP_ADD_MEMBERSHIP Use this option to enable an application to join a multicast group on a specific interface. An interface has to be specified with this option. Only applications that want to receive multicast datagrams need to join multicast groups. This is an IPv4-only socket option.	Contains the IP_MREQ structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 interface address. See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ. See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ.	N/A
IP_ADD_SOURCE_MEMBERSHIP Use this option to enable an application to join a source multicast group on a specific interface and a specific source address. You must specify an interface and a source address with this option. Applications that want to receive multicast datagrams need to join source multicast groups. This is an IPv4-only socket option.	Contains the IP_MREQ_SOURCE structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ_SOURCE structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 source address and a 4-byte IPv4 interface address. See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE. See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ-SOURCE.	N/A
IP_BLOCK_SOURCE Use this option to enable an application to block multicast packets that have a source address that matches the given IPv4 source address. You must specify an interface and a source address with this option. The specified multicast group must have been joined previously. This is an IPv4-only socket option.	Contains the IP_MREQ_SOURCE structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ_SOURCE structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 source address and a 4-byte IPv4 interface address. See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE. See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ-SOURCE.	N/A

Table 20. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
IP_DROP_MEMBERSHIP Use this option to enable an application to exit a multicast group or to exit all sources for a multicast group. This is an IPv4-only socket option.	Contains the IP_MREQ structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 interface address. See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ. See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ.	N/A
IP_DROP_SOURCE_MEMBERSHIP Use this option to enable an application to exit a source multicast group. This is an IPv4-only socket option.	Contains the IP_MREQ_SOURCE structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ_SOURCE structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 source address and a 4-byte IPv4 interface address. See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE. See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ-SOURCE.	N/A
IP_MULTICAST_IF Use this option to set or obtain the IPv4 interface address used for sending outbound multicast datagrams from the socket application. This is an IPv4-only socket option. **Note:** Multicast datagrams can be transmitted only on one interface at a time.	A 4-byte binary field containing an IPv4 interface address.	A 4-byte binary field containing an IPv4 interface address.
IP_MULTICAST_LOOP 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. This is an IPv4-only socket option.	A 1-byte binary field. To enable, set to 1. To disable, set to 0.	A 1-byte binary field. If enabled, will contain a 1. If disabled, will contain a 0.
IP_MULTICAST_TTL 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. This is an IPv4-only socket option.	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.

Table 20. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
IP_UNBLOCK_SOURCE Use this option to enable an application to unblock a previously blocked source for a given IPv4 multicast group. You must specify an interface and a source address with this option. This is an IPv4-only socket option.	Contains the IP_MREQ_SOURCE structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ_SOURCE structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 source address and a 4-byte IPv4 interface address. See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE. See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ-SOURCE.	
IPV6_JOIN_GROUP Use this option to control the reception of multicast packets and specify that the socket join a multicast group. This is an IPv6-only socket option.	Contains the IPV6_MREQ structure as defined in SYS1.MACLIB(BPXYSOCK). The IPV6_MREQ structure contains a 16-byte IPv6 multicast address followed by a 4-byte IPv6 interface index number. If the interface index number is 0, then the stack chooses the local interface. See the SEZAINST(CBLOCK) for the PL/I example of IPV6_MREQ. See SEZAINST(EZACOBOL) for the COBOL example of IPV6-MREQ.	N/A
IPV6_LEAVE_GROUP Use this option to control the reception of multicast packets and specify that the socket leave a multicast group. This is an IPv6-only socket option.	Contains the IPV6_MREQ structure as defined in SYS1.MACLIB(BPXYSOCK). The IPV6_MREQ structure contains a 16-byte IPv6 multicast address followed by a 4-byte IPv6 interface index number. If the interface index number is 0, then the stack chooses the local interface. See the SEZAINST(CBLOCK) for the PL/I example of IPV6_MREQ. See SEZAINST(EZACOBOL) for the COBOL example of IPV6-MREQ.	N/A

Table 20. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
IPV6_MULTICAST_HOPS Use to set or obtain the hop limit used for outgoing multicast packets. This is an IPv6-only socket option.	Contains a 4-byte binary value specifying the multicast hops. If not specified, then the default is 1 hop. -1 indicates use stack default. 0 – 255 is the valid hop limit range. **Note:** An application must be APF authorized to enable it to set the hop limit value above the system defined hop limit value. CICS applications cannot execute as APF authorized.	Contains a 4-byte binary value in the range 0 – 255 indicating the number of multicast hops.
IPV6_MULTICAST_IF Use this option to set or obtain the index of the IPv6 interface used for sending outbound multicast datagrams from the socket application. This is an IPv6-only socket option.	Contains a 4-byte binary field containing an IPv6 interface index number.	Contains a 4-byte binary field containing an IPv6 interface index number.
IPV6_MULTICAST_LOOP Use this option to control or determine whether a multicast datagram is looped back on the outgoing interface by the IP layer for local delivery when datagrams are sent to a group to which the sending host itself belongs. The default is to loop multicast datagrams back. This is an IPv6-only socket option.	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.
IPV6_UNICAST_HOPS Use this option to set or obtain the hop limit used for outgoing unicast IPv6 packets. This is an IPv6-only socket option.	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.
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. This is an IPv6-only socket option.	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.

Table 20. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
MCAST_BLOCK_SOURCE Use this option to enable an application to block multicast packets that have a source address that matches the given source address. You must specify an interface index and a source address with this option. The specified multicast group must have been joined previously.	Contains the GROUP_SOURCE_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_SOURCE_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address and a socket address structure of the source address. 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
MCAST_JOIN_SOURCE_GROUP Use this option to enable an application to join a source multicast group on a specific interface and a source address. You must specify an interface index and the source address. Applications that want to receive multicast datagrams only from specific source addresses need to join source multicast groups.	Contains the GROUP_SOURCE_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_SOURCE_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address and a socket address structure of the source address. See SEZAINST(CBLOCK) for the PL/I example of GROUP_SOURCE_REQ. See SEZAINST(EZACOBOL) for the COBOL example of GROUP-SOURCE-REQ.	N/A

Table 20. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
MCAST_LEAVE_GROUP Use this option to enable an application to exit a multicast group or exit all sources for a given multicast groups.	Contains the GROUP_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address. See SEZAINST(CBLOCK) for the PL/I example of GROUP_REQ. See SEZAINST(EZACOBOL) for the COBOL example of GROUP-REQ.	N/A
MCAST_LEAVE_SOURCE_GROUP Use this option to enable an application to exit a source multicast group.	Contains the GROUP_SOURCE_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_SOURCE_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address and a socket address structure of the source address. See SEZAINST(CBLOCK) for the PL/I example of GROUP_SOURCE_REQ. See SEZAINST(EZACOBOL) for the COBOL example of GROUP-SOURCE-REQ.	N/A
MCAST_UNBLOCK_SOURCE Use this option to enable an application to unblock a previously blocked source for a given multicast group. You must specify an interface index and a source address with this option.	Contains the GROUP_SOURCE_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_SOURCE_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address and a socket address structure of the source address. See SEZAINST(CBLOCK) for the PL/I example of GROUP_SOURCE_REQ. See SEZAINST(EZACOBOL) for the COBOL example of GROUP-SOURCE-REQ.	N/A

Table 20. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
SO_ASCII 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. **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_BROADCAST 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. **Note:** This option has no meaning for stream sockets.	A 4-byte binary field. To enable, set to 1 or a positive value. To disable, set to 0.	A 4-byte field. If enabled, contains a 1. If disabled, contains a 0.
SO_DEBUG 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 20. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<tr><th>OPTNAME options (input)</th><th>SETSOCKOPT, OPTVAL (input)</th><th>GETSOCKOPT, OPTVAL (output)</th></tr>
<tr><td>SO_KEEPALIVE

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>
<tr><td>SO_LINGER

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:
<code>ONOFF   DS F</code>
<code>LINGER DS F</code>

COBOL coding:
<code>ONOFF PIC 9(8) BINARY.</code>
<code>LINGER PIC 9(8) BINARY.</code>

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><td>Contains an 8-byte field containing two 4-byte binary fields.

Assembler coding:
<code>ONOFF   DS F</code>
<code>LINGER DS F</code>

COBOL coding:
<code>ONOFF PIC 9(8) BINARY.</code>
<code>LINGER PIC 9(8) BINARY.</code>

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.</td></tr>
</table>

Table 20. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
SO_OOBINLINE 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.	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.
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 20. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<tr><th>OPTNAME options (input)</th><th>SETSOCKOPT, OPTVAL (input)</th><th>GETSOCKOPT, OPTVAL (output)</th></tr>
<tr><td>SO_REUSEADDR

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, refer to the SHAREPORT option on the PORT statement in TCPIP.PROFILE.</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>
<tr><td>SO_SNDBUF

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 TCPSENDBufrsize keyword on the TCPCONFIG statement in the PROFILE.TCPIP data set for a TCP socket
• The UDPSENDBufrsize keyword on the UDPCONFIG statement in the PROFILE.TCPIP data set for a UDP socket
• The default of 65 535 for a raw socket</td><td>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.

To disable, set to a 0.</td><td>A 4-byte binary field.

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

If disabled, contains a 0.</td></tr>
<tr><td>SO_TYPE

Use this option to return the socket 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>
</table>

Table 20. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
TCP_KEEPALIVE 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-specified timer value remains in effect until respecified by SETSOCKOPT or until the socket is closed. Refer to the *z/OS Communications Server: IP Programmer's Guide and Reference* for more information on the socket option parameters.	A 4-byte binary field. To enable, set to a value in the range of 1 – 2 147 460. To disable, set to a value of 0.	A 4-byte binary field. If enabled, contains the specific timer value (in seconds) that is in effect for the given socket. If disabled, contains a 0 indicating keep alive timing is not active.

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:
`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.`
	A 4-byte binary field.
	
	To enable, set to a 0.
	
	To disable, set to a 1 or nonzero.
		A 4-byte binary field.
		
		If enabled, contains a 0.
		
		If disabled, contains a 1.

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 which have been given, but not taken for a period of four days, are closed and are no longer be available for taking. If a select for the socket is outstanding, it is posted.

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
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 137 shows an example of GIVESOCKET call instructions.

```
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 S9(8) BINARY.

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

Figure 137. GIVESOCKET call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 226.

Parameter values set by the application

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

S A halfword binary number set to the socket descriptor of the socket to be given.

CLIENT
A structure containing the identifier of the application to which the socket should be given.

DOMAIN
A fullword binary number that must be set to a decimal 2, indicating AF_INET, or a decimal 19, indicating AF_INET6.

Rule: A socket given by GIVESOCKET can only be taken by a TAKESOCKET with the same DOMAIN, address family (such as, AF_INET or AF_INET6).

NAME
Specifies an 8-character field, left-aligned, padded to the right with blanks, that can be set to the name of the MVS address space that contains 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 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 eight-character 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.

- If used by CICS IP sockets, the field should be set to blanks.
- If TASK identifier is nonblank, 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 B. Return codes on page 397 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

INITAPI and INITAPIX

The INITAPI and INITAPIX calls connect an application to the TCP/IP interface. The sole difference between INITAPI and INITAPIX is explained in the description of the IDENT parameter. INITAPI is preferred over INITAPIX unless there is a specific need to connect applications to alternate TCP/IP stacks. CICS sockets programs that are written in COBOL, PL/I, or assembler language should issue the INITAPI or INITAPIX macro before they issue other calls to the CICS sockets interface.

If a CICS task's first call to the CICS socket interface is not an INITAPI or INITAPIX, then the CICS socket interface generates a default INITAPI call.

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
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 138 shows an example of INITAPI call instructions. The same example can be used for the INITAPIX call by simply changing the SOC-FUNCTION value to 'INITAPIX'.

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

PROCEDURE DIVISION.
     CALL 'EZASOKET' USING SOC-FUNCTION MAXSOC IDENT SUBTASK
     MAXSNO ERRNO RETCODE.
```

Figure 138. INITAPI call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 226.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing INITAPI or INITAPIX. 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 ever has open at one time. The maximum number is 65535 and the minimum number is 50. This value is used to determine the amount of memory that is allocated for socket control blocks and buffers. If less than 50 are requested, MAXSOC defaults to 50.

IDENT

A 16-byte structure containing the name of the TCP/IP address space (TCPNAME) and the name of calling program's address space (ADSNAME).

The way that the CICS socket interface handles the TCPNAME part of the structure differs between INITAPI and INITAPIX (as explained in the following description of TCPNAME).

TCPNAME

An 8-byte character field which should be set to the MVS jobname of the TCP/IP address space with which you are connecting.

If the function is INITAPI, then the CICS socket interface always overrides this with the value in the TCPADDR configuration parameter. In OS/390® V2R8 and earlier, the INITAPIX functions the same way. In z/OS V1R1 and higher, the TCPNAME passed by the application program on an INITAPIX call overrides the TCPADDR value.

ADSNAME

An 8-byte character field set to the identity of the calling program's address space. It is the name of the CICS startup job. The CICS socket interface always overrides this value with VTAM APPLID of the CICS address space.

SUBTASK

Indicates an 8-byte field containing a unique subtask identifier that is used to distinguish between multiple subtasks within a single address space. For your subtask name, use the zoned decimal value of the CICS task ID (EIBTASKN), plus a unique displayable character. In CICS, if no value is specified, the zoned-decimal value of the CICS task ID appended with the letter C is used.

| **Result:** Using the letter L as the last character in the subtask parameter causes the tasking mechanism to assume the CICS transaction is a listener and schedule it using a non-reusable subtask by way of MVS attach processing when OTE=NO. This has no effect when OTE=YES.

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 zero. 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 B. Return codes on page 397 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

IOCTL

The IOCTL call is used to control certain operating characteristics for a socket.

Before you issue an IOCTL call, you must load a value representing 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 21 on page 287 for information about REQARG and RETARG.

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 “Environmental restrictions and programming requirements” on page 223.
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 139 on page 280 shows an example of IOCTL call instructions.

```
WORKING-STORAGE SECTION.
01  SOKET-FUNCTION        PIC X(16) VALUE 'IOCTL'.
01  S                     PIC 9(4)  BINARY.
01  COMMAND               PIC 9(4)  BINARY.

01  IFREQ.
 05 NAME                  PIC X(16).
 05 FAMILY                PIC 9(4)  BINARY.
 05 PORT                  PIC 9(4)  BINARY.
 05 ADDRESS               PIC 9(8)  BINARY.
 05 FILLER                PIC X(8).

01  IFREQOUT.
 05 NAME                  PIC X(16).
 05 FAMILY                PIC 9(4)  BINARY.
 05 PORT                  PIC 9(4)  BINARY.
 05 ADDRESS               PIC 9(8)  BINARY.
 05 FILLER                PIC X(8).

01  GRP-IOCTL-TABLE.
 05 IOCTL-ENTRY 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 ADDRESS              PIC 9(8)  BINARY.
  10 FILLER               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 139. IOCTL call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 226.

Parameter values set by the application

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

S A halfword binary number set to the descriptor of the socket to be controlled.

COMMAND
To control an operating characteristic, set this field to one of the following symbolic names. A value in a bit mask is associated with each symbolic name. By specifying one of these names, you are turning on a bit in a mask that 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.

SIOCGHOMEIF6
Requests all IPv6 home interfaces. When the SIOCGHOMEIF6 IOCTL is issued, the REQARG must contain a Network

Configuration Header. The NETCONFHDR is defined in SYS1.MACLIB(BPXYIOC6) for Assembler programs.

Requirement: The following input fields 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 is set to ERANGE and the NchNumEntryRet is 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 depends on the number of qualifying interfaces returned.

NchNumEntryRet
: If return code is zero, this is set to number of HOME-IF-ADDRESS returned. If errno is ERANGE, then this is set to number of qualifying interfaces. No interfaces are returned. Recalculate the NchBufferLength based on this value times the size of HOME-IF-ADDRESS.

```
Working-Storage Section.

01 SIOCGHOMEIF6 PIC X(4) VALUE X'C014F608'.


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 soket-ioctl socket-descriptor
        SIOCGHOMEIF6
        NETCONFHDR NETCONFHDR
        errno retcode.
```

Figure 140. COBOL language example for SIOCGHOMEIF6

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 21 on page 287.

SIOCATMARK
: Determines whether the current location in the data input is pointing to out-of-band data.

SIOCGIFADDR
: Requests the network interface address for a given interface name. See the NAME field in Figure 141 on page 283 for the address format.

SIOCGIFBRDADDR
: Requests the network interface broadcast address for a given interface name. See the NAME field in Figure 141 on page 283 for the address format.

SIOCGIFCONF
: Requests the network interface configuration. The configuration is a variable number of 32-byte structures formatted as shown in Figure 141.

- 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.

  Interface request structure (IFREQ) for the IOCTL call

```
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).
```

Figure 141. Interface request structure (IFREQ) for the IOCTL call

SIOCGIFDSTADDR
Requests the network interface destination address for a given interface name. (See IFREQ NAME field, Figure 141 for format.)

SIOCGIFNAMEINDEX
Requests all interface names and 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 the IF_NAMEINDEX structure [defined in SYS1.MACLIB(BPX1IOCC) for assembler programs].

- 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 following steps describe how to compute this length is as follows:
  1. Determine the number of interfaces expected to be returned upon successful completion of this command.
  2. Multiply the number of interfaces by the array element (size of IF-NIINDEX, IF-NINAME, and IF-NIEXT) to get the size of the array element.
  3. To the size of the array, add the size of IF-NITOTALIF and IF-NIENTRIES 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 IOCTL returns.
- The 'SIOCGIFNAMEINDEX' command returns a variable number of all the qualifying network interfaces.

```
WORKING-STORAGE SECTION.
     01 SIOCGIFNAMEINDEX PIC X(4) VALUE X'4000F603'.
     01  reqarg                        pic 9(8) binary.
     01  reqarg-header-only            pic 9(8) binary.

     01  IF-NIHEADER.
         05  IF-NITOTALIF              PIC 9(8) BINARY.
         05  IF-NIENTRIES              PIC 9(8) BINARY.

     01  IF-NAME-INDEX-ENTRY.
         05  IF-NIINDEX                PIC 9(8) BINARY.
         05  IF-NINAME                 PIC X(16).
         05  IF-NINAMETERM             PIC X(1).
         05  IF-NIRESV1                PIC X(3).

     01  OUTPUT-STORAGE                PIC X(500).

     Procedure Division.

     move 8 to reqarg-header-only.
     Call 'EZASOKET' using soket-ioctl socket-descriptor
         SIOCGIFNAMEINDEX
         REQARG-HEADER-ONLY IF-NIHEADER
         errno retcode.

     move 500 to reqarg.
     Call 'EZASOKET' using soket-ioctl socket-descriptor
         SIOCGIFNAMEINDEX
         REQARG OUTPUT-STORAGE
         errno retcode.
```

Figure 142. COBOL language example for SIOCGIFNAMEINDEX

SIOCGIPMSFILTER

Requests a list of the IPv4 source addresses that comprise the source filter along 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 addresses, 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; this structure is defined in SYS1.MACLIB(BPXYIOCC) for assembler, in SEZAINST(CBLOCK) for PL/I, and in SEZAINST(EZACOBOL) for COBOL. The IP_MSFILTER structure 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 contains the number of source addresses that 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). When the process completes, if the number of source addresses contains a larger value, the IOCTL can be repeated with a larger buffer. That is, on output, the number of source addresses is always updated to be the total number of sources in the filter; the array holds as many source addresses as fit, up to the minimum of the array size passed in as the input number.

The size of the IP_MSFILTER value is calculated as follows:

1. Determine the number of source addresses that is expected.

2. Multiply the number of source addresses by the array element (size of IMSF_SrcEntry) to get the size of all array elements.
3. Add the size of all array elements with the size of the IMSF_Header structure to get the total number of bytes needed to accommodate the source address information that is returned.

SIOCGMSFILTER

Requests 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 (MCAST_INCLUDE or MCAST_EXCLUDE).

When the SIOCGMSFILTER IOCTL is issued, the REQARG parameter must contain a GROUP_FILTER structure; this structure is defined in SYS1.MACLIB(BPXYIOCC) for assembler, in SEZAINST(CBLOCK) for PL/I, and in SEZAINST(EZACOBOL) for COBOL. The GROUP_FILTER structure 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 contains the number of source addresses that 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 before processing, it can make a reasonable guess (for example, 0). When the process completes, if the number of source addresses holds a larger value, the IOCTL can be repeated with a larger buffer. That is, on output, the number of source addresses is always updated to be the total number of sources in the filter, and the array holds as many source addresses as fit, up to the minimum of the array size that is passed in as the input number.

The application calculates 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 GF_SrcEntry) to get the size of all array elements.
3. Add the size of all array elements to the size of the GF_Header structure to get the total number of bytes needed to accommodate the source addresses information returned.

SIOCSAPPLDATA

Enables an application to associate 40 bytes of user-specified application data with a socket endpoint. This application data can be used to identify TCP connections in interfaces such as Netstat, SMF, or network management applications.

Requirement: When you issue the SIOCSAPPLDATA IOCTL, ensure that the REQARG parameter contains a SetApplData structure as defined by the EZBYAPPL macro in the SEZANMAC dataset. See the CBLOCK and the EZACOBOL samples for the equivalent SetApplData and SetADcontainer structure definitions for PL/I and COBOL programming environments. See *z/OS Communications Server: IP Programmer's Guide and Reference* for more information about programming the SIOCSAPPLDATA IOCTL.

SetAD_buffer

User-defined application data that comprises 40 bytes of data that is used to identify the TCP connection with the IP CICS socket API sockets application. The application data can be displayed in the following ways:

- By requesting Netstat reports. The information is displayed conditionally by using the modifier APPLDATA on the ALLC/-a and COnn /-c reports, and unconditionally on the ALL/-A report. See the Netstat ALL/-A report, the Netstat ALLConn/-a report, and the Netstat COnn/-c report information in *z/OS Communications Server: IP System Administrator's Commands* for more information about Netstat reports.
- In the SMF 119 TCP connection termination record. See *z/OS Communications Server: IP Configuration Reference* for more information about the application data written on the SMF 119 record.
- By network management applications. See the information in the *z/OS Communications Server: IP Programmer's Guide and Reference* for more information about application data.

Applications using this ioctl need to consider the following guidelines:

- The application is responsible for documenting the content, format, and meaning of the ApplData strings it associates with 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 EZA or EZB, are reserved for IBM use. IBM product identifiers begin with a letter in the range A - I.
- Printable EBCDIC characters should be use 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 along with the current mode on a given interface and a multicast group for a socket. The source filter can either include or exclude the set of source address, depending on the filter mode (MCAST_INCLUDE or MCAST_EXCLUDE). A maximum of 64 source addresses can be specified. When the SIOCSIPMSFILTER IOCTL is issued, the REQARG parameter must contain a IP_MSFILTER structure; this structure is defined in SYS1.MACLIB(BPXYIOCC) for assembler, in SEZAINST(CBLOCK) for PL/I and in SEZAINST(EZACOBOL) for COBOL. The IP_MSFILTER structure 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.

The application program calculates the size of the IP_MSFILTER 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 IMSF_SrcEntry structure) to get the size of all array elements.
3. Add the size of all array elements to the size of IMSF_Header to get the total number of bytes needed to accommodate the source addresses information 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). A maximum of 64 source addresses can be specified. When the SIOCSMSFILTER IOCTL is issued, the REQARG parameter must contain a GROUP_FILTER structure; this structure is defined in SYS1.MACLIB(BPXYIOCC) for assembler, in SEZAINST(CBLOCK) for PL/I, and in SEZAINST(EZACOBOL) for COBOL. The GROUP_FILTER must include an interface index, a socket address structure of the multicast address, filter mode, the number of source addresses in the following array, an array of the socket address structure of source addresses.

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 GF_SrcEntry) to get the size of all array elements.
3. Add the size of all array elements to the size of GF_Header 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 the control block structures in SEZANMAC. EZBZTLS1 defines the PL/I layout, EZBZTLSP defines the assembler layout, and EZBZTLSB defines the COBOL layout. For more usage information and samples, see *z/OS Communications Server: IP Programmer's Guide and Reference*.

REQARG and RETARG
REQARG is used to pass arguments to IOCTL and RETARG receives arguments from IOCTL. The REQARG and RETARG parameters are described in Table 21.

Table 21. IOCTL call arguments

COMMAND/CODE	SIZE	REQARG	SIZE	RETARG
FIONBIO X'8004A77E'	4	Set socket mode to one of the following: X'00'=blocking; X'01'=nonblocking	0	Not used
FIONREAD X'4004A77F'	0	Not used	4	Number of characters available for read
SIOCATMARK X'4004A707'	0	Not used	4	X'00' = at OOB dataX'01' = not at OOB data
SIOCGHOMEIF6 X'C014F608'	20	NetConfHdr		See Figure 140 on page 282.
SIOCGIFADDR X'C020A70D'	32	First 16 bytes is the interface name. Last 16 bytes—not used	32	Network interface address (see Figure 141 on page 283 for format.)
SIOCGIFBRDADDR X'C020A712'	32	First 16 bytes is the interface name. Last 16 bytes—not used	32	Network interface address (see Figure 141 on page 283 for format.)

Table 21. IOCTL call arguments (continued)

COMMAND/CODE	SIZE	REQARG	SIZE	RETARG
SIOCGIFCONF X'C008A714'	8	Size of RETARG		When you call the IOCTL with the SIOCGIFCONF command set, the REQARG parameter should contain the length in bytes of RETARG. Each interface is assigned a 32-byte array element; the REQARG parameter should be set to the number of interfaces multiplied by 32. TCP/IP for z/OS can return up to 100 array elements.
SIOCGIFDSTADDR X'C020A70F'	32	First 16 bytes is the interface name. Last 16 bytes are not used.	32	Destination interface address (See Figure 141 on page 283 for format.)
SIOCGIFNAMEINDEX X'4000F603'	4	First 4 bytes of return the buffer		See Figure 142 on page 284.
SIOCGIPMSFILTER X'C000A724'	–	See the IP_MSFILTER structure in macro BPXYIOCC. See note 1.	0	Not used.
SIOCGMSFILTER X'C000F610'	–	See the GROUP_FILTER structure in macro BPXYIOCC. See note 2.	0	Not used.
SIOCSAPPLDATA X'8018D90C'	–	See the SETAPPLDATA structure in macro EZBYAPPL	0	Not used.
SIOCSIPMSFILTER X'8000A725'	–	See the IP_MSFILTER structure in macro BPXYIOCC. See note 1.	0	Not used.
SIOCSMSFILTER X'8000F611'	–	See the GROUP_FILTER structure in macro BPXYIOCC. See note 2.		
SIOCTTLSCTLX'C038D90B'	56	For the IOCTL structure layout, see SEZANMAC(EZBZTLS1) for PL/I, SEZANMAC(EZBZTLSP) for assembler, and SEZANMAC(EZBZTLSB) for COBOL	56	For the IOCTL structure layout, see SEZANMAC(EZBZTLS1) for PL/I, SEZANMAC(EZBZTLSP) for assembler, and SEZANMAC(EZBZTLSB) for COBOL.

Notes:

1. The size of IP_MSFILTER structure must be equal to or greater than the size of the IMSF_Header stucture.
2. The size of GROUP_FILTER structure must be equal to or greater than the size of the GF_Header structure.

Parameter values returned to the application

RETARG
Returns an array whose size is based on the value in COMMAND. See Table 21 for information about REQARG and RETARG.

ERRNO
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B. Return codes on page 397 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

The COMMAND SIOGIFCONF returns a variable number of network interface configurations. Figure 143 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 143 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 REQARQ.
  CALL 'EZASOKET' USING SOC-FUNCTION S COMMAND
      REQARG RETARG ERRNO RETCODE.
```

Figure 143. 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. After a socket becomes passive, it cannot initiate connection requests.

Note: The BACKLOG value specified on the LISTEN command cannot be greater than 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. If you want a larger backlog, update the SOMAXCONN statement. See the *z/OS Communications Server: IP Configuration Reference* for details.

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
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 144 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 144. LISTEN call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 226.

Parameter values set by the application

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

S A halfword binary number set to the socket descriptor.

BACKLOG
A fullword binary number set to the number of communication requests to be queued.

Parameter values returned to the application

ERRNO
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B. Return codes on page 397 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

NTOP

NTOP 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:

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
Amode:	31-bit or 24-bit

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 145 shows an example of NTOP call instructions.

```
WORKING-STORAGE SECTION.
    01  SOC-NTOP-FUNCTION       PIC X(16)  VALUE IS 'NTOP'.
    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  NTOP-FAMILY PIC 9(8) BINARY.
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.

    01  PRESENTABLE-ADDRESS      PIC X(45).
    01  PRESENTABLE-ADDRESS-LEN  PIC 9(4) BINARY.

PROCEDURE DIVISION.
   CALL 'EZASOKET' USING SOC-NTOP-FUNCTION NTOP-FAMILY
                         IP-ADDRESS
                         PRESENTABLE-ADDRESS
                         PRESENTABLE-ADDRESS-LEN
                         ERRNO RETURN-CODE.
```

Figure 145. NTOP call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 226.

Parameter values set by the application

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

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

PRESENTABLE-ADDRESS
A field used to receive the standard text presentation form of the IPv4 or IPv6 address being converted. For IPv4, the address is in dotted-decimal format and for IPv6 the address is in colon-hexadecimal format. The size of the IPv4 address is a maximum of 15 bytes and the size of the converted IPv6 address is 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 on a successful return) contains the length of converted IP address.

ERRNO
A fullword binary field. If RETCODE is negative, ERRNO contains an error number. See Appendix B. Return codes on page 397 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

PTON

PTON 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:

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
Amode:	31-bit or 24-bit
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 146 on page 293 shows an example of PTON call instructions.

```
WORKING-STORAGE SECTION.
    01  SOC-NTOP-FUNCTION        PIC X(16)  VALUE IS 'PTON'.
    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  AF-INET         PIC 9(8) BINARY VALUE 2.
    01  AF-INET6        PIC 9(8) BINARY VALUE 19.

* IPv4 address.
    01  PRESENTABLE-ADDRESS      PIC X(45).
    01  PRESENTABLE-ADDRESS-IPV4 REDEFINES PRESENTABLE-ADDRESS.
        05  PRESENTABLE-IPV4-ADDRESS PIC X(15)
                                           VALUE '192.26.5.19'.
        05  FILLER      PIC X(30).
    01  PRESENTABLE-ADDRESS-LEN  PIC 9(4) BINARY VALUE 11.

* IPv6 address.
    01  PRESENTABLE-ADDRESS      PIC X(45)
                          VALUE '12f9:0:0:c30:123:457:9cb:1112'.
    01  PRESENTABLE-ADDRESS-LEN  PIC 9(4) BINARY VALUE 29.

* IPv4-mapped IPv6 address.
    01  PRESENTABLE-ADDRESS      PIC X(45)
                          VALUE '12f9:0:0:c30:123:457:192.26.5.19'.
    01  PRESENTABLE-ADDRESS-LEN  PIC 9(4) BINARY VALUE 32.

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

    01  PRESENTABLE-ADDRESS      PIC X(45).
    01  PRESENTABLE-ADDRESS-LEN  PIC 9(4) BINARY.

PROCEDURE DIVISION.

* IPv4 address.
  CALL 'EZASOKET' USING SOC-PTON-FUNCTION AF-INET
                        PRESENTABLE-ADDRESS
                        PRESENTABLE-ADDRESS-LEN
                        IP-ADDRESS
                        ERRNO RETURN-CODE.
* IPv6 address.
  CALL 'EZASOKET' USING SOC-PTON-FUNCTION AF-INET6
                        PRESENTABLE-ADDRESS
                        PRESENTABLE-ADDRESS-LEN
                        IP-ADDRESS
                        ERRNO RETURN-CODE.
```

Figure 146. PTON call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 226.

Parameter values set by the application

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

FAMILY
The addressing family for the IP address being converted. The value of decimal 2 must be specified for AF_INET and 19 for AF_INET6.

PRESENTABLE-ADDRESS
A field containing the standard text presentation form of the IPv4 or IPv6 address being converted. For IPv4, the address is in dotted-decimal format and for IPv6 the address is in colon-hexadecimal format.

PRESENTABLE-ADDRESS-LEN
An input parameter. The address of a binary halfword field that must contain the length of IP address to be converted.

Parameter values returned to the application

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 in network byte order.

ERRNO
A fullword binary field. If RETCODE is negative, ERRNO contains an error number. See Appendix B. Return codes on page 397 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 sockets. 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 is 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 352 for a subroutine that translates ASCII input data to EBCDIC.

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 "Environmental restrictions and programming requirements" on page 223.
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 147 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 147. READ call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 226.

Parameter values set by the application

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

S A halfword binary number set to the socket descriptor of the socket that is going to read the data.

NBYTE
A fullword binary number set to the size of BUF. READ does not return more than the number of bytes of data in NBYTE even if more data is available.

Parameter values returned to the application

BUF On input, a buffer to be filled by completion of the call. The length of BUF must be at least as long as the value of NBYTE.

ERRNO
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B. Return codes on page 397 for information about ERRNO return codes.

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

Value	Description
0	A 0 return code indicates that the connection is closed and no data is available.

>0 A positive value indicates the number of bytes copied into the buffer.

–1 Check ERRNO for an error code.

READV

The READV function reads data on a socket and stores it in a set of buffers. If a datagram packet is too long to fit in the supplied buffers, datagram sockets discard extra bytes.

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
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 148 shows an example of READV call instructions.

```
WORKING-STORAGE SECTION.
   01  SOKET-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 'EZASOKET' USING SOC-FUNCTION S IOV IOVCNT ERRNO RETCODE.
```

Figure 148. READV call instruction example

Parameter values set by the application

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 B. Return codes on page 397 for information about ERRNO return codes.

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

Value	Description
0	A 0 return code indicates that the connection is closed and no data is available.
>0	A positive value indicates the number of bytes copied into the buffer.
–1	Check ERRNO for an error code.

RECV

The RECV call, like READ, receives data on a socket with descriptor S. RECV applies only to connected sockets. If a datagram packet is too long to fit in the supplied buffers, datagram sockets discard extra bytes.

For additional control of the incoming data, RECV can:
- Peek at the incoming message without having it removed from the buffer.
- Read out-of-band data.

For stream sockets, data is processed as streams of information with no boundaries separating the data. For example, if programs A and B are connected with a stream socket and program A sends 1000 bytes, each call to this function can return any number of bytes up to the entire 1000 bytes. The number of bytes returned are 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 236 or “IOCTL” on page 278 for a description of how to set nonblocking mode.

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

Note: See “EZACIC05” on page 352 for a subroutine that translates ASCII input data to EBCDIC.

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
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 149 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.
    01  NO-FLAG         PIC 9(8) BINARY  VALUE IS 0.
    01  OOB             PIC 9(8) BINARY  VALUE IS 1.
    01  PEEK            PIC 9(8) BINARY  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 149. RECV call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 226.

Parameter values set by the application

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

S A halfword binary number set to the socket descriptor of the socket to receive the data.

FLAGS
A fullword binary field with values as follows:

Literal value	Binary value	Description
NO-FLAG	0	Read data.
OOB	1	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.
PEEK	2	Peek at the data, but do not destroy data. If the peek flag is set, the next RECV call reads the same data.

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 B. Return codes on page 397 for information about ERRNO return codes.

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

Value	**Description**
0	The socket is closed
>0	A positive return code indicates the number of bytes copied into the buffer.
–1	Check ERRNO for an error code

RECVFROM

The RECVFROM call receives data on a socket with descriptor S and stores it in a buffer. The RECVFROM call applies to both connected and unconnected sockets. The IPv4 or IPv6 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, the RECVFROM call returns the source address associated with each incoming datagram. For connection-oriented protocols like TCP, the GETPEERNAME call returns the address associated with the other end of the connection.

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 are 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 236 or "IOCTL" on page 278 for a description of how to set nonblocking mode.

Note: See "EZACIC05" on page 352 for a subroutine that translates ASCII input data to EBCDIC.

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
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 150 on page 301 shows an example of RECVFROM call instructions.

```
WORKING-STORAGE SECTION.
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'RECVFROM'.
    01  S               PIC 9(4) BINARY.
    01  FLAGS           PIC 9(8) BINARY.
    01  NO-FLAG         PIC 9(8) BINARY  VALUE IS 0.
    01  OOB             PIC 9(8) BINARY  VALUE IS 1.
    01  PEEK            PIC 9(8) BINARY  VALUE IS 2.
    01  NBYTE           PIC 9(8) BINARY.
    01  BUF             PIC X(length of buffer).
*
* 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  FLOW-INFO   PIC 9(8) BINARY.
        03  IP-ADDRESS.
            05  FILLER  PIC 9(16) BINARY.
            05  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 FLAGS
                    NBYTE BUF NAME ERRNO RETCODE.
```

Figure 150. RECVFROM call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 226.

Parameter values set by the application

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

S
A halfword binary number set to the socket descriptor of the socket to receive the data.

FLAGS
A fullword binary field containing flag values as follows:

Literal value	Binary value	Description
NO-FLAG	0	Read data.
OOB	1	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.
PEEK	2	Peek at the data, but do not destroy data. If the peek flag is set, the next RECVFROM call reads the same data.

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 structure containing the address of the socket that sent the data. The structure is:

FAMILY
A halfword binary number specifying the addressing family. The value is a 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 structure containing the address of the socket that sent the data. The structure is:

FAMILY
A halfword binary number specifying the addressing family. The value is a decimal 19, indicating AF_INET6.

PORT A halfword binary number specifying the port number of the sending socket.

FLOW-INFO
A fullword binary field specifying the traffic class and flow label. The value of this field is undefined.

IP-ADDRESS
A 16-byte binary number specifying the 128-bit IPv6 Internet address of the sending socket.

SCOPE-ID
A fullword binary field that identifies a set of interfaces as appropriate for the scope of the address carried in the IP-ADDRESS field. For a link scope IP-ADDRESS, SCOPE-ID contains the link index for the IP-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 B. Return codes on page 397 for information about ERRNO return codes.

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

Value	Description
0	The socket is closed.
>0	A positive return code indicates the number of bytes of data transferred by the read call.
–1	Check ERRNO for an error code.

RECVMSG

The RECVMSG call receives messages on a socket with descriptor S and stores them in an array of message headers. If a datagram packet is too long to fit in the supplied buffers, datagram sockets discard extra bytes.

For datagram protocols, the RECVMSG call returns the source address associated with each incoming datagram. For connection-oriented protocols like TCP, the GETPEERNAME call returns the address associated with the other end of the connection.

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
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 151 on page 304 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.
                03  NAME            USAGE IS POINTER.
                03  NAME-LEN        USAGE IS POINTER.
                03  IOV             USAGE IS POINTER.
                03  IOVCNT          USAGE IS POINTER.
                03  ACCRIGHTS       USAGE IS POINTER.
                03  ACCRLEN         USAGE IS POINTER.

            01  FLAGS           PIC 9(8)   BINARY.
            01  NO-FLAG         PIC 9(8)   BINARY VALUE IS 0.
            01  OOB             PIC 9(8)   BINARY VALUE IS 1.
            01  PEEK            PIC 9(8)   BINARY VALUE IS 2.
            01  ERRNO           PIC 9(8)   BINARY.
            01  RETCODE         PIC S9(8)  BINARY.

       LINKAGE SECTION.

            01 L1.
               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 RECVMSG-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 RECVMSG-NAME.
                 05 FAMILY            PIC 9(4) BINARY.
                 05 PORT              PIC 9(4) BINARY.
                 05 FLOW-INFO         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 151. RECVMSG call instruction example (Part 1 of 2)

```
PROCEDURE DIVISION USING L1.

                  SET NAME TO ADDRESS OF RECVMSG-NAME.
                  MOVE LENGTH OF RECVMSG-NAME TO NAME-LEN.
                  SET IOV TO ADDRESS OF RECVMSG-IOVECTOR.
                  MOVE 3 TO RECVMSG-BUFNO.
                  SET IOVCNT TO ADDRESS OF RECVMSG-BUFNO.
                  SET IOV1A TO ADDRESS OF RECVMSG-BUFFER1.
                  MOVE 0 TO MSG-IOV1AL.
                  MOVE LENGTH OF RECVMSG-BUFFER1 TO IOV1L.
                  SET IOV2A TO ADDRESS OF RECVMSG-BUFFER2.
                  MOVE 0 TO IOV2AL.
                  MOVE LENGTH OF RECVMSG-BUFFER2 TO IOV2L.
                  SET IOV3A TO ADDRESS OF RECVMSG-BUFFER3.
                  MOVE 0 TO IOV3AL.
                  MOVE LENGTH OF RECVMSG-BUFFER3 TO IOV3L.
                  SET ACCRIGHTS TO NULLS.
                  SET ACCRLEN TO NULLS.
                  MOVE 0 TO FLAGS.
                  MOVE SPACES TO RECVMSG-BUFFER1.
                  MOVE SPACES TO RECVMSG-BUFFER2.
                  MOVE SPACES TO RECVMSG-BUFFER3.

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

Figure 151. RECVMSG call instruction example (Part 2 of 2)

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 226.

Parameter values set by the application

S
A value or the address of a halfword binary number specifying the socket descriptor.

MSG
On input, a pointer to a message header into which the message is received upon completion of the call.

Field **Description**

NAME
On input, a pointer to a buffer where the sender address is stored upon completion of the call. The storage being pointed to should be for an IPv4 socket address or an IPv6 socket address.

The IPv4 socket address structure contains the following fields:

Field	Description
FAMILY	Output parameter. A halfword binary number specifying the IPv4 addressing family. The value for IPv4 socket descriptor (for example, S parameter) is a 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 eight-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 field specifying the IPv6 addressing family. The value for IPv6 socket descriptor (for example, S parameter) is a decimal 19, indicating AF_INET6.
PORT	Output parameter. A halfword binary number specifying the port number of the sending socket.
FLOW-INFO	Output parameter. A fullword binary field specifying the traffic class and flow label. The value of this field is undefined.
IP-ADDRESS	Output parameter. A two doubleword, 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 IP-ADDRESS field. For a link scope IP-ADDRESS, SCOPE-ID contains the link index for the IP-ADDRESS. For all other address scopes, SCOPE-ID is undefined.

NAME-LEN
On input, a pointer to the size of the NAME buffer that is filled in on completion of the call.

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. The data buffer must be in the home address space.

Fullword 2
Reserved. This storage is 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 portion, as shown in the example.

IOVCNT
On input, a pointer to a fullword binary field specifying the number of data buffers provided for this call.

ACCRIGHTS
On input, a pointer to the access rights received. This field is ignored.

ACCRLEN
On input, a pointer to the length of the access rights received. This field is ignored.

FLAGS
A fullword binary field with values as follows:

Literal value	Binary value	Description
NO-FLAG	0	Read data.
OOB	1	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.
PEEK	2	Peek at the data, but do not destroy data. If the peek flag is set, the next RECVMSG call reads the same data.

Parameter values returned by the application

ERRNO
: A fullword binary field. If RETCODE is negative, this contains an error number. See Appendix B. Return codes on page 397 for information about ERRNO return codes.

RETCODE
: A fullword binary field with the following values:

Value	Description
<0	Call returned error. See ERRNO field.
0	Connection partner has closed connection.
>0	Number of bytes read.

SELECT

In a process where multiple I/O operations can occur, it is necessary for the program to be able to wait on one or several of the operations to complete.

For example, consider a program that issues a READ to multiple sockets whose blocking mode is set. Because the socket would block on a READ call, only one socket could be read at a time. Setting the sockets nonblocking would solve this problem, but would require polling each socket repeatedly until data became available. The SELECT call allows you to test several sockets and to execute a subsequent I/O call only when one of the tested sockets is ready, thereby ensuring that the I/O call does not block.

To use the SELECT call as a timer in your program, do one of the following:
- Set the read, write, and except arrays to zeros.
- Specify MAXSOC <= 0.

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
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 does not block.
  - A connection has been requested on that socket.
- When a socket is ready to write, TCP/IP stacks can accommodate additional output data. If TCP/IP stacks can accept additional output for a given socket, a write operation on that socket does not block.
- When an exception condition has occurred on a specified socket it is an indication that a TAKESOCKET has occurred for that socket.

Each socket descriptor is represented by a bit in a bit string. The bit strings are contained in 32-bit fullwords, numbered from right to left. The rightmost bit of the first fullword represents socket descriptor 0 and the leftmost bit of the first fullword represents socket descriptor 31. If your process uses 32 or fewer sockets, the bit string is one fullword. If your process uses 33 sockets, the bit string is two fullwords. The rightmost bit of the second fullword represents socket descriptor 32, and the leftmost bit of the second fullword represents socket descriptor 63. This pattern repeats itself for each subsequent fullword. That is, the leftmost bit of fullword n represents socket 32n-1 and the rightmost bit represents socket 32(n-1).

You define the sockets that you want to test by turning on bits in the string. Although the bits in the fullwords are numbered from right to left, the fullwords are numbered from left to right with the leftmost fullword representing socket descriptor 0–31. For example:

```
First fullword            Second fullword           Third fullword
socket descriptor 31...0   socket descriptor 63...32   socket descriptor 95...64
```

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 354.

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 ready for reading.

Write operations

A socket is selected for writing (ready to be written) when:

- TCP/IP stacks 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 is selected for write when the CONNECT completes.

A call to SEND, SENDTO, WRITE, or WRITEV blocks when the amount of data to be sent exceeds the amount of data TCP/IP stacks can accept. To avoid this, you can precede the write operation with a SELECT call to ensure that the socket is ready for writing. After a socket is selected for WRITE, the program can determine the amount of TCP/IP stacks 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 one before issuing the SELECT call. When the SELECT call returns, the corresponding bits in the WRETMSK indicate sockets 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 returns 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 one. 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 the call returns any 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 in the range 0 up to the MAXSOC value minus 1. For example, if the MAXSOC parameter is set to 50, the range is 0-49.

TIMEOUT parameter

If the time specified in the TIMEOUT parameter elapses before any event is detected, the SELECT call returns and RETCODE is set to 0.

Figure 152 on page 310 shows an example of SELECT call instructions.

```
WORKING-STORAGE SECTION.
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'SELECT'.
    01  MAXSOC          PIC 9(8) BINARY.
    01  TIMEOUT.
        03  TIMEOUT-SECONDS   PIC 9(8) BINARY.
        03  TIMEOUT-MICROSEC  PIC 9(8) BINARY.
    01  RSNDMSK        PIC X(*).
    01  WSNDMSK        PIC X(*).
    01  ESNDMSK        PIC X(*).
    01  RRETMSK        PIC X(*).
    01  WRETMSK        PIC X(*).
    01  ERETMSK        PIC X(*).
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.

PROCEDURE DIVISION.
     CALL 'EZASOKET' USING SOC-FUNCTION MAXSOC TIMEOUT
                    RSNDMSK WSNDMSK ESNDMSK
                    RRETMSK WRETMSK ERETMSK
                    ERRNO RETCODE.
```

* The bit mask lengths can be determined from the expression:

```
((maximum socket number +32)/32 (drop the remainder))*4
```

Figure 152. SELECT call instruction example

Bit masks are 32-bit fullwords with one bit for each socket. Up to 32 sockets fit into one 32-bit mask [PIC X(4)]. If you have 33 sockets, you must allocate two 32-bit masks [PIC X(8)].

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 226.

Parameter values set by the application

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

MAXSOC
A fullword binary field specifying the largest socket descriptor number that is being checked.

Guideline: For the INITAPI call, the MAXSOC field is a halfword binary field. Therefore, do not reuse this field for the SELECT and INITAPI calls.

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 or an ECB in a list is posted. 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 timeout after 3.5 seconds, set TIMEOUT-SECONDS to 3 and TIMEOUT-MICROSEC to 500000.

RSNDMSK
A bit string sent to request read event status.

- For each socket to be checked for pending read events, the corresponding bit in the string should be set to 1.
- For sockets to be ignored, the value of the corresponding bit should be set to 0.

If this parameter is set to all zeros, the SELECT does 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 set.
- 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 does 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 set.
- For each socket to be ignored, the corresponding bit should be set to 0.

If this parameter is set to all zeros, the SELECT does 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 is set to 1; bits that represent sockets that are not ready to read are 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 is set to 1; bits that represent sockets that are not ready to be written are 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 is set to 1; bits that represent sockets that do not have exception status are set to 0.

ERRNO
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B. Return codes on page 397 for information about ERRNO return codes.

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

Value **Description**

>0 Indicates the sum of all ready sockets in the three masks

0 Indicates that the SELECT time limit has expired

–1 Check ERRNO for an error code

SELECTEX

The SELECTEX call monitors a set of sockets, a time value and an ECB or list of ECBs. 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 except arrays to zeros
- Specify MAXSOC <= 0

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
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 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 does not block.
  - A connection has been requested on that socket.
- When a socket is ready to write, TCP/IP stacks can accommodate additional output data. If TCP/IP stacks can accept additional output for a given socket, a write operation on that socket does not block.
- When an exception condition has occurred on a specified socket it is an indication that a TAKESOCKET has occurred for that socket.

Each socket descriptor is represented by a bit in a bit string. The bit strings are contained in 32-bit fullwords, numbered from right to left. The rightmost bit of the first fullword represents socket descriptor 0 and the leftmost bit of the first fullword represents socket descriptor 31. If your process uses 32 or fewer sockets, the bit string is one fullword. If your process uses 33 sockets, the bit string is two fullwords. The rightmost bit of the second fullword represents socket descriptor 32, and the leftmost bit of the second fullword represents socket descriptor 63. This pattern repeats itself for each subsequent fullword. That is, the leftmost bit of fullword n represents socket 32n-1 and the rightmost bit represents socket 32(n-1).

You define the sockets that you want to test by turning on bits in the string. Although the bits in the fullwords are numbered from right to left, the fullwords are numbered from left to right with the leftmost fullword representing socket descriptor 0-31. For example:

```
First fullword           Second fullword          Third fullword
socket descriptor 31...0     socket descriptor 63...32    socket descriptor 95...64
```

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 the EZACIC06 topic.

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 ready for reading.

Write operations

A socket is selected for writing (ready to be written) when:

- TCP/IP stacks 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 is selected for write when the CONNECT completes.

A call to SEND, SENDTO, WRITE, or WRITEV blocks when the amount of data to be sent exceeds the amount of data TCP/IP stacks can accept. To avoid this, you can precede the write operation with a SELECTEX call to ensure that the socket is ready for writing. After a socket is selected for WRITE, the program can determine the amount of TCP/IP stacks 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 one before issuing the SELECTEX call. When the SELECTEX call returns, the corresponding bits in the WRETMSK indicate sockets 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 returns 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 one. 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 the returns any 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 in the range 0 up to the MAXSOC value minus 1. For example, if MAXSOC is set to 50, the range is 0-49.

TIMEOUT parameter

If the time specified in the TIMEOUT parameter elapses before any event is detected, the SELECTEX call returns and RETCODE is set to 0.

Figure 153 on page 315 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 follows:

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

Where * is the size of the select mask

 PROCEDURE DIVISION.
    CALL 'EZASOKET' USING SOC-FUNCTION MAXSOC TIMEOUT
                    RSNDMSK WSNDMSK ESNDMSK
                    RRETMSK WRETMSK ERETMSK
                    SELECB ERRNO RETCODE.
```

Where * is the size of the select mask.

```
PROCEDURE DIVISION.
    CALL 'EZASOKET' USING SOC-FUNCTION MAXSOC TIMEOUT
                    RSNDMSK WSNDMSK ESNDMSK
                    RRETMSK WRETMSK ERETMSK
                    SELECB ERRNO RETCODE.
```

However, if the application intends to pass the address of an ECB list on the SELECTEX call, then the application must set the high-order bit in the ECB list address and pass that address using the BY VALUE option as in the following example. The remaining parameters must be reset to the default value by specifying BY REFERENCE before the ERRNO value:

```
WORKING-STORAGE SECTION.
     01  SOC-FUNCTION    PIC X(16)  VALUE IS 'SELECTEX'.
     01  MAXSOC          PIC 9(8)   BINARY.
     01  TIMEOUT.
         03  TIMEOUT-SECONDS    PIC 9(8) BINARY.
         03  TIMEOUT-MINUTES    PIC 9(8) BINARY.
     01  RSNDMSK         PIC X(*).
     01  WSNDMSK         PIC X(*).
     01  ESNDMSK         PIC X(*).
     01  RRETMSK         PIC X(*).
     01  WRETMSK         PIC X(*).
     01  ERETMSK         PIC X(*).
     01  ECBLIST-PTR     USAGE IS POINTER.
     01  ERRNO           PIC 9(8)   BINARY.
     01  RETCODE         PIC S9(8)  BINARY.
```

An asterisk (*) represents the size of the select mask.

```
PROCEDURE DIVISION.
    CALL 'EZASOKET' USING SOC-FUNCTION MAXSOC TIMEOUT
                    RSNDMSK WSNDMSK ESNDMSK
                    RRETMSK WRETMSK ERETMSK
                    BY VALUE ECBLIST-PTR
                    BY REFERENCE ERRNO RETCODE.
```

Figure 153. SELECTEX call instruction example

Parameter values set by the application

MAXSOC
: Input parameter. A fullword binary field specifying the largest socket descriptor number that is being checked.

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 SELECT call blocks until a socket becomes ready. 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 timeout after 3.5 seconds, set TIMEOUT-SECONDS to 3 and TIMEOUT-MICROSEC to 500000.

RSNDMSK
: The bit-mask array to control checking for read interrupts. If this parameter is not specified or the specified bit-mask is zeros, the SELECT does 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 does 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 does 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.

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(8) BINARY.
   01 TIMEOUT.
   03 TIMEOUT-SECONDS PIC 9(8) BINARY.
   03 TIMEOUT-MINUTES PIC 9(8) BINARY.
   01 RSNDMSK PIC X(*).
   01 WSNDMSK PIC X(*).
   01 ESNDMSK PIC X(*).
   01 RRETMSK PIC X(*).
   01 WRETMSK PIC X(*).
   01 ERETMSK PIC X(*).
   01 ECBLIST-PTR USAGE IS POINTER.
   01 ERRNO PIC 9(8) BINARY.
   01 RETCODE PIC S9(8) BINARY.
```

```
Where * is the size of the select mask

PROCEDURE DIVISION.
   CALL 'EZASOKET' USING SOC-FUNCTION MAXSOC TIMEOUT
   RSNDMSK WSNDMSK ESNDMSK
   RRETMSK WRETMSK ERETMSK
   BY VALUE ECBLIST-PTR
   BY REFERENCE ERRNO RETCODE.
```

Notes:

1. The maximum number of ECBs that can be specified in a list is 63
2. Perform an MVS POST (not a CICS POST) to post the ECB.

Parameter values returned by the application

ERRNO
A fullword binary field; if RETCODE is negative, this contains an error number. See Appendix B. Return codes on page 397 for information about ERRNO return codes.

RETCODE
A fullword binary field

Value	Meaning
>0	The number of ready sockets.
0	Either the SELECTEX time limit has expired (ECB value is 0) or one of the caller's ECBs has been posted (ECB value is nonzero and the caller's descriptor sets are set to 0). The caller must initialize the ECB values to 0 before issuing the SELECTEX call.
-1	Error; check ERRNO.

RRETMSK
The bit-mask array returned by the SELECT if RSNDMSK is specified. The length of this bit-mask array is dependent on the value in MAXSOC.

WRETMSK
The bit-mask array returned by the SELECT if WSNDMSK is specified. The length of this bit-mask array is dependent on the value in MAXSOC.

ERETMSK
The bit-mask array returned by the SELECT if ESNDMSK is specified. The length of this bit-mask array is dependent on the value in MAXSOC.

Note: See EZACIC06 for information about bits mask conversion.

Note: See Appendix E, "Sample programs," on page 463 for sample programs.

SEND

The SEND call sends data on a specified connected socket.

The FLAGS field allows you to:

- Send out-of-band data, for example, interrupts, aborts, and data marked urgent. Only stream sockets created in the AF_INET or AF_INET6 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 350 for a subroutine that translates EBCDIC input data to ASCII.

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
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 154 shows an example of SEND call instructions.

```
WORKING-STORAGE SECTION.
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'SEND'.
    01  S               PIC 9(4) BINARY.
    01  FLAGS           PIC 9(8) BINARY.
    01  NO-FLAG         PIC 9(8) BINARY  VALUE IS 0.
    01  OOB             PIC 9(8) BINARY  VALUE IS 1.
    01  DONT-ROUTE      PIC 9(8) BINARY  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 154. SEND call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 226.

Parameter values set by the application

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

S
A halfword binary number specifying the socket descriptor of the socket that is sending data.

FLAGS
A fullword binary field with values as follows:

Literal value	Binary value	Description
NO-FLAG	0	No flag is set. The command behaves like a WRITE call.
OOB	1	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.
DONT-ROUTE	4	Do not route. Routing is provided by the calling program.

NBYTE
A fullword binary number set to the number of bytes of data to be transferred.

BUF The buffer containing the data to be transmitted. BUF should be the size specified in NBYTE.

Parameter values returned to the application

ERRNO
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B. Return codes on page 397 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

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:

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
Amode:	31-bit or 24-bit
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 155 on page 320 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.
         03  NAME            USAGE IS POINTER.
         03  NAME-LEN        USAGE IS POINTER.
         03  IOV             USAGE IS POINTER.
         03  IOVCNT          USAGE IS POINTER.
         03  ACCRIGHTS       USAGE IS POINTER.
         03  ACCRLEN         USAGE IS POINTER.

     01  FLAGS           PIC 9(8)  BINARY.
     01  NO-FLAG         PIC 9(8)  BINARY  VALUE IS 0.
     01  OOB             PIC 9(8)  BINARY  VALUE IS 1.
     01  DONTROUTE       PIC 9(8)  BINARY  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.
         03  FILLER      PIC 9(16) BINARY.
         03  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.

*
* IPv4 Socket Address Structure.
*
        03 SENDMSG-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 SENDMSG-NAME.
           05 FAMILY        PIC 9(4) BINARY.
           05 PORT          PIC 9(4) BINARY.
           05 FLOW-INFO     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.

        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.
```

Figure 155. SENDMSG call instruction example (Part 1 of 2)

```
PROCEDURE DIVISION USING L1.

      * For IPv6
                 MOVE 19 TO FAMILY.
                 MOVE 1234 TO PORT.
                 MOVE 0 TO FLOW-INFO.
                 MOVE SENDMSG-IPV6ADDR TO IP-ADDRESS.
                 MOVE 0 TO SCOPE-ID.
      * For IPv4
                 MOVE 2 TO FAMILY.
                 MOVE 1234 TO PORT.
                 MOVE SENDMSG-IPV4ADDR TO IP-ADDRESS.

                 SET NAME TO ADDRESS OF SENDMSG-NAME.
                 SET IOV TO ADDRESS OF SENDMSG-IOVECTOR.
                 MOVE LENGTH OF SENDMSG-NAME TO NAME-LEN.
                 SET IOVCNT TO ADDRESS OF SENDMSG-BUFNO.
                 SET IOV1A TO ADDRESS OF SENDMSG-BUFFER1.
                 MOVE 0 TO IOV1AL.
                 MOVE LENGTH OF SENDMSG-BUFFER1 TO IOV1L.
                 SET IOV2A TO ADDRESS OF SENDMSG-BUFFER2.
                 MOVE 0 TO IOV2AL.
                 MOVE LENGTH OF SENDMSG-BUFFER2 TO IOV2L.
                 SET IOV3A TO ADDRESS OF SENDMSG-BUFFER3.
                 MOVE 0 TO IOV3AL.
                 MOVE LENGTH OF SENDMSG-BUFFER3 TO IOV3L.
                 SET ACCRIGHTS TO NULLS.
                 SET ACCRLEN TO NULLS.
                 MOVE 0 TO FLAGS.
                 MOVE "MESSAGE TEXT 1" TO SENDMSG-BUFFER1.
                 MOVE "MESSAGE TEXT 2" TO SENDMSG-BUFFER2.
                 MOVE "MESSAGE TEXT 3" TO SENDMSG-BUFFER3.

          CALL 'EZASOKET' USING SOC-FUNCTION MSG FLAGS ERRNO RETCODE.
```

Figure 155. SENDMSG call instruction example (Part 2 of 2)

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 226.

Parameter values set by the application

S A value or the address of a halfword binary number specifying the socket descriptor.

MSG A pointer to an array of message headers from which messages are sent.

Field **Description**

NAME
On input, a pointer to a buffer where the sender's address is stored upon completion of the call. The storage being pointed to should be for an IPv4 socket address or an IPv6 socket address.

The IPv4 socket address structure contains the following fields:

Field	Description
FAMILY	A halfword binary number specifying the IPv4 addressing family. The value for IPv4 socket descriptor (that is, S parameter) is a 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 eight-byte reserved field. This field is required, but is not used.

The IPv6 socket address structure contains the following fields:

Field	**Description**
FAMILY	A halfword binary field specifying the IPv6 addressing family. The value for IPv6 socket descriptor (for example, S parameter) is a decimal 19, indicating AF_INET6.
PORT	A halfword binary number specifying the port number of the sending socket.
FLOW-INFO	A fullword binary field specifying the traffic class and flow label. This field must be set to zero.
IP-ADDRESS	A two doubleword, 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 IP-ADDRESS field. A value of zero indicates the SCOPE-ID field does not identify the set of interfaces to be used, and can be specified for any address types and scopes. For a link scope IP-ADDRESS, SCOPE-ID can specify a link index which identifies a set of interfaces. For all other address scopes, SCOPE-ID must be set to zero.

NAME-LEN
On input, a pointer to the size of the address buffer that is filled in on completion of the call.

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 portion, 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 field containing the following:

Literal value	Binary value	Description
NO-FLAG	0	No flag is set. The command behaves like a WRITE call.
OOB	1	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.
DONT-ROUTE	4	Do not route. Routing is provided by the calling program.

Parameter values returned by the application

ERRNO
A fullword binary field. If RETCODE is negative, this contains an error number. See Appendix B. Return codes on page 397 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.

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 350 for a subroutine that translates EBCDIC input data to ASCII.

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
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 156 shows an example of SENDTO call instructions.

```
WORKING-STORAGE SECTION.
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'SENDTO'.
    01  S               PIC 9(4) BINARY.
    01  FLAGS.          PIC 9(8) BINARY.
    01  NO-FLAG         PIC 9(8) BINARY  VALUE IS 0.
    01  OOB             PIC 9(8) BINARY  VALUE IS 1.
    01  DONT-ROUTE      PIC 9(8) BINARY  VALUE IS 4.
    01  NBYTE           PIC 9(8) BINARY.
    01  BUF             PIC X(length of buffer).
*
* 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  FLOW-INFO   PIC 9(8) BINARY.
        03  IP-ADDRESS.
            05  FILLER  PIC 9(16) BINARY.
            05  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 FLAGS NBYTE
                    BUF NAME ERRNO RETCODE.
```

Figure 156. SENDTO call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 226.

Parameter values set by the application

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

S A halfword binary number set to the socket descriptor of the socket sending the data.

FLAGS
A fullword field that returns one of the following:

Literal value	Binary value	Description
NO-FLAG	0	No flag is set. The command behaves like a WRITE call.
OOB	1	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.
DONT-ROUTE	4	Do not route. Routing is provided by the calling program.

NBYTE
A fullword binary number set to the number of bytes to transmit.

BUF Specifies the buffer containing the data to be transmitted. BUF should be the size specified in NBYTE.

NAME

Specifies the IPv4 socket address structure as follows:

FAMILY
A halfword binary field containing the addressing family. For TCP/IP the value must be a 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 addressing family. For TCP/IP stacks the value must be a decimal 19, indicating AF_INET6.

PORT
A halfword binary field containing the port number bound to the socket.

FLOW-INFO
A fullword binary field specifying the traffic class and flow label. This field must be set to zero.

IP-ADDRESS
A 16-byte binary field containing the socket's 128-bit IPv6 Internet address.

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

Parameter values returned to the application

ERRNO
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B. Return codes on page 397 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.

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:

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
Amode:	31-bit or 24-bit
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 157 on page 327 shows an example of SETSOCKOPT call instructions.

```
WORKING-STORAGE SECTION.
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'SETSOCKOPT'.
    01  S               PIC 9(4)  BINARY.
    01  OPTNAME         PIC 9(8)  BINARY.
    01  OPTVAL          PIC 9(8)  BINARY.
    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 157. SETSOCKOPT call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 226.

Parameter values set by the application

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

S A halfword binary number set to the socket whose options are to be set.

OPTNAME
Input parameter. See the table below for a list of the options and their unique requirements. See Appendix C, “GETSOCKOPT/SETSOCKOPT command values,” on page 415 for the numeric values of OPTNAME.

Note: COBOL programs cannot contain field names with the underscore character. Fields representing the option name should contain dashes instead.

OPTVAL
Input parameter. Contains data that further defines the option specified in OPTNAME. See the table below for a list of the options and their unique requirements.

OPTLEN
Input parameter. A fullword binary field specifying the length of the data specified in OPTVAL. See the table below for how to determine 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 B, “Return codes,” on page 397 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.

Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
IP_ADD_MEMBERSHIP Use this option to enable an application to join a multicast group on a specific interface. An interface has to be specified with this option. Only applications that want to receive multicast datagrams need to join multicast groups. This is an IPv4-only socket option.	Contains the IP_MREQ structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 interface address. See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ. See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ.	N/A
IP_ADD_SOURCE_MEMBERSHIP Use this option to enable an application to join a source multicast group on a specific interface and a specific source address. You must specify an interface and a source address with this option. Applications that want to receive multicast datagrams need to join source multicast groups. This is an IPv4-only socket option.	Contains the IP_MREQ_SOURCE structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ_SOURCE structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 source address and a 4-byte IPv4 interface address. See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE. See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ-SOURCE.	N/A
IP_BLOCK_SOURCE Use this option to enable an application to block multicast packets that have a source address that matches the given IPv4 source address. You must specify an interface and a source address with this option. The specified multicast group must have been joined previously. This is an IPv4-only socket option.	Contains the IP_MREQ_SOURCE structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ_SOURCE structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 source address and a 4-byte IPv4 interface address. See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE. See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ-SOURCE.	N/A

Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
IP_DROP_MEMBERSHIP Use this option to enable an application to exit a multicast group or to exit all sources for a multicast group. This is an IPv4-only socket option.	Contains the IP_MREQ structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 interface address. See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ. See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ.	N/A
IP_DROP_SOURCE_MEMBERSHIP Use this option to enable an application to exit a source multicast group. This is an IPv4-only socket option.	Contains the IP_MREQ_SOURCE structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ_SOURCE structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 source address and a 4-byte IPv4 interface address. See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE. See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ-SOURCE.	N/A
IP_MULTICAST_IF Use this option to set or obtain the IPv4 interface address used for sending outbound multicast datagrams from the socket application. This is an IPv4-only socket option. **Note:** Multicast datagrams can be transmitted only on one interface at a time.	A 4-byte binary field containing an IPv4 interface address.	A 4-byte binary field containing an IPv4 interface address.
IP_MULTICAST_LOOP 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. This is an IPv4-only socket option.	A 1-byte binary field. To enable, set to 1. To disable, set to 0.	A 1-byte binary field. If enabled, will contain a 1. If disabled, will contain a 0.
IP_MULTICAST_TTL 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. This is an IPv4-only socket option.	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.

Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
IP_UNBLOCK_SOURCE Use this option to enable an application to unblock a previously blocked source for a given IPv4 multicast group. You must specify an interface and a source address with this option. This is an IPv4-only socket option.	Contains the IP_MREQ_SOURCE structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ_SOURCE structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 source address and a 4-byte IPv4 interface address. See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE. See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ-SOURCE.	
IPV6_JOIN_GROUP Use this option to control the reception of multicast packets and specify that the socket join a multicast group. This is an IPv6-only socket option.	Contains the IPV6_MREQ structure as defined in SYS1.MACLIB(BPXYSOCK). The IPV6_MREQ structure contains a 16-byte IPv6 multicast address followed by a 4-byte IPv6 interface index number. If the interface index number is 0, then the stack chooses the local interface. See the SEZAINST(CBLOCK) for the PL/I example of IPV6_MREQ. See SEZAINST(EZACOBOL) for the COBOL example of IPV6-MREQ.	N/A
IPV6_LEAVE_GROUP Use this option to control the reception of multicast packets and specify that the socket leave a multicast group. This is an IPv6-only socket option.	Contains the IPV6_MREQ structure as defined in SYS1.MACLIB(BPXYSOCK). The IPV6_MREQ structure contains a 16-byte IPv6 multicast address followed by a 4-byte IPv6 interface index number. If the interface index number is 0, then the stack chooses the local interface. See the SEZAINST(CBLOCK) for the PL/I example of IPV6_MREQ. See SEZAINST(EZACOBOL) for the COBOL example of IPV6-MREQ.	N/A

Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
IPV6_MULTICAST_HOPS Use to set or obtain the hop limit used for outgoing multicast packets. This is an IPv6-only socket option.	Contains a 4-byte binary value specifying the multicast hops. If not specified, then the default is 1 hop. -1 indicates use stack default. 0 – 255 is the valid hop limit range. **Note:** An application must be APF authorized to enable it to set the hop limit value above the system defined hop limit value. CICS applications cannot execute as APF authorized.	Contains a 4-byte binary value in the range 0 – 255 indicating the number of multicast hops.
IPV6_MULTICAST_IF Use this option to set or obtain the index of the IPv6 interface used for sending outbound multicast datagrams from the socket application. This is an IPv6-only socket option.	Contains a 4-byte binary field containing an IPv6 interface index number.	Contains a 4-byte binary field containing an IPv6 interface index number.
IPV6_MULTICAST_LOOP Use this option to control or determine whether a multicast datagram is looped back on the outgoing interface by the IP layer for local delivery when datagrams are sent to a group to which the sending host itself belongs. The default is to loop multicast datagrams back. This is an IPv6-only socket option.	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.
IPV6_UNICAST_HOPS Use this option to set or obtain the hop limit used for outgoing unicast IPv6 packets. This is an IPv6-only socket option.	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.
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. This is an IPv6-only socket option.	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.

Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
MCAST_BLOCK_SOURCE Use this option to enable an application to block multicast packets that have a source address that matches the given source address. You must specify an interface index and a source address with this option. The specified multicast group must have been joined previously.	Contains the GROUP_SOURCE_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_SOURCE_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address and a socket address structure of the source address. 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
MCAST_JOIN_SOURCE_GROUP Use this option to enable an application to join a source multicast group on a specific interface and a source address. You must specify an interface index and the source address. Applications that want to receive multicast datagrams only from specific source addresses need to join source multicast groups.	Contains the GROUP_SOURCE_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_SOURCE_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address and a socket address structure of the source address. See SEZAINST(CBLOCK) for the PL/I example of GROUP_SOURCE_REQ. See SEZAINST(EZACOBOL) for the COBOL example of GROUP-SOURCE-REQ.	N/A

Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
MCAST_LEAVE_GROUP Use this option to enable an application to exit a multicast group or exit all sources for a given multicast groups.	Contains the GROUP_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address. See SEZAINST(CBLOCK) for the PL/I example of GROUP_REQ. See SEZAINST(EZACOBOL) for the COBOL example of GROUP-REQ.	N/A
MCAST_LEAVE_SOURCE_GROUP Use this option to enable an application to exit a source multicast group.	Contains the GROUP_SOURCE_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_SOURCE_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address and a socket address structure of the source address. See SEZAINST(CBLOCK) for the PL/I example of GROUP_SOURCE_REQ. See SEZAINST(EZACOBOL) for the COBOL example of GROUP-SOURCE-REQ.	N/A
MCAST_UNBLOCK_SOURCE Use this option to enable an application to unblock a previously blocked source for a given multicast group. You must specify an interface index and a source address with this option.	Contains the GROUP_SOURCE_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_SOURCE_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address and a socket address structure of the source address. See SEZAINST(CBLOCK) for the PL/I example of GROUP_SOURCE_REQ. See SEZAINST(EZACOBOL) for the COBOL example of GROUP-SOURCE-REQ.	N/A

Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
SO_ASCII 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. **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_BROADCAST 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. **Note:** This option has no meaning for stream sockets.	A 4-byte binary field. To enable, set to 1 or a positive value. To disable, set to 0.	A 4-byte field. If enabled, contains a 1. If disabled, contains a 0.
SO_DEBUG 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 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<tr><th>OPTNAME options (input)</th><th>SETSOCKOPT, OPTVAL (input)</th><th>GETSOCKOPT, OPTVAL (output)</th></tr>
<tr><td>SO_KEEPALIVE

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>
<tr><td>SO_LINGER

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:
<code>ONOFF   DS F</code>
<code>LINGER DS F</code>

COBOL coding:
<code>ONOFF PIC 9(8) BINARY.</code>
<code>LINGER PIC 9(8) BINARY.</code>

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><td>Contains an 8-byte field containing two 4-byte binary fields.

Assembler coding:
<code>ONOFF   DS F</code>
<code>LINGER DS F</code>

COBOL coding:
<code>ONOFF PIC 9(8) BINARY.</code>
<code>LINGER PIC 9(8) BINARY.</code>

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.</td></tr>
</table>

Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
SO_OOBINLINE 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.	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.
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 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
SO_REUSEADDR 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, refer to the SHAREPORT option on the PORT statement in TCPIP.PROFILE.	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.
SO_SNDBUF 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 TCPSENDBufrsize keyword on the TCPCONFIG statement in the PROFILE.TCPIP data set for a TCP socket • The UDPSENDBufrsize 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 send 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 send buffer. If disabled, contains a 0.
SO_TYPE Use this option to return the socket type.	N/A	A 4-byte binary field indicating the socket type: X'1' indicates SOCK_STREAM. X'2' indicates SOCK_DGRAM. X'3' indicates SOCK_RAW.

Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
TCP_KEEPALIVE 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-specified timer value remains in effect until respecified by SETSOCKOPT or until the socket is closed. Refer to the *z/OS Communications Server: IP Programmer's Guide and Reference* for more information on the socket option parameters.	A 4-byte binary field. To enable, set to a value in the range of 1 – 2 147 460. To disable, set to a value of 0.	A 4-byte binary field. If enabled, contains the specific timer value (in seconds) that is in effect for the given socket. If disabled, contains a 0 indicating keep alive timing is not active.

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:
`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.`
	A 4-byte binary field.
	
	To enable, set to a 0.
	
	To disable, set to a 1 or nonzero.
		A 4-byte binary field.
		
		If enabled, contains a 0.
		
		If disabled, contains a 1.

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 waits before releasing the connection. For example, with a LINGER value of 30 seconds, system resources (including the IMS or CICS transaction) 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 Table 23 to determine the effects of this operation on the outstanding socket calls.

Table 23. Effect of SHUTDOWN socket call

Socket calls in local program	Local program		Remote program	
	SHUTDOWN SEND	**SHUTDOWN RECEIVE**	**SHUTDOWN RECEIVE**	**SHUTDOWN SEND**
Write calls	Error number EPIPE on first call		Error number EPIPE on second call*	
Read calls		Zero length return code		Zero length return code
* If you issue two write calls immediately, both might be successful, and an EPIPE error number might not be returned until a third write call is issued.				

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
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 158 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.
    01 END-FROM      PIC 9(8)  BINARY VALUE 0.
    01 END-TO        PIC 9(8)  BINARY VALUE 1.
    01 END-BOTH      PIC 9(8)  BINARY 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 158. SHUTDOWN call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 226.

Parameter values set by the application

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

S A halfword binary number set to the socket descriptor of the socket to be shutdown.

HOW A fullword binary field. Set to specify whether all or part of a connection is to be shut down. The following values can be set:

Value	Description
0 (END-FROM)	Ends further receive operations.
1 (END-TO)	Ends further send operations.
2 (END-BOTH)	Ends further send and receive operations.

Parameter values returned to the application

ERRNO
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B. Return codes on page 397 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:

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
Amode:	31-bit or 24-bit
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 159 on page 341 shows an example of SOCKET call instructions.

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

    01 PROTO         PIC 9(8)  BINARY.
    01 ERRNO         PIC 9(8)  BINARY.
    01 RETCODE       PIC S9(8) BINARY.

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

Figure 159. SOCKET call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 226.

Parameter values set by the application

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

AF
A fullword binary field set to the addressing family. For TCP/IP the value is set to a decimal 2 for AF_INET, or a decimal 19, indicating AF_INET6.

SOCTYPE
A fullword binary field set to the type of socket required. The types are:

Value	Description
1	Stream sockets provide sequenced, two-way byte streams that are reliable and connection-oriented. They support a mechanism for out-of-band data.
2	Datagram sockets provide datagrams, which are connectionless messages of a fixed maximum length whose reliability is not guaranteed. Datagrams can be corrupted, received out of order, lost, or delivered multiple times.

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.

Parameter values returned to the application

ERRNO
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B. Return codes on page 397 for information about ERRNO return codes.

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

Value **Description**

> or = 0
Contains the new socket descriptor

–1 Check ERRNO for an error code

TAKESOCKET

The TAKESOCKET call acquires a socket from another program and creates a new socket. Typically, a child server issues this call using client ID and socket descriptor data that it obtained from the concurrent server. See "GIVESOCKET" on page 274 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:

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
Amode:	31-bit or 24-bit
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 160 shows an example of TAKESOCKET call instructions.

```
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 160. TAKESOCKET call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 226.

Parameter values set by the application

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

SOCRECV
A halfword binary field set to the descriptor of the socket to be taken. The socket to be taken is passed by the concurrent server.

CLIENT
Specifies the client ID of the program that is giving the socket. In CICS, these parameters are passed by the listener program to the program that issues the TAKESOCKET call. The information is obtained using EXEC CICS RETRIEVE.

DOMAIN
A fullword binary field set to the domain of the program giving the socket. It is always a decimal 2, indicating AF_INET, or a decimal 19, indicating AF_INET6.

Rule: 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 character 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 RETCODE is negative, the field contains an error number. See Appendix B. Return codes on page 397 for information about ERRNO return codes.

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

Value	Description
> or = 0	Contains the new socket descriptor
–1	Check ERRNO for an error code

TERMAPI

This call terminates the session created by INITAPI. All TCP/IP stacks resources allocated to the task are cleaned up. This includes any outstanding open sockets or sockets that have been given away with the GIVESOCKET call but have not been taken with a TAKESOCKET call.

In the CICS environment, the use of TERMAPI is not recommended. CICS task termination processing automatically performs the functions of TERMAPI. A CICS application program should only issue TERMAPI if there is a particular need to terminate the session before task termination.

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
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 161 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 161. TERMAPI call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 226.

Parameter values set by the application

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

WRITE

The WRITE call writes data on a connected socket. This call is similar to SEND, except that it lacks the control flags available with SEND.

For datagram sockets the WRITE call writes the entire datagram if it fits into the receiving buffer.

Stream sockets act like streams of information with no boundaries separating data. For example, if a program 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 are 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 350 for a subroutine that translates EBCDIC output data to ASCII.

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 "Environmental restrictions and programming requirements" on page 223.
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 162 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 162. WRITE call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 226.

Parameter values set by the application

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

S A halfword binary field set to the socket descriptor.

NBYTE
A fullword binary field set to the number of bytes of data to be transmitted.

BUF Specifies the buffer containing the data to be transmitted.

Parameter values returned to the application

ERRNO
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B. Return codes on page 397 for information about ERRNO return codes.

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

Value	Description
≥0	A successful call. A return code greater than zero indicates the number of bytes of data written.
–1	Check ERRNO for an error code.

WRITEV

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

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
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 163 shows an example of WRITEV call instructions.

```
WORKING-STORAGE SECTION.
       01 SOKET-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) 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 163. WRITEV call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 226.

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 B. Return codes on page 397 for information about ERRNO return codes.

RETCODE
A fullword binary field.

Value	Meaning
<0	Error. Check ERRNO.
0	Connection partner has closed connection.
>0	Number of bytes sent.

Using data translation programs for socket call interface

In addition to the socket calls, you can use 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 an EBCDIC-to-ASCII translation table as described in*z/OS Communications Server: IP Configuration Reference*.

EZACIC05
Translates ASCII data to EBCDIC data using an ASCII-to-EBCDIC translation table as described in *z/OS Communications Server: IP Configuration Reference*.

EZACIC14
An alternative to EZACIC04 that translates EBCDIC data to ASCII data using the translation table listed in "EZACIC14" on page 363.

EZACIC15
An alternative to EZACIC05 that translates ASCII data to EBCDIC data using the translation table listed in "EZACIC15" on page 365.

A sample program that performs these translations is also available; you can modify them to perform any translations not provided by these routines. See the EZACICTR member in the SEZAINST data set for more information.

It is not necessary to define these programs to CICS. If your application dynamically links these programs, then you must define them to CICS as follows:

```
DEFINE PROGRAM(EZACIC04)
DESCRIPTION(TRANSLATE EBCDIC-8 BIT TO ASCII-8 BIT)
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(USER)
RELOAD(NO) RESIDENT(NO) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(NORMAL)
CONCURRENCY(THREADSAFE)

DEFINE PROGRAM(EZACIC05)
DESCRIPTION(TRANSLATE ASCII-8 BIT TO EBCDIC-8 BIT)
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(USER)
RELOAD(NO) RESIDENT(NO) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(NORMAL)
CONCURRENCY(THREADSAFE)

DEFINE PROGRAM(EZACIC14)
DESCRIPTION(TRANSLATE EBCDIC-8 BIT TO ASCII-8 BIT)
GROUP(SOCKETS)
```

```
CEDF(YES) DATALOCATION(ANY) EXECKEY(USER)
RELOAD(NO) RESIDENT(NO) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(NORMAL)
CONCURRENCY(THREADSAFE)

DEFINE PROGRAM(EZACIC15)
DESCRIPTION(TRANSLATE ASCII-8 BIT TO EBCDIC-8 BIT)
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(USER)
RELOAD(NO) RESIDENT(NO) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(NORMAL)
CONCURRENCY(THREADSAFE)
```

| For more information about specifying the key that CICS uses to give control to
| the program, see the CICS Transaction Server information in *CICS Resource
| Definition Guide* for details about RDO resource types and their attributes, Program
| Definition Attributes, and the EXECKEY attribute.

Bit string processing

In C-language, bit strings are often used to convey flags, switch settings, and so on; TCP/IP stacks makes frequent uses of bit strings. However, because bit strings are difficult to decode in COBOL, TCP/IP includes:

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.

It is not necessary to define these programs to CICS. If your application dynamically links these programs, then you must define them to CICS as follows:

```
DEFINE PROGRAM(EZACIC06)
DESCRIPTION(TRANSLATE EBCDIC-8 BIT TO ASCII-8 BIT)
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(USER)
RELOAD(NO) RESIDENT(NO) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(NORMAL)
CONCURRENCY(THREADSAFE)

DEFINE PROGRAM(EZACIC08)
DESCRIPTION(INTERPRET HOSTENT)
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(USER)
RELOAD(NO) RESIDENT(NO) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(NORMAL)
CONCURRENCY(THREADSAFE)

DEFINE PROGRAM(EZACIC09)
DESCRIPTION(INTERPRET ADDRINFO)
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(USER)
RELOAD(NO) RESIDENT(NO) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(NORMAL)
CONCURRENCY(THREADSAFE)
```

| For more information about specifying the key that CICS uses to give control to
| the program, see *CICS Resource Definition Guide* .

CALL instruction utility programs

This topic describes the CALL instruction API for TCP/IP application programs written in the COBOL, PL/I, or High Level Assembler language. The format and parameters are described for each utility call.

Note: For a PL/I program, include the following statement before your first call instruction:

```
DCL EZASOKET ENTRY OPTIONS(RETCODE,ASM,INTER) EXT;
```

Understanding COBOL, assembler, and PL/I call formats: These utility programs are invoked by calling the EZACICnn program. The parameters look differently due to the differences in the programming languages.

COBOL language call format: The following sample illustrates the utility program call format for COBOL language programs:

```
>>-- CALL 'EZACICnn' USING parm1, parm2, ... . --><
```

parm *n*
: A variable number of parameters that depends on the type call.

See the utility programs in this topic for an explanation of the parameters.

Assembler language call format: The following sample illustrates the utility program call format for assembler language programs. Because DATAREG is used to access the application's working storage, applications using the assembler language format should not code DATAREG but should let it default to the CICS data register.

```
>>-- CALL EZACICnn,(parm1, parm2, ... ),VL,MF=(E, PARMLIST) --><
```

PARMLIST is a remote parameter list defined in dynamic storage DFHEISTG. This list contains addresses of 30 parameters that can be referenced by all execute forms of the CALL.

Note: This form of CALL is necessary to meet the CICS requirement for quasi-reentrant programming

parm *n*
: A variable number of parameters that depends on the type call.

See the utility programs in this topic for an explanation of the parameters.

PL/I language call format: The following sample illustrates the utility program call format for PL/I language programs:

```
>>-- CALL EZACICnn (parm1, parm2, ... ); --><
```

parm *n*

parm *n*
: A variable number of parameters that depends on the type call.

See the utility programs in this topic for an explanation of the parameters.

EZACIC04

The EZACIC04 program is used to translate EBCDIC data to ASCII data.

Figure 164 shows an example of how EZACIC04 translates a byte of EBCDIC data to ASCII data.

ASCII output by EZACIC04	second hex digit of byte of EBCDIC data															
first hex digit of byte of EBCDIC data	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	01	02	03	1A	09	1A	7F	1A	1A	1A	0B	0C	0D	0E	0F
1	10	11	12	13	1A	0A	08	1A	18	19	1A	1A	1C	1D	1E	1F
2	1A	1A	1C	1A	1A	0A	17	1B	1A	1A	1A	1A	1A	05	06	07
3	1A	1A	16	1A	1A	1E	1A	04	1A	1A	1A	1A	14	15	1A	1A
4	20	A6	E1	80	EB	90	9F	E2	AB	8B	9B	2E	3C	28	2B	7C
5	26	A9	AA	9C	DB	A5	99	E3	A8	9E	21	24	2A	29	3B	5E
6	2D	2F	DF	DC	9A	DD	DE	98	9D	AC	BA	2C	25	5F	3E	3F
7	D7	88	94	B0	B1	B2	FC	D6	FB	60	3A	23	40	27	3D	22
8	F8	61	62	63	64	65	66	67	68	69	96	A4	F3	AF	AE	C5
9	8C	6A	6B	6C	6D	6E	6F	70	71	72	97	87	CE	93	F1	FE
A	C8	7E	73	74	75	76	77	78	79	7A	EF	C0	DA	5B	F2	AE
B	B5	B6	FD	B7	B8	B9	E6	BB	BC	BD	8D	D9	BF	5D	D8	C4
C	7B	41	42	43	44	45	46	47	48	49	CB	CA	BE	E8	EC	ED
D	7D	4A	4B	4C	4D	4E	4F	50	51	52	A1	AD	F5	F4	A3	8F
E	5C	E7	53	54	55	56	57	58	59	5A	A0	85	8E	E9	E4	D1
F	30	31	32	33	34	35	36	37	38	39	B3	F7	F0	FA	A7	FF

Figure 164. EZACIC04 EBCDIC-to-ASCII table

Figure 165 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 165. EZACIC04 call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 226.

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 166 shows an example of how EZACIC05 translates a byte of ASCII data to EBCDIC data.

EBCDIC output by EZACIC05	second hex digit of byte of ASCII data																
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
first hex digit of byte of ASCII data	0	00	01	02	03	37	2D	2E	2F	16	05	25	0B	0C	0D	0E	0F
	1	10	11	12	13	3C	3D	32	26	18	19	3F	27	22	1D	35	1F
	2	40	5A	7F	7B	5B	6C	50	7D	4D	5D	5C	4E	6B	60	4B	61
	3	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	7A	5E	4C	7E	6E	6F
	4	7C	C1	C2	C3	C4	C5	C6	C7	C8	C9	D1	D2	D3	D4	D5	D6
	5	D7	D8	D9	E2	E3	E4	E5	E6	E7	E8	E9	AD	E0	BD	5F	6D
	6	79	81	82	83	84	85	86	87	88	89	91	92	93	94	95	96
	7	97	98	99	A2	A3	A4	A5	A6	A7	A8	A9	C0	4F	D0	A1	07
	8	00	01	02	03	37	2D	2E	2F	16	05	25	0B	0C	0D	0E	0F
	9	10	11	12	13	3C	3D	32	26	18	19	3F	27	22	1D	35	1F
	A	40	5A	7F	7B	5B	6C	50	7D	4D	5D	5C	4E	6B	60	AF	61
	B	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	7A	5E	4C	7E	6E	6F
	C	7C	C1	C2	C3	C4	C5	C6	C7	C8	C9	D1	D2	D3	D4	D5	D6
	D	D7	D8	D9	E2	E3	E4	E5	E6	E7	E8	E9	AD	E0	BD	5F	6D
	E	79	81	82	83	84	85	86	87	88	89	91	92	93	94	95	96
	F	97	98	99	A2	A3	A4	A5	A6	A7	A8	A9	C0	4F	D0	A1	07

Figure 166. EZACIC05 ASCII-to-EBCDIC

Figure 167 shows an example of EZACIC05 call instructions.

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

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

Figure 167. EZACIC05 call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 226.

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.

EZACIC06

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, use the assembler language program EZACIC06 to translate them to character strings to be used with the SELECT call.

Figure 168 shows an example of EZACIC06 call instructions.

```
WORKING STORAGE
      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 'EZACIC06' USING CTOB
                             BIT-MASK
                             CHAR-MASK
                             CHAR-MASK-LENGTH
                             RETCODE.

  PROCEDURE CALL (to convert from binary to character)
       CALL 'EZACIC06' USING BTOC
                             BIT-MASK
                             CHAR-MASK
                             CHAR-MASK-LENGTH
                             RETCODE.
```

Figure 168. EZACIC06 call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 226.

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. Keep in mind that the index is 1 greater than the socket number. That is, CHAR-ENTRY(1) represents socket 0, CHAR-ENTRY(2) represents socket 1, and so on.

BIT-MASK
: Specifies the bit string to be translated for the SELECT call. Within each fullword of the bit string, the bits are ordered right to left. The rightmost bit in the first fullword represents socket 0 and the leftmost bit represents socket 31. The rightmost bit in the second fullword represents socket 32 and the leftmost bit represents socket 63. The number of fullwords in the bit string should be calculated by dividing the sum of 31 and the character array length by 32 (truncate the remainder).

COMMAND

BTOC—Specifies bit string to character array translation.

CTOB—Specifies character array to bit string translation.

CHAR-MASK-LENGTH

Specifies the length of the character array. This field should be no greater than 1 plus the MAXSNO value returned on the INITAPI (which is usually the same as the MAXSOC value specified on the INITAPI).

RETCODE

A binary field that returns one of the following:

Value	**Description**
0	Successful call
–1	Check ERRNO for an error code

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 the following example, index position 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 are checked. The second word of BIT-MASK contains B'00000000000000000000000000000001' to indicate that socket 32 is 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 stacks host can have multiple alias names and host Internet addresses.

TCP/IP stacks uses indirect addressing to connect the variable number of alias names and Internet addresses in the HOSTENT structure that is 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 straightforward 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:

- GETHOSTBYADDR or GETHOSTBYNAME returns a HOSTENT structure that indirectly addresses the lists of alias names and Internet addresses.
- 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:
  1. The length of host name, if present
  2. The host name
  3. The number of alias names for the host
  4. The alias name sequence number
  5. The length of the alias name
  6. The alias name
  7. The host Internet address type, always 2 for AF_INET
  8. The host Internet address length, always 4 for AF_INET
  9. The number of host Internet addresses for this host
  10. The host Internet address sequence number
  11. The host Internet address
- If the GETHOSTBYADDR or GETHOSTBYNAME call returns more than one alias name or host Internet address (steps 3 and 9 above), the application program should repeat the call to EZACIC08 until all alias names and host Internet addresses have been retrieved.

Figure 169 on page 357 shows an example of EZACIC08 call instructions.

```
WORKING-STORAGE SECTION.

    01  HOSTENT-ADDR       PIC 9(8) BINARY.
    01  HOSTNAME-LENGTH    PIC 9(4) BINARY.
    01  HOSTNAME-VALUE     PIC X(255).
    01  HOSTALIAS-COUNT    PIC 9(4) BINARY.
    01  HOSTALIAS-SEQ      PIC 9(4) BINARY.
    01  HOSTALIAS-LENGTH   PIC 9(4) BINARY.
    01  HOSTALIAS-VALUE    PIC X(255).
    01  HOSTADDR-TYPE      PIC 9(4) BINARY.
    01  HOSTADDR-LENGTH    PIC 9(4) BINARY.
    01  HOSTADDR-COUNT     PIC 9(4) BINARY.
    01  HOSTADDR-SEQ       PIC 9(4) BINARY.
    01  HOSTADDR-VALUE     PIC 9(8) BINARY.
    01  RETURN-CODE        PIC 9(8) BINARY.

PROCEDURE DIVISION.

   CALL 'EZASOKET' USING 'GETHOSTBYADDR'
                   HOSTADDR HOSTENT-ADDR
                   RETCODE.

   CALL 'EZASOKET' USING 'GETHOSTBYNAME'
                   NAMELEN NAME HOSTENT-ADDR
                   RETCODE.

   CALL 'EZACIC08' USING HOSTENT-ADDR HOSTNAME-LENGTH
                   HOSTNAME-VALUE HOSTALIAS-COUNT HOSTALIAS-SEQ
                   HOSTALIAS-LENGTH HOSTALIAS-VALUE
                   HOSTADDR-TYPE HOSTADDR-LENGTH HOSTADDR-COUNT
                   HOSTADDR-SEQ HOSTADDR-VALUE RETURN-CODE
```

Figure 169. EZAZIC08 call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 226.

Parameter values set by the application

HOSTENT-ADDR
: This fullword binary field must contain the address of the HOSTENT structure (as returned by the GETHOSTBY*xxxx* call). This variable is the same as the variable HOSTENT in the GETHOSTBYADDR and GETHOSTBYNAME socket calls.

HOSTALIAS-SEQ
: This halfword field is used by EZACIC08 to index the list of alias names. When EZACIC08 is called, it adds one 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 one 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:

Value	Description
0	Successful completion
-1	Invalid HOSTENT address
-2	Invalid HOSTALIAS-SEQ value
-3	Invalid HOSTADDR-SEQ value

EZACIC09

The GETADDRINFO call was derived from the C socket call that returns a structure known as RES. A given TCP/IP stacks host can have multiple sets of NAMES. TCP/IP stacks uses indirect addressing to connect the variable number of NAMES in the RES structure that the GETADDRINFO call returns. If you are coding in PL/I or Assembler language, the RES structure can be processed in a relatively straightforward 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:

- GETADDRINFO returns a RES structure that indirectly addresses the lists of socket address structures.
- 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:
  1. The socket address structure
  2. The next address information structure
- 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 170 on page 360 shows an example of EZACIC09 call instructions.

```
WORKING-STORAGE SECTION.
     *
     * Variables used for the GETADDRINFO call
     *
      01  getaddrinfo-parms.
          02  node-name                  pic x(255).
          02  node-name-len              pic 9(8) binary.
          02  service-name               pic x(32).
          02  service-name-len           pic 9(8) binary.
          02  canonical-name-len         pic 9(8) binary.
          02  ai-passive                 pic 9(8) binary value 1.
          02  ai-canonnameok             pic 9(8) binary value 2.
          02  ai-numerichost             pic 9(8) binary value 4.
          02  ai-numericserv             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.

     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.
```

Figure 170. EZACIC09 call instruction example (Part 1 of 2)

```
*
* SOCKET ADDRESS STRUCTURE FROM EZACIC09.
*
 01  OUTPUT-NAME-PTR                USAGE IS POINTER.
 01  OUTPUT-IP-NAME.
     03  OUTPUT-IP-FAMILY           PIC 9(4) BINARY.
     03  OUTPUT-IP-PORT             PIC 9(4) BINARY.
     03  OUTPUT-IP-SOCK-DATA        PIC X(24).
     03  OUTPUT-IPV4-SOCK-DATA REDEFINES OUTPUT-IP-SOCK-DATA.
         05  OUTPUT-IPV4-IPADDR     PIC 9(8) BINARY.
         05  FILLER                 PIC X(20).
     03  OUTPUT-IPV6-SOCK-DATA REDEFINES OUTPUT-IP-SOCK-DATA.
         05  OUTPUT-IPV6-FLOWINFO   PIC 9(8) BINARY.
         05  OUTPUT-IPV6-IPADDR.
             10 FILLER              PIC 9(16) BINARY.
             10 FILLER              PIC 9(16) BINARY.
         05  OUTPUT-IPV6-SCOPEID    PIC 9(8) BINARY.

 PROCEDURE DIVISION USING L1.

*
* Get an address from the resolver.
*
     move 'yournodename' to node-name.
     move 12 to node-name-len.
     move spaces to service-name.
     move 0 to service-name-len.
     move af-inet6 to hints-ai-family.
     move 49 to hints-ai-flags
     move 0 to hints-ai-socktype.
     move 0 to hints-ai-protocol.
     set address of results-addrinfo to res-addrinfo-ptr.
     set hints-addrinfo-ptr to address of hints-addrinfo.
     call 'EZASOKET' using soket-getaddrinfo
                       node-name node-name-len
                       service-name service-name-len
                       hints-addrinfo-ptr
                       res-addrinfo-ptr
                       canonical-name-len
                       errno retcode.
*
* Use EZACIC09 to extract the IP address
*
     set address of results-addrinfo to res-addrinfo-ptr.
     set res to address of results-addrinfo.
     move zeros to res-name-len.
     move spaces to res-canonical-name.
     set res-name to nulls.
     set res-next-addrinfo to nulls.
     call 'EZACIC09' using res
                       res-name-len
                       res-canonical-name
                       res-name
                       res-next-addrinfo
                       retcode.
     set address of output-ip-name to res-name.
     move output-ipv6-ipaddr to server-ipaddr.
```

Figure 170. EZACIC09 call instruction example (Part 2 of 2)

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 226.

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 contains the length of the socket address structure as returned by the GETADDRINFO call.

Parameter values returned to the application

RES-CANONICAL-NAME
A field large enough to hold the canonical name. The maximum field size is 256 bytes. The canonical name length field indicates 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
This fullword binary field contains the EZACIC09 return code:

Value	Description
0	Successful completion
-1	Invalid HOSTENT address

EZACIC14

The EZACIC14 program is an alternative to EZACIC04, which is used to translate EBCDIC data to ASCII data.

Figure 171 shows an example of how EZACIC14 translates a byte of EBCDIC data.

ASCII output by EZACIC14	second hex digit of byte of EBCDIC data															
first hex digit of byte of EBCDIC data	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	01	02	03	9C	09	86	7F	97	8D	8E	0B	0C	0D	0E	0F
1	10	11	12	13	9D	85	08	87	18	19	92	8F	1C	1D	1E	1F
2	80	81	82	83	84	0A	17	1B	88	89	8A	8B	8C	05	06	07
3	90	91	16	93	94	95	96	04	98	99	9A	9B	14	15	9E	1A
4	20	A0	E2	E4	E0	E1	E3	E5	E7	F1	A2	2E	3C	28	2B	7C
5	26	E9	EA	EB	E8	ED	EE	EF	EC	DF	21	24	2A	29	3B	5E
6	2D	2F	C2	C4	C0	C1	C3	C5	C7	D1	A6	2C	25	5F	3E	3F
7	F8	C9	CA	CB	C8	CD	CE	CF	CC	60	3A	23	40	27	3D	22
8	D8	61	62	63	64	65	66	67	68	69	AB	BB	F0	FD	FE	B1
9	B0	6A	6B	6C	6D	6E	6F	70	71	72	AA	BA	E6	B8	C6	A4
A	B5	7E	73	74	75	76	77	78	79	7A	A1	BF	D0	5B	DE	AE
B	AC	A3	A5	B7	A9	A7	B6	BC	BD	BE	DD	A8	AF	5D	B4	D7
C	7B	41	42	43	44	45	46	47	48	49	AD	F4	F6	F2	F3	F5
D	7D	4A	4B	4C	4D	4E	4F	50	51	52	B9	FB	FC	F9	FA	FF
E	5C	F7	53	54	55	56	57	58	59	5A	B2	D4	D6	D2	D3	D5
F	30	31	32	33	34	35	36	37	38	39	B4	DB	DC	D9	DA	9F

Figure 171. EZACIC14 EBCDIC-to-ASCII table

Figure 172 shows an example of EZACIC14 call instructions.

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

PROCEDURE DIVISION.
    CALL 'EZACIC14' USING OUT-BUFFER LENGTH.
```

Figure 172. EZACIC14 call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 226.

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 is used to translate ASCII data to EBCDIC data.

Figure 173 shows an example of how EZACIC15 translates a byte of ASCII data.

EBCDIC output by EZACIC15		second hex digit of byte of ASCII data															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
first hex digit of byte of ASCII data	0	00	01	02	03	37	2D	2E	2F	16	05	25	0B	0C	0D	0E	0F
	1	10	11	12	13	3C	3D	32	26	18	19	3F	27	1C	1D	1E	1F
	2	40	5A	7F	7B	5B	6C	50	7D	4D	5D	5C	4E	6B	60	4B	61
	3	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	7A	5E	4C	7E	6E	6F
	4	7C	C1	C2	C3	C4	C5	C6	C7	C8	C9	D1	D2	D3	D4	D5	D6
	5	D7	D8	D9	E2	E3	E4	E5	E6	E7	E8	E9	AD	E0	BD	5F	6D
	6	79	81	82	83	84	85	86	87	88	89	91	92	93	94	95	96
	7	97	98	99	A2	A3	A4	A5	A6	A7	A8	A9	C0	4F	D0	A1	07
	8	20	21	22	23	24	15	06	17	28	29	2A	2B	2C	09	0A	1B
	9	30	31	1A	33	34	35	36	08	38	39	3A	3B	04	14	3E	FF
	A	41	AA	4A	B1	9F	B2	6A	B5	BB	B4	9A	8A	B0	CA	AF	BC
	B	90	8F	EA	FA	BE	A0	B6	B3	9D	DA	9B	8B	B7	B8	B9	A9
	C	64	65	62	66	63	67	9E	68	74	71	72	73	78	75	76	77
	D	AC	69	ED	EE	EB	EF	EC	BF	80	FD	FE	FB	FC	BA	AE	59
	E	44	45	42	46	43	47	9C	48	54	51	52	53	58	55	56	57
	F	8C	49	CD	CE	CB	CF	CC	E1	70	DD	DE	DB	DC	8D	8E	DF

Figure 173. EZACIC15 ASCII-to-EBCDIC table

Figure 174 shows an example of EZACIC15 call instructions.

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

PROCEDURE DIVISION.
    CALL 'EZACIC15' USING OUT-BUFFER LENGTH.
```

Figure 174. EZACIC15 call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 226.

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.

Appendix A. Original COBOL application programming interface (EZACICAL)

The EZACICAL does not formally support IPv6 and it is not a recommended API.

This topic describes the first COBOL API provided with TCP/IP Version 2.2.1 for MVS. It is referred to as the EZACICAL API to distinguish it from the Sockets Extended API. (EZACICAL is the routine that is called for this API.)

It gives the format of each socket call and describes the call parameters. It starts with guidance on compiling COBOL programs.

Using the EZACICAL or Sockets Extended API

The EZACICAL API (described in this topic) and the Sockets Extended API (described in Chapter 8) both provide sockets APIs for COBOL, PL/I, and Assembler language programs.

The Sockets Extended API is recommended because it has a simpler set of parameters for each call.

You might want to use the EZACICAL API if you have existing TCP/IP Version 2.2.1. for MVS COBOL/assembler language programs that require maintenance or modification.

COBOL compilation

The procedure that you use to compile a (non-CICS TCP/IP) source VS COBOL II CICS program can be used for CICS TCP/IP programs, but it needs some modification.

The modified JCL procedure is shown in Figure 175 on page 368. The procedure contains 3 steps:

1. **TRN** translates the COBOL program
2. **COB** compiles the translated COBOL program
3. **LKED** link-edits the final module to a LOADLIB

```
//CICSRS2C JOB (999,POK),'CICSRS2',NOTIFY=CICSRS2,
//    CLASS=A,MSGCLASS=T,TIME=1439,
//    REGION=5000K,MSGLEVEL=(1,1)
//DFHEITVL PROC SUFFIX=1$,
//       INDEX='CICS410',
//       INDEX2='CICS410',
//       OUTC=*,
//       REG=2048K,
//       LNKPARM='LIST,XREF',
//       WORK=SYSDA
//TRN    EXEC PGM=DFHECP&SUFFIX,
//            PARM='COBOL2',
//            REGION=&REG
//STEPLIB  DD DSN=&INDEX2..SDFHLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=&OUTC
//SYSPUNCH DD DSN=&&SYSCIN,
//            DISP=(,PASS),UNIT=&WORK,
//            DCB=BLKSIZE=400,
//            SPACE=(400,(400,100))
//*
//COB    EXEC PGM=IGYCRCTL,REGION=&REG,
//       PARM='NODYNAM,LIB,OBJECT,RENT,RES,APOST,MAP,XREF'
//STEPLIB  DD DSN=COBOL.V1R3M2.COB2COMP,DISP=SHR
//SYSLIB   DD DSN=&INDEX..SDFHCOB,DISP=SHR
//         DD DSN=&INDEX..SDFHMAC,DISP=SHR
//         DD DSN=CICSRS2.MAPA.DATA,DISP=SHR
//SYSPRINT DD SYSOUT=&OUTC
//SYSIN    DD DSN=&&SYSCIN,DISP=(OLD,DELETE)
//SYSLIN   DD DSN=&&LOADSET,DISP=(MOD,PASS),
//            UNIT=&WORK,SPACE=(80,(250,100))
//SYSUT1   DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT2   DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT3   DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT4   DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT5   DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT6   DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT7   DD UNIT=&WORK,SPACE=(460,(350,100))
//*                                                                     X
//*
//LKED   EXEC PGM=IEWL,REGION=&REG,
//            PARM='&LNKPARM',COND=(5,LT,COB)
//SYSLIB   DD DSN=&INDEX2..SDFHLOAD,DISP=SHR
//         DD DSN=SYS1.COBOL.V1R3M2.COB2CICS,DISP=SHR
//         DD DSN=COBOL.V1R3M2.COB2LIB,DISP=SHR
//         DD DSN=hlq.SEZATCP,DISP=SHR
//SYSLMOD  DD DSN=CICSRS2.CICS410.PGMLIB,DISP=SHR
//SYSUT1   DD UNIT=&WORK,DCB=BLKSIZE=1024,
//            SPACE=(1024,(200,20))
//SYSPRINT DD SYSOUT=&OUTC
//*                                                                      X
//SYSLIN   DD DSN=&&LOADSET,DISP=(OLD,DELETE)
//         DD DDNAME=SYSIN
//   PEND
//APPLPROG EXEC DFHEITVL
//TRN.SYSIN  DD DISP=SHR,DSN=CICSRS2.JCL.DATA(SISSRR1C)
//LKED.SYSIN DD *
   INCLUDE SYSLIB(EZACICAL)
   NAME SISSRR1C(R)
/*
```

Figure 175. Modified JCL for COBOL compilation

The EZACICAL API

The EZACICAL API can be used by assembler language, COBOL, or PL/I programs and is invoked by calling the EZACICAL routine. Although the calls to this routine perform the same function as the C language calls described in Chapter 7, the parameters are presented differently because of the differences in the languages. The equivalent to the return code provided by all C function calls is found in a decimal value parameter included as the last parameter variable.

COBOL

The following is the 'EZACICAL' call format for COBOL:

```
►►──CALL 'EZACICAL' USING TOKEN COMMAND──parm1, parm2, ...──ERRNO RETCODE.──────────────────────►◄
```

TOKEN
A 16-character field with the value 'TCPIPIUCVSTREAMS'

COMMAND
A binary halfword of value from 1 to 32, identifying the socket call.

parm*n* The parameters particular to each socket call. For example, BIND, described in "BIND" on page 371, has two such parameters: S (socket), which is a halfword binary, and NAME, which is a structure specifying a port name.

ERRNO
There is an error number in this field if the RETCODE is negative. This field is used in most, but not all, of the calls. It corresponds to the global errno variable in C.

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

PL/I

The following is the 'EZACICAL' call format for PL/I:

```
►►──CALL EZACICAL (TOKEN COMMAND──parm1, parm2, ...──ERRNO RETCODE);──────────────────────────►◄
```

TOKEN
A 16-character field with the value 'TCPIPIUCVSTREAMS'

COMMAND
A binary halfword of value from 1 to 32, identifying the socket call.

parm*n* The parameters particular to each socket call. For example, BIND, described in "BIND" on page 371, has two such parameters: S (socket), which is a halfword binary, and NAME, which is a structure specifying a port name.

ERRNO
There is an error number in this field if the RETCODE is negative. This field is used in most, but not all, of the calls. It corresponds to the global errno variable in C.

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

Assembler language

The following is the EZACICAL call format for assembler language:

```
►►──CALL EZACICAL,(TOKEN,COMMAND,──parm1, parm2, ...──ERRNO RETCODE),VL──────────────────────────►◄
```

The parameter descriptions in this topic are written using the COBOL language syntax and conventions. For assembler language, use the following conversions:

```
COBOL PIC

  PIC S9(4) COMP                      HALFWORD BINARY VALUE
  PIC S9(8) COMP                      FULLWORD BINARY VALUE
  PIC   X(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
```

COBOL and assembler language socket calls

The remainder of this topic describes the EZACICAL API call formats.

The descriptions assume you are using VS COBOL II. If you are using an earlier version, the picture clauses should read COMP rather than BINARY.

The following abbreviations are used:

H Halfword

F Fullword

D Doubleword

CL*n* Character format, length *n* bytes

XL*n* Hexadecimal format, length *n* bytes

ACCEPT

This call functions in the same way as the equivalent call described "ACCEPT" on page 226. The format of the COBOL call for ACCEPT is:

```
CALL 'EZACICAL' USING TOKEN COMMAND S ZERO-FWRD NEW-S NAME ERRNO RETCODE.
```

In assembler language, issue the macro call `CALL EZACICAL`, using standard assembler call syntax (for the call format, see "Assembler language").

Parameter lengths in assembler language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY
S	H	PIC 9(4) BINARY
ZERO-FWRD	F	PIC 9(8) BINARY
NEW-S	F	PIC S9(8) BINARY
NAME STRUCTURE:		

Internet Family	H	PIC 9(4) BINARY
Port	H	PIC 9(4) BINARY
Internet Address	F	PIC 9(8) BINARY
Zeros	XL8	PIC X(8)
ERRNO	F	PIC 9(8) BINARY
RETCODE	F	PIC S9(8) BINARY

Parameter values to be set by the application

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
Must be set to 1 for the ACCEPT command

S The descriptor of the local socket on which the connection is accepted

ZERO-FWRD
Set to zeros

NEW-S
Set to −1. The system returns the socket number in the RETCODE field.

Note: Be sure to use **only** the socket number returned by the system.

Parameter values returned to the application

NAME
Structure giving the name of the port to which the new socket is connected

Internet Family
`AF-INET` is always returned

Port The port address of the new socket

Internet Address
The IP address of the new socket

Zeros Set to binary zeros or LOW VALUES

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 397.

RETCODE
The socket number for new socket is returned. A RETCODE of −1 indicates an error.

BIND

This call functions in the same way as the equivalent call described in “BIND” on page 229. The format of the COBOL call for the BIND function is:

```
CALL 'EZACICAL' USING TOKEN COMMAND S NAME ERRNO RETCODE.
```

In assembler language, issue the macro call `CALL EZACICAL`, using standard assembler call syntax (for the call format, see “Assembler language” on page 370).

Parameter lengths in assembler language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY

S	H	PIC 9(4) BINARY
NAME STRUCTURE:		
Internet Family	H	PIC 9(4) BINARY
Port	H	PIC 9(4) BINARY
Internet Address	F	PIC 9(8) BINARY
Zeros	XL8	PIC X(8)
ERRNO	F	PIC 9(8) BINARY
RETCODE	F	PIC S9(8) BINARY

Parameter values to be set by the application

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
Must be set to 2 for the BIND command

S The descriptor of the local socket to be bound

NAME
Structure giving the name of the port to which the socket is to be bound, consisting of:

Internet Family
Must be set to 2 (AF-INET)

Port The local port address to which the socket is to be bound

Internet Address
The local IP address to which the socket is to be bound

Zeros Set to binary zeros or low values

Parameter values returned to the application

NAME (*Port***)**
If *Port* was set to 0, the system returns an available port.

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 397.

RETCODE
A return of 0 indicates a successful call. A return of −1 indicates an error.

CLOSE

This call functions in the same way as the equivalent call described in “CLOSE” on page 232. The format of the COBOL call for the CLOSE function is:

```
CALL 'EZACICAL' USING TOKEN COMMAND S DZERO ERRNO RETCODE.
```

In assembler language, issue the macro call `CALL EZACICAL`, using standard assembler call syntax (for the call format, see “Assembler language” on page 370).

Parameter lengths in assembler language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY
S	H	PIC 9(4) BINARY
DZERO	D	PIC X(8)
ERRNO	F	PIC S9(8) BINARY

RETCODE	F	PIC S9(8) BINARY

Parameter values to be set by the application

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
Must be set to 3 for the CLOSE command

S The descriptor of the socket to be closed

DZERO
Set to binary zeros or low values

Parameter values returned to the application

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, "Return codes," on page 397.

RETCODE
A return of 0 indicates a successful call. A return of −1 indicates an error.

CONNECT

This call functions in the same way as the equivalent call described in "CONNECT" on page 233. The format of the COBOL call for the CONNECT function is:

```
CALL 'EZACICAL' USING TOKEN COMMAND S NAME ERRNO RETCODE.
```

In assembler language, issue the macro call `CALL EZACICAL`, using standard assembler call syntax (for the call format, see "Assembler language" on page 370).

Parameter lengths in assembler language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY
S	H	PIC 9(4) BINARY
NAME STRUCTURE:		
Internet Family	H	PIC 9(4) BINARY
Port	H	PIC 9(4) BINARY
Internet Address	F	PIC 9(8) BINARY
Zeros	XL8	PIC X(8)
ERRNO	F	PIC 9(8) BINARY
RETCODE	F	PIC S9(8) BINARY

Parameter values to be set by the application

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
Must be set to 4 for the CONNECT command

S The descriptor of the local socket to be used to establish a connection

NAME
Structure giving the name of the port to which the socket is to be connected, consisting of:

Internet Family
: Must be set to 2 (AF-INET)

Port The remote port number to which the socket is to be connected

Internet Address
: The remote IP address to which the socket is to be connected

Zeros Set to binary zeros or low values

Parameter values returned to the application

ERRNO
: If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 397.

RETCODE
: A return of 0 indicates a successful call. A return of −1 indicates an error.

FCNTL

This call functions in the same way as the equivalent call described in “FCNTL” on page 236. The format of the COBOL call for the FCNTL function is:

```
CALL 'EZACICAL' USING TOKEN COMMAND S CMD ARG ERRNO RETCODE.
```

In assembler language, issue the macro call `CALL EZACICAL`, using standard assembler call syntax (for the call format, see “Assembler language” on page 370).

Parameter lengths in assembler language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY
S	H	PIC 9(4) BINARY
CMD	F	PIC 9(8) BINARY
ARG	F	PIC 9(8)
ERRNO	F	PIC 9(8) BINARY
RETCODE	F	PIC S9(8) BINARY

Parameter values to be set by the application

TOKEN
: Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
: Must be set to 5 for the FCNTL command

S The socket descriptor whose FNDELAY flag is to be set or queried

CMD Set a value of 3 to query the FNDELAY flag of socket `s`. This is equivalent to setting the *cmd* parameter to F-GETFL in the fcntl() C call.

Set a value of 4 to set the FNDELAY flag of socket `s`. This is equivalent to setting the *cmd* parameter to F-SETFL in the fcntl() C call.

ARG If CMD is set to 4, setting ARG to 4 sets the FNDELAY flag; setting ARG to 3 resets the FNDELAY flag.

Parameter values returned to the application

ERRNO
: If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 397.

RETCODE
: If CMD was set to 3, a bit mask is returned. If CMD was set to 4, a successful call is indicated by 0 in this field. In both cases, a RETCODE of −1 indicates an error.

GETCLIENTID

This call functions in the same way as the equivalent call described in “GETCLIENTID” on page 247. The format of the COBOL call for the GETCLIENTID function is:

```
CALL 'EZACICAL' USING TOKEN COMMAND HZERO DZERO CLIENTID ERRNO RETCODE.
```

In assembler language, issue the macro call `CALL EZACICAL`, using standard assembler call syntax (for the call format, see “Assembler language” on page 370).

Parameter lengths in assembler language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY
HZERO	H	PIC 9(4) BINARY
DZERO	D	PIC X(8)
CLIENTID STRUCTURE:		
Domain	F	PIC 9(8) BINARY
Name	CL8	PIC X(8)
Task	CL8	PIC X(8)
Reserved	XL20	PIC X(20)
ERRNO	F	PIC 9(8) BINARY
RETCODE	F	PIC S9(8) BINARY

Parameter values to be set by the application

TOKEN
: Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
: Must be set to `30` for the GETCLIENTID command

HZERO
: Set to binary zeros or LOW VALUES

DZERO
: Set to binary zeros or LOW VALUES

CLIENTID

Domain
: Must be set to 2 (AF-INET)

Parameter values returned to the application

CLIENTID
: Structure identifying the client as follows:

Name Address space identification is returned

Task Task identification is returned

Reserved
: Zeros or LOW VALUES are returned

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 397.

RETCODE
A return of 0 indicates a successful call. A return of −1 indicates an error.

GETHOSTID

This call functions in the same way as the equivalent call described in “GETHOSTBYADDR” on page 248. The format of the COBOL call for the GETHOSTID function is:

```
CALL 'EZACICAL' USING TOKEN COMMAND HZERO DZERO ERRNO RETCODE.
```

In assembler language, issue the macro call `CALL EZACICAL`, using standard assembler call syntax (for the call format, see “Assembler language” on page 370).

Parameter lengths in assembler language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY
HZERO	H	PIC 9(4) BINARY
DZERO	D	PIC X(8)
ERRNO	F	PIC 9(8) BINARY
RETCODE	F	PIC S9(8) BINARY

Parameter values to be set by the application

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
Must be set to 7 for the GETHOSTID command

HZERO
Set to binary zeros or low values

DZERO
Set to binary zeros or low values

Parameter values returned to the application

ERRNO
This field is not used

RETCODE
Returns a fullword binary field containing the 32-bit Internet address of the host. A value of -1 is a call failure, probably indicating that an INITAPI call has not been issued. There is no ERRNO parameter for this call.

GETHOSTNAME

This call functions in the same way as the equivalent call described in “GETHOSTBYNAME” on page 250.

Result: The host name returned is the host name the TCPIP stack learned at startup from the TCPIP.DATA file.

The format of the COBOL call for the GETHOSTNAME function is:

```
CALL 'EZACICAL' USING TOKEN COMMAND HZERO DZERO NAMELEN NAME ERRNO RETCODE.
```

In assembler language, issue the macro call `CALL EZACICAL`, using standard assembler call syntax (for the call format, see “Assembler language” on page 370).

Parameter lengths in assembler language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY
HZERO	H	PIC 9(4) BINARY
DZERO	D	PIC X(8)
NAMELEN	F	PIC 9(8) BINARY
NAME	NAMELEN or larger	NAMELEN or larger
ERRNO	F	PIC 9(8) BINARY
RETCODE	F	PIC S9(8) BINARY

Parameter values to be set by the application

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
Must be set to 8 for the GETHOSTNAME command

HZERO
Set to 0

DZERO
Set to binary zeros or low values

NAMELEN
The length of the NAME field. The minimum length of the NAME field is 1 character. The maximum length of the NAME field is 255 characters.

Parameter values returned to the application

NAME
The host name returned from the call. If the host name is shorter than the NAMELEN value, then the NAME field is filled with binary zeros after the host name. If the host name is longer than the NAMELEN value, then the name is truncated.

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 397.

RETCODE
A return of 0 indicates a successful call. A return of −1 indicates an error.

GETPEERNAME

This call functions in the same way as the equivalent call described in “GETPEERNAME” on page 258. The format of the COBOL call for the GETPEERNAME function is:

```
CALL 'EZACICAL' USING TOKEN COMMAND S DZERO NAME ERRNO RETCODE.
```

In assembler language, issue the macro call `CALL EZACICAL`, using standard assembler call syntax (for the call format, see “Assembler language” on page 370).

Parameter lengths in assembler language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY
S	H	PIC 9(4) BINARY
DZERO	D	PIC X(8)
NAME	CL16	PIC X(16)
ERRNO	F	PIC 9(8) BINARY
RETCODE	F	PIC S9(8) BINARY

Parameter values to be set by the application

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
Must be set to 9 for the GETPEERNAME command

S The descriptor of the local socket connected to the requested peer

DZERO
Set to binary zeros or low values

Parameter values returned to the application

NAME
The peer name returned from the call

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, "Return codes," on page 397.

RETCODE
A return of 0 indicates a successful call. A return of −1 indicates an error.

GETSOCKNAME

This call functions in the same way as the equivalent call described in "GETSOCKNAME" on page 260. The format of the COBOL call for the GETSOCKNAME function is:

```
CALL 'EZACICAL' USING TOKEN COMMAND S DZERO NAME ERRNO RETCODE.
```

In assembler language, issue the macro call `CALL EZACICAL`, using standard assembler call syntax (for the call format, see "Assembler language" on page 370).

Parameter lengths in assembler language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY
S	H	PIC 9(4) BINARY
DZERO	D	PIC X(8)
NAME STRUCTURE:		
Internet Family	H	PIC 9(4) BINARY
Port	H	PIC 9(4) BINARY
Internet Address	F	PIC 9(8) BINARY
Zeros	XL8	PIC X(8)
ERRNO	F	PIC 9(8) BINARY
RETCODE	F	PIC S9(8) BINARY

Parameter values to be set by the application

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
Must be set to `10` for the GETSOCKNAME command

S The descriptor of the local socket whose address is required

DZERO
Set to binary zeros or low values

NAME
Structure giving the name of the port to which the socket is bound, consisting of:

Internet Family
Must be set to 2 (AF-INET).

Port The local port address to which the socket is bound

Internet Address
The local IP address to which the socket is bound

Zeros Set to binary zeros or low values

Parameter values returned to the application

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 397.

RETCODE
A return of 0 indicates a successful call. A return of –1 indicates an error.

GETSOCKOPT

This call functions in the same way as the equivalent call described in “GETSOCKOPT” on page 262. The format of the COBOL call for the GETSOCKOPT function is:

```
CALL 'EZACICAL'
     USING TOKEN COMMAND S LEVEL OPTNAME OPTLEN OPTVAL ERRNO RETCODE.
```

In assembler language, issue the macro call `CALL EZACICAL`, using standard assembler call syntax (for the call format, see “Assembler language” on page 370).

Parameter lengths in assembler language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY
S	H	PIC 9(4) BINARY
LEVEL	F	PIC X(4)
OPTNAME	F	PIC X(4)
OPTLEN	F	PIC 9(8) BINARY
OPTVAL	CL4	PIC X(4)
ERRNO	F	PIC 9(8) BINARY
RETCODE	F	PIC S9(8) BINARY

Parameter values to be set by the application

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
Must be set to 11 for the GETSOCKOPT command

S The descriptor of the socket whose option settings are required

LEVEL
This must be set to X'0000FFFF'.

OPTNAME
Set this field to specify the option to be queried, as shown below. For a description of these options, see "GETSOCKOPT" on page 262

Value	Meaning
X'00000004'	SO-REUSEADDR
X'00000020'	SO-BROADCAST
X'00001007'	SO-ERROR
X'00000080'	SO-LINGER
X'00000100'	SO-OOBINLINE
X'00001001'	SO-SNDBUF
X'00001008'	SO-TYPE
X'80000008'	TCP_KEEPALIVE
X'80000001'	TCP_NODELAY

Parameter values returned to the application

OPTLEN
The length of the option data

OPTVAL
The value of the option. For all options except SO-LINGER, an integer indicates that the option is enabled, while a 0 indicates it is disabled. For SO-LINGER, the following structure is returned:

```
ONOFF    F    PIC X(4)
LINGER   F    PIC 9(4)
```

A nonzero value of ONOFF indicates that the option is enabled, and 0, that it is disabled. The LINGER value indicates the amount of time to linger after close.

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, "Return codes," on page 397.

RETCODE
A return of 0 indicates a successful call. A return of –1 indicates an error.

GIVESOCKET

This call functions in the same way as the equivalent call described in "GIVESOCKET" on page 274. The format of the COBOL call for the GIVESOCKET function is:

```
CALL 'EZACICAL' USING TOKEN COMMAND S CLIENTID ERRNO RETCODE.
```

In assembler language, issue the macro call `CALL EZACICAL`, using standard assembler call syntax (for the call format, see "Assembler language" on page 370).

Parameter lengths in assembler language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY
S	H	PIC 9(4) BINARY
CLIENTID STRUCTURE:		
Domain	F	PIC 9(8) BINARY
Name	CL8	PIC X(8)
Task	CL8	PIC X(8)
Reserved	XL20	PIC X(20)
ERRNO	F	PIC 9(8) BINARY
RETCODE	F	PIC S9(8) BINARY

Parameter values to be set by the application

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
Must be set to 31 for the GIVESOCKET command

S The socket descriptor of the socket to be given

CLIENTID
Structure identifying the client ID of this application, as follows:

Domain
Must be set to 2 (AF-INET)

Name Set to the address space identifier obtained from GETCLIENTID

Task Set to blanks

Reserved
Set to binary zeros or low values

Parameter values returned to the application

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 397.

RETCODE
A return of 0 indicates a successful call. A return of –1 indicates an error.

INITAPI

The format of the COBOL call for the INITAPI function is:

```
CALL 'EZACICAL'
    USING TOKEN COMMAND FZERO MAX-SOCK API SUBTASK FZERO ERRNO RETCODE.
```

In assembler language, issue the macro call `CALL EZACICAL`, using standard assembler call syntax (for the call format, see “Assembler language” on page 370).

Parameter lengths in assembler language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY
MAX-SOCK	H	PIC 9(4) BINARY
API	H	PIC 9(4) BINARY
SUBTASK	XL8	PIC X(8)

FZERO	F	PIC 9(8) BINARY
ERRNO	F	PIC 9(8) BINARY
RETCODE	F	PIC S9(8) BINARY

Parameter values to be set by the application

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
Must be set to `0` for the INITAPI command

MAX-SOCK
The maximum number of sockets to be supported in this application. This value cannot exceed 65535. The minimum value is 50.

API Must be set to `2`, indicating use of the sockets API

SUBTASK
A unique subtask identifier. It should consist of the 7-character CICS task number and any printable character.

Note: Using the letter L as the last character in the subtask parameter causes the tasking mechanism to assume the CICS transaction is a Listener and schedule it using a non-reusable subtask by way of MVS attach processing when OTE=NO. This has no effect when OTE=YES.

FZERO
Zeros

Parameter values returned to the application

ERRNO
If RETCODE=0, contains the highest socket number available to this program.

RETCODE
A return of 0 indicates a successful call. A return of −1 indicates an error.

IOCTL

This call functions in the same way as the equivalent call described in “IOCTL” on page 278. The format of the COBOL call for the IOCTL function is:

```
CALL 'EZACICAL'
    USING TOKEN COMMAND S IOCTLCMD REQARG RETARG ERRNO RETCODE.
```

In assembler language, issue the macro call `CALL EZACICAL`, using standard assembler call syntax (for the call format, see “Assembler language” on page 370).

Parameter lengths in assembler language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY
S	H	PIC 9(4) BINARY
IOCTLCMD	F	PIC 9(8)
REQARG	var	var
RETARG	var	var
ERRNO	F	PIC S9(8) BINARY

RETCODE	F	PIC S9(8) BINARY

Parameter values to be set by the application

TOKEN
Must be set to ’TCPIPIUCVSTREAMS’

COMMAND
Must be set to 12 for the IOCTL command

S The descriptor of the socket to be controlled

IOCTLCMD
Set to the command value to be passed to IOCTL. See “IOCTL” on page 278 for values and descriptions.

REQARG
The request argument associated with the command. See “IOCTL” on page 278 for a list and description of possible argument values.

Parameter values returned to the application

RETARG
The return argument. See “IOCTL” on page 278 for a description of the return argument for each command.

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 397.

RETCODE
A return value of 0 indicates a successful call. A return value of −1 indicates an error.

LISTEN

This call functions in the same way as the equivalent call described in “LISTEN” on page 289. The format of the COBOL call for the LISTEN function is:

```
CALL 'EZACICAL' USING TOKEN COMMAND S FZERO BACKLOG ERRNO RETCODE.
```

In assembler language, issue the macro call `CALL EZACICAL`, using standard assembler call syntax (for the call format, see “Assembler language” on page 370).

Parameter lengths in assembler language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY
S	H	PIC 9(4) BINARY
FZERO	F	PIC 9(8) BINARY
BACKLOG	F	PIC 9(8) BINARY
ERRNO	F	PIC 9(8) BINARY
RETCODE	F	PIC S9(8) BINARY

Parameter values to be set by the application

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
Must be set to 13 for the LISTEN command

S The descriptor of the socket that is going to listen for incoming connection requests

FZERO
Set to binary zeros or low values

BACKLOG
Set to the number of connection requests to be queued.

Note: The BACKLOG value specified on the LISTEN command cannot be greater than 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. If you want a larger backlog, update the SOMAXCONN statement. See *z/OS Communications Server: IP Configuration Reference* for details.

Parameter values returned to the application

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, "Return codes," on page 397.

RETCODE
A return value of 0 indicates a successful call. A return value of −1 indicates an error.

READ

This call functions in the same way as the equivalent call described in "READ" on page 294. The format of the COBOL call for the READ function is:

```
CALL 'EZACICAL'
    USING TOKEN COMMAND S DZERO NBYTE FILLER BUF ERRNO RETCODE.
```

In assembler language, issue the macro call `CALL EZACICAL`, using standard assembler call syntax (for the call format, see"Assembler language" on page 370).

Parameter lengths in assembler language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY
S	H	PIC 9(4) BINARY
DZERO	D	PIC X(8)
NBYTE	F	PIC 9(8) BINARY
FILLER	CL16	PIC X(16)
BUF	NBYTE or larger	NBYTE or larger
ERRNO	F	PIC 9(8) BINARY
RETCODE	F	PIC S9(8) BINARY

Parameter values to be set by the application

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
Must be set to 14 for the READ command

S The descriptor of the socket that is going to read data

DZERO
Set to binary zeros or low values

NBYTE
Set to the length of the buffer (maximum 32 767 bytes)

Parameter values returned to the application

FILLER
Your program should ignore this field.

BUF The input buffer.

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 397.

RETCODE
A positive value indicates the number of bytes copied into the buffer. A value of 0 indicates that the socket is closed. A value of −1 indicates an error.

See “EZACIC05” on page 352 for a subroutine that translates ASCII data to EBCDIC.

RECVFROM

This call functions in the same way as the equivalent call described in “RECV” on page 297. The format of the COBOL call for the RECVFROM function is:

```
CALL 'EZACICAL'
    USING TOKEN COMMAND S FZERO FLAGS NBYTE FROM BUF ERRNO RETCODE.
```

In assembler language, issue the macro call `CALL EZACICAL`, using standard assembler call syntax (for the call format, see “Assembler language” on page 370).

Parameter lengths in assembler language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY
S	H	PIC 9(4) BINARY
FZERO	F	PIC 9(8) BINARY
FLAGS	F	PIC 9(8) BINARY
NBYTE	F	PIC 9(8) BINARY
FROM	CL16	PIC X(16)
BUF	NBYTE or larger	NBYTE or larger
ERRNO	F	PIC 9(8) BINARY
RETCODE	F	PIC S9(8) BINARY

Parameter values to be set by the application

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
Must be set to `16` for the RECVFROM command

S The descriptor of the socket receiving data

FZERO
Set to binary zeros or low values

FLAGS
: Set to 2 to peek at (read) data, but not destroy it, so that any subsequent RECVFROM calls reads the same data. CICS TCP/IP does not support out-of-band data.

NBYTE
: Set to the length of the input buffer. This length cannot exceed 32 768 bytes.

Parameter values returned to the application

FROM
: The socket address structure identifying the `from` address of the data.

BUF
: The input buffer.

ERRNO
: If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, "Return codes," on page 397.

RETCODE
: A positive value indicates the number of bytes copied into the buffer. A value of 0 indicates that the socket is closed. A value of −1 indicates an error.

See "EZACIC05" on page 352 for a subroutine that translates ASCII data to EBCDIC.

SELECT

This call functions in the same way as the equivalent call describedin "SELECT" on page 307. The format of the COBOL call for the SELECT function is:

```
CALL 'EZACICAL' USING TOKEN COMMAND LOM NUM-FDS
TIME-SW RD-SW WR-SW EX-SW
TIMEOUT RD-MASK WR-MASK EX-MASK
DZERO R-R-MASK R-W-MASK R-E-MASK
ERRNO RETCODE.
```

In assembler language, issue the macro call `CALL EZACICAL`, using standard assembler call syntax (for the call format, see "Assembler language" on page 370).

Parameter lengths in assembler language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY
LOM	H	PIC 9(4) BINARY
NUM-FDS	F	PIC 9(8) BINARY
TIME-SW	F	PIC 9(8) BINARY
RD-SW	F	PIC 9(8) BINARY
WR-SW	F	PIC 9(8) BINARY
EX-SW	F	PIC 9(8) BINARY
TIMEOUT STRUCTURE:		
Seconds	F	PIC 9(8) BINARY
Milliseconds	F	PIC 9(8) BINARY
RD-MASK	Length Of Mask*	Length Of Mask*
WR-MASK	Length of Mask*	Length of Mask*
EX-MASK	Length of Mask*	Length of Mask*
DZERO	D	PIC X(8)
R-R-MASK	Length of Mask*	Length of Mask*

R-W-MASK	Length of Mask*	Length of Mask*
R-E-MASK	Length of Mask*	Length of Mask*
ERRNO	F	PIC 9(8) BINARY
RETCODE	F	PIC S9(8) BINARY

***How to calculate Length of Mask (LOM):**

1. LOM = ((NUM-FDS + 31)/32) * 4, using integer arithmetic.
2. So, for NUM-FDS ≤ 32, LOM = 4 bytes.
3. For 33 ≤ NUM-FDS ≤ 64, LOM = 8 bytes, and so on.

Parameter values to be set by the application

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
Must be set to 19 for the SELECT command

LOM Set to the length of mask. The calculation method is given above.

NUM-FDS
The number of socket descriptors to check. For efficiency, it should be set to the largest number of socket descriptors plus 1.

TIME-SW
Set to 0 to specify a wait forever on socket descriptor activity. Set to 1 to specify a timeout value; this blocks the call until the timeout value is exceeded or until there is socket activity.

RD-SW
Set either 0 (do not check for read interrupts) or 1 (check for read interrupts).

WR-SW
Set either 0 (do not check for write interrupts) or 1 (check for write interrupts).

EX-SW
Set either 0 (do not check for exception interrupts) or 1 (check for exception interrupts).

TIMEOUT
Use this structure to set the timeout value if no activity is detected. Setting this structure to (0,0) indicates that SELECT should act as a polling function; that is, as nonblocking.

Seconds
Set to the seconds component of the timeout value.

Milliseconds
Set to the milliseconds component of the timeout value (in the range 0 through 999).

RD-MASK
Set the bit mask array for reads. See *z/OS Communications Server: IP Programmer's Guide and Reference* for more information.

WR-MASK
Set the bit mask array for writes. See *z/OS Communications Server: IP Programmer's Guide and Reference* for more information.

EX-MASK
Set the bit mask array for exceptions. See *z/OS Communications Server: IP Programmer's Guide and Reference* for more information.

DZERO
Set to binary zeros or low values.

Parameter values returned to the application

R-R-MASK
Returned bit mask array for reads. See *z/OS Communications Server: IP Programmer's Guide and Reference* for more information.

R-W-MASK
Returned bit mask array for writes. See *z/OS Communications Server: IP Programmer's Guide and Reference* for more information.

R-E-MASK
Returned bit mask array for exceptions. See *z/OS Communications Server: IP Programmer's Guide and Reference* for more information.

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 397.

RETCODE
A positive value indicates the total number of ready sockets in all bit masks. A value of 0 indicates an expired time limit. A value of −1 indicates an error.

SEND

This call functions in the same way as the equivalent call described in “SEND” on page 317. The format of the COBOL call for the SEND function is:

```
CALL 'EZACICAL' USING TOKEN COMMAND S NBYTE FLAGS DZERO BUF ERRNO RETCODE.
```

In assembler language, issue the macro call `CALL EZACICAL`, using standard assembler call syntax (for the call format, see “Assembler language” on page 370).

Parameter lengths in assembler language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY
S	H	PIC 9(4) BINARY
NBYTE	F	PIC 9(8) BINARY
FLAGS	F	PIC 9(8) BINARY
DZERO	D	PIC X(8)
BUF	NBYTE or larger	NBYTE or larger
ERRNO	F	PIC 9(8) BINARY
RETCODE	F	PIC S9(8) BINARY

Parameter values to be set by the application

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
Must be set to `20` for the SEND command

S The descriptor of the socket sending the data

NBYTE
Set to the number of bytes to be transmitted (maximum 32 768 bytes)

FLAGS
Set to 0 (no flags) or 4 (do not route, routing is provided). CICS TCP/IP does not support out-of-band data.

DZERO
Set to binary zeros or low values

BUF Buffer from which data is transmitted

Parameter values returned to the application

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, "Return codes," on page 397.

RETCODE
A value of −1 indicates an error. Other values have no meaning.

See "EZACIC04" on page 350 for a subroutine that translates EBCDIC data to ASCII.

SENDTO

This call functions in the same way as the equivalent call described in "SENDTO" on page 323. The format of the COBOL call for the SENDTO function is:

```
CALL 'EZACICAL' USING TOKEN COMMAND S LEN FLAGS NAME BUF ERRNO RETCODE.
```

In assembler language, issue the macro call `CALL EZACICAL`, using standard assembler call syntax (for the call format, see "Assembler language" on page 370).

Parameter lengths in assembler language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY
S	H	PIC 9(4) BINARY
LEN	F	PIC 9(8) BINARY
FLAGS	F	PIC 9(8) BINARY
NAME STRUCTURE:		
in-family	H	PIC 9(4) BINARY
in-port	H	PIC 9(4) BINARY
in-address	F	PIC 9(8) BINARY
dzero	D	PIC X(8)
BUF	LEN or larger	LEN or larger
ERRNO	F	PIC 9(8) BINARY
RETCODE	F	PIC S9(8) BINARY

Parameter values to be set by the application

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
Must be set to 22 for the SENDTO command

S The descriptor of the socket sending the data

LEN The number of bytes to be transmitted (maximum 32 768 bytes)

FLAGS
Set to 0 (no flags) or 4 (do not route, routing is provided)

NAME
Structure specifying the address to which data is to be sent, as follows:

in-family
Must be set to 2 (AF-INET)

in-port Set to the port number for receiver

in-address
Set to the IP address for receiver

dzero Set to binary zeros or low values

BUF Set to the buffer from which data is transmitted

Parameter values returned to the application

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, "Return codes," on page 397.

RETCODE
A value of −1 indicates an error. Other values have no meaning.

See "EZACIC04" on page 350 for a subroutine that translates EBCDIC data to ASCII.

SETSOCKOPT

This call functions in the same way as the equivalent call described "GETSOCKOPT" on page 262. The format of the COBOL call for the SETSOCKOPT function is:

```
CALL 'EZACICAL'
    USING TOKEN COMMAND S LEN LEVEL OPTNAME OPTVAL ERRNO RETCODE.
```

In assembler language, issue the macro call `CALL EZACICAL`, using standard assembler call syntax (for the call format, see "Assembler language" on page 370).

Parameter lengths in assembler language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY
S	H	PIC 9(4) BINARY
LEN	F	PIC 9(8) BINARY
LEVEL	F	PIC X(4)
OPTNAME	F	PIC 9(8) BINARY
OPTVAL	CL4	PIC X(4)
ERRNO	F	PIC 9(8) BINARY
RETCODE	F	PIC S9(8) BINARY

Parameter values to be set by the application

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
Must be set to 23 for the SETSOCKOPT command

S The descriptor of the socket whose options are to be set

LEN Set to the length of OPTVAL

LEVEL
This must be set to X'0000FFFF'.

OPTNAME
Set this field to specify the option to be set, as shown below. See “SETSOCKOPT” on page 326 for a description of these settings.

Value	**Meaning**
X'00000020'	SO-BROADCAST
X'00000080'	SO-LINGER
X'00000100'	SO-OOBINLINE
X'00000004'	SO-REUSEADDR
X'80000008'	TCP_KEEPALIVE
X'80000001'	TCP_NODELAY

OPTVAL
For SO-BROADCAST, SO-OOBINLINE, and SO-REUSEADDR, set to a nonzero integer to enable the option specified in OPTNAME, and set to 0 to disable the option. For SO-LINGER, see the equivalent OPTVAL parameter in “SETSOCKOPT” on page 326.

Parameter values returned to the application

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 397.

RETCODE
A return value of 0 indicates a successful call. A return value of −1 indicates an error.

SHUTDOWN

This call functions in the same way as the equivalent call described in “SHUTDOWN” on page 338. The format of the COBOL call for the SHUTDOWN function is:

```
CALL 'EZACICAL' USING TOKEN COMMAND S FZERO HOW ERRNO RETCODE.
```

In assembler language, issue the macro call `CALL EZACICAL`, using standard assembler call syntax (for the call format, see “Assembler language” on page 370).

Parameter lengths in assembler language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY
S	H	PIC 9(4) BINARY
FZERO	F	PIC 9(8) BINARY
HOW	F	PIC 9(8) BINARY
ERRNO	F	PIC 9(8) BINARY
RETCODE	F	PIC S9(8) BINARY

Parameter values to be set by the application

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
: Must be set to `24` for the SHUTDOWN command

S The descriptor of the socket to be shut down

FZERO
: Set to zeros

HOW Set this to specify whether all or part of a connection is to be shut down, as follows:

Value	**Meaning**
0	Ends communication from the socket
1	Ends communication to the socket
2	Ends communication both to and from the socket

Parameter values returned to the application

ERRNO
: If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, "Return codes," on page 397.

RETCODE
: A return value of 0 indicates a successful call. A return value of –1 indicates an error.

SOCKET

This call functions in the same way as the equivalent call described in "SOCKET" on page 340. The format of the COBOL call for the SOCKET function is:

```
CALL 'EZACICAL'
    USING TOKEN COMMAND HZERO AF TYPE PROTOCOL SOCKNO ERRNO RETCODE.
```

In assembler language, issue the macro call `CALL EZACICAL`, using standard assembler call syntax (for call format, see "Assembler language" on page 370).

Parameter lengths in assembler language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY
HZERO	H	PIC 9(4) BINARY
AF	F	PIC 9(8) BINARY
TYPE	F	PIC 9(8) BINARY
PROTOCOL	F	PIC 9(8) BINARY
SOCKNO	F	PIC S9(8) BINARY
ERRNO	F	PIC 9(8) BINARY
RETCODE	F	PIC S9(8) BINARY

Parameter values to be set by the application

TOKEN
: Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
: Must be set to `25` for the SOCKET command

HZERO
: Set to binary zeros or low values

AF Must be set to 2 (AF-INET)

TYPE Set to 1 for TCP sockets; 2 for UDP sockets.

PROTOCOL
Set to `0`. (The system selects the appropriate protocol for the TYPE specified above.)

SOCKNO
Set to –1. The system returns the socket number in the RETCODE field.

Note: Use only the socket number returned by the system.

Parameter values returned to the application

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, "Return codes," on page 397.

RETCODE
The socket number for the new socket is returned. A RETCODE of –1 indicates an error.

TAKESOCKET

This call functions in the same way as the equivalent call described in "TAKESOCKET" on page 342. The format of the COBOL call for the TAKESOCKET function is:

```
CALL 'EZACICAL'
    USING TOKEN COMMAND HZERO CLIENTID L-DESC SOCKNO ERRNO RETCODE.
```

In assembler language, issue the macro call `CALL EZACICAL`, using standard assembler call syntax (for the call format, see "Assembler language" on page 370).

Parameter lengths in assembler language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY
HZERO	H	PIC 9(4) BINARY
CLIENTID STRUCTURE:		
Domain	F	PIC 9(8) BINARY
Name	CL8	PIC X(8)
Task	CL8	PIC X(8)
Reserved	CL20	PIC X(20)
L-DESC	F	PIC 9(8) BINARY
SOCKNO	F	PIC S9(8) BINARY
ERRNO	F	PIC 9(8) BINARY
RETCODE	F	PIC 9(8) BINARY

Parameter values to be set by the application

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
Must be set to `32` for the TAKESOCKET command

HZERO
Set to zeros

CLIENTID
Structure specifying the client ID of this program:

Domain
Must be set to 2 (AF-INET)

Name Set to address space identifier, obtained from GETCLIENTID

Task Set to CICS task number with `L` at the right end

Reserved
Set to binary zeros or LOW VALUES

L-DESC
Set to the descriptor (as used by the socket-giving program) of the socket being passed.

SOCKNO
Set to −1. The system returns the socket number in the RETCODE field.

Note: Be sure to use **only** the socket number returned by the system.

Parameter values returned to the application

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 397.

RETCODE
The socket number for the new socket is returned. A RETCODE of −1 indicates an error.

WRITE

This call functions in the same way as the equivalent call described in “WRITE” on page 344. The format of the COBOL call for the WRITE function is:

```
CALL 'EZACICAL' USING TOKEN COMMAND S NBYTE FZERO SZERO BUF ERRNO RETCODE.
```

In assembler language, issue the macro call `CALL EZACICAL`, using standard assembler call syntax (for the call format, see “Assembler language” on page 370).

Parameter lengths in assembler language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY
S	H	PIC 9(4) BINARY
NBYTE	F	PIC 9(8) BINARY
FZERO	F	PIC 9(8) BINARY
SZERO	XL16	PIC X(16)
BUF	NBYTE or larger	NBYTE or larger
ERRNO	F	PIC 9(8) BINARY
RETCODE	F	PIC S9(8) BINARY

Parameter values to be set by the application

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
Must be set to `26` for the WRITE command

S The descriptor of the socket from which data is to be transmitted

NBYTE
Set to the number of bytes of data to be transmitted. This value cannot exceed 32 768 bytes.

FZERO
Set to binary zeros or LOW VALUES

SZERO
Set to binary zeros or LOW VALUES

BUF Buffer containing data to be transmitted

Parameter values returned to the application

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, "Return codes," on page 397.

RETCODE
The number of bytes written is returned. A RETCODE of −1 indicates an error.

See "EZACIC04" on page 350 for a subroutine that translates EBCDIC data to ASCII.

Appendix B. Return codes

This topic covers the following return codes and error messages:
- Error numbers from z/OS 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 24. Sockets ERRNOs

Error number	Message name	Socket type	Error description	Programmer's response
1	EAI_NONAME	GETADDRINFO GETNAMEINFO	NODE or HOST cannot be found.	Ensure the NODE or HOST name can be resolved.
1	EPERM	All	Permission is denied. No owner exists.	Check that TPC/IP is still active; check protocol value of socket () call.
1	EPERM	IOCTL (SIOCTTLSCTL requesting both TTLS_INIT_ CONNECTION and TTLS_RESET_ SESSION or both TTLS_INIT_ CONNECTION and TTLS_RESET_ CIPHER)	The combination of requests specified is not permitted.	Request TTLS_RESET_SESSION and TTLS_RESET_CIPHER only when TTLS_INIT_ CONNECTION has been previously requested for the connection.
1	EDOM	All	Argument too large.	Check parameter values of the function call.
2	EAI_AGAIN	FREEADDRINFO GETADDRINFO GETNAMEINFO	For GETADDRINFO, NODE could not be resolved within the configured time interval. For GETNAMEINFO, HOST could not be resolved within the configured time interval. The Resolver address space has not been started. The request can be retried later.	Ensure the Resolver is active, then retry the request.
2	ENOENT	All	The data set or directory was not found.	Check files used by the function call.
2	ERANGE	All	The result is too large.	Check parameter values of the function call.

Table 24. Sockets ERRNOs (continued)

Error number	Message name	Socket type	Error description	Programmer's response
3	EAI_FAIL	FREEADDRINFO GETADDRINFO GETNAMEINFO	This is an unrecoverable error. NODELEN, HOSTLEN, or SERVLEN is incorrect. For FREEADDRINFO, the resolver storage does not exist.	Correct the NODELEN, HOSTLEN, or SERVLEN. Otherwise, call your system administrator.
3	ESRCH	All	The process was not found. A table entry was not located.	Check parameter values and structures pointed to by the function parameters.
4	EAI_OVERFLOW	GETNAMEINFO	The output buffer for the host name or service name was too small.	Increase the size of the buffer to 255 characters, which is the maximum size permitted.
4	EINTR	All	A system call was interrupted.	Check that the socket connection and TCP/IP are still active.
5	EAI_FAMILY	GETADDRINFO GETNAMEINFO	The AF or the FAMILY is incorrect.	Correct the AF or the FAMILY.
5	EIO	All	An I/O error occurred.	Check status and contents of source database if this occurred during a file access.
6	EAI_MEMORY	GETADDRINFO GETNAMEINFO	The resolver cannot obtain storage to process the host name.	Contact your system administrator.
6	ENXIO	All	The device or driver was not found.	Check status of the device attempting to access.
7	E2BIG	All	The argument list is too long.	Check the number of function parameters.
7	EAI_BADFLAGS	GETADDRINFO GETNAMEINFO	FLAGS has an incorrect value.	Correct the FLAGS.
8	EAI_SERVICE	GETADDRINFO	The SERVICE was not recognized for the specified socket type.	Correct the SERVICE.
8	ENOEXEC	All	An EXEC format error occurred.	Check that the target module on an exec call is a valid executable module.
9	EAI_SOCKTYPE	GETADDRINFO	The SOCTYPE was not recognized.	Correct the SOCTYPE.
9	EBADF	All	An incorrect socket descriptor was specified.	Check socket descriptor value. It might be currently not in use or incorrect.
9	EBADF	Givesocket	The socket has already been given. The socket domain is not AF_INET or AF_INET6.	Check the validity of function parameters.
9	EBADF	Select	One of the specified descriptor sets is an incorrect socket descriptor.	Check the validity of function parameters.

Table 24. Sockets ERRNOs (continued)

Error number	Message name	Socket type	Error description	Programmer's response
9	EBADF	Takesocket	The socket has already been taken.	Check the validity of function parameters.
9	EAI_SOCKTYPE	GETADDRINFO	The SOCTYPE was not recognized.	Correct the SOCTYPE.
10	ECHILD	All	There are no children.	Check if created subtasks still exist.
11	EAGAIN	All	There are no more processes.	Retry the operation. Data or condition might not be available at this time.
12	ENOMEM	All	There is not enough storage.	Check the validity of function parameters.
13	EACCES	All	Permission denied, caller not authorized.	Check access authority of file.
13	EACCES	Takesocket	The other application (listener) did not give the socket to your application. Permission denied, caller not authorized.	Check access authority of file.
13	EACCES	IOCTL (SIOCTTLSCTL)	The IOCTL is requesting a function that requires that the socket be mapped to policy that specifies ApplicationControlled On.	Check policy and add ApplicationControlled On if the application should be permitted to issue the controlled SIOCTTLSCTL functions.
14	EFAULT	All	An incorrect storage address or length was specified.	Check the validity of function parameters.
14	EFAULT	IOCTL (SIOCSAPPLDATA)	An abend occurred while attempting to copy the SetADcontainer structure from the address provided in the SetAD_ptr field.	Check the validity of function parameters.
15	ENOTBLK	All	A block device is required.	Check device status and characteristics.
16	EBUSY	All	Listen has already been called for this socket. Device or file to be accessed is busy.	Check if the device or file is in use.
17	EEXIST	All	The data set exists.	Remove or rename existing file.
18	EXDEV	All	This is a cross-device link. A link to a file on another file system was attempted.	Check file permissions.
19	ENODEV	All	The specified device does not exist.	Check file name and if it exists.
20	ENOTDIR	All	The specified directory is not a directory.	Use a valid file that is a directory.
21	EISDIR	All	The specified directory is a directory.	Use a valid file that is not a directory.

Table 24. Sockets ERRNOs (continued)

Error number	Message name	Socket type	Error description	Programmer's response
22	EINVAL	All types	An incorrect argument was specified.	Check the validity of function parameters.
23	ENFILE	All	Data set table overflow occurred.	Reduce the number of open files.
24	EMFILE	All	The socket descriptor table is full.	Check the maximum sockets specified in MAXDESC().
25	ENOTTY	All	An incorrect device call was specified.	Check specified IOCTL() values.
26	ETXTBSY	All	A text data set is busy.	Check the current use of the file.
27	EFBIG	All	The specified data set is too large.	Check size of accessed dataset.
28	ENOSPC	All	There is no space left on the device.	Increase the size of accessed file.
29	ESPIPE	All	An incorrect seek was attempted.	Check the offset parameter for seek operation.
30	EROFS	All	The data set system is Read only.	Access data set for read only operation.
31	EMLINK	All	There are too many links.	Reduce the number of links to the accessed file.
32	EPIPE	All	The connection is broken. For socket write/send, peer has shut down one or both directions.	Reconnect with the peer.
33	EDOM	All	The specified argument is too large.	Check and correct function parameters.
34	ERANGE	All	The result is too large.	Check function parameter values.
35	EWOULDBLOCK	Accept	The socket is in nonblocking mode and connections are not queued. This is not an error condition.	Reissue Accept().
35	EWOULDBLOCK	Read Recvfrom	The socket is in nonblocking mode and read data is not available. This is not an error condition.	Issue a select on the socket to determine when data is available to be read or reissue the Read()/Recvfrom().
35	EWOULDBLOCK	Send Sendto Write	The socket is in nonblocking mode and buffers are not available.	Issue a select on the socket to determine when data is available to be written or reissue the Send(), Sendto(), or Write().
35	EWOULDBLOCK	IOCTL (SIOCTTLSCTL)	The initial handshake is in progress and the socket is a non-blocking socket.	For a non-blocking socket, you can wait for the handshake to complete by issuing Select or Poll for Socket Writable.

Table 24. Sockets ERRNOs (continued)

Error number	Message name	Socket type	Error description	Programmer's response
36	EINPROGRESS	Connect	The socket is marked nonblocking and the connection cannot be completed immediately. This is not an error condition.	See the Connect() description for possible responses.
36	EINPROGRESS	IOCTL (SIOCTTLSCTL requesting TTLS_INIT_ CONNECTION)	The initial handshake is already in progress and the socket is a non-blocking socket.	For a non-blocking socket, you can wait for the handshake to complete by issuing Select or Poll for Socket Writable.
37	EALREADY	Connect	The socket is marked nonblocking and the previous connection has not been completed.	Reissue Connect().
37	EALREADY	IOCTL (SIOCTTLSCTL requesting TTLS_INIT_ CONNECTION)	The socket is already secure.	Correct application to issue SIOCTTLSCTL IOCTL that requests TTLS_INIT_ CONNECTION only when the socket is not already secure.
37	EALREADY	Maxdesc	A socket has already been created calling Maxdesc() or multiple calls to Maxdesc().	Issue Getablesize() to query it.
37	EALREADY	Setibmopt	A connection already exists to a TCP/IP image. A call to SETIBMOPT (IBMTCP_IMAGE), has already been made.	Only call Setibmopt() once.
38	ENOTSOCK	All	A socket operation was requested on a nonsocket connection. The value for socket descriptor was not valid.	Correct the socket descriptor value and reissue the function call.
39	EDESTADDRREQ	All	A destination address is required.	Fill in the destination field in the correct parameter and reissue the function call.
40	EMSGSIZE	Sendto Sendmsg Send Write	The message is too long. It exceeds the IP limit of 64K or the limit set by the setsockopt() call.	Either correct the length parameter, or send the message in smaller pieces.
41	EPROTOTYPE	All	The specified protocol type is incorrect for this socket.	Correct the protocol type parameter.
41	EPROTOTYPE	IOCTL (SIOCTTLSCTL)	Socket is not a TCP socket.	Issue the SIOCTTLSCTL IOCTL on TCP sockets only.
41	EPROTOTYPE	IOCTL (SIOCSAPPLDATA)	The request was not successful. The socket is not a stream (TCP) socket.	Issue the SIOCSAPPLDATA IOCTL on TCP sockets only.

Table 24. Sockets ERRNOs (continued)

Error number	Message name	Socket type	Error description	Programmer's response
42	ENOPROTOOPT	Getsockopt Setsockopt	The socket option specified is incorrect or the level is not SOL_SOCKET. Either the level or the specified optname is not supported.	Correct the level or optname.
42	ENOPROTOOPT	Getibmsockopt Setibmsockopt	Either the level or the specified optname is not supported.	Correct the level or optname.
43	EPROTONOSUPPORT	Socket	The specified protocol is not supported.	Correct the protocol parameter.
44	ESOCKTNOSUPPORT	All	The specified socket type is not supported.	Correct the socket type parameter.
45	EOPNOTSUPP	IOCTL	The specified IOCTL command is not supported by this socket API.	Correct the IOCTL COMMAND.
45	EOPNOTSUPP	IOCTL (SIOCTTLSCTL requesting TTLS_INIT_ CONNECTION, TTLS_RESET_ SESSION, or TTLS_RESET_ CIPHER)	Mapped policy indicates that AT-TLS is not enabled for the connection.	Modify policy to enable AT-TLS for the connection.
45	EOPNOTSUPP	RECV, RECVFROM, RECVMSG, SEND, SENDTO, SENDMSG	The specified flags are not supported on this socket type or protocol.	Correct the FLAG.
45	EOPNOTSUPP	Accept Givesocket	The selected socket is not a stream socket.	Use a valid socket.
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.
45	EOPNOTSUPP	Getibmopt Setibmopt	The socket does not support this function call. This command is not supported for this function.	Correct the command parameter. See Getibmopt() for valid commands. Correct by ensuring a Listen() was not issued before the Connect().
46	EPFNOSUPPORT	All	The specified protocol family is not supported or the specified domain for the client identifier is not AF_INET=2.	Correct the protocol family.
47	EAFNOSUPPORT	Bind Connect Socket	The specified address family is not supported by this protocol family.	For Socket(), set the domain parameter to AF_INET. For Bind() and Connect(), set Sin_Family in the socket address structure to AF_INET.
47	EAFNOSUPPORT	Getclient Givesocket	The socket specified by the socket descriptor parameter was not created in the AF_INET domain.	The Socket() call used to create the socket should be changed to use AF_INET for the domain parameter.

Table 24. Sockets ERRNOs (continued)

Error number	Message name	Socket type	Error description	Programmer's response
48	EADDRINUSE	Bind	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.	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.
49	EADDRNOTAVAIL	Bind	The specified address is incorrect for this host.	Correct the function address parameter.
49	EADDRNOTAVAIL	Connect	The calling host cannot reach the specified destination.	Correct the function address parameter.
50	ENETDOWN	All	The network is down.	Retry when the connection path is up.
51	ENETUNREACH	Connect	The network cannot be reached.	Ensure that the target application is active.
52	ENETRESET	All	The network dropped a connection on a reset.	Reestablish the connection between the applications.
53	ECONNABORTED	All	The software caused a connection abend.	Reestablish the connection between the applications.
54	ECONNRESET	All	The connection to the destination host is not available.	N/A
54	ECONNRESET	Send Write	The connection to the destination host is not available.	The socket is closing. Issue Send() or Write() before closing the socket.
55	ENOBUFS	All	No buffer space is available.	Check the application for massive storage allocation call.
55	ENOBUFS	Accept	Not enough buffer space is available to create the new socket.	Call your system administrator.
55	ENOBUFS	Send Sendto Write	Not enough buffer space is available to send the new message.	Call your system administrator.
55	ENOBUFS	IOCTL (SIOCTTLSCTL requesting TTLS_RETURN_ CERTIFICATE)	The buffer size provided is too small.	Use the returned certificate length to allocate a larger buffer and reissue IOCTL with the larger buffer.
55	ENOBUFS	Takesocket	Not enough buffer space is available to create the new socket.	Call your system administrator.
55	ENOBUF	IOCTL (SIOCSAPPLDATA)	There was no storage available to store the associated data.	Call your system administrator.

Table 24. Sockets ERRNOs (continued)

Error number	Message name	Socket type	Error description	Programmer's response
56	EISCONN	Connect	The socket is already connected.	Correct the socket descriptor on Connect() or do not issue a Connect() twice for the socket.
57	ENOTCONN	All	The socket is not connected.	Connect the socket before communicating.
57	ENOTCONN	IOCTL (SIOCTTLSCTL)	The socket is not connected.	Issue the SIOCTTLSCTL IOCTL only after the socket is connected.
58	ESHUTDOWN	All	A Send cannot be processed after socket shutdown.	Issue read/receive before shutting down the read side of the socket.
59	ETOOMANYREFS	All	There are too many references. A splice cannot be completed.	Call your system administrator.
60	ETIMEDOUT	Connect	The connection timed out before it was completed.	Ensure the server application is available.
61	ECONNREFUSED	Connect	The requested connection was refused.	Ensure server application is available and at specified port.
62	ELOOP	All	There are too many symbolic loop levels.	Reduce symbolic links to specified file.
63	ENAMETOOLONG	All	The file name is too long.	Reduce size of specified file name.
64	EHOSTDOWN	All	The host is down.	Restart specified host.
65	EHOSTUNREACH	All	There is no route to the host.	Set up network path to specified host and verify that host name is valid.
66	ENOTEMPTY	All	The directory is not empty.	Clear out specified directory and reissue call.
67	EPROCLIM	All	There are too many processes in the system.	Decrease the number of processes or increase the process limit.
68	EUSERS	All	There are too many users on the system.	Decrease the number of users or increase the user limit.
69	EDQUOT	All	The disk quota has been exceeded.	Call your system administrator.
70	ESTALE	All	An old NFS** data set handle was found.	Call your system administrator.
71	EREMOTE	All	There are too many levels of remote in the path.	Call your system administrator.
72	ENOSTR	All	The device is not a stream device.	Call your system administrator.
73	ETIME	All	The timer has expired.	Increase timer values or reissue function.
74	ENOSR	All	There are no more stream resources.	Call your system administrator.

Table 24. Sockets ERRNOs (continued)

Error number	Message name	Socket type	Error description	Programmer's response
75	ENOMSG	All	There is no message of the desired type.	Call your system administrator.
76	EBADMSG	All	The system cannot read the message.	Verify that z/OS Communications Server installation was successful and that message files were properly loaded.
77	EIDRM	All	The identifier has been removed.	Call your system administrator.
78	EDEADLK	All	A deadlock condition has occurred.	Call your system administrator.
78	EDEADLK	Select Selectex	None of the sockets in the socket descriptor sets are either AF_INET or AF_IUCV sockets and there is no timeout value or no ECB specified. The select/selectex would never complete.	Correct the socket descriptor sets so that an AF_INET or AF_IUCV socket is specified. A timeout or ECB value can also be added to avoid the select/selectex from waiting indefinitely.
79	ENOLCK	All	No record locks are available.	Call your system administrator.
80	ENONET	All	The requested machine is not on the network.	Call your system administrator.
81	ERREMOTE	All	The object is remote.	Call your system administrator.
82	ENOLINK	All	The link has been severed.	Release the sockets and reinitialize the client-server connection.
83	EADV	All	An ADVERTISE error has occurred.	Call your system administrator.
84	ESRMNT	All	An SRMOUNT error has occurred.	Call your system administrator.
85	ECOMM	All	A communication error has occurred on a Send call.	Call your system administrator.
86	EPROTO	All	A protocol error has occurred.	Call your system administrator.
86	EPROTO	IOCTL (SIOCTTLSCTL requesting TTLS_RESET_ SESSION or TTLS_RESET_ CIPHER)	A TTLS_INIT_ CONNECTION request has not been received for the connection or TTLS_RESET_CIPHER was requested on a connection that is secured using SSL version 2.	Request TTLS_INIT_ CONNECTION prior to requesting TTLS_RESET_SESSION or TTLS_RESET_CIPHER. Request TTLS_RESET_CIPHER only on connections secured using SSL version 3 or TLS version 1.
87	EMULTIHOP	All	A multihop address link was attempted.	Call your system administrator.
88	EDOTDOT	All	A cross-mount point was detected. This is not an error.	Call your system administrator.

Table 24. Sockets ERRNOs (continued)

Error number	Message name	Socket type	Error description	Programmer's response
89	EREMCHG	All	The remote address has changed.	Call your system administrator.
90	ECONNCLOSED	All	The connection was closed by a peer.	Check that the peer is running.
113	EBADF	All	Socket descriptor is not in correct range. The maximum number of socket descriptors is set by MAXDESC(). The default range is 0–49.	Reissue function with corrected socket descriptor.
113	EBADF	Bind socket	The socket descriptor is already being used.	Correct the socket descriptor.
113	EBADF	Givesocket	The socket has already been given. The socket domain is not AF_INET.	Correct the socket descriptor.
113	EBADF	Select	One of the specified descriptor sets is an incorrect socket descriptor.	Correct the socket descriptor. Set on Select() or Selectex().
113	EBADF	Takesocket	The socket has already been taken.	Correct the socket descriptor.
113	EBADF	Accept	A Listen() has not been issued before the Accept().	Issue Listen() before Accept().
121	EINVAL	All	An incorrect argument was specified.	Check and correct all function parameters.
121	EINVAL	IOCTL (SIOCSAPPLDATA)	The input parameter is not a correctly formatted SetApplData structure. • The SetAD_eye1 value is not valid • The SetAD_ver value is not valid. • Storage pointed to by SetAD_ptr does not contain a properly formatted SetADcontainer structure. • The SetAD_eye2 value is not valid. • the SetAD_len value contains an incorrect length for the SetAD_ver version of the SetADcontainer structure.	Check and correct all function parameters.
122	ECLOSED			
126	ENMELONG			
136	ENOTEMPT			
145	E2BIG	All	The argument list is too long.	Eliminate excessive number of arguments.

Table 24. Sockets ERRNOs (continued)

Error number	Message name	Socket type	Error description	Programmer's response
156	EMVSINITIAL	All	Process initialization error. This indicates an z/OS UNIX process initialization failure. This is usually an indication that a proper OMVS RACF segment is not defined for the user ID associated with application. The RACF OMVS segment may not be defined or may contain errors such as an improper HOME() directory specification.	Attempt to initialize again. After ensuring that an OMVS Segment is defined, if the errno is still returned, call your MVS system programmer to have IBM service contacted.
157	EMISSED			
1002	EIBMSOCKOUTOFRANGE	Socket	A socket number assigned by the client interface code is out of range.	Check the socket descriptor parameter.
1003	EIBMSOCKINUSE	Socket	A socket number assigned by the client interface code is already in use.	Use a different socket descriptor.
1004	EIBMIUCVERR	All	The request failed because of an IUCV error. This error is generated by the client stub code.	Ensure IUCV/VMCF is functional.
1008	EIBMCONFLICT	All	This request conflicts with a request already queued on the same socket.	Cancel the existing call or wait for its completion before reissuing this call.
1009	EIBMCANCELLED	All	The request was canceled by the CANCEL call.	Informational, no action needed.
1011	EIBMBADTCPNAME	All	A TCP/IP name that is not valid was detected.	Correct the name specified in the IBM_TCPIMAGE structure.
1011	EIBMBADTCPNAME	Setibmopt	A TCP/IP name that is not valid was detected.	Correct the name specified in the IBM_TCPIMAGE structure.
1011	EIBMBADTCPNAME	INITAPI	A TCP/IP name that is not valid was detected.	Correct the name specified on the IDENT option TCPNAME field.
1012	EIBMBADREQUESTCODE	All	A request code that is not valid was detected.	Contact your system administrator.
1013	EIBMBADCONNECTIONSTATE	All	A connection token that is not valid was detected; bad state.	Verify TCP/IP is active.
1014	EIBMUNAUTHORIZEDCALLER	All	An unauthorized caller specified an authorized keyword.	Ensure user ID has authority for the specified operation.

Table 24. Sockets ERRNOs (continued)

Error number	Message name	Socket type	Error description	Programmer's response
1015	EIBMBADCONNECTIONMATCH	All	A connection token that is not valid was detected. There is no such connection.	Verify TCP/IP is active.
1016	EIBMTCPABEND	All	An abend occurred when TCP/IP was processing this request.	Verify that TCP/IP has restarted.
1023	EIBMTERMERROR	All	Encountered a terminating error while processing.	Call your system administrator.
1026	EIBMINVDELETE	All	Delete requestor did not create the connection.	Delete the request from the process that created it.
1027	EIBMINVSOCKET	All	A connection token that is not valid was detected. No such socket exists.	Call your system programmer.
1028	EIBMINVTCPCONNECTION	All	Connection terminated by TCP/IP. The token was invalidated by TCP/IP.	Reestablish the connection to TCP/IP.
1032	EIBMCALLINPROGRESS	All	Another call was already in progress.	Reissue after previous call has completed.
1036	EIBMNOACTIVETCP	All	TCP/IP is not installed or not active.	Correct TCP/IP name used.
1036	EIBMNOACTIVETCP	Select	EIBMNOACTIVETCP	Ensure TCP/IP is active.
1036	EIBMNOACTIVETCP	Getibmopt	No TCP/IP image was found.	Ensure TCP/IP is active.
1037	EIBMINVTSRBUSERDATA	All	The request control block contained data that is not valid.	Call your system programmer.
1038	EIBMINVUSERDATA	All	The request control block contained user data that is not valid.	Check your function parameters and call your system programmer.
1040	EIBMSELECTEXPOST	SELECTEX	SELECTEX passed an ECB that was already posted.	Check whether the user's ECB was already posted.
1112	ECANCEL			
2001	EINVALIDRXSOCKETCALL	REXX	A syntax error occurred in the RXSOCKET parameter list.	Correct the parameter list passed to the REXX socket call.
2002	ECONSOLEINTERRUPT	REXX	A console interrupt occurred.	Retry the task.
2003	ESUBTASKINVALID	REXX	The subtask ID is incorrect.	Correct the subtask ID on the INITIALIZE call.
2004	ESUBTASKALREADYACTIVE	REXX	The subtask is already active.	Only issue the INITIALIZE call once in your program.
2005	ESUBTASKALNOTACTIVE	REXX	The subtask is not active.	Issue the INITIALIZE call before any other socket call.

Table 24. Sockets ERRNOs (continued)

Error number	Message name	Socket type	Error description	Programmer's response
2006	ESOCKNETNOTALLOCATED	REXX	The specified socket could not be allocated.	Increase the user storage allocation for this job.
2007	EMAXSOCKETSREACHED	REXX	The maximum number of sockets has been reached.	Increase the number of allocate sockets, or decrease the number of sockets used by your program.
2009	ESOCKETNOTDEFINED	REXX	The socket is not defined.	Issue the SOCKET call before the call that fails.
2011	EDOMAINSERVERFAILURE	REXX	A Domain Name Server failure occurred.	Call your MVS system programmer.
2012	EINVALIDNAME	REXX	An incorrect *name* was received from the TCP/IP server.	Call your MVS system programmer.
2013	EINVALIDCLIENTID	REXX	An incorrect *clientid* was received from the TCP/IP server.	Call your MVS system programmer.
2014	ENIVALIDFILENAME	REXX	An error occurred during NUCEXT processing.	Specify the correct translation table file name, or verify that the translation table is valid.
2016	EHOSTNOTFOUND	REXX	The host is not found.	Call your MVS system programmer.
2017	EIPADDRNOTFOUND	REXX	Address not found.	Call your MVS system programmer.
3412	ENODATA		Message does not exist.	
3416	ELINKED		Stream is linked.	
3419	ERECURSE		Recursive attempt rejected.	
3420	EASYNC		Asynchronous I/O scheduled. This is a normal, internal event that is NOT returned to the user.	
3448	EUNATCH		The protocol required to support the specified address family is not available.	
3464	ETERM		Operation terminated.	
3474	EUNKNOWN		Unknown system state.	
3495	EBADOBJ		You attempted to reference a object that does not exist.	
3513	EOUTOFSTATE		Protocol engine has received a command that is not acceptable in its current state.	

Sockets extended ERRNOs

Table 25. Sockets extended ERRNOs

Error code	Problem description	System action	Programmer's response
10100	An ESTAE macro did not complete normally.	End the call.	Call your MVS system programmer.
10101	A STORAGE OBTAIN failed.	End the call.	Increase MVS storage in the application's address space.
10108	The first call issued was not a valid first call.	End the call.	For a list of valid first calls, refer to the section on special considerations in the general programming information.
10110	LOAD of EZBSOH03 (alias EZASOH03) failed.	End the call.	Call the IBM Software Support Center.
10154	Errors were found in the parameter list for an IOCTL call.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the IOCTL call. You might have incorrect sequencing of socket calls.
10155	The length parameter for an IOCTL call is less than or equal to 0.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the IOCTL call. You might have incorrect sequencing of socket calls.
10156	The length parameter for an IOCTL call is 3200 (32 x 100).	Disable the subtask for interrupts. Return an error code to the caller.	Correct the IOCTL call. You might have incorrect sequencing of socket calls.
10159	A 0 or negative data length was specified for a READ or READV call.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the length in the READ call.
10161	The REQARG parameter in the IOCTL parameter list is 0.	End the call.	Correct the program.
10163	A 0 or negative data length was found for a RECV, RECVFROM, or RECVMSG call.	Disable the subtask for interrupts. Sever the DLC path. Return an error code to the caller.	Correct the data length.
10167	The descriptor set size for a SELECT or SELECTEX call is less than or equal to 0.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the SELECT or SELECTEX call. You might have incorrect sequencing of socket calls.
10168	The descriptor set size *in bytes* for a SELECT or SELECTEX call is greater than 8192. A number greater than the maximum number of allowed sockets (65534 is the maximum) has been specified.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the descriptor set size.
10170	A 0 or negative data length was found for a SEND or SENDMSG call.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the data length in the SEND call.

Table 25. Sockets extended ERRNOs (continued)

Error code	Problem description	System action	Programmer's response
10174	A 0 or negative data length was found for a SENDTO call.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the data length in the SENDTO call.
10178	The SETSOCKOPT option length is less than the minimum length.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the OPTLEN parameter.
10179	The SETSOCKOPT option length is greater than the maximum length.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the OPTLEN parameter.
10184	A data length of 0 was specified for a WRITE call.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the data length in the WRITE call.
10186	A negative data length was specified for a WRITE or WRITEV call.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the data length in the WRITE call.
10190	The GETHOSTNAME option length is not in the range of 1–255..	Disable the subtask for interrupts. Return an error code to the caller.	Correct the length parameter.
10193	The GETSOCKOPT option length is less than the minimum or greater than the maximum length.	End the call.	Correct the length parameter.
10197	The application issued an INITAPI call after the connection was already established.	Bypass the call.	Correct the logic that produces the INITAPI call that is not valid.
10198	The maximum number of sockets specified for an INITAPI exceeds 65535.	Return to the user.	Correct the INITAPI call.
10200	The first call issued was not a valid first call.	End the call.	For a list of valid first calls, refer to the section on special considerations in the general programming information.
10202	The RETARG parameter in the IOCTL call is 0.	End the call.	Correct the parameter list. You might have incorrect sequencing of socket calls.
10203	The requested socket number is a negative value.	End the call.	Correct the requested socket number.
10205	The requested socket number is a duplicate.	End the call.	Correct the requested socket number.
10208	The NAMELEN parameter for a GETHOSTBYNAME call was not specified.	End the call.	Correct the NAMELEN parameter. You might have incorrect sequencing of socket calls.
10209	The NAME parameter on a GETHOSTBYNAME call was not specified.	End the call.	Correct the NAME parameter. You might have incorrect sequencing of socket calls.

Table 25. Sockets extended ERRNOs (continued)

Error code	Problem description	System action	Programmer's response
10210	The HOSTENT parameter on a GETHOSTBYNAME or GETHOSTBYADDR call was not specified.	End the call.	Correct the HOSTENT parameter. You might have incorrect sequencing of socket calls.
10211	The HOSTADDR parameter on a GETHOSTBYNAME or GETHOSTBYADDR call is incorrect.	End the call.	Correct the HOSTADDR parameter. You might have incorrect sequencing of socket calls.
10212	The resolver program failed to load correctly for a GETHOSTBYNAME or GETHOSTBYADDR call.	End the call.	Check the JOBLIB, STEPLIB, and linklib datasets and rerun the program.
10213	Not enough storage is available to allocate the HOSTENT structure.	End the call.	Increase the user storage allocation for this job.
10214	The HOSTENT structure was not returned by the resolver program.	End the call.	Ensure that the domain name server is available. This can be a nonerror condition indicating that the name or address specified in a GETHOSTBYADDR or GETHOSTBYNAME call could not be matched.
10215	The APITYPE parameter on an INITAPI call instruction was not 2 or 3.	End the call.	Correct the APITYPE parameter.
10218	The application programming interface (API) cannot locate the specified TCP/IP.	End the call.	Ensure that an API that supports the performance improvements related to CPU conservation is installed on the system and verify that a valid TCP/IP name was specified on the INITAPI call. This error call might also mean that EZASOKIN could not be loaded.
10219	The NS parameter is greater than the maximum socket for this connection.	End the call.	Correct the NS parameter on the ACCEPT, SOCKET or TAKESOCKET call.
10221	The AF parameter of a SOCKET call is not AF_INET.	End the call.	Set the AF parameter equal to AF_INET.
10222	The SOCTYPE parameter of a SOCKET call must be stream, datagram, or raw (1, 2, or 3).	End the call.	Correct the SOCTYPE parameter.
10223	No ASYNC parameter specified for INITAPI with APITYPE=3 call.	End the call.	Add the ASYNC parameter to the INITAPI call.
10224	The IOVCNT parameter is less than or equal to 0, for a READV, RECVMSG, SENDMSG, or WRITEV call.	End the call.	Correct the IOVCNT parameter.
10225	The IOVCNT parameter is greater than 120, for a READV, RECVMSG, SENDMSG, or WRITEV call.	End the call.	Correct the IOVCNT parameter.

Table 25. Sockets extended ERRNOs (continued)

Error code	Problem description	System action	Programmer's response
10226	Not valid COMMAND parameter specified for a GETIBMOPT call.	End the call.	Correct the COMMAND parameter of the GETIBMOPT call.
10229	A call was issued on an APITYPE=3 connection without an ECB or REQAREA parameter.	End the call.	Add an ECB or REQAREA parameter to the call.
10300	Termination is in progress for either the CICS transaction or the socket interface.	End the call.	None.
10330	A SELECT call was issued without a MAXSOC value and a TIMEOUT parameter.	End the call.	Correct the call by adding a TIMEOUT parameter.
10331	A call that is not valid was issued while in SRB mode.	End the call.	Get out of SRB mode and reissue the call.
10332	A SELECT call is invoked with a MAXSOC value greater than that which was returned in the INITAPI function (MAXSNO field).	End the call.	Correct the MAXSOC parameter and reissue the call.
10334	An error was detected in creating the data areas required to process the socket call.	End the call.	Call the IBM Software Support Center.
10999	An abend has occurred in the subtask.	Write message EZY1282E to the system console. End the subtask and post the TRUE ECB.	If the call is correct, call your system programmer.
20000	An unknown function code was found in the call.	End the call.	Correct the SOC-FUNCTION parameter.
20001	The call passed an incorrect number of parameters.	End the call.	Correct the parameter list.
20002	The user ID associated with the program linking EZACIC25 does not have the proper authority to execute a CICS EXTRACT EXIT.	End the call.	Start the CICS socket interface before executing this call.
20003	The CICS socket interface is not in operation.	End the call.	Contact the CICS system programmer. Ensure that the user ID being used is permitted to have at least UPDATE access to the EXITPROGRAM resource.

Appendix C. GETSOCKOPT/SETSOCKOPT command values

You can use the table below to determine the decimal or hexadecimal value associated with the GETSOCKOPT/SETSOCKOPT OPTNAMES supported by the APIs discussed in this document.

The command names are shown with underscores for the assembler language. The underscores should be changed to dashes if using the COBOL programming language.

Languages that cannot easily handle binary values, such as COBOL, should use the decimal value associated with the command where necessary.

The hexadecimal value can be used in Macro, Assembler and PL/I programs.

Table 26. GETSOCKOPT/SETSOCKOPT command values for Macro, Assembler, COBOL and PL/I

Command name	Decimal value	Hex value
IP_ADD_MEMBERSHIP	1048581	X'00100005'
IP_ADD_SOURCE_MEMBERSHIP	1048588	X'0010000C'
IP_BLOCK_SOURCE	1048586	X'0010000A'
IP_DROP_MEMBERSHIP	1048582	X'00100006'
IP_DROP_SOURCE_MEMBERSHIP	1048589	X'0010000D'
IP_MULTICAST_IF	1048583	X'00100007'
IP_MULTICAST_LOOP	1048580	X'00100004'
IP_MULTICAST_TTL	1048579	X'00100003'
IP_UNBLOCK_SOURCE	1048587	X'0010000B'
IPV6_JOIN_GROUP	65541	X'00010005'
IPV6_LEAVE_GROUP	65542	X'00010006'
IPV6_MULTICAST_HOPS	65545	X'00010009'
IPV6_MULTICAST_IF	65543	X'00010007'
IPV6_MULTICAST_LOOP	65540	X'00010004'
IPV6_UNICAST_HOPS	65539	X'00010003'
IPV6_V6ONLY	65546	X'0001000A'
MCAST_BLOCK_SOURCE	1048620	X'0010002C'
MCAST_JOIN_GROUP	1048616	X'00100028'
MCAST_JOIN_SOURCE_GROUP	1048618	X'0010002A'
MCAST_LEAVE_GROUP	1048617	X'00100029'
MCAST_LEAVE_SOURCE_GROUP	1048619	X'0010002B'
MCAST_UNBLOCK_SOURCE	1048621	X'0010002D'
SO_BROADCAST	32	X'00000020'
SO_ERROR	4103	X'00001007'
SO_LINGER	128	X'00000080'
SO_KEEPALIVE	8	X'00000008'

Table 26. GETSOCKOPT/SETSOCKOPT command values for Macro, Assembler, COBOL and PL/I (continued)

Command name	Decimal value	Hex value
SO_OOBINLINE	256	X'00000100'
SO_RCVBUF	4098	X'00001002'
SO_REUSEADDR	4	X'00000004'
SO_SNDBUF	4097	X'00001001'
SO_TYPE	4104	X'00001008'
TCP_KEEPALIVE	2147483654	X'80000008'
TCP_NODELAY	2147483649	X'80000001'

Table 27. GETSOCKOPT/SETSOCKOPT optname value for C programs

Option name	Decimal value
IP_ADD_MEMBERSHIP	5
IP_ADD_SOURCE_MEMBERSHIP	12
IP_BLOCK_SOURCE	10
IP_DROP_MEMBERSHIP	6
IP_DROP_SOURCE_MEMBERSHIP	13
IP_MULTICAST_IF	7
IP_MULTICAST_LOOP	4
IP_MULTICAST_TTL	3
IP_UNBLOCK_SOURCE	11
MCAST_BLOCK_SOURCE	44
MCAST_JOIN_GROUP	40
MCAST_JOIN_SOURCE_GROUP	42
MCAST_LEAVE_GROUP	41
MCAST_LEAVE_SOURCE_GROUP	43
MCAST_UNBLOCK_SOURCE	45
SO_ACCEPTCONN	2
SO_BROADCAST	32
SO_CLUSTERCONNTYPE	16385
SO_DEBUG	1
SO_ERROR	4103
SO_KEEPALIVE	8
SO_LINGER	128
SO_OOBINLINE	256
SO_RCVBUF	4098
SO_REUSEADDR	4
SO_SNDBUF	4097
SO_TYPE	4104
TCP_KEEPALIVE	8
TCP_NODELAY	1

Appendix D. CICS sockets messages

This topic contains CICS socket interface messages.

EZY1218—EZY1366

EZY1218E *mm/dd/yy hh:mm:ss* **PROGRAM** *programname* **DISABLED TRANID=** *transactionid* **PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: The Listener checked the status of the program associated with the transaction. It was not enabled.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

programname is the name of the program that is associated with the transaction requested by the connecting client.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: Listener continues.

Operator response: Use CEMT to determine and correct the status of the program.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1219E *mm/dd/yy hh:mm:ss* **UNEXPECTED** *eventtype* **EVENT IN LISTENER** *transactionid* **FROM CLIENT IP ADDRESS** *ipaddress* **PORT** *portnumber*

Explanation: The CICS Listener was notified about an unexpected event.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

eventtype is the type of event: READ, WRITE, or EXCEPTION.

transactionid is the name of the Listener's CICS transaction.

ipaddress is the remote IP address of the client.

portnumber is the remote port number of the client.

System action: The Listener closes the connection and continues processing.

Operator response: Contact the system programmer.

System programmer response: If the event type is EXCEPTION, investigate whether or not the client is attempting to send out-of-band data. If necessary, have the client avoid sending out-of-band data. If the event type is not EXCEPTION or the client is not attempting to send out-of-band data, then contact the IBM Software Support Center.

Module: EZACIC02

Destination: LISTENER

EZY1220E *mm/dd/yy hh:mm:ss* **READ FAILURE ON CONFIGURATION FILE PHASE=***phase* **EIBRESP2=***response*

Explanation: EZACIC21 was unable to read the IP CICS Sockets configuration file, EZACONFG.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

phase is the IP CICS Sockets initialization phase.

response is the response from CICS when reading the IP CICS Sockets configuration file.

System action: If the ABEND code is AEXY, then the listener ends normally. Otherwise, the listener ends with an ABEND code of EZAL.

Operator response: Notify the CICS system programmer.

System programmer response: Use the EIBRESP2 value to determine the problem and correct the file. See the *CICS Application Programming Reference* for information about EIBRESP2 values. If the EIBRESP2 value is zero, then the EZACONFG file has been defined as remote. If this is the configuration file you want, then verify that no CICS Sockets programs can run directly in the file owning region. This can cause the file to become disabled. Ensure that EZACIC20 is not in the file owning region PLT, and that the EZAC and EZAO transactions are unable to run directly in the file owning region. Attempts to open the file will fail if the file is defined with a value of YES specified in the ADD, DELETE, or UPDATE parameters in the CICS file definition in more than one CICS region.

Module: EZACIC21

Destination: INITIALIZATION

EZY1221E *mm/dd/yy hh:mm:ss* **CICS SOCKETS ENABLE FAILURE EIBRCODE BYTE2 =** *resp_code*

Explanation: The attempt to enable the task related user exit (TRUE) failed.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

resp_code is the CICS response code from attempting to enable IP CICS Sockets Task Related User Exit (TRUE).

System action: Terminate the transaction.

Operator response: Notify the CICS system programmer.

System programmer response: Use the EIBRESP2 value to determine the problem and correct the file. An EIBRCODE BYTE2 value of 20 indicates the TRUE is already enabled. This will occur if you disable the interface using EZAO,STOP,CICS transaction and then immediately issue EZAO,START,CICS transaction before the Task Related User Exit (TRUE) is completely disabled from the previous EZAO,STOP,CICS transaction. See the *CICS Application Programming Reference* for information about EIBRCODEs.

Module: EZACIC21

Destination: INITIALIZATION

EZY1222E *mm/dd/yy hh:mm:ss* **CICS/SOCKETS REGISTRATION FAILURE RETURN code=** *return_code*

Explanation: The attempt to register the CICS Sockets Feature to z/OS failed.

System action: Terminate the transaction.

Operator response: Contact your System Administrator.

System programmer response: See the *z/OS MVS Programming: Product Registration* for information about the values for *return_code*.

Module: EZACIC21

Destination: INITIALIZATION

EZY1223E *mm/dd/yy hh:mm:ss* **CICS/SOCKETS ATTACH FAILURE RETURN CODE =** *return_code* **REASON CODE =** *reason_code*

Explanation: An attempt to attach one of the pool subtasks failed.

System action: Stop attaching pool subtasks. The size of the pool is determined by the number of subtasks successfully attached.

Operator response: Contact the CICS system programmer.

System programmer response: See the *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN* for information about the values for *return_code* and *reason_code* and make appropriate adjustments to your CICS environment.

Module: EZACIC21

Destination: INITIALIZATION

EZY1224I *mm/dd/yy hh:mm:ss* **CICS/SOCKETS INITIALIZATION SUCCESSFUL USING** *tasking_method*

Explanation: The CICS socket interface has completed initialization successfully.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

tasking_method is the tasking method used to support the EZASOKET calls. The possible methods are:

Reusable MVS subtasks
: Signifies that the IP CICS socket interface is using MVS subtasks from the pool generated according to the value specified on the NTASKS configuration parameter.

Non-reusable MVS subtasks
: Signifies that the IP CICS socket interface is attaching an MVS subtask for each IP CICS Sockets-enabled application because NTASKS=0.

Open Transaction Environment
: Signifies that the IP CICS socket interface is enabled to use CICS Open Transaction Environment. All EZASOKET calls will be processed on an Open API, L8, TCB. Programs calling EZASOKET should be coded to threadsafe programming standards and defined to CICS as CONCURRENCY(THREADSAFE) to benefit from this environment.

System action: Continue with execution.

Operator response: None.

System programmer response: None.

Module: EZACIC21

Destination: INITIALIZATION

EZY1225E *mm/dd/yy hh:mm:ss* **STARTBR FAILURE ON CICS/SOCKETS CONFIGURATION FILE PHASE=***xx* **EIBRESP2=***rrrrrr*

Explanation: The STARTBR command used for the configuration file has failed.

System action: Terminate the transaction.

Operator response: Contact the CICS system programmer.

System programmer response: Use the EIBRESP2 value to determine the problem. Check the CICS definition of the Configuration file to ensure the browse operation is permitted. See the *CICS Application Programming Reference* for information about EIBRESP2 values.

Module: EZACIC21

Destination: INITIALIZATION

EZY1226E *mm/dd/yy hh:mm:ss* **READNEXT FAILURE ON CICS/SOCKETS CONFIGURATION FILE PHASE=***xx* **EIBRESP2=***rrrrrr*

Explanation: The READNEXT command used for the configuration file has failed.

System action: Terminate the transaction.

Operator response: Contact the CICS system programmer.

System programmer response: Use the EIBRESP2 value to determine the problem. Check the CICS definition of the Configuration file to ensure the browse operation is permitted. See the *CICS Application Programming Reference* for information about EIBRESP2 values.

Module: EZACIC21

Destination: INITIALIZATION

EZY1227E *mm/dd/yy hh:mm:ss* **CICS/SOCKETS INVALID LISTENER TRANID =** *tran*

Explanation: The Listener transaction *tran* was not defined to CICS.

System action: Terminate Listener Initialization.

Operator response: Use CICS facilities to define the Listener transaction and program. Then use EZAO to start the Listener.

System programmer response: None.

Module: EZACIC21

Destination: INITIALIZATION

EZY1228E *mm/dd/yy hh:mm:ss* **CICS/SOCKETS LISTENER TRANSACTION** *tran* **DISABLED**

Explanation: The Listener transaction *tran* could not be started because it was disabled.

System action: Terminate Listener Initialization.

Operator response: Use CICS facilities to enable the transaction and then start the Listener using EZAO.

System programmer response: None.

Module: EZACIC21

Destination: INITIALIZATION

EZY1229E *mm/dd/yy hh:mm:ss* **CICS SOCKETS LISTENER TRANSACTION** *tran* **NOT AUTHORIZED**

Explanation: The Listener transaction *tran* could not be started because it was not authorized.

System action: Terminate Listener Initialization.

Operator response: Use CICS facilities to authorize starting the Listener transaction and then start the Listener using EZAO.

System programmer response: None.

Module: EZACIC21

Destination: INITIALIZATION

EZY1246E *mm/dd/yy hh:mm:ss* **CICS SOCKETS LISTENER PROGRAM ID** *mmmmmmmm* **INVALID**

Explanation: The Listener transaction could not be started because program *mmmmmmmm* is not defined.

System action: Terminate Listener Initialization.

Operator response: If the program ID is correct, use CICS facilities to define it. If it is not correct, use the EZAC transaction to correct the CICS Sockets Configuration file.

System programmer response: None.

Module: EZACIC21

Destination: INITIALIZATION

EZY1247E *mm/dd/yy hh:mm:ss* **CICS SOCKETS LISTENER PROGRAM ID** *mmmmmmmm* **DISABLED**

Explanation: The Listener transaction could not be started because program *mmmmmmmm* is disabled.

System action: Terminate Listener Initialization.

Operator response: Use CICS facilities to enable the program and then use EZAO to start the Listener.

System programmer response: None.

Module: EZACIC21

Destination: INITIALIZATION

EZY1250E *mm/dd/yy hh:mm:ss* **CICS/SOCKETS LISTENER** *tran* **NOT ON CONFIGURATION FILE**

Explanation: The Listener transaction *tran* is not defined on the CICS Sockets configuration file.

System action: Terminate Listener Initialization.

Operator response: If the Listener transaction name is correct, use the EZAC transaction to define it on the CICS Configuration file. If the name is not correct, correct it on the EZAO transaction.

System programmer response: None.

Module: EZACIC21

Destination: INITIALIZATION

EZY1251E *mm/dd/yy hh:mm:ss* **CICS SOCKETS MODULE** *mmmmmmmm* **ABEND** *xxxx*

Explanation: The CICS Sockets module *mmmmmmmm* has abended.

System action: Terminate the transaction.

Operator response: Contact the IBM Software Support Center.

System programmer response: None.

Module: EZACIC21

Destination: INITIALIZATION

EZY1252E *mm/dd/yy hh:mm:ss* **UNABLE TO LOAD EZASOH03 ERROR CODE=** *error_code* **REASON CODE=** *reason_code*

Explanation: During CICS Sockets initialization, the attempt to load module EZASOH03 failed.

System action: Terminate Initialization.

Operator response: Contact the CICS system programmer.

System programmer response: See the *z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU* for information about the values for *error_code* and *reason_code* to determine why the module would not load. Also, look for associated MVS messages.

Module: EZACIC21

EZY1253E *mm/dd/yy hh:mm:ss* **CICS/SOCKETS LISTENER** *tran* **NOT ON CONFIGURATION FILE**

Explanation: An EZAO STOP LISTENER transaction was run with an invalid Listener name.

System action: Present the panel to correct the name.

Operator response: Correct the name and retry termination.

System programmer response: None.

Module: EZACIC22

Destination: TERMINATION

EZY1254E *mm/dd/yy hh:mm:ss* **CACHE FILE ERROR RESP2 VALUE ****** CALL # ***

Explanation: An error occurred on a cache file operation.

System action: Return to the calling program with an error response.

Operator response: Contact the CICS system programmer.

System programmer response: Use the RESP2 value to determine the error and correct the cache file. See the *CICS Application Programming Reference* for information about RESP2 values.

Module: EZACIC25

Destination: DOMAIN NAME SERVER FUNCTION

EZY1255E *mm/dd/yy hh:mm:ss* **TEMPORARY STORAGE ERROR RESP2 VALUE ****** CALL # ***

Explanation: An error occurred on a temporary storage operation in EZACIC25.

System action: Return to the calling program with an error response.

Operator response: Use the RESP2 value to determine the error. Contact the IBM Software Support Center. See the *CICS Application Programming Reference* for information about RESP2 values.

System programmer response: None.

Module: EZACIC25

Destination: DOMAIN NAME SERVER FUNCTION

EZY1256E *mm/dd/yy hh:mm:ss* **CICS SOCKETS INTERFACE NOT ENABLED PRIOR TO LISTENER STARTUP**

Explanation: An attempt to start a Listener was made when the CICS socket interface was inactive.

System action: Return error and terminate transaction EZAO.

Operator response: Use transaction EZAO to start the CICS socket interface prior to starting the Listener.

System programmer response: None.

Module: EZACIC21

Destination: INITIALIZATION

EZY1258I *module* **ENTRY POINT IS** *address*

Explanation: This message displays the entry point address of a module.

module is the name of the module.

address is the entry point address of the module.

System action: Processing continues.

Operator response: None.

System programmer response: None.

Module: EZACIC01, EZACIC02

EZY1259E *mm/dd/yy hh:mm:ss* **IOCTL CALL FAILURE TRANSACTION=***transactionid* **TASKID=***tasknumber* **ERRNO=***errno*

Explanation: Listener transaction *transactionid* experienced a failure on the IOCTL call.

In the message text:

mm/dd/yy
: The date (month/day/year) of the message.

hh:mm:ss
: The time (hours:minutes:seconds) of the message.

transactionid
: The name of the transaction under which the Listener is executing.

tasknumber
: The CICS task number of the Listener task.

errno
: The UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

System action: If the error is during initialization of the Listener, then the Listener transaction *transactionid* terminates. Otherwise, the Listener closes the socket that was being processed and resumes normal processing.

Operator response: Use the *errno* value to determine the cause of the failure.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1260E *mm/dd/yy hh:mm:ss* **EZACIC03 ATTACH FAILED GPR15=***xxxxxxxx* **ERRNO=***errno* **TRAN=***tran* **TASK=***cicstask*

Explanation: An ATTACH for an MVS subtask has failed. The reason code is in GPR 15.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

System action: The task related user exit (TRUE) for this transaction is disabled. The transaction abends with an AEY9.

Operator response: Contact the CICS system programmer.

System programmer response: Determine the cause for the ATTACH failure and correct.

Module: EZACIC01

Destination: TASK RELATED USER EXIT (TRUE)

EZY1261I *mm/dd/yy hh:mm:ss* **EZACIC03 ATTACH SUCCESSFUL, TCB ADDRESS=** *tcbaddr* **TERM=***term* **TRAN=***tran* **TASK=***cicstask*

Explanation: An ATTACH for an MVS subtask was successful. This message is produced only for Listeners and for those tasks that cannot be accommodated within the pool of reusable tasks.

| **Result:** If you specify the character L as the last character in the subtask ID parameter of an INITAPI socket command, then the IP CICS Socket task related user exit (TRUE) assumes that the CICS transaction is a listener causing the TRUE to attach a new task to support the listener's socket commands.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

tcbaddr is the address of the Task Control Block (TCB) being attached.

term is the CICS terminal ID associated with the CICS transaction identified by *tran*.

tran is the name of the CICS transaction that was requested.

cicstask is the task number of the CICS transaction identified by *tran*.

System action: Processing continues.

Operator response: If this message happens frequently, increase the size of the reusable task pool, NTASKS, for this CICS. Increasing NTASKS appropriately will prevent overhead incurred with attaching the subtask. See "TYPE parameter" on page 54 for information the NTASKS value.

System programmer response: None.

Module: EZACIC01

Destination: TASK RELATED USER EXIT (TRUE)

EZY1262E *mm/dd/yy hh:mm:ss* **GWA ADDRESS INVALID UEPGAA=***xxxxxxxx* **TRAN=***tran* **TASK=***cicstask*

Explanation: The task related user exit (TRUE) detected an invalid GWA address.

System action: The TRUE is disabled and the task abends with an AEY9.

Operator response: Use EZAO to stop (immediate) and start the CICS socket interface. If the problem repeats, contact the IBM Software Support Center.

System programmer response: None.

Module: EZACIC01

Destination: TASK RELATED USER EXIT (TRUE)

EZY1263E *mm/dd/yy hh:mm:ss* **TIE ADDRESS INVALID UEPGAA=***xxxxxxxx* **TRAN=***tran* **TASK=***cicstask*

Explanation: The task related user exit (TRUE) detected an invalid TIE address.

System action: The TRUE is disabled and the task abends with an AEY9.

Operator response: Use EZAO to stop (immediate) and start the CICS socket interface. If the problem repeats, contact the IBM Software Support Center.

System programmer response: None.

Module: EZACIC01

Destination: TASK RELATED USER EXIT (TRUE)

EZY1264E *mm/dd/yy hh:mm:ss* **FLAG WORD ADDRESS INVALID UEPFLAGS=** *xxxxxxxx* **ERRNO=***errno* **TRAN=***tran* **TASK=***cicstask*

Explanation: The task related user exit (TRUE) detected an invalid flag word address.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

System action: The TRUE is disabled and the task abends with an AEY9.

Operator response: Use EZAO to stop (immediate) and start the CICS socket interface. If the problem repeats, contact the IBM Software Support Center.

System programmer response: None.

Module: EZACIC01

Destination: TASK RELATED USER EXIT (TRUE)

EZY1265E *mm/dd/yy hh:mm:ss* **CICS VERSION UNSUPPORTED GWACIVRM=***xxxx* **ERRNO=***errno* **TRAN=***tran* **TASK=***cicstask*

Explanation: The task related user exit (TRUE) detected a version of CICS which it does not support. The CICS version must be 3 or above.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

System action: The TRUE is disabled and the task abends with an AEY9.

Operator response: Contact the CICS system programmer.

System programmer response: The CICS socket interface requires CICS V3R3 or later.

Module: EZACIC01

Destination: TASK RELATED USER EXIT (TRUE)

EZY1267E *mm/dd/yy hh:mm:ss* **ROUTING TASK FUNCTION INVALID UERTIFD=xx ERRNO=***errno* **TRAN=***tran* **TASK=***cicstask*

Explanation: The task related user exit (TRUE) detected an invalid routing task function.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

System action: The TRUE is disabled and the task abends with an AEY9.

Operator response: If this happens repeatedly, use EZAO to STOP (immediate) the CICS socket interface and then START it. If it still happens, contact the IBM Software Support Center.

System programmer response: None.

Module: EZACIC01

Destination: TASK RELATED USER EXIT (TRUE)

EZY1268E *mm/dd/yy hh:mm:ss* **SAVE AREA ADDRESS INVALID UEPHSMA=** *xxxxxxxx* **ERRNO=***errno* **TRAN=***tran* **TASK=***cicstask*

Explanation: The task related user exit (TRUE) detected an invalid save area address.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

System action: The TRUE is disabled and the task abends with an AEY9.

Operator response: Contact the IBM Software Support Center.

System programmer response: None.

Module: EZACIC01

Destination: TASK RELATED USER EXIT (TRUE)

EZY1269E *mm/dd/yy hh:mm:ss* **PARM LIST ADDRESS INVALID GPR1=** *xxxxxxxx* **ERRNO=***errno* **TRAN=***tran* **TASK=***cicstask*

Explanation: The task related user exit (TRUE) detected an invalid parameter list on a call request from the CICS application program.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

System action: The TRUE is disabled and the task abends with an AEY9.

Operator response: Check the application program calls to the CICS socket interface to ensure that each call has the correct number and type of parameters.

System programmer response: None.

Module: EZACIC01

Destination: TASK RELATED USER EXIT (TRUE)

EZY1270E *mm/dd/yy hh:mm:ss* **PARM nn ADDRESS INVALID ADDRESS=** *xxxxxxxx* **ERRNO=***errno* **TRAN=***tran* **TASK=***cicstask*

Explanation: The task related user exit (TRUE) detected an invalid parameter address on a call request from the CICS application program. nn is the number of the parameter.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

System action: The TRUE is disabled and the task abends with an AEY9.

Operator response: Check the application program calls to the CICS socket interface to ensure that the parameter addresses are valid (not zero). This problem is most common in assembler language and C applications.

System programmer response: None.

Module: EZACIC01

Destination: TASK RELATED USER EXIT (TRUE)

EZY1271E *mm/dd/yy hh:mm:ss* **TOKERR=***xxxxxxxx* **ERRNO=***errno* **TRAN=***tran* **TASK=***cicstask*

Explanation: The task related user exit (TRUE) detected a token error on an internal token used to coordinate CICS transaction activity with TCP/IP activity.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

System action: The TRUE is disabled and the task abends with an AEY9.

Operator response: Contact the IBM Software Support Center.

System programmer response: None.

Module: EZACIC01

Destination: TASK RELATED USER EXIT (TRUE)

EZY1272E *mm/dd/yy hh:mm:ss* **INVALID SOCKET/FUNCTION CALL FUNCTION= xxxx ERRNO=***errno* **TRAN=***tran* **TASK=***cicstask*

Explanation: A call to EZASOKET specified in invalid function.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

System action: The TRUE is disabled and the task abends with an AEY9.

Operator response: Correct the call and retry.

System programmer response: None.

Module: EZACIC01

Destination: task related user exit (TRUE)

EZY1273E *mm/dd/yy hh:mm:ss* **IUCV SOCK/FUNC TABLE INVALID FUNCTION= xxxx ERRNO=***errno* **TRAN=***tran* **TASK=***cicstask*

Explanation: A call to EZACICAL specified a function that was not valid.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

System action: The TRUE is disabled and the task abends with an AEY9.

Operator response: Correct the call and retry.

System programmer response: None.

Module: EZACIC01

Destination: TASK RELATED USER EXIT (TRUE)

EZY1274E *mm/dd/yy hh:mm:ss* **INCORRECT EZASOKET PARM COUNT FUNCTION= xxxx ERRNO=***errno* **TRAN=***tran* **TASK=***cicstask*

Explanation: A call to EZASOKET specified in invalid number of parameters.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

System action: The TRUE is disabled and the task abends with an AEY9.

Operator response: Correct the call and retry.

System programmer response: None.

Module: EZACIC01

Destination: TASK RELATED USER EXIT (TRUE)

EZY1275E *mm/dd/yy hh:mm:ss* **MONITOR CALLS NOT SUPPORTED UERTFID=xx ERRNO=***errno* **TRAN=***tran* **TASK=***cicstask*

Explanation: The task related user exit (TRUE) detected a monitor call which is not supported for this version of CICS.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

System action: The TRUE is disabled and the task abends with an AEY9.

Operator response: Contact the IBM Software Support Center.

System programmer response: None.

Module: EZACIC01

Destination: TASK RELATED USER EXIT (TRUE)

EZY1276E *mm/dd/yy hh:mm:ss* **EDF CALLS NOT SUPPORTED UERTFID=xx ERRNO=***errno* **TRAN=***tran* **TASK=***cicstask*

Explanation: The task related user exit (TRUE) detected an EDF (Execute Diagnostic Facility) call. This TRUE does not support EDF calls.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

System action: The TRUE is disabled and the task abends with an AEY9.

Operator response: Contact the IBM Software Support Center.

System programmer response: None.

Module: EZACIC01

Destination: TASK RELATED USER EXIT (TRUE)

EZY1277I *mm/dd/yy hh:mm:ss* **EZACIC03 DETACHED TCB ADDRESS=***xxxxxxxx* **ERRNO=***errno* **TRAN=***tran* **TASK=***cicstask*

Explanation: An attached subtask is terminating.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

System action: The TRUE detaches the MVS subtask.

Operator response: None.

System programmer response: None.

Module: EZACIC01

Destination: TASK RELATED USER EXIT (TRUE)

EZY1278I *mm/dd/yy hh:mm:ss* **EZACIC03 DETACH SUCCESSFUL TCB ADDRESS=** *xxxxxxxx* **TRAN=***tran* **TASK=***cicstask*

Explanation: An attached subtask is terminating.

System action: The TRUE detaches the MVS subtask.

Operator response: None.

System programmer response: None.

Module: EZACIC01

Destination: TASK RELATED USER EXIT (TRUE)

EZY1279E *mm/dd/yy hh:mm:ss* **INVALID SYNC PT COMMAND DISP=xx TRAN=***tran* **TASK=***cicstask*

Explanation: The task related user exit (TRUE) Detected an invalid Sync Point command.

System action: Disable the TRUE and return to the caller.

Operator response: Contact the IBM Software Support Center.

System programmer response: None.

Module: EZACIC01

Destination: TASK RELATED USER EXIT (TRUE)

EZY1280E *mm/dd/yy hh:mm:ss* **INVALID RESYNC COMMAND DISP=xx TRAN=***tran* **TASK=***cicstask*

Explanation: The task related user exit (TRUE) Detected an invalid Resync command.

System action: Disable the TRUE and return to the caller.

Operator response: Contact the IBM Software Support Center.

System programmer response: None.

Module: EZACIC01

EZY1282E *mm/dd/yy hh:mm:ss* **10999 ABEND** *reasonxx*

Explanation: The ESTAE processing in EZACIC03 could not be completed because of *reasonxx*.

System action: Allow the ABEND to percolate.

Operator response: Contact the IBM Software Support Center. See the *CICS Application Programming Reference* for information about abend codes.

System programmer response: None.

Module: EZACIC03

Destination: MVS SUBTASK

EZY1285E *mm/dd/yy hh:mm:ss* **CICS/SOCKETS LISTENER TRANSACTION** *tran* **NOT ON CONFIGURATION FILE**

Explanation: The Listener attempting to start does not have a description record on the CICS Sockets configuration file.

System action: Listener terminates.

Operator response: Contact CICS system programmer.

System programmer response: Add the Listener to the configuration file using EZAC and retry.

Module: EZACIC02

Destination: LISTENER

EZY1286E *mm/dd/yy hh:mm:ss* **READ FAILURE ON CICS/SOCKETS CONFIGURATION FILE TRANSACTION=** *tran* **EIBRESP2= rrrrr**

Explanation: The Listener could not read the configuration file.

System action: Listener terminates.

Operator response: Contact CICS system programmer.

System programmer response: Use the CICS APR to interpret the value of EIBRESP2. If the file is not known to CICS, perform the installation steps for the configuration file.

See the *CICS Application Programming Reference* for information about EIBRESP2 values.

Module: EZACIC02

Destination: LISTENER

EZY1287E *mm/dd/yy hh:mm:ss* **EZYCIC02 GETMAIN FAILURE FOR VARIABLE STORAGE TRANSACTION=** *tran* **EIBRESP2=***rrrrr*

Explanation: EZACIC02 could not obtain the variable storage it requires to execute.

System action: Listener terminates.

Operator response: Contact CICS system programmer.

System programmer response: Use the CICS APR to interpret the value of EIBRESP2. Correct your CICS configuration as indicated.

See the *CICS Application Programming Reference* for information about EIBRESP2 values.

Module: EZACIC02

Destination: LISTENER

EZY1288E *mm/dd/yy hh:mm:ss* **CICS SOCKETS MODULE** *mmmmmmmm* **ABEND** *aaaa*

Explanation: An abend has occurred in module *mmmmmmmm* of the CICS socket interface.

System action: Listener terminates.

Operator response: See the *CICS Application Programming Reference* for information about abend codes. Contact the IBM Software Support Center.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1289E *mm/dd/yy hh:mm:ss* **CICS LISTENER TRANSACTION** *tran* **TERMINATING**

Explanation: The Listener is terminating. This could be a normal shutdown situation or a failure related to the Listener socket. If it is the latter, a previous message will describe the failure.

System action: Continue termination of the Listener.

Operator response: None.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1290I *mm/dd/yy hh:mm:ss* **LISTENER TRANSACTION** *tran* **STARTING**

Explanation: Transaction *tran*, Listener program EZACIC02 has been given control.

System action: Listener *tran* continues.

Operator response: None.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1291I *mm/dd/yy hh:mm:ss* **LISTENER TRANSACTION** *transactionid* **TASKID=** *taskno* **ACCEPTING REQUESTS VIA PORT** *port*

Explanation: The specified transaction can now receive connection requests on the specified port.

This message is issued when any of the following events occur:

- The listener is initialized and was able to connect to its TCP/IP.
- The listener reconnects to its TCP/IP after its TCP/IP has been restarted.
- The listener's socket descriptor table is no longer full and the table is now accepting client connections.

In the message text:

mm/dd/yy
The date (month/day/year) of the message.

hh:mm:ss
The time (hours:minutes:seconds) of the message.

transactionid
The name of the listener's transaction that can now accept new client connections.

taskno
The task number assigned by CICS.

port
The port number on which the listener identified by the *transactionid* value is listening.

Example:

```
EZY1291I 01/19/06 10:07:33 LISTENER TRANSACTION= CSKL TASKID= 0000079L ACCEPTING REQUESTS VIA PORT   3010
```

System action: The listener transaction continues.

Operator response: No action needed.

User response: None.

System programmer response: No action needed.

Problem determination: None.

Source: Not applicable.

Module: EZACIC02

Routing code: Not applicable.

Descriptor code: Not applicable.

EZY1292E *mm/dd/yy hh:mm:ss* **CANNOT START LISTENER, TRUE NOT ACTIVE TRANSACTION=** *tran* **TASKID=** *cicstask* **EIBRCODE BYTE3=rr**

Explanation: The initialization of the CICS socket interface did not complete successfully and this Listener cannot continue.

System action: Listener transaction *tran* terminates.

Operator response: If EZAO is being used to start the Listener, ensure that the CICS socket interface has successfully completed initialization first. If this happens during automatic initialization, look for other messages which would indicate why the initialization of the CICS socket interface failed.

See the *CICS Application Programming Reference* for information about EIBRCODEs.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1293E *mm/dd/yy hh:mm:ss* **INITAPI CALL FAILURE TRANSACTION=***tran* **TASKID=** *cicstask* **ERRNO=***errno*

Explanation: Listener transaction *tran* experienced a failure on the INITAPI call.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

System programmer response: None.

System action: Listener transaction *tran* terminates.

Operator response: Use the *errno* value to determine the cause of the failure.

Module: EZACIC02

Destination: LISTENER

EZY1294E *mm/dd/yy hh:mm:ss* **SOCKET CALL FAILURE TRANSACTION=** *tran* **TASKID=** *cicstask* **ERRNO=** *errno*

Explanation: Listener transaction *tran* experienced a failure on the SOCKET call.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

System programmer response: None.

System action: Listener transaction *tran* terminates.

Operator response: Use the *errno* value to determine the cause of the failure.

Module: EZACIC02

Destination: LISTENER

EZY1295E *mm/dd/yy hh:mm:ss* **BIND CALL FAILURE TRANSACTION=** *tran* **TASKID=** *cicstask* **ERRNO=** *errno*

Explanation: Listener transaction *tran* experienced a failure on the BIND call.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

System action: Listener transaction *tran* terminates.

Operator response: Use the *errno* value to determine the cause of the failure.

Notes:

1. An ERRNO=13 could indicate that the port and jobname specified in the PORT statement in *hlq*.TCPIP.PROFILE does not match the port and jobname used by the CICS Listener.
2. An ERRNO=48 could indicate that the port is not reserved in *hlq*.TCPIP.PROFILE.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1296E *mm/dd/yy hh:mm:ss* **LISTEN CALL FAILURE TRANSACTION=** *tran* **TASKID=** *cicstask* **ERRNO=** *errno*

Explanation: Listener transaction *tran* experienced a failure on the LISTEN call.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

System action: Listener transaction *tran* terminates.

Operator response: Use the *errno* value to determine the cause of the failure.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1297E *mm/dd/yy hh:mm:ss* **GETCLIENTID CALL FAILURE TRANSACTION=***tran* **TASKID=** *cicstask* **ERRNO=***errno*

Explanation: Listener transaction *tran* experienced a failure on the GETCLIENTID call.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

System action: Listener transaction *tran* terminates.

Operator response: Use the *errno* value to determine the cause of the failure.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1298E *mm/dd/yy hh:mm:ss* **CLOSE FAILURE TRANID=** *tran* **TASKID=** *cicstask* **ERRNO=** *errno*

Explanation: Listener transaction *tran* experienced a failure on the CLOSE call.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

System action: Listener transaction *tran* continues.

Operator response: Use the *errno* value to determine the cause of the failure.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1299E *mm/dd/yy hh:mm:ss* **SELECT CALL FAILURE TRANSACTION=** *tran* **TASKID= xxxxx ERRNO=** *errno*

Explanation: Listener transaction *tran* experienced a failure on the SELECT call.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

System action: Listener transaction *tran* terminates.

Operator response: Use the *errno* value to determine the cause of the failure.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1300E *mm/dd/yy hh:mm:ss* **RECV FAILURE TRANSID=** *transactionid* **TASKID=** *tasknumber* **ERRNO=** *errno* **INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: The Listener transaction *transactionid* experienced a failure on the RECV call.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the Listener transaction performing the RECV Socket.

tasknumber is the CICS task number assigned to the CICS transaction transactionid.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: The Listener transaction *transactionid* continues.

Operator response: Use the *errno* value to determine the cause of the failure.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1301E *mm/dd/yy hh:mm:ss* **CONNECTION CLOSED BY CLIENT TRANSACTION=** *transactionid* **PARTNER INET ADDR=** *ipaddr* **PORT=** *port*

Explanation: A remote client connected to the CICS Listener but then closed the connection before sending the entire amount of data required by the Listener as determined by the MINMSGL standard Listener configuration parameter or the MSGLEN enhanced Listener configuration parameter.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the transaction name of the CICS Listener.

ipaddr is the internet address of the remote client.

port is the port number of the remote client.

System action: The Listener transaction *transactionid* continues.

Operator response: Correct the client program.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1302I *mm/dd/yy hh:mm:ss* **READ TIMEOUT PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: The initial message from the client did not arrive within the read timeout value specified for this Listener in the CICS Sockets configuration file.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: The Listener closes the connection socket and does not attempt to start a server transaction.

Operator response: Determine the cause of the delay and correct it.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1303I *mm/dd/yy hh:mm:ss* **EZACIC02 GIVESOCKET TIMEOUT TRANS** *transactionid* **PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: The started server transaction did not perform the takesocket within the timeout value specified for this Listener in the CICS Sockets configuration file.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: Send an error message to the client and close the socket.

Operator response: Determine the reason for the delay in the server transaction. Possible causes are an overloaded CICS system or excessive processing in the server transaction before the takesocket is issued. Correct the situation and retry.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1304I *mm/dd/yy hh:mm:ss* **UNEXPECTED INPUT EVENT TRANSACTION** *transactionid* **PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: The Listener received data from the client after the end of the transaction input message.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: The Listener ignores this data.

Operator response: Ensure that the minimum message length specification for this Listener in the CICS Sockets Configuration file is correct. If it is, determine why the client is sending this additional data.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1305E *mm/dd/yy hh:mm:ss* **UNEXPECTED EXCEPTION EVENT TRANS** *transactionid* **PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: The Listener received an exception event on this connection other than the event showing a successful takesocket was issued by the server.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: Ignore the event.

Operator response: Ensure the client is not doing anything that would cause an exception event such the use of out-of-band data.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1306E *mm/dd/yy hh:mm:ss* **SECURITY EXIT** *mmmmmmmm* **IS NOT DEFINED TRANID=** *tran* **TASKID=***xxxxxxxx*

Explanation: The security exit specified for this Listener in the CICS Sockets configuration file is not defined to CICS.

System action: Close the socket and terminate the connection.

Operator response: Use CICS RDO to define the security exit.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1307E *mm/dd/yy hh:mm:ss* **MAXIMUM # OF SOCKETS USED TRANS=** *tran* **TASKID=** *cicstask* **ERRNO=** *errno*

Explanation: All of the sockets allocated to Listener transaction xxxx are in use.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

System action: The ACCEPT call is delayed until a socket is available.

Operator response: Use the EZAC transaction to increase the number of sockets allocated Listener *tran* and then stop and restart Listener transaction *tran*.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1308E *mm/dd/yy hh:mm:ss* **ACCEPT CALL FAILURE TRANSACTION=** *tran* **TASKID=** *cicstask* **ERRNO=** *errno*

Explanation: Listener transaction *tran* experienced a failure on the ACCEPT call.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

System action: Listener transaction *tran* terminates.

Operator response: Use the *errno* value to determine the cause of the failure.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1309E *mm/dd/yy hh:mm:ss* **GIVESOCKET FAILURE TRANS** *transactionid* **TASKID=***tasknumber* **ERRNO=***errno* **INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: The Listener transaction *transactionid* experienced a failure on the GIVESOCKET call.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction that was requested by the connecting client.

tasknumber is the CICS task number assigned to the CICS transaction transactionid.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: The Listener transaction *transactionid* terminates.

Operator response: Use the *errno* value to determine the cause of the failure.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1310E *mm/dd/yy hh:mm:ss* **IC VALUE NOT NUMERIC TRANID=***transactionid* **PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: The interval specified in the transaction input message contains one or more non-numeric characters.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: The interval is ignored, and the transaction is started immediately.

Operator response: Correct the client program which is sending this transaction input message.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1311E *mm/dd/yy hh:mm:ss* **CICS TRANID** *transactionid* **NOT AUTHORIZED PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: The transaction name specified in the transaction input message is not RSL authorized.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: The transaction is not started.

Operator response: Correct the CICS transaction definition if the transaction should be authorized or the client program if it is sending the wrong transaction name.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1312E *mm/dd/yy hh:mm:ss* **SECURITY EXIT** *mmmmmmmm* **CANNOT BE LOADED TRANID=** *tran* **TASKID=***cicstask*

Explanation: Listener transaction *tran* experienced a failure when it attempted to load security exit program *mmmmmmmm*.

System action: Listener transaction *tran* continues but the server transaction associated with this transaction input message is not started.

Operator response: Use CEMT to determine the status of the exit program and correct whatever problems are found.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1313E *mm/dd/yy hh:mm:ss* **LISTENER NOT AUTHORIZED TO ACCESS SECURITY EXIT** *mmmmmmmm* **TRANID=** *tran* **TASKID=***xxxxxxxx*

Explanation: Listener transaction *tran* is not authorized to access security exit program *mmmmmmmm*.

System action: Listener transaction *tran* continues but the server transaction associated with this transaction input message is not started.

Operator response: If the security exit program name is incorrect, use EZAC to correct the definition of this Listener on the CICS Sockets Configuration file. If the security exit program is correct, use the CICS RDO facility to authorize Listener transaction xxxx to use security exit program *mmmmmmmm*.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1314E *mm/dd/yy hh:mm:ss* **SECURITY EXIT** *mmmmmmmm* **IS DISABLED TRANID=** *tran* **TASKID=***xxxxxxxx*

Explanation: Security exit program *mmmmmmmm* is disabled.

System action: Listener transaction *tran* continues but the server transaction associated with this transaction input message is not started.

Operator response: Use CEMT to enable the security exit program.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1315E *mm/dd/yy hh:mm:ss* **INVALID TRANSID** *transactionid* **PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: The transaction input message from the client specified transaction *transactionid* but this transaction is not defined to CICS.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction that was requested by the connecting client. The *transactionid* field will be blank if no printable name was passed by the client or the security exit.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: The Listener transaction continues but the server transaction associated with this transaction input message is not started.

Operator response: If the transaction name is incorrect, correct the client program. If the transaction name is correct, correct the CICS transaction definition.

System programmer response: If *transactionid* is blank, then there is a possible mismatch because the Listener is expecting the first message segment to start with a transaction name but it does not. A packet trace might be helpful in determining whether there is such a mismatch. For example, if the packet trace shows that the first message segment starts with X'160300' or X'160301' then possibly a **clienthello** message was received, which indicates that there is an Application Transparent Transport Layer Security (AT-TLS) policy on the client side of the TCP connection but no matching AT-TLS policy (or AT-TLS is not enabled) on the Listener side of the TCP connection. This would need to be addressed by the AT-TLS administrator. See Application Transparent Transport Layer Security (AT-TLS) Data Protection in *z/OS Communications Server: IP Configuration Guide* and Diagnosing AT-TLS problems in *z/OS Communications Server: IP Diagnosis Guide* for more information.

Module: EZACIC02

Destination: LISTENER

EZY1316E *mm/dd/yy hh:mm:ss* **TRANSID** *transactionid* **IS DISABLED PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: Transaction *transactionid* is disabled.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: The Listener transaction continues but the server transaction associated with this transaction input message is not started.

Operator response: Use CEMT to enable the server transaction.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1317E *mm/dd/yy hh:mm:ss* **TRANSID** *transactionid* **IS NOT AUTHORIZED PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: The Listener transaction *transactionid* is not authorized to start the transaction name specified in the transaction input message.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: The transaction is not started.

Operator response: Authorize Listener transaction *transactionid* to start the transaction.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1318E *mm/dd/yy hh:mm:ss* **TD START SUCCESSFUL QUEUEID=** *que*

Explanation: The Listener transaction started a server transaction through transient data queue *que*

System action: Listener transaction continues and the server transaction is ready to start.

Operator response: None.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1319E *mm/dd/yy hh:mm:ss* **QIDERR FOR TD DESTINATION** *queuename* **PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: The Listener transaction was unable to start a CICS transaction through transient data queue *queuename*. DFHRESP was QIDERR.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

queuename is the name of the transient data queue that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: The Listener transaction continues.

Operator response: If the queue name is incorrect, correct the client program sending this transaction input message. If the queue name is correct, correct the CICS Destination Control Table.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1320E *mm/dd/yy hh:mm:ss* **I/O ERROR FOR TD DESTINATION** *queuename* **PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: The Listener transaction was unable to start a CICS transaction through transient data queue *queuename*. DFHRESP was IOERR.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

queuename is the name of the transient data queue that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: The Listener transaction continues.

Operator response: Contact the CICS system programmer.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1321E *mm/dd/yy hh:mm:ss* **LENGTH ERROR FOR TD DESTINATION** *queuename* **PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: The Listener transaction was unable to start a CICS transaction through transient data queue *queuename*. DFHRESP was LENGERR.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

queuename is the name of the transient data queue that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: The Listener transaction continues.

Operator response: Contact the CICS system programmer. The minimum length for this queue should be greater than 72.

System programmer response: Change definition of Transient Data Queue to accommodate length of this message.

Module: EZACIC02

Destination: LISTENER

EZY1322E *mm/dd/yy hh:mm:ss* **TD DESTINATION** *queuename* **DISABLED PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: The Listener transaction was unable to start a CICS transaction through transient data queue *queuename*. DFHRESP was DISABLED.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

queuename is the name of the transient data queue that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: The Listener transaction continues.

Operator response: Use CEMT to enable the destination.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1323E *mm/dd/yy hh:mm:ss* **TD DESTINATION** *queuename* **OUT OF SPACE PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: The Listener transaction was unable to start a CICS transaction through transient data queue *queuename*. DFHRESP was NOSPACE.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

queuename is the name of the transient data queue that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: The Listener transaction continues.

Operator response: Contact the CICS system programmer.

System programmer response: Allocate space for this Transient Data Queue.

Module: EZACIC02

Destination: LISTENER

EZY1324E *mm/dd/yy hh:mm:ss* **TD START FAILED QUEUE ID=***queuename* **PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: The Listener transaction was unable to start a CICS transaction through transient data queue *queuename*.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

queuename is the name of the transient data queue that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: The Listener transaction continues.

Operator response: Contact the CICS system programmer.

System programmer response: Determine the problem with the Transient Data Queue and correct it.

Module: EZACIC02

Destination: LISTENER

EZY1325I *mm/dd/yy hh:mm:ss* **START SUCCESSFUL TRANID=***transactionid* **PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: The Listener transaction was able to start a CICS transaction *transactionid* transient data queue.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: The Listener transaction continues.

Operator response: None.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1326E *mm/dd/yy hh:mm:ss* **START I/O ERROR TRANID=***transactionid* **PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: The Listener transaction was unable to start a CICS transaction *transactionid*. DFHRESP was IOERR.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: The Listener transaction continues.

Operator response: Contact the CICS system programmer.

System programmer response: Determine the cause of the I/O error and correct it.

Module: EZACIC02

Destination: LISTENER

EZY1327E *mm/dd/yy hh:mm:ss* **START TRANSACTION ID** *transactionid* **INVALID PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: The Listener transaction was unable to start a CICS transaction *transactionid*. DFHRESP was TRANSIDERR.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: The Listener transaction continues.

Operator response: Contact the CICS system programmer.

System programmer response: Check the transaction definition in RDO to ensure it is correct.

Module: EZACIC02

Destination: LISTENER

EZY1328E *mm/dd/yy hh:mm:ss* **START TRANSACTION ID** *transactionid* **NOT AUTHORIZED PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: The Listener transaction was unable to start a CICS transaction *transactionid*. DFHRESP was NOTAUTH.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: The Listener transaction continues.

Operator response: If the transaction ID is incorrect, correct the client program which sent this transaction input message. If the transaction ID is correct, authorize Listener transaction to start this transaction.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1329E *mm/dd/yy hh:mm:ss* **START FAILED (99) TRANSID=***transactionid* **PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: The Listener transaction was unable to start a CICS transaction *transactionid*. DFHRESP was 99.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: The Listener transaction continues.

Operator response: Contact the CICS system programmer.

System programmer response: Check the transaction definition in RDO. Look for associated messages in the MSGUSR queue, which might indicate why the transaction would not start.

Module: EZACIC02

Destination: LISTENER

EZY1330E *mm/dd/yy hh:mm:ss* **IC START SUCCESSFUL TRANID=***transactionid* **PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: The Listener transaction was able to start a CICS transaction *transactionid*.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: The Listener transaction continues.

Operator response: None.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1331E *mm/dd/yy hh:mm:ss* **IC START I/O ERROR TRANID=***transactionid* **PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: Listener transaction was unable to start a CICS transaction *transactionid*. DFHRESP was IOERR.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: Listener transaction continues.

Operator response: Contact the CICS system programmer.

System programmer response: Look for other messages in the MSGUSR queue, which provide specific information on the I/O error and correct the problem.

Module: EZACIC02

Destination: LISTENER

EZY1332E *mm/dd/yy hh:mm:ss* **IC START INVALID REQUEST TRANID=***transactionid* **PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: Listener transaction was unable to start a CICS transaction *transactionid*. DFHRESP was INVREQ.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: Listener transaction continues.

Operator response: Collect the messages written to the console and MSGUSR queue, client input data, and a SOCKAPI component trace and contact the IBM Software Support Center.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1333E *mm/dd/yy hh:mm:ss* **IC START FAILED TRANID=***transactionid* **PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: Listener transaction was unable to start a CICS transaction *transactionid*.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: Listener transaction continues.

Operator response: Contact the CICS system programmer.

System programmer response: Check the RDO definition of the transaction. Collect the messages written to the console and MSGUSR queue, client input data, and a SOCKAPI component trace and contact the IBM Software Support Center.

Module: EZACIC02

Destination: LISTENER

EZY1334E *mm/dd/yy hh:mm:ss* **INVALID USER TRANID=***transactionid* **PARTNER INET ADDR =** *inetaddress* **PORT =** *portnumber*

Explanation: This message indicates that the user security exit has given the Listener an invalid USERID field.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: The server transaction does not start.

Operator response: Correct the invalid USERID in the security exit.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1335E *mm/dd/yy hh:mm:ss* **WRITE FAILED ERRNO=***errno* **TRANID=***transactionid***. PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: Listener transaction had a failure on a WRITE command.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: The Listener transaction continues.

Operator response: Use the *errno* value to determine the cause of the failure.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1336E *mm/dd/yy hh:mm:ss* **TAKESOCKET FAILURE TRANS** *transactionid* **TASKID=***tasknumber* **ERRNO=***errno* **INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: The Listener transaction had a failure on a TAKESOCKET command.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: The Listener transaction continues.

Operator response: Use the *errno* value to determine the cause of the failure.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1337E *mm/dd/yy hh:mm:ss* **CICS IN QUIESCE, LISTENER TERMINATING TRANSID=** *tran* **TASKID=** *cicstask*

Explanation: Listener transaction *tran* is terminating because it detected a CICS quiesce in progress.

System action: Listener transaction *tran* terminates.

Operator response: None.

System programmer response: None.

Module: EZACIC02

Destination: LISTENER

EZY1338E *mm/dd/yy hh:mm:ss* **PROGRAM** *programname* **NOT FOUND TRANID=***transactionid* **PARTNER INET ADDR=***inetaddress* **PORT=***portnumber*

Explanation: The Listener checked the status of the program associated with the transaction. It was not found.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

programname is the name of the program which is associated with the transaction requested by the connecting client.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action: Listener continues.

Operator response: If *transactionid* is incorrect, correct the client program that sent the transaction input message. If the transaction ID is correct, check the transaction and program definitions in CICS.

System programmer response: None.

Module: EZACIC02

EZY1339E *mm/dd/yy hh:mm:ss* **EXIT PROGRAM (EZACIC01) IS NOT ENABLED. DISABLE IGNORED TERM=***term* **TRAN=***tranxxx*

Explanation: A termination of the CICS socket interface was requested but the interface is not enabled.

System action: The termination request is ignored.

Operator response: None.

System programmer response: None.

Module: EZACIC22

Destination: TERMINATION

EZY1340E *mm/dd/yy hh:mm:ss* **API ALREADY QUIESCING DUE TO PREVIOUS REQ. EZAO IGNORED TERM=***term* **TRAN=***tranxxx*

Explanation: A request for a quiesce of the CICS socket interface has been made but one is already is progress.

System action: Ignore the second request.

Operator response: None.

System programmer response: None.

Module: EZACIC22

Destination: TERMINATION

EZY1341E *mm/dd/yy hh:mm:ss* **API ALREADY IN IMMED MODE DUE TO PREV. REQ. EZAO IGNORED TERM=***term* **TRAN=***tranxxx*

Explanation: A request for an immediate of the CICS socket interface has been made but one is already is progress.

System action: Ignore the second request.

Operator response: None.

System programmer response: None.

Module: EZACIC22

Destination: TERMINATION

EZY1342I *mm/dd/yy hh:mm:ss* **DISABLE DELAYED UNTIL ALL USING TASKS COMPLETE TERM=***termid* **TRAN=***transid*

Explanation: A quiesce is in progress and is waiting for all outstanding CICS tasksto complete using the CICS socket interface.

When an IP CICS interface is being shut down the following actions occur:

- All listeners are posted to end.
- If the interface is configured as OTE=NO, then all non-listener tasks have their MVS subtask posted and their CICS task ends.

- If the interface is configured as OTE=YES, then any non-listener transaction that is running a blocking socket command is forced to end by a CICS FORCE PURGE action.

See the information about the “TYPE=CICS” on page 55 for information about the OTE configuration option.

In the message text:

mm/dd/yy
: The date (month/day/year) of the message.

hh:mm:ss
: The time (hours:minutes:seconds) of the message.

termid
: The CICS terminal ID on which the IP CICS socket shutdown is occuring.

transid
: The CICS transaction ID that requested that the IP CICS socket be shut down.

System action: The system continues to shut down.

Operator response: None.

System programmer response: None.

Module: EZACIC22

Destination: TERMINATION

EZY1343I *mm/dd/yy hh:mm:ss* **CICS/SOCKETS INTERFACE IMMEDIATELY DISABLED TERM=***term* **TRAN=***tranxxx*

Explanation: A request for immediate termination of the CICS socket interface has been successfully completed.

System action: Terminate the CICS socket interface.

Operator response: None.

System programmer response: None.

Module: EZACIC22

Destination: TERMINATION

EZY1344I *mm/dd/yy hh:mm:ss* **CICS/SOCKETS INTERFACE QUIESCENTLY DISABLED TERM=***term* **TRAN=***tranxxx*

Explanation: A request for deferred termination of the CICS socket interface has been successfully completed.

System action: Terminate the CICS socket interface.

Operator response: None.

System programmer response: None.

Module: EZACIC22

EZY1345E *mm/dd/yy hh:mm:ss* **CICS/SOCKETS WLM REGISTER FAILURE. RETURN CODE =** *return_code***, GROUP =** *groupname***, LISTNER =** *list*

Explanation: The CICS Listener received an error response when attempting to register WLM group with the Workload manager.

mm/dd/yy hh:mm:ss
: Date and time of the message.

return_code
: The return code from the WLM registration.

groupname
Name of the WLM group.

list Name of the CICS Listener.

System action: The Listener continues initialization but will not use *groupname* to participate in workload connection balancing.

Operator response: Verify that the WLM group name is correct and correctly defined to the Workload manager. If it is incorrect, either change it in the EZACICD TYPE=LISTENER macro that was used to define the Listener, or change it via the EZAC transaction. See the *z/OS MVS Programming: Workload Management Services* for more information about *return_code*.

System programmer response: None

Module: EZACIC12

EZY1346E *mm/dd/yy hh:mm:ss* **CICS SOCKETS WLM DEREGISTER FAILED RETURN CODE =** *return_code***, GROUP =** *groupname***, LISTNER =** *list*

Explanation: The CICS Listener received an error response when attempting to deregister WLM group with the Workload manager.

mm/dd/yy hh:mm:ss
Date and time of the message.

return_code
The return code from the WLM deregistration.

groupname
Name of the WLM group.

list Name of the CICS Listener.

System action: The Listener continues termination.

Operator response: See the *z/OS MVS Programming: Workload Management Services* for more information about *return_code*.

System programmer response: None.

Module: EZACIC12

EZY1347I *mm/dd/yy hh:mm:ss* **PROGRAM** *programname* **ASSUMED TO BE AUTOINSTALLED TRANID=***transactionid* **IP ADDR=***inetaddress* **PORT=***portnumber*

Explanation: The Listener checked the status of the program associated with the transaction. It was not found. Since program autoinstall is active in the CICS region, the Listener assumes that the program definition will automatically be installed by CICS.

mm/dd/yy
The date (month/day/year) of the message.

hh:mm:ss
The time (hours:minutes:seconds) of the message.

programname
The name of the undefined program which is associated with the transaction requested by the connecting client.

transactionid
The name of the transaction that was requested by the connecting client.

inetaddress
The internet address of the connecting client.

portnumber
The connecting client's port number.

System action: Listener continues.

Operator response: None.

System programmer response: Verify that the program name in the transaction definition is correct. Verify that the program is intended to be autoinstalled rather than explicitly defined in the PPT.

Module: EZACIC02

Destination: LISTENER

EZY1348E *mm/dd/yy hh:mm:ss* **INVALID SOCKET FUNCTION** *function* **ERRNO** *errno* **TRAN** *tranid* **TASK** *taskid*

Explanation: The task related user exit (TRUE) detected an invalid socket function on a call request from the CICS application program.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

function is the invalid socket function.

errno is the UNIX System Services return code. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

tranid is the name of the CICS transaction.

taskid is the CICS task ID number.

System action: The TRUE is disabled and the task abends with an AEY9 CICS abend code.

Operator response: Correct the invalid socket function and retry.

The most probable *errno* is 10011 ″INVALID SOCKET FUNCTION″. If the socket function name appears correct, ensure that the application padded the function call with blanks.

System programmer response: None.

Module: EZACIC01

Destination: Task Related User Exit (TRUE)

EZY1349E *mm/dd/yy hh:mm:ss* **UNABLE TO OPEN CONFIGURATION FILE TRANSACTION=***transactionid* **EIBRESP2=***eibresp2*

Explanation: The CICS Listener received an abnormal response from CICS when attempting to open the CICS Sockets configuration file (EZACONFG) using an EXEC CICS SET FILE call.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction under which the Listener is executing.

eibresp2 is the EIBRESP2 value returned by CICS on the EXEC CICS SET FILE call as described in *CICS System Programming Reference*.

System action: The Listener ends.

Operator response: Contact the CICS system programmer.

System programmer response: Use the *CICS System Programming Reference* to interpret the value of EIBRESP2. If the file is not known to CICS, perform the installation steps for the configuration file.

Module: EZACIC02

Destination: LISTENER

EZY1350E *mm/dd/yy hh:mm:ss* **NOT AUTHORIZED TO USE** *api_function***,** *action* **IGNORED. TERM=***termid* **TRAN=***transid*

Explanation: The IP CICS socket interface uses a CICS EXTRACT EXIT command to determine whether the IP CICS Sockets Task Related User Exit (TRUE) is enabled. This action is performed by IP CICS socket interface initialization and shutdown programs, the Listener, and by any user application linking to the IP CICS domain name server module.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

api_function is the CICS command performed.

action is the action intended.

- ENABLE means the IP CICS socket interface is being enabled.
- DISABLE means the IP CICS socket interface is being disabled.
- STARTUP means the IP CICS socket interface is being started.

termid is the terminal ID where the transaction receiving the error is executing.

transid is the name of the transaction that is incurring the security violation.

System action:

- If the TRUE is being enabled when the IP CICS socket interface is initializing, then the enable action is ignored and the interface is not activated.
- If the TRUE is being disabled when the IP CICS socket interface is shutting down, then the disable action is ignored and the interface remains active.
- If the IP CICS socket interface is being started, then the startup action is ignored and the interface remains inactive.

Operator response: Contact the CICS system programmer.

System programmer response: Ensure that the user ID being used is allowed at least UPDATE access to the EXITPROGRAM resource.

Module: EZACIC02, EZACIC21, EZACIC22

Destination: Listener, Initialization, Shutdown

EZY1351E *mm/dd/yy hh:mm:ss* **EXIT PROGRAM (EZACIC01) IS NOT ENABLED,** *action* **IGNORED. TERM=***termid* **TRAN=***transid*

Explanation: The IP CICS socket interface uses a CICS ENABLE PROGRAM command to enable the IP CICS Sockets Task Related User Exit (TRUE). This action is performed by IP CICS socket interface initialization.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

action is the action intended.

- ENABLE means the IP CICS socket interface is being enabled.
- DISABLE means the IP CICS socket interface is being disabled.

termid is the terminal ID where the transaction receiving the error is executing.

transid is the name of the transaction that is incurring the security violation.

System action: The IP CICS socket interface is not initialized.

Operator response: Contact the CICS system programmer.

System programmer response: Ensure that the user ID being used is allowed at least UPDATE access to the EXITPROGRAM resource.

Module: EZACIC21

Destination: Initialization

EZY1352E *mm/dd/yy hh:mm:ss* **SUBTASK ENDED UNEXPECTEDLY TRANSACTION=** *transactionid* **TASKID=** *taskid*

Explanation: The current tasks CICS Sockets subtask ended unexpectedly. This is probably caused by an ABEND of the subtask.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the CICS transaction whose subtask ended unexpectedly.

taskid is the CICS task number of the task whose subtask ended unexpectedly.

System action: The CICS socket interface is disabled for the current task. Any subsequent CICS Sockets calls by that task will result in CICS ABEND code AEY9. Other tasks are not affected.

Operator response: Contact the CICS system programmer.

System programmer response: Check the console log for previous messages that explain what happened to the subtask.

Module: EZACIC01

Destination: TASK RELATED USER EXIT (TRUE)

EZY1353E *mm/dd/yy hh:mm:ss* **COMMA MISSING AFTER IC TRANS ID =** *transactionid* **PARTNER IP ADDR =** *inetaddress* **PORT =** *portnumber*

Explanation: The listener did not find a comma delimiter after the interval control (IC) start type indicator in the client's transaction request message.

In the message text:

mm/dd/yy
: The date (month/day/year) of the message.

hh:mm:ss
: The time (hours:minutes:seconds) of the message.

transactionid
: The name of the transaction that was requested by the connecting client.

inetaddress
: The internet address of the connecting client.

portnumber
: The connecting client's port number.

Example: An example of a transaction request message for the standard listener:

```
SCCS,DATA,IC000010

EZY1258I 10/11/05 14:01:55 EZACIC02 ENTRY POINT IS 17CB2028
EZY1258I 10/11/05 14:01:55 EZACIC01 ENTRY POINT IS 177E2518
EZY1291I 10/11/05 14:01:56 LISTENER TRANSACTION= CSKL TASKID= 0000032L ACCEPTING REQUESTS VIA PORT   3010
EZY1353E 10/11/05 14:02:56 COMMA MISSING AFTER IC TRANSACTION ID= SCCS     PARTNER INET ADDR=10.1.1.2 PORT=    1076
```

System action: The listener does not start the transaction specified by the client's transaction request message and ends the connection. This message is also returned to the client.

Operator response: Ensure that a comma delimiter separates the IC start type and the IC start time. See "Listener input format" on page 135 for information about the client's transaction request message.

User response: Not applicable.

System programmer response: None.

Problem determination: Not applicable.

Source:

Module: EZACIC02

Routing code: Not applicable.

Descriptor code: Not applicable.

EZY1354I *mm/dd/yy hh:mm:ss* **CICS/SOCKETS CICS TRACING IS status**

Explanation: This message shows the status of changing IP CICS Sockets CICS tracing and is issued when one of the following occurs:

- The operator issued the EZAO,START,TRACE transaction.
- The operator issued the EZAO,STOP,TRACE transaction.
- The CICS Master User Trace Flag is specified as OFF and the IP CICS Sockets TRACE configuration is specified as YES.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

status is the status of CICS tracing for the IP CICS socket interface.

- ENABLED indicates that the IP CICS socket interface will generate CICS trace data when CICS tracing is active.
- DISABLED indicates that the IP CICS socket interface will not generate CICS trace data.

System action: When *status* is ENABLED, IP CICS Sockets will generate CICS trace data when CICS tracing is active. When *status* is DISABLED, IP CICS Sockets will not generate CICS trace data.

Operator response: None.

System programmer response: None.

Module: EZACIC00, EZACIC01

Destination: TRC00000, SUB05100

EZY1355I *mm/dd/yy hh:mm:ss* **CICS/SOCKETS TCBLIM EXCEEDS MAXOPENTCBS**

Explanation: IP CICS Sockets has determined that the value specified for TCBLIM exceeds the value of MAXOPENTCBS allowed at the time the interface was enabled. TCBLIM will be forced to the same value as MAXOPENTCBS.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

System action: IP CICS Sockets TCBLIM will default to the value of MAXOPENTCBS. IP CICS Sockets processing continues.

Operator response: Contact the CICS system programmer.

System programmer response: Adjust the value specified by the TCBLIM configuration option using one or more of the following methods:

- Specify an appropriate TCBLIM value on the EZACICD TYPE=CICS,TCBLIM= macro.
- Specify an appropriate TCBLIM value using the EZAC Configuration transaction.
- Specify an appropriate TCBLIM value dynamically by using the EZAO Operator transaction.
- Specify an appropriate MAXOPENTCBS value using the CICS System Initialization parameters.
- Specify an appropriate MAXOPENTCBS value using the CICS Master Terminal transaction, CEMT SET DISPATCHER MAXOPENTCBS.

Refer to the following sections:

- "Building the configuration data set with EZACICD" on page 51 for information about using the EZACICD macro.
- "Configuration transaction (EZAC)" on page 70 for information about the EZAC Configuration transaction.
- "SET function" on page 107 and "INQUIRE function" on page 105 for information about the EZAO Operator transaction.
- "TYPE parameter" on page 54 for a description of the TCBLIM parameter.

Refer to the *CICS System Definition Guide* for a description of the MAXOPENTCBS parameter. Refer to *CICS Supplied Transactions* for information about using the CEMT transaction.

Module: EZACIC21

Destination: Initialization

EZY1356E *mm/dd/yy hh:mm:ss* **CICS/SOCKETS TCBLIM HAS BEEN REACHED**

Explanation: The number of IP CICS Sockets-enabled CICS tasks using an Open API, L8, TCB is equal to the value specified by the TCBLIM configuration option.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

System action: The IP CICS socket interface will suspend any new tasks until one of the following actions occur:

- The IP CICS Sockets TCBLIM value is increased.
- Existing transactions using IP CICS Sockets end.

This message will be issued only when the interface detects that it has reached TCBLIM. EZY1360I will be issued when this condition is relieved.

Operator response: Contact the CICS system programmer.

System programmer response: Use the CICS Master Terminal transaction, CEMT INQ TASK HVALUE(ATTCBLIM), to determine which IP CICS Sockets-enabled CICS transactions are subject to TCBLIM. Either take action to reduce the IP CICS Sockets work load or increase the IP CICS Socket TCBLIM configuration option. You can use the EZAO,SET,CICS Operator transaction to dynamically increase TCBLIM. The new value you set for the TCBLIM configuration option must be less than or equal to the value specified by MAXOPENTCBS.

Module: EZACIC01

Destination: SUB16000

EZY1357I *mm/dd/yy hh:mm:ss* **TRANSIENT DATA QUEUE SPECIFIED ON ERRORTD IS NOT DEFINED TO CICS**

Explanation: IP CICS Sockets has determined that the CICS transient data queue specified by the ERRORTD configuration option was not defined to the CICS region where the IP CICS socket interface is enabled.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

System action: The CSMT transient data queue will be used for reporting all IP CICS Sockets interface messages. CSMT is the default CICS transient data queue name.

Operator response: Contact the CICS system programmer.

System programmer response: Ensure that the CICS transient data queue specified by the ERRORTD configuration option is properly defined to CICS.

See “Transient data definition” on page 36 for more information.

Module: EZACIC21

Destination: Initialization

EZY1358E **10999 ABEND - IP CICS SOCKETS USING OTE**

Explanation: IP CICS Sockets has incorrectly called the MVS subtask wrapper module when the interface was enabled to use CICS Open Transaction Environment.

System action: The IP CICS socket interface will stop.

Operator response: Contact the CICS system programmer.

System programmer response: Contact the IBM Software Support Center. See the *CICS Application Programming Reference* for information about abend codes.

Module: EZACIC03

Destination: MVS SUBTASK

EZY1359I *mm/dd/yy hh:mm:ss* **CICS/SOCKETS APPLICATIONS WILL USE THE QR TCB**

Explanation: IP CICS Sockets has determined that CICS FORCEQR=YES is specified.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

System action: CICS will force all user application programs, including those enabled to IP CICS Sockets, that are specified as threadsafe to run under the CICS Quasi-Reentrant (QR) TCB, as if they were specified as quasi-reentrant programs.

Operator response: Contact the CICS system programmer.

System programmer response: If you do not want to incur the overhead of CICS switching Open API-enabled tasks back to the QR TCB, then change the value of FORCEQR to NO. Refer to the *CICS System Definition Guide* for more information about the FORCEQR CICS System Initialization parameter. Refer to *CICS Supplied Transactions* for more information about the CICS Master Terminal transaction that is used to dynamically change the FORCEQR setting.

Module: EZACIC21

Destination: Initialization

EZY1360I *mm/dd/yy hh:mm:ss* **CICS/SOCKETS TCBLIM CONDITION HAS BEEN RELIEVED**

Explanation: IP CICS Sockets enable transactions are no longer suspended due to TCBLIM.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

System action: Any new or suspended IP CICS Sockets work will now be processed without being suspended due to IP CICS Sockets being at TCBLIM.

Operator response: None.

System programmer response: None.

Module: EZACIC01

Destination: SUB16000, Task termination

EZY1361E *mm/dd/yy hh:mm:ss* **CICS/TS OPEN TRANSACTION ENVIRONMENT SUPPORT IS NOT AVAILABLE**

Explanation: The IP CICS Sockets OTE configuration parameter is specified as YES. IP CICS Sockets determined that the CICS environment that is required to support the exploitation of CICS Open Transaction Environment by IP CICS Sockets is not available.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

System action: The IP CICS socket interface is not enabled to use CICS Open Transaction Environment.

Operator response: Contact the system programmer.

System programmer response: Perform one of the following:

- Upgrade the level of CICS to support Open Transaction Environment. The CICS Open Transaction Environment requires CICS/TS V2R2 or later.
- Change the IP CICS socket interface configuration to use MVS subtasks when configuring it by using the EZAC configuration transaction or the EZACICD macro.

Module: EZACIC21

Destination: Initialization

EZY1362E *mm/dd/yy hh:mm:ss* **CICS/SOCKETS START OF LISTENER** *transactionid* **FAILED RESP1=** *resp1* **RESP2=***resp2*

Explanation: CICS Sockets attempted to start the specified listener, but the EXEC CICS START command failed with the RESP1 and RESP2 values listed in the message text.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the transaction name of the listener that the CICS Sockets attempted to start.

resp1 is the RESP1 value returned by the EXEC CICS START transaction.

resp2 is the RESP2 value returned by the EXEC CICS START transaction.

System action: The CICS Listener does not start.

Operator response: None.

System programmer response: Refer to the description of the START command in the *CICS Application Programming Reference* for information about why the START command failed.

- If the RESP2 value is 8 or 9, then the problem is related to the USERID parameter in the definition of the listener. Verify that the USERID parameter is correct. See Chapter 2, "Setting up and configuring CICS TCP/IP," on page 23 for a description of the USERID parameter.
- If the RESP2 value is 8, then the USERID parameter of the listener definition specifies a user ID that is not known to RACF. Therefore, either change the USERID parameter or define the user ID to RACF.
- If the RESP2 value is 9, then the user ID under which the EXEC CICS START was issued does not have SURROGAT security access to the user ID that is specified in the USERID parameter. For example, if the failure occurs during CICS PLT processing, then the PLT user ID does not have SURROGAT security access to the listener's user ID. Refer to the *CICS RACF Security Guide* for more information.

Module: EZACIC21

Destination: INITIALIZATION

EZY1363I *mm/dd/yy hh:mm:ss* **LISTENER** *transactionid taskno* **HAD** *threads* **THREADS ACTIVE WHEN STACK** *tcpname* **ENDED**

Explanation: This message displays the number of listener threads that were active when the TCP/IP stack that is specified ended. This message is followed by one or more EZY1368I messages that describe the clients that are affected.

In the message text:

mm/dd/yy
: The date (month/day/year) of the message.

hh:mm:ss
: The time (hours:minutes:seconds) of the message.

transactionid
: The listener's transaction ID.

taskno
: The task number assigned by CICS.

threads
: The number of threads that were active when the specified TCP/IP stack ended.

tcpname
: The TCP/IP procedure name with which the listener had affinity.

Example: Following is an example of the messages that are displayed when the stack has ended while the listener was processing data.

```
EZY1369E 01/10/06 12:59:32 LISTENER CSKL 10295 IS DELAYED, STACK TCPCS IS UNAVAILABLE
EZY1363I 01/10/06 12:59:33 LISTENER CSKL 10295 HAD 5 THREADS ACTIVE WHEN STACK TCPCS ENDED
EZY1367I 01/10/06 12:59:33 SOCK# IP ADDRESS                                    PORT  CHILD
```

```
EZY1368I 01/10/06 12:59:33     2 10.11.1.2                              10245 PAYR
EZY1368I 01/10/06 12:59:33    12 2001:DB8:10::11:2:1                    21089
EZY1368I 01/10/06 12:59:33    15 10.91.1.1                              10245 INVN
EZY1368I 01/10/06 12:59:33    19 10.81.1.1                              21212 ACCT
EZY1368I 01/10/06 12:59:33   999 2001:DB8:10::11:1:2                    00901 ORDR
```

System action: Processing continues.

Operator response: No action needed.

User response: No action needed.

System programmer response: No action needed.

Problem determination: Not applicable.

Source: z/OS Communications Server TCP/IP: CICS Socket Interface and API

Module: EZACIC02

Routing code: 10

Descriptor code: 12

Automation: This message is sent to the CICS transient data queue that is specified by the IP CICS Sockets ERRORTD configuration option.

EZY1364I *mm/dd/yy hh:mm:ss* **LISTENER** *transactionid* **DETECTED THAT TTLS IS** *status* **ON STACK** *tcpname*

Explanation: The CICS Listener is defined with a GETTID parameter of YES which indicates that the listener is requested to attempt to obtain the connecting client certificates and user IDs from Application Transparent Transport Layer Security (AT-TLS). If status is DISABLED, then AT-TLS is disabled in the TCP/IP stack. Therefore, the listener is unable to obtain client certificates and user IDs as requested by the GETTID parameter. If status is ENABLED, then AT-TLS has been enabled in the TCP/IP stack, making it possible for the listener to obtain client certificates and user IDs.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the listeners CICS transaction.

status is the status of AT-TLS in the TCP/IP stack. *status* is either DISABLED or ENABLED.

tcpname is the name of the TCP/IP stack.

System action: The listener continues its normal processing, which includes attempting to obtain client certificates and User IDs.

Operator response: Contact the system programmer.

System programmer response: No response is needed if status is ENABLED. If status is DISABLED, then verify that the GETTID parameter of YES is correct in the listener definition. If so, request that your AT-TLS administrator investigate why AT-TLS is not enabled in the TCP/IP stack. See Chapter 2, “Setting up and configuring CICS TCP/IP,” on page 23 for a description of the GETTID parameter.

See Application Transparent Transport Layer Security (AT-TLS) Data Protection in *z/OS Communications Server: IP Configuration Guide* and Diagnosing AT-TLS problems in *z/OS Communications Server: IP Diagnosis Guide* for more information.

Module: EZACIC02

Destination: LISTENER

EZY1365E *mm/dd/yy hh:mm:ss* **LISTENER** *transactionid taskno* **IS NOT ACCEPTING REQUESTS ON PORT** *port*

Explanation: The listener identified by the specified transaction ID and task number cannot process inbound connections because the listener’s socket descriptor table is full.

In the message text:

mm/dd/yy
The date (month/day/year) of the message.

hh:mm:ss
The time (hours:minutes:seconds) of the message.

transactionid
The name of the listener's transaction that cannot accept new client connections.

taskno
The task number assigned by CICS.

port
The port number on which the specified listener is listening.

Example:

```
EZY1365E 01/19/06 10:07:33 LISTENER CSKL 0000079 IS NOT ACCEPTING REQUESTS AT  PORT   3010
```

System action: The listener does not accept new connections until the number of socket descriptors currently being processed by the listener is less than the value specified by the lesser of either the system MAXFILEPROC parameter or the listener user ID's FILEPROCMAX parameter.

Operator response: Contact the system programmer.

User response: No action needed.

System programmer response: Perform any of the following actions as appropriate:

- If the ERRORTD log indicates that the child server transaction failed to take the client's given socket, then investigate the CICS region where the child server transaction runs.

  See the steps for diagnosing TCP/IP clients that are unable to connect in *z/OS Communications Server: IP Diagnosis Guide* for information about diagnosing child server transactions problems.

  See *CICS Problem Determination Guide* for information about CICS/TS problems.
- If the listeners NUMSOCK value is greater than or equal to the value specified by the MAXFILEPROC parameter, then perform one of the following actions:
  - Set the NUMSOCK value to be less than the MAXFILEPROC value using either the EZACICD macro or the EZAC configuration transaction and then restart the listener. See the information about "Configuring the CICS TCP/IP environment" on page 51 for more information about using the EZACICD macro and the EZAC configuration transaction.
  - Set the MAXFILEPROC value to be greater than the NUMSOCK value using the SETOMVS system command. See the SETOMVS command information in *z/OS MVS System Commands* for information about dynamically changing the MAXFILEPROC option that z/OS UNIX System Services is currently using.
- If the listener user ID FILEPROCMAX value is less than the value specified by the NUMSOCK parameter, set the FILEPROCMAX value to be greater than the value specified by the NUMSOCK parameter. For more information about the FILEPROCMAX specification, see the documentation provided for the SAF product that is in use on your system. If you are using RACF, see the information about the FILEPROCMAX parameter in the *z/OS Security Server RACF Security Administrator's Guide*.

Problem determination: See the system programmer response.

Source: z/OS Communications Server TCP/IP: CICS Socket Interface and API

Module: EZACIC02

Routing code: 1

Descriptor code: 2

Automation: This message is sent to the system console and to the CICS transient data queue that is specified by the IP CICS Sockets ERRORTD configuration option.

EZY1366E *mm/dd/yy hh:mm:ss* **CICS/SOCKETS LISTENER TRANSACTION** *tranid* **IS ALREADY ACTIVE**

Explanation: The IP CICS Sockets Listener determined that another listener with the same transaction ID is already active.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

tranid is the CICS transaction identifier of the duplicate IP CICS Sockets Listener.

System action: The IP CICS Sockets Listener that issued this message ends.

Operator response: Contact the system programmer.

System programmer response: Change the Listeners CICS transaction identifier or port number to ensure that the definition is unique. See Chapter 2, "Setting up and configuring CICS TCP/IP," on page 23 for more information about configuring the IP CICS Sockets Listener.

Module: EZACIC02

Destination: Initialization

EZY1367I *mm/dd/yy hh:mm:ss* **SOCK# IP ADDRESS PORT CHILD**

Explanation: The listener was processing client connections when its TCP/IP stack ended. This message is issued when the listener has accepted sockets that were not taken by child server tasks. This message is a header message for the EZY1368I detail messages that follow. This message accompanies an EZY1363I message.

In the message text:

mm/dd/yy
: The date (month/day/year) of the message.

hh:mm:ss
: The time (hours:minutes:seconds) of the message.

Example: Following is an example of the messages displayed when the stack has ended while the listener was processing data.

```
EZY1369E 01/10/06 12:59:32 LISTENER CSKL 10295 IS DELAYED, STACK TCPCS IS UNAVAILABLE
EZY1363I 01/10/06 12:59:33 LISTENER CSKL 10295 HAD 5 THREADS ACTIVE WHEN STACK TCPCS ENDED
EZY1367I 01/10/06 12:59:33 SOCK# IP ADDRESS                              PORT  CHILD
EZY1368I 01/10/06 12:59:33     2 10.11.1.2                               10245 PAYR
EZY1368I 01/10/06 12:59:33    12 2001:DB8:10::11:2:1                     21089
EZY1368I 01/10/06 12:59:33    15 10.91.1.1                               10245 INVN
EZY1368I 01/10/06 12:59:33    19 10.81.1.1                               21212 ACCT
EZY1368I 01/10/06 12:59:33   999 2001:DB8:10::11:1:2                     00901 ORDR
```

System action: Processing continues.

Operator response: No action needed.

User response: No action needed.

System programmer response: No action needed.

Problem determination: Not applicable.

Source: z/OS Communications Server TCP/IP: CICS Socket Interface and API

Module: EZACIC02

Routing code: 10

Descriptor code: 12

Automation: This message is sent to the CICS transient data queue that is specified by the IP CICS Sockets ERRORTD configuration option.

EZY1368I *mm/dd/yy hh:mm:ss sock# ipaddr port tran*

Explanation: The listener was processing client connections when its TCP/IP stack ended. This message is issued when the listener has accepted sockets that were not taken by child server tasks. One EZY1368I message is issued for each client connection that is being processed.

In the message text:

mm/dd/yy
The date (month/day/year) of the message.

hh:mm:ss
The time (hours:minutes:seconds) of the message.

sock#
The listener's socket number.

ipaddr
The client's IP address.

port
The client's port number.

tran
The child server's transaction ID. A blank child server transaction ID indicates that the ID has not yet been determined.

Example: Following is an example of the messages displayed when the stack has ended while the listener was processing data.

```
EZY1369E 01/10/06 12:59:32 LISTENER CSKL 10295 IS DELAYED, STACK TCPCS IS UNAVAILABLE
EZY1363I 01/10/06 12:59:33 LISTENER CSKL 10295 HAD 5 THREADS ACTIVE WHEN STACK TCPCS ENDED
EZY1367I 01/10/06 12:59:33 SOCK# IP ADDRESS                             PORT  CHILD
EZY1368I 01/10/06 12:59:33     2 10.11.1.2                              10245 PAYR
EZY1368I 01/10/06 12:59:33    12 2001:DB8:10::11:2:1                    21089
EZY1368I 01/10/06 12:59:33    15 10.91.1.1                              10245 INVN
EZY1368I 01/10/06 12:59:33    19 10.81.1.1                              21212 ACCT
EZY1368I 01/10/06 12:59:33   999 2001:DB8:10::11:1:2                    00901 ORDR
```

System action: Processing continues.

Operator response: No action needed.

User response: No action needed.

System programmer response: No action needed.

Problem determination: Not applicable.

Source: z/OS Communications Server TCP/IP: CICS Socket Interface and API

Module: EZACIC02

Routing code: 10

Descriptor code: 12

Automation: This message is sent to the CICS transient data queue that is specified by the IP CICS Sockets ERRORTD configuration option.

EZY1369E *mm/dd/yy hh:mm:ss* **LISTENER** *transactionid taskno* **IS DELAYED, STACK** *tcpname* **IS UNAVAILABLE.**

Explanation: The TCP/IP stack assigned to the specified listener is not active.

In the message text:

mm/dd/yy
The date (month/day/year) of the message.

hh:mm:ss
The time (hours:minutes:seconds) of the message.

transactionid
The listener's transaction ID.

taskno
The task number assigned by CICS.

tcpname
The TCP/IP procedure name with which the listener had affinity.

Example: The following is an example of the messages displayed when the stack has ended while the listener was processing data.

```
EZY1369E 01/10/06 12:59:32 LISTENER CSKL 10295 IS DELAYED, STACK TCPCS IS UNAVAILABLE
EZY1363I 01/10/06 12:59:33 LISTENER CSKL 10295 HAD 5 THREADS ACTIVE WHEN STACK TCPCS ENDED
EZY1367I 01/10/06 12:59:33 SOCK# IP ADDRESS                              PORT  CHILD
EZY1368I 01/10/06 12:59:33     2 10.11.1.2                               10245 PAYR
EZY1368I 01/10/06 12:59:33    12 2001:DB8:10::11:2:1                     21089
EZY1368I 01/10/06 12:59:33    15 10.91.1.1                               10245 INVN
EZY1368I 01/10/06 12:59:33    19 10.81.1.1                               21212 ACCT
EZY1368I 01/10/06 12:59:33   999 2001:DB8:10::11:1:2                     00901 ORDR
```

System action: The listener releases any resources and connects to the TCP/IP stack specified by the *tcpname* value. If the connection fails because the stack is not active, then the listener delays using the time value specified by its RTYTIME configuration option and attempts to reconnect. See the "TYPE=LISTENER" on page 59 for information about setting the listener's RTYTIME value.

Operator response: Start or restart the TCP/IP address space specified by the *tcpname* value.

User response: No action needed.

System programmer response: No action needed.

Problem determination: Not applicable.

Source: z/OS Communications Server TCP/IP: CICS Socket Interface and API

Module: EZACIC02

Routing code: 1

Descriptor code: 2

Automation: This message is sent to the system console and to the CICS transient data queue that is specified by the IP CICS Sockets ERRORTD configuration option.

EZY1370I *mm/dd/yy hh:mm:ss* **LISTENER** *transactionid* **NUMSOCK** *numsock* **IS EQUAL TO OR GREATER THAN MAXFILEPROC** *maxfileproc*

Explanation: A listener startup run-time check determined that the z/OS UNIX System Services MAXFILEPROC value was less than or equal to the listener's NUMSOCK value. The listener's accept processing pauses when the number of sockets that are supported by this listener exceeds the MAXFILEPROC value. No new connections are accepted until the number of sockets that are supported by this listener is less than the MAXFILEPROC value.

In the message text:

mm/dd/yy
: The date (month/day/year) of the message.

hh:mm:ss
: The time (hours:minutes:seconds) of the message.

transactionid
: The listener's transaction ID.

numsock
: The number of sockets supported by this listener.

maxfileproc
: The maximum number of descriptors for files, sockets, directories, and any other file-system objects that can be concurrently active or allocated by a single process.

Example:

```
EZY1370I 01/19/06 10:07:33 LISTENER CSKL NUMSOCK 2000 IS EQUAL TO OR GREATER  THAN MAXFILEPROC 250
```

System action: Processing continues.

Operator response: Contact the system programmer.

User response: No action needed.

System programmer response: Perform one of the following actions:

- Set the NUMSOCK value to be less than the MAXFILEPROC value using either the EZACICD macro or the EZAC configuration transaction, and then restart the listener. See the information about "Configuring the CICS TCP/IP environment" on page 51 for more information about using the EZACICD macro and the EZAC configuration transaction.
- Set the MAXFILEPROC value to be greater than the NUMSOCK value using the SETOMVS system command. See the SETOMVS command information in *z/OS MVS System Commands* for information about dynamically changing the MAXFILEPROC option that z/OS UNIX System Services is currently using.

Problem determination: Not applicable.

Source: z/OS Communications Server TCP/IP: CICS Socket Interface and API

Module: EZACIC21

Routing code: 10

Descriptor code: 12

Automation: This message is sent to the CICS transient data queue that is specified by the IP CICS Sockets ERRORTD configuration option.

EZY1371E *mm/dd/yy hh:mm:ss* **AUTOMATIC APPLDATA REGISTRATION FAILED FOR TRANSACTION=** *transactionid* **TASKNO=** *taskno* **ERRNO=** *errno*

Explanation: The automatic registration of application data failed for the reason described by the *errno* value.

In the message text:

mm/dd/yy
The date (month/day/year) of the message.

hh:mm:ss
The time (hours:minutes:seconds) of the message.

transactionid
The listener's transaction ID.

taskno
The task number assigned by CICS.

errno
The UNIX System Services return code for the SIOCSAPPLDATA IOCTL socket command. These return codes are listed and described in the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes*.

Example:

```
EZY1371E 07/01/06 10:07:33 AUTOMATIC APPLDATA REGISTRATION FAILED FOR
     TRANSACTION= CSKL TASKNO= 00000022L ERRNO= 55
```

System action: The application continues.

Operator response: Contact the system programmer.

User response: Not applicable.

System programmer response: See the information about automatically registering application data in *z/OS Communications Server: IP Configuration Reference* for information about the socket commands affected by the automatic registration of application data. See the return codes (errnos) information in *z/OS UNIX System Services Messages and Codes* for the action that you should take based on the SIOCSAPPLDATA IOCTL socket command return code.

Problem determination: See the system programmer response.

Source: z/OS Communications Server TCP/IP: CICS Socket Interface and API

Module: EZACIC01, EZACIC02

Routing code: 10

Descriptor code: 12

| **Automation:** This message is sent to the CICS transient data queue that is specified by the IP CICS Sockets
| ERRORTD configuration option.

Appendix E. Sample programs

This topic contains the following samples:

- EZACICSC - An IPv4 child server, see EZACICSC
- EZACICSS - An IPv4 iterative server, see EZACICSS
- EZACIC6C - An IPv6 child server, see EZACIC6C
- EZACIC6S - An IPv6 iterative server, see EZACIC6S
- EZACICAC - An assembler child server, see EZACICAC
- EZACICAS - An assembler iterative server, see SELECTEX

EZACICSC

The following COBOL socket program is in the SEZAINST data set.


```
*************************************************************
*                                                           *
* Communications Server for z/OS,   Version 1, Release 9    *
*                                                           *
*                                                           *
* Copyright:    Licensed Materials - Property of IBM        *
*                                                           *
*               "Restricted Materials of IBM"               *
*                                                           *
*               5694-A01                                    *
*                                                           *
*               Copyright IBM Corp. 1993, 2007              *
*                                                           *
*               US Government Users Restricted Rights -     *
*               Use, duplication or disclosure restricted by *
*               GSA ADP Schedule Contract with IBM Corp.    *
*                                                           *
* Status:       CSV1R9                                      *
*                                                           *
* $MOD(EZACICSC),COMP(CICS),PROD(TCPIP):                    *
*                                                           *
*************************************************************
* $SEG(EZACICSC)
*--------------------------------------------------------------*
*                                                              *
*   Module Name : EZACICSC                                     *
*                                                              *
*   Description :                                              *
*                                                              *
*      This is a sample CICS/TCP application program. It issues*
*      TAKESOCKET to obtain the socket passed from MASTER      *
*      SERVER and perform dialog function with CLIENT program. *
*                                                              *
*--------------------------------------------------------------*
*
 IDENTIFICATION DIVISION.
 PROGRAM-ID. EZACICSC.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
*
 WORKING-STORAGE SECTION.
 77  TASK-START                     PIC X(40)
      VALUE IS 'TASK STARTING THRU CICS/TCPIP INTERFACE '.
 77  TAKE-ERR                       PIC X(24)
      VALUE IS ' TAKESOCKET FAIL       '.
 77  TAKE-SUCCESS                    PIC X(24)
      VALUE IS ' TAKESOCKET SUCCESSFUL '.
 77  READ-ERR                        PIC X(24)
      VALUE IS ' READ SOCKET FAIL       '.
```

Figure 176. EZACICSC IPv4 child server sample (Part 1 of 8)

```
77  READ-SUCCESS                    PIC X(24)
     VALUE IS ' READ SOCKET SUCCESSFUL '.
77  WRITE-ERR                       PIC X(24)
     VALUE IS ' WRITE SOCKET FAIL      '.
77  WRITE-END-ERR                       PIC X(32)
     VALUE IS ' WRITE SOCKET FAIL - PGM END MSG'.
77  WRITE-SUCCESS                   PIC X(25)
     VALUE IS ' WRITE SOCKET SUCCESSFUL '.
77  CLOS-ERR                        PIC X(24)
     VALUE IS ' CLOSE SOCKET FAIL      '.
77  CLOS-SUCCESS                   PIC X(24)
     VALUE IS 'CLOSE SOCKET SUCCESSFUL '.
77  INVREQ-ERR                     PIC X(24)
     VALUE IS 'INTERFACE IS NOT ACTIVE '.
77  IOERR-ERR                     PIC X(24)
     VALUE IS 'IOERR OCCURRS           '.
77  LENGERR-ERR                   PIC X(24)
     VALUE IS 'LENGERR ERROR           '.
77  ITEMERR-ERR                   PIC X(24)
     VALUE IS 'ITEMERR ERROR           '.
77  NOSPACE-ERR                   PIC X(24)
     VALUE IS 'NOSPACE CONDITION       '.
77  QIDERR-ERR                   PIC X(24)
     VALUE IS 'QIDERR  CONDITION       '.
77  ENDDATA-ERR                  PIC X(30)
     VALUE IS 'RETRIEVE DATA CAN NOT BE FOUND'.
77  WRKEND                       PIC X(20)
     VALUE 'CONNECTION END      '.
77  WRITE-SW                     PIC X(1)
     VALUE 'N'.
77  FORCE-ERROR-MSG              PIC X(1)
     VALUE 'N'.
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-GETCLIENTID     PIC X(16) VALUE 'GETCLIENTID     '.
    02 SOKET-GETHOSTBYADDR   PIC X(16) VALUE 'GETHOSTBYADDR   '.
    02 SOKET-GETHOSTBYNAME   PIC X(16) VALUE 'GETHOSTBYNAME   '.
    02 SOKET-GETHOSTID       PIC X(16) VALUE 'GETHOSTID       '.
    02 SOKET-GETHOSTNAME     PIC X(16) VALUE 'GETHOSTNAME     '.
    02 SOKET-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-IOCTL           PIC X(16) VALUE 'IOCTL           '.
    02 SOKET-LISTEN          PIC X(16) VALUE 'LISTEN          '.
    02 SOKET-READ            PIC X(16) VALUE 'READ            '.
    02 SOKET-RECV            PIC X(16) VALUE 'RECV            '.
    02 SOKET-RECVFROM        PIC X(16) VALUE 'RECVFROM        '.
    02 SOKET-SELECT          PIC X(16) VALUE 'SELECT          '.
    02 SOKET-SEND            PIC X(16) VALUE 'SEND            '.
```

Figure 176. EZACICSC IPv4 child server sample (Part 2 of 8)

```
           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           '.
       01  WRKMSG.
           02 WRKM                         PIC X(14)
              VALUE IS 'DATA RECEIVED '.
      *----------------------------------------------------------------*
      *    program's variables                                         *
      *----------------------------------------------------------------*
       77  SUBTRACE                PIC X(8)  VALUE 'CONTRACE'.
       77  RESPONSE                       PIC 9(9) COMP.
       77  TASK-FLAG                      PIC X(1) VALUE '0'.
       77  TAKE-SOCKET                    PIC 9(8) COMP.
       77  SOCKID                         PIC 9(4) COMP.
       77  SOCKID-FWD                     PIC 9(8) COMP.
       77  ERRNO                          PIC 9(8) COMP.
       77  RETCODE                        PIC S9(8) COMP.
       77  AF-INET                        PIC 9(8) COMP VALUE 2.
       01  TCP-BUF.
           05 TCP-BUF-H                   PIC X(3) VALUE IS SPACES.
           05 TCP-BUF-DATA                PIC X(197) VALUE IS SPACES.
       77  TCPLENG                        PIC 9(8) COMP.
       77  RECV-FLAG                      PIC 9(8) COMP.
       77  CLENG                          PIC 9(4) COMP.
       77  CNT                            PIC 9(4) COMP.
       01  ZERO-PARM                  PIC X(16) VALUE LOW-VALUES.
       01  DUMMY-MASK REDEFINES ZERO-PARM.
           05 DUMYMASK                PIC X(8).
           05 ZERO-FLD-8              PIC X(8).
       01  ZERO-FLD REDEFINES ZERO-PARM.
           05 ZERO-FWRD               PIC 9(8)  COMP.
           05 ZERO-HWRD               PIC 9(4)  COMP.
           05 ZERO-DUM                PIC X(10).
       01  TD-MSG.
           03 TASK-LABEL              PIC X(07) VALUE 'TASK # '.
           03 TASK-NUMBER             PIC 9(07).
           03 TASK-SEP                PIC X     VALUE ' '.
           03  CICS-MSG-AREA          PIC X(70).
       01  CICS-ERR-AREA.
           03  ERR-MSG            PIC X(24).
           03  SOCK-HEADER        PIC X(08) VALUE ' SOCKET='.
           03  ERR-SOCKET         PIC 9(05).
           03  RETC-HEADER        PIC X(09) VALUE ' RETCDE=-'.
           03  ERR-RETCODE        PIC 9(05).
           03  ERRN-HEADER        PIC X(07) VALUE ' ERRNO='.
           03  ERR-ERRNO          PIC 9(05).
      *
       01  CLIENTID-LSTN.
           05 CID-DOMAIN-LSTN               PIC 9(8) COMP.
```

Figure 176. EZACICSC IPv4 child server sample (Part 3 of 8)

```
           05 CID-NAME-LSTN                 PIC X(8).
           05 CID-SUBTASKNAME-LSTN          PIC X(8).
           05 CID-RES-LSTN                  PIC X(20).
       01  CLIENTID-APPL.
           05 CID-DOMAIN-APPL               PIC 9(8) COMP.
           05 CID-NAME-APPL                 PIC X(8).
           05 CID-SUBTASKNAME-APPL          PIC X(8).
           05 CID-RES-APPL                  PIC X(20).
       01  TCPSOCKET-PARM.
           05 GIVE-TAKE-SOCKET              PIC 9(8) COMP.
           05 LSTN-NAME                     PIC X(8).
           05 LSTN-SUBTASKNAME              PIC X(8).
           05 CLIENT-IN-DATA                PIC X(35).
           05 THREADSAFE-INDICATOR          PIC X(1).
              88 INTERFACE-IS-THREADSAFE             VALUE '1'.
           05 SOCKADDR-IN.
             10  SIN-FAMILY                 PIC 9(4) COMP.
             10  SIN-PORT                   PIC 9(4) COMP.
             10  SIN-ADDR                   PIC 9(8) COMP.
             10  SIN-ZERO                   PIC X(8).
       PROCEDURE DIVISION.
           MOVE 'Y' TO WRITE-SW.
           EXEC CICS HANDLE CONDITION INVREQ  (INVREQ-ERR-SEC)
                                      IOERR   (IOERR-SEC)
                                      ENDDATA (ENDDATA-SEC)
                                      LENGERR (LENGERR-SEC)
                                      NOSPACE (NOSPACE-ERR-SEC)
                                      QIDERR  (QIDERR-SEC)
                                      ITEMERR (ITEMERR-SEC)
                END-EXEC.
           PERFORM INITIAL-SEC     THRU   INITIAL-SEC-EXIT.
           PERFORM TAKESOCKET-SEC  THRU   TAKESOCKET-SEC-EXIT.
           MOVE '0' TO TASK-FLAG.
           PERFORM CLIENT-TASK     THRU   CLIENT-TASK-EXIT
               VARYING CNT FROM 1 BY 1  UNTIL TASK-FLAG = '1'.
       CLOSE-SOCK.
      *----------------------------------------------------------------*
      *                                                                *
      *   CLOSE 'accept descriptor'                                    *
      *                                                                *
      *----------------------------------------------------------------*
           CALL 'EZASOKET' USING SOKET-CLOSE SOCKID
                 ERRNO RETCODE.
           IF RETCODE <  0 THEN
              MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
              MOVE CLOS-ERR TO ERR-MSG
              MOVE SOCKID TO ERR-SOCKET
              MOVE RETCODE TO ERR-RETCODE
              MOVE ERRNO TO ERR-ERRNO
              MOVE CICS-ERR-AREA TO CICS-MSG-AREA
           ELSE
              MOVE CLOS-SUCCESS TO CICS-MSG-AREA.
           PERFORM WRITE-CICS  THRU WRITE-CICS-EXIT.
       PGM-EXIT.
           IF RETCODE < 0 THEN
```

Figure 176. EZACICSC IPv4 child server sample (Part 4 of 8)

```
         EXEC CICS ABEND ABCODE('TCPC') END-EXEC.
      MOVE SPACES TO CICS-MSG-AREA.
      MOVE 'END OF EZACICSC PROGRAM' TO CICS-MSG-AREA.
      PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
      EXEC CICS RETURN  END-EXEC.
      GOBACK.
*---------------------------------------------------------------*
*
*  RECEIVE PASSED PARAMETER WHICH ARE CID                       *
*
*---------------------------------------------------------------*
 INITIAL-SEC.
      MOVE SPACES TO CICS-MSG-AREA.
      MOVE 50 TO CLENG.
      MOVE 'TCPC TRANSACTION START UP     ' TO CICS-MSG-AREA.
      PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
      MOVE 72 TO CLENG.
      EXEC CICS RETRIEVE INTO(TCPSOCKET-PARM) LENGTH(CLENG)
                          END-EXEC.
 INITIAL-SEC-EXIT.
      EXIT.
*---------------------------------------------------------------*
*                                                               *
*  Perform TCP SOCKET functions by passing socket command to    *
*  EZASOKET routine.  SOCKET command are translated to pre-     *
*  define integer.                                              *
*                                                               *
*---------------------------------------------------------------*
 TAKESOCKET-SEC.
*---------------------------------------------------------------*
*                                                               *
*   Issue 'TAKESOCKET' call to acquire a socket which was       *
*   given by the LISTENER program.                              *
*                                                               *
*---------------------------------------------------------------*
      MOVE AF-INET TO CID-DOMAIN-LSTN CID-DOMAIN-APPL.
      MOVE LSTN-NAME TO CID-NAME-LSTN.
      MOVE LSTN-SUBTASKNAME TO CID-SUBTASKNAME-LSTN.
      MOVE GIVE-TAKE-SOCKET TO TAKE-SOCKET SOCKID SOCKID-FWD.
      CALL 'EZASOKET' USING SOKET-TAKESOCKET SOCKID
           CLIENTID-LSTN ERRNO RETCODE.
      IF RETCODE <  0 THEN
         MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
         MOVE TAKE-ERR TO ERR-MSG
         MOVE SOCKID TO ERR-SOCKET
         MOVE RETCODE TO ERR-RETCODE
         MOVE ERRNO TO ERR-ERRNO
         MOVE CICS-ERR-AREA TO CICS-MSG-AREA
         PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
         GO TO PGM-EXIT
      ELSE
          MOVE SPACES TO CICS-MSG-AREA
          MOVE TAKE-SUCCESS TO CICS-MSG-AREA
          PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
      MOVE RETCODE TO SOCKID.
```

Figure 176. EZACICSC IPv4 child server sample (Part 5 of 8)

```
           MOVE SPACES TO TCP-BUF.
           MOVE TASK-START TO TCP-BUF.
           MOVE 50  TO TCPLENG.
      *
      *    REMOVE FOLLOWING STATEMENT IF USING EBCDIC CLIENT
      *
           CALL 'EZACIC04' USING TCP-BUF TCPLENG.
           CALL 'EZASOKET' USING SOKET-WRITE SOCKID TCPLENG
                 TCP-BUF ERRNO RETCODE.
           IF RETCODE <  0 THEN
              MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
              MOVE WRITE-ERR TO ERR-MSG
              MOVE SOCKID TO ERR-SOCKET
              MOVE RETCODE TO ERR-RETCODE
              MOVE ERRNO TO ERR-ERRNO
              MOVE CICS-ERR-AREA TO CICS-MSG-AREA
              PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
              GO TO PGM-EXIT
           ELSE
              MOVE WRITE-SUCCESS TO CICS-MSG-AREA
              PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
       TAKESOCKET-SEC-EXIT.
           EXIT.
       CLIENT-TASK.
      *---------------------------------------------------------------*
      *                                                               *
      *  Issue 'RECV' socket to receive input data from client       *
      *                                                               *
      *---------------------------------------------------------------*
           MOVE LOW-VALUES TO TCP-BUF.
           MOVE 200 TO TCPLENG.
           MOVE ZEROS TO RECV-FLAG.
           CALL 'EZASOKET' USING SOKET-RECV SOCKID
                RECV-FLAG TCPLENG TCP-BUF ERRNO RETCODE.
           IF RETCODE <  0 THEN
              MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
              MOVE READ-ERR TO ERR-MSG
              MOVE SOCKID TO ERR-SOCKET
              MOVE RETCODE TO ERR-RETCODE
              MOVE ERRNO TO ERR-ERRNO
              MOVE CICS-ERR-AREA TO CICS-MSG-AREA
              PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
              GO TO PGM-EXIT
           ELSE
              MOVE READ-SUCCESS TO CICS-MSG-AREA
              PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
      *
      *    REMOVE FOLLOWING STATEMENT IF USING EBCDIC CLIENT
      *
           CALL 'EZACIC05' USING TCP-BUF TCPLENG.
      *
```

Figure 176. EZACICSC IPv4 child server sample (Part 6 of 8)

```
*    DETERMINE WHETHER THE CLIENT IS FINISHED SENDING DATA
*
     IF TCP-BUF-H = 'END' OR TCP-BUF-H = 'end' THEN
        MOVE '1' TO TASK-FLAG
        PERFORM CLIENT-TALK-END THRU CLIENT-TALK-END-EXIT
        GO TO CLIENT-TASK-EXIT.
     IF RETCODE = 0  THEN
        MOVE '1' TO TASK-FLAG
        GO TO CLIENT-TASK-EXIT.
*---------------------------------------------------------------*
**  ECHO RECEIVING DATA
*---------------------------------------------------------------*
     MOVE TCP-BUF TO CICS-MSG-AREA.
     PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
     MOVE RETCODE TO TCPLENG.
*
*    REMOVE FOLLOWING STATEMENT IF USING EBCDIC CLIENT
*
     CALL 'EZACIC04' USING TCP-BUF TCPLENG.
     CALL 'EZASOKET' USING SOKET-WRITE SOCKID TCPLENG
          TCP-BUF ERRNO RETCODE.
     IF RETCODE <  0 THEN
        MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
        MOVE WRITE-ERR TO ERR-MSG
        MOVE SOCKID TO ERR-SOCKET
        MOVE RETCODE TO ERR-RETCODE
        MOVE ERRNO TO ERR-ERRNO
        MOVE CICS-ERR-AREA TO CICS-MSG-AREA
        PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
        GO TO PGM-EXIT
     ELSE
        MOVE WRITE-SUCCESS TO CICS-MSG-AREA
        PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
 CLIENT-TASK-EXIT.
     EXIT.
 WRITE-CICS.
     MOVE 78 TO CLENG.
     MOVE EIBTASKN TO TASK-NUMBER.
     IF WRITE-SW = 'Y' THEN
         IF INTERFACE-IS-THREADSAFE THEN
             IF FORCE-ERROR-MSG = 'Y' THEN
                 EXEC CICS WRITEQ TD QUEUE('CSMT') FROM(TD-MSG)
                      LENGTH(CLENG) NOHANDLE
                 END-EXEC
             ELSE
                 NEXT SENTENCE
         ELSE
             EXEC CICS WRITEQ TD QUEUE('CSMT') FROM(TD-MSG)
                  LENGTH(CLENG) NOHANDLE
             END-EXEC
     ELSE
         NEXT SENTENCE.
     MOVE SPACES TO CICS-MSG-AREA.
```

Figure 176. EZACICSC IPv4 child server sample (Part 7 of 8)

```
 WRITE-CICS-EXIT.
     EXIT.
 CLIENT-TALK-END.
        MOVE LOW-VALUES TO TCP-BUF.
        MOVE WRKEND TO TCP-BUF CICS-MSG-AREA.
        MOVE 50 TO TCPLENG.
*
*    REMOVE FOLLOWING STATEMENT IF USING EBCDIC CLIENT
*
        CALL 'EZACIC04' USING TCP-BUF TCPLENG.
        CALL 'EZASOKET' USING SOKET-WRITE SOCKID TCPLENG
             TCP-BUF ERRNO RETCODE.
        IF RETCODE <  0 THEN
           MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
           MOVE WRITE-END-ERR TO ERR-MSG
           MOVE SOCKID TO ERR-SOCKET
           MOVE RETCODE TO ERR-RETCODE
           MOVE ERRNO TO ERR-ERRNO
           MOVE CICS-ERR-AREA TO CICS-MSG-AREA
           PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
           GO TO PGM-EXIT.
 CLIENT-TALK-END-EXIT.
     EXIT.
 INVREQ-ERR-SEC.
     MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
     MOVE INVREQ-ERR TO CICS-MSG-AREA.
     PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
     GO TO PGM-EXIT.
 IOERR-SEC.
     MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
     MOVE IOERR-ERR TO CICS-MSG-AREA.
     PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
     GO TO PGM-EXIT.
 LENGERR-SEC.
     MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
     MOVE LENGERR-ERR TO CICS-MSG-AREA.
     PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
     GO TO PGM-EXIT.
 NOSPACE-ERR-SEC.
     MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
     MOVE NOSPACE-ERR TO CICS-MSG-AREA.
     PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
     GO TO PGM-EXIT.
 QIDERR-SEC.
     MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
     MOVE QIDERR-ERR TO CICS-MSG-AREA.
     PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
     GO TO PGM-EXIT.
 ITEMERR-SEC.
     MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
     MOVE ITEMERR-ERR TO CICS-MSG-AREA.
     PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
     GO TO PGM-EXIT.
 ENDDATA-SEC.
     MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
     MOVE ENDDATA-ERR TO CICS-MSG-AREA.
     PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
     GO TO PGM-EXIT.
```

Figure 176. EZACICSC IPv4 child server sample (Part 8 of 8)

EZACICSS

The following COBOL socket program is in the SEZAINST data set.

```
*************************************************************
*                                                           *
* Communications Server for z/OS,   Version 1, Release 9    *
*                                                           *
*                                                           *
* Copyright:    Licensed Materials - Property of IBM        *
*                                                           *
*               "Restricted Materials of IBM"               *
*                                                           *
*               5694-A01                                    *
*                                                           *
*               Copyright IBM Corp. 1977, 2007              *
*                                                           *
*               US Government Users Restricted Rights -     *
*               Use, duplication or disclosure restricted by *
*               GSA ADP Schedule Contract with IBM Corp.    *
*                                                           *
* Status:       CSV1R9                                      *
*                                                           *
* $MOD(EZACICSS),COMP(CICS),PROD(TCPIP):                    *
*                                                           *
*************************************************************
* $SEG(EZACICSS)
*--------------------------------------------------------------*
*                                                              *
*   Module Name :  EZACICSS                                    *
*                                                              *
*   Description :  This is a sample server program.  It        *
*                  establishes a connection between            *
*                  CICS & TCPIP to process client requests.    *
*                  The server expects the data received        *
*                  from a host / workstation in ASCII.         *
*                  All responses sent by the server to the     *
*                  CLIENT are in ASCII.  This server is        *
*                  started using CECI or via the LISTENER.     *
*                                                              *
*                    CECI START TRANS(xxxx) from(yyyy)         *
*                       where xxxx is this servers CICS        *
*                       transaction id and yyyy is the         *
*                       port this server will listen on.       *
*                                                              *
*                  It processes request received from          *
*                  clients for updates to a hypothetical       *
*                  DB2 database.  Any and all references to    *
```

Figure 177. EZACICSS IPv4 iterative server sample (Part 1 of 22)

```
      *                  DB2 or SQL are commented out as this       *
      *                  sample is to illustrate CICS Sockets.      *
      *                                                             *
      *                  A client connection is broken when the     *
      *                  client transmits and 'END' token to the    *
      *                  server.  All processing is terminated      *
      *                  when an 'TRM' token is received from a     *
      *                  client.                                    *
      *                                                             *
      *                                                             *
      *-------------------------------------------------------------*
      *                                                             *
      *   LOGIC       :  1.  Establish server setup                 *
      *                      a).  TRUE Active                       *
      *                      b).  CAF Active                        *
      *                  2.  Assign user specified port at          *
      *                      start up or use the program            *
      *                      declared default.                      *
      *                  3.  Initialize the Socket.                 *
      *                  4.  Bind the port.                         *
      *                  5.  Set Bit Mask to accept incoming        *
      *                      read request.                          *
      *                  6.  Process request from clients.          *
      *                      a).  Wait for connection               *
      *                      b).  Process request until 'END'       *
      *                           token is receive from client.     *
      *                      c).  Close connection.                 *
      *                      note:  The current client request      *
      *                             ends when the client closes     *
      *                             the connection or sends an      *
      *                             'END' token to the server.      *
      *                      d).  If the last request received by   *
      *                           the current client is not a       *
      *                           request to the server to          *
      *                           terminate processing ('TRM'),     *
      *                           continue at step 6A.              *
      *                  7.  Close the server's connection.         *
      *                                                             *
      *-------------------------------------------------------------*
       IDENTIFICATION DIVISION.
       PROGRAM-ID. EZACICSS.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
      *--------------------------------------------------------------*
      *   MESSAGES                                                   *
      *--------------------------------------------------------------*
       77  BITMASK-ERR                     PIC X(30)
           VALUE IS 'BITMASK CONVERSION - FAILED   '.
       77  ENDDATA-ERR                     PIC X(30)
           VALUE IS 'RETRIEVE DATA CAN NOT BE FOUND'.
       77  INIT-MSG                        PIC X(30)
           VALUE IS 'INITAPI COMPLETE              '.
       77  IOERR-ERR                       PIC X(30)
```

Figure 177. EZACICSS IPv4 iterative server sample (Part 2 of 22)

```
         VALUE IS 'IOERR OCCURRS                 '.
       77  ITEMERR-ERR                     PIC X(30)
            VALUE IS 'ITEMERR ERROR                 '.
       77  KEYWORD-ERR                     PIC X(30)
            VALUE IS 'INPUT KEYWORD ERROR           '.
       77  LENGERR-ERR                     PIC X(30)
            VALUE IS 'LENGERR ERROR                 '.
       77  NOSPACE-ERR                     PIC X(30)
            VALUE IS 'NOSPACE CONDITION             '.
       77  NULL-DATA                       PIC X(30)
            VALUE IS 'READ NULL DATA                '.
       77  QIDERR-ERR                      PIC X(30)
            VALUE IS 'TRANSIENT DATA QUEUE NOT FOUND'.
       77  START-MSG                       PIC X(30)
            VALUE IS 'SERVER PROGRAM IS STARTING    '.
       77  TCP-EXIT-ERR                    PIC X(30)
            VALUE IS 'SERVER STOPPED:TRUE NOT ACTIVE'.
       77  TCP-SERVER-OFF                  PIC X(30)
            VALUE IS 'SERVER IS ENDING              '.
       77  TS-INVREQ-ERR                   PIC X(30)
            VALUE IS 'WRITE TS FAILED  - INVREQ     '.
       77  TS-NOTAUTH-ERR                  PIC X(30)
            VALUE IS 'WRITE TS FAILED  - NOTAUTH    '.
       77  TS-IOERR-ERR                    PIC X(30)
            VALUE IS 'WRITE TS FAILED  - IOERR      '.
       77  WRITETS-ERR                     PIC X(30)
            VALUE IS 'WRITE TS FAILED               '.
       01  ACCEPT-ERR.
           05  ACCEPT-ERR-M                PIC X(25)
                VALUE IS 'SOCKET CALL FAIL - ACCEPT'.
           05  FILLER                      PIC X(9)
                VALUE IS ' ERRNO = '.
           05  ACCEPT-ERRNO                PIC 9(8) DISPLAY.
           05  FILLER                      PIC X(13)
                VALUE IS SPACES.
       01  BIND-ERR.
           05  BIND-ERR-M                  PIC X(25)
                VALUE IS 'SOCKET CALL FAIL  -  BIND'.
           05  FILLER                      PIC X(9)
                VALUE IS ' ERRNO = '.
           05  BIND-ERRNO                  PIC 9(8) DISPLAY.
           05  FILLER                      PIC X(13)
                VALUE IS SPACES.
       01  CLOSE-ERR.
           05  CLOSE-ERR-M                 PIC X(30)
                VALUE IS 'CLOSE SOCKET DESCRIPTOR FAILED'.
           05  FILLER                      PIC X(9)
                VALUE IS ' ERRNO = '.
           05  CLOSE-ERRNO                 PIC 9(8)  DISPLAY.
           05  FILLER                      PIC X(8)
                VALUE IS SPACES.
       01  DB2END.
           05  FILLER                      PIC X(16)
                VALUE IS 'DB2 PROCESS ENDS'.
           05  FILLER                      PIC X(39)
```

Figure 177. EZACICSS IPv4 iterative server sample (Part 3 of 22)

```
           VALUE IS SPACES.
     01  DB2-CAF-ERR.
         05  FILLER                      PIC X(24)
             VALUE IS 'CONNECT NOT ESTABLISHED '.
         05  FILLER                      PIC X(30)
             VALUE IS 'ATTACHMENT FACILITY NOT ACTIVE'.
         05  FILLER                      PIC X(1)
             VALUE IS SPACES.
     01  DB2MSG.
         05  DB2-ACT                     PIC X(6)  VALUE SPACES.
             88 DAINSERT                           VALUE 'INSERT'.
             88 DADELETE                           VALUE 'DELETE'.
             88 DAUPDATE                           VALUE 'UPDATE'.
         05  DB2M                        PIC X(18)
             VALUE IS ' COMPLETE - #ROWS '.
         05  DB2M-VAR                    PIC X(10).
         05  FILLER                      PIC X(2)  VALUE SPACES.
         05  DB2CODE                     PIC -(9)9.
         05  FILLER                      PIC X(11)
             VALUE IS SPACES.
     01  INITAPI-ERR.
         05  INITAPI-ERR-M               PIC X(35)
             VALUE IS 'INITAPI FAILED - SERVER NOT STARTED'.
         05  FILLER                      PIC X(9)
             VALUE IS ' ERRNO = '.
         05  INIT-ERRNO                  PIC 9(8) DISPLAY.
         05  FILLER                      PIC X(3)
             VALUE IS SPACES.
     01  LISTEN-ERR.
         05  LISTEN-ERR-M                PIC X(25)
             VALUE IS 'SOCKET CALL FAIL - LISTEN'.
         05  FILLER                      PIC X(9)
             VALUE IS ' ERRNO = '.
         05  LISTEN-ERRNO                PIC 9(8) DISPLAY.
         05  FILLER                      PIC X(13)
             VALUE IS SPACES.
     01  LISTEN-SUCC.
         05  FILLER                      PIC X(34)
             VALUE IS 'READY TO ACCEPT REQUEST ON PORT:  '.
         05  BIND-PORT                   PIC X(4).
         05  FILLER                      PIC X(10)  VALUE SPACES.
         05  FILLER                      PIC X(7)
             VALUE IS SPACES.
     01  PORTNUM-ERR.
         05  INVALID-PORT                PIC X(33)
             VALUE IS 'SERVER NOT STARTED - INVALID PORT'.
         05  FILLER                      PIC X(10)
             VALUE IS ' NUMBER = '.
         05  PORT-ERRNUM                 PIC X(4).
         05  FILLER                      PIC X(8)
             VALUE IS SPACES.
     01  RECVFROM-ERR.
         05  RECVFROM-ERR-M              PIC X(24)
             VALUE IS 'RECEIVE SOCKET CALL FAIL'.
         05  FILLER                      PIC X(9)
```

Figure 177. EZACICSS IPv4 iterative server sample (Part 4 of 22)

```
            VALUE IS ' ERRNO = '.
         05  RECVFROM-ERRNO              PIC 9(8) DISPLAY.
         05  FILLER                      PIC X(14)
            VALUE IS SPACES.
     01  SELECT-ERR.
         05  SELECT-ERR-M                PIC X(24)
            VALUE IS 'SELECT CALL FAIL        '.
         05  FILLER                      PIC X(9)
            VALUE IS ' ERRNO = '.
         05  SELECT-ERRNO                PIC 9(8) DISPLAY.
         05  FILLER                      PIC X(14)
            VALUE IS SPACES.
     01  SQL-ERROR.
         05  FILLER                      PIC X(35)
            VALUE IS 'SQLERR -PROG TERMINATION,SQLCODE = '.
         05  SQL-ERR-CODE                PIC -(9)9.
         05  FILLER                      PIC X(11)
            VALUE IS SPACES.
     01  SOCKET-ERR.
         05  SOCKET-ERR-M                PIC X(25)
            VALUE IS 'SOCKET CALL FAIL - SOCKET'.
         05  FILLER                      PIC X(9)
            VALUE IS ' ERRNO = '.
         05  SOCKET-ERRNO                PIC 9(8) DISPLAY.
         05  FILLER                      PIC X(13)
            VALUE IS SPACES.
     01  TAKE-ERR.
         05  TAKE-ERR-M                  PIC X(17)
            VALUE IS 'TAKESOCKET FAILED'.
         05  FILLER                      PIC X(9)
            VALUE IS ' ERRNO = '.
         05  TAKE-ERRNO                  PIC 9(8) DISPLAY.
         05  FILLER                      PIC X(21)
            VALUE IS SPACES.
     01  WRITE-ERR.
         05  WRITE-ERR-M                 PIC X(33)
            VALUE IS 'WRITE SOCKET FAIL'.
         05  FILLER                      PIC X(9)
            VALUE IS ' ERRNO = '.
         05  WRITE-ERRNO                 PIC 9(8) DISPLAY.
         05  FILLER                      PIC X(21)
            VALUE IS SPACES.
    *----------------------------------------------------------------*
    *    PROGRAM'S CONSTANTS                                         *
    *----------------------------------------------------------------*
|    77  CTOB                     PIC X(4)  VALUE 'CTOB'.
     77  DEL-ID                   PIC X(1)  VALUE ','.
     77  BACKLOG                  PIC 9(8)  COMP VALUE 5.
     77  NONZERO-FWRD             PIC 9(8)  VALUE 256.
     77  TCP-FLAG                 PIC 9(8)  COMP VALUE 0.
     77  SOCK-TYPE                PIC 9(8)  COMP VALUE 1.
     77  AF-INET                  PIC 9(8)  COMP VALUE 2.
     77  NUM-FDS                  PIC 9(8)  COMP VALUE 5.
```

Figure 177. EZACICSS IPv4 iterative server sample (Part 5 of 22)

```
 77  LOM                     PIC 9(4)  COMP VALUE 4.
 77  CECI-LENG               PIC 9(8)  COMP VALUE 5.
 77  BUFFER-LENG             PIC 9(8)  COMP VALUE 55.
 77  GWLENG                  PIC 9(4)  COMP VALUE 256.
 77  DEFAULT-PORT            PIC X(4)  VALUE '????'.
     88  DEFAULT-SPECIFIED             VALUE '1950'.
 01  INADDR-ANY.
     05 FILLER               PIC 9(8) BINARY VALUE 0.
 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-GETCLIENTID    PIC X(16) VALUE 'GETCLIENTID     '.
     02 SOKET-GETHOSTBYADDR  PIC X(16) VALUE 'GETHOSTBYADDR   '.
     02 SOKET-GETHOSTBYNAME  PIC X(16) VALUE 'GETHOSTBYNAME   '.
     02 SOKET-GETHOSTID      PIC X(16) VALUE 'GETHOSTID       '.
     02 SOKET-GETHOSTNAME    PIC X(16) VALUE 'GETHOSTNAME     '.
     02 SOKET-GETPEERNAME    PIC X(16) VALUE 'GETPEERNAME     '.
     02 SOKET-GETNAMEINFO    PIC X(16) VALUE 'GETNAMEINFO     '.
     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-IOCTL          PIC X(16) VALUE 'IOCTL           '.
     02 SOKET-LISTEN         PIC X(16) VALUE 'LISTEN          '.
     02 SOKET-NTOP           PIC X(16) VALUE 'NTOP            '.
     02 SOKET-READ           PIC X(16) VALUE 'READ            '.
     02 SOKET-RECV           PIC X(16) VALUE 'RECV            '.
     02 SOKET-RECVFROM       PIC X(16) VALUE 'RECVFROM        '.
     02 SOKET-SELECT         PIC X(16) VALUE 'SELECT          '.
     02 SOKET-SELECTEX       PIC X(16) VALUE 'SELECTEX        '.
     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           '.
*---------------------------------------------------------------*
*    PROGRAM'S VARIABLES                                        *
*---------------------------------------------------------------*
 77  PROTOCOL                PIC 9(8)  COMP VALUE 0.
 77  SRV-SOCKID              PIC 9(4)  COMP VALUE 0.
 77  SRV-SOCKID-FWD          PIC 9(8)  COMP VALUE 0.
 77  CLI-SOCKID              PIC 9(4)  COMP VALUE 0.
 77  CLI-SOCKID-FWD          PIC S9(8) COMP VALUE 0.
 77  LENG                    PIC 9(4)  COMP.
 77  WSLENG                  PIC 9(4)  COMP.
```

Figure 177. EZACICSS IPv4 iterative server sample (Part 6 of 22)

```
       77  RESPONSE                 PIC 9(9)  COMP.
       77  TSTAMP                   PIC 9(8).
       77  TASK-FLAG                PIC X(1)  VALUE '0'.
           88  TASK-END             VALUE '1'.
           88  TASK-TERM            VALUE '2'.
       77  GWPTR                    PIC S9(8) COMP.
       77  WSPTR                    PIC S9(8) COMP.
       77  TCP-INDICATOR            PIC X(1)  VALUE IS SPACE.
       77  TAKESOCKET-SWITCH        PIC X(1)  VALUE IS SPACE.
           88  DOTAKESOCKET         VALUE '1'.
       77  TCPLENG                  PIC 9(8)  COMP VALUE 0.
       77  ERRNO                    PIC 9(8)  COMP.
       77  RETCODE                  PIC S9(8) COMP.
       77  TRANS                    PIC X(4).
       01  CLIENTID-LSTN.
           05  CID-DOMAIN-LSTN      PIC 9(8)  COMP VALUE 2.
           05  CID-LSTN-INFO.
               10  CID-NAME-LSTN    PIC X(8).
               10  CID-SUBTNAM-LSTN PIC X(8).
           05  CID-RES-LSTN         PIC X(20) VALUE LOW-VALUES.
       01  INIT-SUBTASKID.
           05  SUBTASKNO            PIC X(7)  VALUE LOW-VALUES.
           05  SUBT-CHAR            PIC A(1)  VALUE 'L'.
       01  IDENT.
           05  TCPNAME              PIC X(8) VALUE 'TCPCS   '.
           05  ADSNAME              PIC X(8) VALUE 'EZACIC6S'.
       01  MAXSOC                   PIC 9(4) BINARY VALUE 0.
       01  MAXSNO                   PIC 9(8) BINARY VALUE 0.
       01  NFDS                     PIC 9(8) BINARY.
       01  PORT-RECORD.
           05  PORT                 PIC X(4).
           05  FILLER               PIC X(36).
       01  SELECT-CSOCKET.
           05  READMASK             PIC X(4)  VALUE LOW-VALUES.
           05  DUMYMASK             PIC X(4)  VALUE LOW-VALUES.
           05  REPLY-RDMASK         PIC X(4)  VALUE LOW-VALUES.
           05  REPLY-RDMASK-FF      PIC X(4).
       01 SOCKADDR-IN.
          05 SAIN-FAMILY                PIC 9(4) BINARY VALUE 0.
             88 SAIN-FAMILY-IS-AFINET   VALUE 2.
          05 SAIN-DATA                  PIC X(14).
          05 SAIN-SIN REDEFINES SAIN-DATA.
             10 SAIN-SIN-PORT           PIC 9(4) BINARY.
             10 SAIN-SIN-ADDR           PIC 9(8) BINARY.
             10 FILLER                  PIC X(8).
       01  SOCKET-CONV.
           05  SOCKET-TBL  OCCURS 6 TIMES.
               10  SOCK-CHAR        PIC X(1)  VALUE '0'.
       01  TCP-BUF.
           05  TCP-BUF-H            PIC X(3).
           05  TCP-BUF-DATA         PIC X(52).
```

Figure 177. EZACICSS IPv4 iterative server sample (Part 7 of 22)

```
 01  TCPCICS-MSG-AREA.
     02  TCPCICS-MSG-1.
         05  MSGDATE          PIC 9(8).
         05  FILLER           PIC X(2)  VALUE SPACES.
         05  MSGTIME          PIC 9(8).
         05  FILLER           PIC X(2)  VALUE SPACES.
         05  MODULE           PIC X(10) VALUE 'EZACICSS: '.
     02  TCPCICS-MSG-2.
         05  MSG-AREA         PIC X(55) VALUE SPACES.
 01  TCP-INPUT-DATA              PIC X(85) VALUE LOW-VALUES.
 01  TCPSOCKET-PARM REDEFINES TCP-INPUT-DATA.
     05 GIVE-TAKE-SOCKET             PIC 9(8) COMP.
     05 CLIENTID-PARM.
        10 LSTN-NAME                 PIC X(8).
        10 LSTN-SUBTASKNAME          PIC X(8).
     05 CLIENT-DATA-FLD.
        10 CLIENT-IN-DATA            PIC X(35).
        10 FILLER                    PIC X(1).
     05 TCPSOCKADDR-IN.
        10 SOCK-FAMILY               PIC 9(4) BINARY.
           88 SOCK-FAMILY-IS-AFINET   VALUE 2.
           88 SOCK-FAMILY-IS-AFINET6  VALUE 19.
        10 SOCK-DATA                 PIC X(26).
        10 SOCK-SIN REDEFINES SOCK-DATA.
           15 SOCK-SIN-PORT          PIC 9(4) BINARY.
           15 SOCK-SIN-ADDR          PIC 9(8) BINARY.
           15 FILLER                 PIC X(8).
           15 FILLER                 PIC X(12).
        10 SOCK-SIN6 REDEFINES SOCK-DATA.
           15 SOCK-SIN6-PORT         PIC 9(4) BINARY.
           15 SOCK-SIN6-FLOWINFO     PIC 9(8) BINARY.
           15 SOCK-SIN6-ADDR.
              20 FILLER              PIC 9(16) BINARY.
              20 FILLER              PIC 9(16) BINARY.
           15 SOCK-SIN6-SCOPEID      PIC 9(8) BINARY.
     05 FILLER                       PIC X(68).
     05 CLIENT-IN-DATA-LENGTH        PIC 9(4) COMP.
     05 CLIENT-IN-DATA-2             PIC X(999).
 01  SOCK-TO-RECV-FWD.
     02 FILLER                PIC 9(4) BINARY.
     02 SOCK-TO-RECV          PIC 9(4) BINARY.
 01  TIMEVAL.
     02 TVSEC                 PIC 9(8)  COMP VALUE 180.
     02 TVUSEC                PIC 9(8)  COMP VALUE 0.
 01  ZERO-PARM                PIC X(16) VALUE LOW-VALUES.
 01  ZERO-FLD REDEFINES ZERO-PARM.
     02  ZERO-8               PIC X(8).
     02  ZERO-DUM             PIC X(2).
     02  ZERO-HWRD            PIC 9(4)  COMP.
     02  ZERO-FWRD            PIC 9(8)  COMP.
* ********************************************** *
*  INPUT FORMAT FOR UPDATING THE SAMPLE DB2 TABLE *
* ********************************************** *
```

Figure 177. EZACICSS IPv4 iterative server sample (Part 8 of 22)

```
 01  INPUT-DEPT.
     05  IN-ACT                PIC X(3).
     05  IN-DEPTNO             PIC X(3).
     05  IN-DEPTN              PIC X(36).
     05  IN-MGRNO              PIC X(6).
     05  IN-ADMRDEPT           PIC X(3).
*----------------------------------------------------------------*
*    SQL STATEMENTS:  SQL COMMUNICATION AREA                     *
*----------------------------------------------------------------*
***  EXEC SQL INCLUDE SQLCA    END-EXEC.
*----------------------------------------------------------------*
*    SQL STATEMENTS:  DEPARTMENT TABLE CREATE STATEMENT FOR DB2 *
*                                                                *
*              CREATE TABLE TCPCICS.DEPT                         *
*                     (DEPTNO      CHAR(03),                     *
*                      DEPTNAME    CHAR(36),                     *
*                      MGRNO       CHAR(06),                     *
*                      ADMRDEPT    CHAR(03));                    *
*                                                                *
*----------------------------------------------------------------*
*    DCLGEN GENERATED FROM DB2 FOR THE DEPARTMENT TABLE.         *
*----------------------------------------------------------------*
* ***EXEC SQL INCLUDE DCLDEPT  END-EXEC.
******************************************************************
* DCLGEN TABLE(TCPCICS.DEPT)                                     *
*        LIBRARY(SYSADM.CICS.SPUFI(DCLDEPT))                     *
*        LANGUAGE(COBOL)                                         *
*        QUOTE                                                   *
* ... IS THE DCLGEN COMMAND THAT MADE THE FOLLOWING STATEMENTS   *
******************************************************************
***  EXEC SQL DECLARE TCPCICS.DEPT TABLE
***  ( DEPTNO                          CHAR(3),
***    DEPTNAME                        CHAR(36),
***    MGRNO                           CHAR(6),
***    ADMRDEPT                        CHAR(3)
***  ) END-EXEC.
******************************************************************
* COBOL DECLARATION FOR TABLE TCPCICS.DEPT                       *
******************************************************************
 01  DCLDEPT.
     10 DEPTNO               PIC X(3).
     10 DEPTNAME             PIC X(36).
     10 MGRNO                PIC X(6).
     10 ADMRDEPT             PIC X(3).
******************************************************************
* THE NUMBER OF COLUMNS DESCRIBED BY THIS DECLARATION IS 4       *
******************************************************************
 PROCEDURE DIVISION.
```

Figure 177. EZACICSS IPv4 iterative server sample (Part 9 of 22)

```
      *** EXEC SQL WHENEVER SQLERROR    GO TO SQL-ERROR-ROU END-EXEC.
      *** EXEC SQL WHENEVER SQLWARNING  GO TO SQL-ERROR-ROU END-EXEC.
           EXEC CICS IGNORE CONDITION TERMERR
                                      EOC
                                      SIGNAL
           END-EXEC.
           EXEC CICS HANDLE CONDITION ENDDATA   (ENDDATA-SEC)
                                      IOERR     (IOERR-SEC)
                                      LENGERR   (LENGERR-SEC)
                                      NOSPACE   (NOSPACE-ERR-SEC)
                                      QIDERR    (QIDERR-SEC)
           END-EXEC.
           MOVE START-MSG                   TO MSG-AREA.
           PERFORM HANDLE-TCPCICS           THRU HANDLE-TCPCICS-EXIT.
      *---------------------------------------------------------------*
      *                                                               *
      *  BEFORE SERVER  STARTS, TRUE MUST BE ACTIVE.  ISSUE 'EXTRACT  *
      *  EXIT' COMMAND TO CHECK IF TRUE IS ACTIVE OR NOT              *
      *                                                               *
      *---------------------------------------------------------------*
           EXEC CICS PUSH HANDLE END-EXEC.
           EXEC CICS HANDLE CONDITION
                INVEXITREQ(TCP-TRUE-REQ)
           END-EXEC.
           EXEC CICS EXTRACT EXIT
                PROGRAM ('EZACIC01')
                GASET   (GWPTR)
                GALENGTH(GWLENG)
           END-EXEC.
           EXEC CICS POP HANDLE END-EXEC.
      *---------------------------------------------------------------*
      *                                                               *
      *  CICS ATTACH FACILITY MUST BE STARTED FOR THE APPROPRIATE DB2 *
      *  SUBSYSTEM BEFORE YOU EXECUTE CICS TRANSACTIONS REQUIRING     *
      *  ACCESS TO DB2 DATABASES.                                     *
      *                                                               *
      *---------------------------------------------------------------*
      *    EXEC CICS PUSH HANDLE END-EXEC.
      *
      *    EXEC CICS HANDLE CONDITION
      *         INVEXITREQ(DB2-TRUE-REQ)
      *    END-EXEC.
      *
      *    EXEC CICS EXTRACT EXIT
      *         PROGRAM   ('DSNCEXT1')
      *         ENTRYNAME ('DSNCSQL')
      *         GASET     (WSPTR)
      *         GALENGTH  (WSLENG)
      *    END-EXEC.
      *
      *    EXEC CICS POP HANDLE END-EXEC.
      *
      *
```

Figure 177. EZACICSS IPv4 iterative server sample (Part 10 of 22)

```
*-------------------------------------------------------------*
*                                                             *
*  AT START UP THE SERVER REQUIRES THE PORT NUMBER FOR TCP/IP *
*  IT WILL USE.  THE PORT NUMBER SUPPORTED BY THIS SAMPLE IS  *
*  4 DIGITS IN LENGTH.                                        *
*                                                             *
*  INVOCATION:  <server>,<port number>                        *
*   LISTENER => SRV2,4000  - OR -  SRV2,4    -                *
*   CECI     => CECI START TR(SRV2) FROM(4000)                *
*                                                             *
*  THE LEADING SPACES ARE SIGNIFICANT.                        *
*                                                             *
*-------------------------------------------------------------*
     MOVE EIBTRNID                    TO TRANS.
     EXEC CICS RETRIEVE
          INTO   (TCP-INPUT-DATA)
          LENGTH (LENG)
     END-EXEC.
* *********************************************************** *
* THE PORT CAN SPECIFIED IN THE FROM(????) OPTION OF THE CECI *
* COMMAND OR THE DEFAULT PORT IS USED.                        *
* THE PORT FOR THE LISTENER STARTED SERVER IS THE PORT        *
* SPECIFIED IN THE CLIENT-DATA-FLD OR THE DEFAULT PORT        *
* IS USED.                                                    *
* *********************************************************** *
*       THE DEFAULT PORT MUST BE SET, BY THE PROGRAMMER.      *
* *********************************************************** *
     IF LENG < CECI-LENG
        THEN MOVE TCP-INPUT-DATA      TO PORT
        ELSE
          MOVE CLIENT-DATA-FLD        TO PORT-RECORD
          MOVE '1'                    TO TAKESOCKET-SWITCH
     END-IF.
     INSPECT PORT REPLACING LEADING SPACES BY '0'.
     IF PORT IS NUMERIC
        THEN MOVE PORT                TO BIND-PORT
        ELSE
          IF DEFAULT-SPECIFIED
             THEN  MOVE DEFAULT-PORT  TO PORT
                                         BIND-PORT
             ELSE
               MOVE PORT              TO PORT-ERRNUM
               MOVE PORTNUM-ERR       TO MSG-AREA
               PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT
               GO TO PGM-EXIT
          END-IF
     END-IF.
     IF DOTAKESOCKET
        THEN PERFORM LISTENER-STARTED-TASK THRU
                LISTENER-STARTED-TASK-EXIT
        ELSE PERFORM INIT-SOCKET           THRU
                INIT-SOCKET-EXIT
     END-IF.
     PERFORM SCKET-BIND-LSTN          THRU SCKET-BIND-LSTN-EXIT.
```

Figure 177. EZACICSS IPv4 iterative server sample (Part 11 of 22)

```
           MOVE 2                           TO CLI-SOCKID
                                               CLI-SOCKID-FWD.
           MOVE LISTEN-SUCC                 TO MSG-AREA.
           PERFORM HANDLE-TCPCICS           THRU HANDLE-TCPCICS-EXIT.
           COMPUTE NFDS = NUM-FDS + 1.
           MOVE LOW-VALUES                  TO READMASK.
           MOVE 6                           TO TCPLENG.
|          CALL 'EZACIC06' USING CTOB
|                                READMASK
|                                SOCKET-CONV
|                                TCPLENG
|                                RETCODE.
           IF RETCODE = -1
              THEN
                MOVE BITMASK-ERR            TO MSG-AREA
                PERFORM HANDLE-TCPCICS      THRU HANDLE-TCPCICS-EXIT
              ELSE
                PERFORM ACCEPT-CLIENT-REQ   THRU
                        ACCEPT-CLIENT-REQ-EXIT
                        UNTIL TASK-TERM
           END-IF.
           PERFORM CLOSE-SOCKET             THRU CLOSE-SOCKET-EXIT.
           MOVE TCP-SERVER-OFF              TO MSG-AREA.
           PERFORM HANDLE-TCPCICS           THRU HANDLE-TCPCICS-EXIT.
      *---------------------------------------------------------------*
      *                                                               *
      *    END OF PROGRAM                                             *
      *                                                               *
      *---------------------------------------------------------------*
       PGM-EXIT.
           EXEC CICS
                RETURN
           END-EXEC.
           GOBACK.
      *---------------------------------------------------------------*
      *                                                               *
      *          TRUE IS NOT ENABLED                                  *
      *                                                               *
      *---------------------------------------------------------------*
       TCP-TRUE-REQ.
           MOVE TCP-EXIT-ERR      TO MSG-AREA.
           PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
           GO TO PGM-EXIT.
      *---------------------------------------------------------------*
      *                                                               *
      *          DB2 CALL ATTACH FACILITY IS NOT ENABLED              *
      *                                                               *
      *---------------------------------------------------------------*
       DB2-TRUE-REQ.
           MOVE DB2-CAF-ERR       TO MSG-AREA.
           PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
           GO TO PGM-EXIT.
      *---------------------------------------------------------------*
```

Figure 177. EZACICSS IPv4 iterative server sample (Part 12 of 22)

```
*                                                               *
*  LISTENER STARTED TASK                                        *
*                                                               *
*---------------------------------------------------------------*
 LISTENER-STARTED-TASK.
     MOVE CLIENTID-PARM                TO CID-LSTN-INFO.
     MOVE GIVE-TAKE-SOCKET             TO SOCK-TO-RECV-FWD.
     CALL 'EZASOKET' USING SOKET-TAKESOCKET
                           SOCK-TO-RECV
                           CLIENTID-LSTN
                           ERRNO
                           RETCODE.
     IF RETCODE < 0
        THEN
          MOVE ERRNO                   TO TAKE-ERRNO
          MOVE TAKE-ERR                TO MSG-AREA
          PERFORM HANDLE-TCPCICS       THRU HANDLE-TCPCICS-EXIT
          GO TO PGM-EXIT
        ELSE
          MOVE BUFFER-LENG             TO TCPLENG
          MOVE START-MSG               TO TCP-BUF
          MOVE RETCODE                 TO SRV-SOCKID
          CALL 'EZACIC04' USING TCP-BUF TCPLENG
          CALL 'EZASOKET' USING SOKET-WRITE
                                SRV-SOCKID
                                TCPLENG
                                TCP-BUF
                                ERRNO
                                RETCODE
          IF RETCODE < 0
             THEN
               MOVE ERRNO              TO WRITE-ERRNO
               MOVE WRITE-ERR          TO MSG-AREA
               PERFORM HANDLE-TCPCICS THRU
                       HANDLE-TCPCICS-EXIT
               GO TO PGM-EXIT
             ELSE
               CALL 'EZASOKET' USING SOKET-CLOSE
                                     SRV-SOCKID
                                     ERRNO
                                     RETCODE
               IF RETCODE < 0
                  THEN
                    MOVE ERRNO         TO CLOSE-ERRNO
                    MOVE CLOSE-ERR     TO MSG-AREA
                    PERFORM HANDLE-TCPCICS  THRU
                            HANDLE-TCPCICS-EXIT
                    GO TO PGM-EXIT
                  ELSE NEXT SENTENCE
```

Figure 177. EZACICSS IPv4 iterative server sample (Part 13 of 22)

```
                 END-IF
             END-IF
         END-IF.
         MOVE LOW-VALUES                  TO TCP-BUF.
     LISTENER-STARTED-TASK-EXIT.
         EXIT.
    *-----------------------------------------------------------------*
    *                                                                 *
    *  START SERVER  PROGRAM                                          *
    *                                                                 *
    *-----------------------------------------------------------------*
     INIT-SOCKET.
         MOVE EIBTASKN                TO SUBTASKNO.
         CALL 'EZASOKET' USING SOKET-INITAPI
                               MAXSOC
                               IDENT
                               INIT-SUBTASKID
                               MAXSNO
                               ERRNO
                               RETCODE.
         IF RETCODE < 0
            THEN
              MOVE ERRNO              TO INIT-ERRNO
              MOVE INITAPI-ERR        TO MSG-AREA
              PERFORM HANDLE-TCPCICS  THRU HANDLE-TCPCICS-EXIT
              GO TO PGM-EXIT
            ELSE
              MOVE INIT-MSG           TO MSG-AREA
              PERFORM HANDLE-TCPCICS  THRU HANDLE-TCPCICS-EXIT
            END-IF.
     INIT-SOCKET-EXIT.
         EXIT.
     SCKET-BIND-LSTN.
         MOVE  -1                    TO SRV-SOCKID-FWD.
    *----------------------------------------------------------------*
    *                                                                *
    *   CREATING A SOCKET TO ALLOCATE                                *
    *   AN OPEN SOCKET FOR INCOMING CONNECTIONS                      *
    *                                                                *
    *----------------------------------------------------------------*
         CALL 'EZASOKET' USING SOKET-SOCKET
                               AF-INET
                               SOCK-TYPE
                               PROTOCOL
                               ERRNO
                               RETCODE.
         IF RETCODE < 0
```

Figure 177. EZACICSS IPv4 iterative server sample (Part 14 of 22)

```
          THEN
            MOVE ERRNO             TO SOCKET-ERRNO
            MOVE SOCKET-ERR        TO MSG-AREA
            PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT
            GO TO PGM-EXIT
          ELSE MOVE RETCODE        TO SRV-SOCKID
               MOVE  '1' TO SOCK-CHAR(RETCODE + 1)
       END-IF.
  *---------------------------------------------------------------*
  *                                                               *
| *  BIND THE SOCKET TO THE SERVICE PORT                          *
  *  TO ESTABLISH A LOCAL ADDRESS FOR PROCESSING INCOMING         *
  *  CONNECTIONS.                                                 *
  *                                                               *
  *---------------------------------------------------------------*
|      MOVE AF-INET                TO SAIN-FAMILY.
|      MOVE INADDR-ANY             TO SAIN-SIN-ADDR.
|      MOVE PORT                   TO SAIN-SIN-PORT.
|      CALL 'EZASOKET' USING SOKET-BIND
|                            SRV-SOCKID
|                            SOCKADDR-IN
|                            ERRNO
|                            RETCODE.
       IF RETCODE < 0 THEN
          MOVE ERRNO               TO BIND-ERRNO
          MOVE BIND-ERR            TO MSG-AREA
          PERFORM HANDLE-TCPCICS   THRU HANDLE-TCPCICS-EXIT
          GO TO PGM-EXIT.
  *---------------------------------------------------------------*
  *                                                               *
| *  CALL THE LISTEN COMMAND TO ALLOWS SERVERS TO                 *
  *  PREPARE A SOCKET FOR INCOMING CONNECTIONS AND SET MAXIMUM    *
  *  CONNECTIONS.                                                 *
  *                                                               *
  *---------------------------------------------------------------*
|      CALL 'EZASOKET' USING SOKET-LISTEN
|                            SRV-SOCKID
|                            BACKLOG
|                            ERRNO
                             RETCODE.
       IF RETCODE < 0 THEN
          MOVE ERRNO               TO LISTEN-ERRNO
          MOVE LISTEN-ERR          TO MSG-AREA
          PERFORM HANDLE-TCPCICS   THRU HANDLE-TCPCICS-EXIT
          GO TO PGM-EXIT.
   SCKET-BIND-LSTN-EXIT.
       EXIT.
```

Figure 177. EZACICSS IPv4 iterative server sample (Part 15 of 22)

```
      *-------------------------------------------------------------*
      *                                                             *
|     *  SOCKET HAS BEEN SET UP, THEN CALL 'ACCEPT' TO              *
      *  ACCEPT A REQUEST WHEN A CONNECTION ARRIVES.                *
      *                                                             *
      *  THIS SAMPLE PROGRAM WILL ONLY USE 5 SOCKETS.               *
      *                                                             *
      *-------------------------------------------------------------*
       ACCEPT-CLIENT-REQ.
|          CALL 'EZASOKET' USING SOKET-SELECT
|                                NFDS
|                                TIMEVAL
|                                READMASK
|                                DUMYMASK
|                                DUMYMASK
|                                REPLY-RDMASK
|                                DUMYMASK
|                                DUMYMASK
|                                ERRNO
|                                RETCODE.
           IF RETCODE < 0
              THEN
                MOVE ERRNO             TO SELECT-ERRNO
                MOVE SELECT-ERR        TO MSG-AREA
                PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT
                GO TO PGM-EXIT.
           IF RETCODE = 0
              THEN GO TO ACCEPT-CLIENT-REQ-EXIT.
      *-------------------------------------------------------------*
      *                                                             *
      *  ACCEPT REQUEST                                             *
      *                                                             *
      *-------------------------------------------------------------*
|          CALL 'EZASOKET' USING SOKET-ACCEPT
|                                SRV-SOCKID
|                                SOCKADDR-IN
|                                ERRNO
|                                RETCODE.
           IF RETCODE < 0 THEN
              MOVE ERRNO               TO ACCEPT-ERRNO
              MOVE ACCEPT-ERR          TO MSG-AREA
              PERFORM HANDLE-TCPCICS   THRU HANDLE-TCPCICS-EXIT
              GO TO PGM-EXIT.
           MOVE RETCODE TO CLI-SOCKID.
           PERFORM ACCEPT-RECV         THRU ACCEPT-RECV-EXIT
                   UNTIL TASK-END OR TASK-TERM.
           MOVE DB2END                 TO MSG-AREA.
           PERFORM HANDLE-TCPCICS      THRU HANDLE-TCPCICS-EXIT.
|
```

Figure 177. EZACICSS IPv4 iterative server sample (Part 16 of 22)

```
|            CALL 'EZASOKET' USING SOKET-CLOSE
|                                  CLI-SOCKID
|                                  ERRNO
|                                  RETCODE.
             IF RETCODE < 0 THEN
                MOVE ERRNO               TO CLOSE-ERRNO
                MOVE CLOSE-ERR           TO MSG-AREA
                PERFORM HANDLE-TCPCICS   THRU HANDLE-TCPCICS-EXIT.
             IF NOT TASK-TERM
                MOVE '0'                 TO TASK-FLAG.
         ACCEPT-CLIENT-REQ-EXIT.
             EXIT.
        *----------------------------------------------------------------*
        *                                                                *
        *  RECEIVING DATA THROUGH A SOCKET BY ISSUING 'RECVFROM'         *
        *  COMMAND.                                                      *
        *                                                                *
        *----------------------------------------------------------------*
         ACCEPT-RECV.
             MOVE 'T'                              TO TCP-INDICATOR.
             MOVE BUFFER-LENG                      TO TCPLENG.
             MOVE LOW-VALUES                       TO TCP-BUF.
|            CALL 'EZASOKET' USING SOKET-RECVFROM
|                                  CLI-SOCKID
|                                  TCP-FLAG
|                                  TCPLENG
|                                  TCP-BUF
|                                  SOCKADDR-IN
|                                  ERRNO
                                   RETCODE.
             IF RETCODE EQUAL 0 AND TCPLENG EQUAL 0
                THEN NEXT SENTENCE
                ELSE
                  IF RETCODE < 0
                     THEN
                       MOVE ERRNO                  TO RECVFROM-ERRNO
                       MOVE RECVFROM-ERR           TO MSG-AREA
                       PERFORM HANDLE-TCPCICS      THRU
                               HANDLE-TCPCICS-EXIT
                       MOVE '1'                    TO TASK-FLAG
                     ELSE
|                      CALL 'EZACIC05' USING TCP-BUF TCPLENG
                       IF TCP-BUF-H = LOW-VALUES OR SPACES
                          THEN
                            MOVE NULL-DATA         TO MSG-AREA
                            PERFORM HANDLE-TCPCICS THRU
                                    HANDLE-TCPCICS-EXIT
                          ELSE
                            IF TCP-BUF-H =  'END'
                               THEN MOVE '1'       TO TASK-FLAG
```

Figure 177. EZACICSS IPv4 iterative server sample (Part 17 of 22)

```
                        ELSE IF TCP-BUF-H = 'TRM'
                                THEN MOVE '2' TO TASK-FLAG
                                ELSE PERFORM TALK-CLIENT THRU
                                             TALK-CLIENT-EXIT
                             END-IF
                     END-IF
                END-IF
           END-IF
      END-IF.
  ACCEPT-RECV-EXIT.
      EXIT.
 *****************************************************************
 **    PROCESSES TALKING TO CLIENT THAT WILL UPDATE DB2  **
 **    TABLES.                                           **
 *****************************************************************
 **    DATA PROCESS:                                     **
 **                                                      **
 **    INSERT REC -  INS,X81,TEST DEPT,A0213B,Y94        **
 **    UPDATE REC -  UPD,X81,,A1234C,                    **
 **    DELETE REC -  DEL,X81,,,                          **
 **    END CLIENT -  END,{end client connection     }    **
 **    END SERVER -  TRM,{terminate server          }    **
 **                                                      **
 *****************************************************************
  TALK-CLIENT.
      UNSTRING TCP-BUF DELIMITED BY DEL-ID OR ALL '*'
          INTO IN-ACT
               IN-DEPTNO
               IN-DEPTN
               IN-MGRNO
               IN-ADMRDEPT.
      IF IN-ACT EQUAL 'END'
         THEN
           MOVE '1'                            TO TASK-FLAG
         ELSE
           IF IN-ACT EQUAL 'U' OR EQUAL 'UPD'
              THEN
| ***           EXEC SQL UPDATE TCPCICS.DEPT
| ***             SET    MGRNO  = :IN-MGRNO
| ***             WHERE  DEPTNO = :IN-DEPTNO
| ***           END-EXEC
|               MOVE 'UPDATE'                   TO DB2-ACT
|               MOVE 'UPDATED:  '               TO DB2M-VAR
|             ELSE
|               IF IN-ACT EQUAL 'I' OR EQUAL 'INS'
|                  THEN
| ***                EXEC SQL INSERT
| ***                  INTO TCPCICS.DEPT (DEPTNO,     DEPTNAME,
| ***                                     MGRNO,      ADMRDEPT)
| ***                  VALUES           (:IN-DEPTNO, :IN-DEPTN,
| ***                                    :IN-MGRNO,  :IN-ADMRDEPT)
| ***                END-EXEC
|                    MOVE 'INSERT'              TO DB2-ACT
|                    MOVE 'INSERTED: '          TO DB2M-VAR
```

Figure 177. EZACICSS IPv4 iterative server sample (Part 18 of 22)

```
                         ELSE
                           IF IN-ACT EQUAL 'D' OR EQUAL 'DEL'
                              THEN
      ***                       EXEC SQL DELETE
      ***                         FROM  TCPCICS.DEPT
      ***                         WHERE DEPTNO = :IN-DEPTNO
      ***                       END-EXEC
                                MOVE 'DELETE'          TO DB2-ACT
                                MOVE 'DELETED: '       TO DB2M-VAR
                              ELSE
                                MOVE KEYWORD-ERR       TO MSG-AREA
                                PERFORM HANDLE-TCPCICS THRU
                                        HANDLE-TCPCICS-EXIT
                           END-IF
                      END-IF
                 END-IF
            END-IF.
            IF DADELETE OR DAINSERT OR DAUPDATE
               THEN
      *          MOVE SQLERRD(3)                       TO DB2CODE
                 MOVE DB2MSG                           TO MSG-AREA
                 MOVE LENGTH OF TCPCICS-MSG-AREA       TO LENG
                 EXEC CICS SYNCPOINT END-EXEC
                 EXEC CICS WRITEQ TD
                      QUEUE    ('CSMT')
                      FROM     (TCPCICS-MSG-AREA)
                      LENGTH   (LENG)
                      NOHANDLE
                 END-EXEC
      **********************************************************
      **          WRITE THE DB2 MESSAGE TO CLIENT.            **
      **********************************************************
                 MOVE TCPCICS-MSG-2                    TO TCP-BUF
                 CALL 'EZACIC04' USING TCP-BUF TCPLENG
                 CALL 'EZASOKET' USING SOKET-WRITE
                                       CLI-SOCKID
                                       TCPLENG
                                       TCP-BUF
                                       ERRNO
                                       RETCODE
                 MOVE LOW-VALUES                       TO TCP-BUF
                                                          TCP-INDICATOR
                                                          DB2-ACT
                 IF RETCODE < 0
                    THEN
                      MOVE ERRNO                       TO WRITE-ERRNO
                      MOVE WRITE-ERR                   TO MSG-AREA
                      PERFORM HANDLE-TCPCICS           THRU
                              HANDLE-TCPCICS-EXIT
                      MOVE '1'                         TO TASK-FLAG
```

Figure 177. EZACICSS IPv4 iterative server sample (Part 19 of 22)

```
           END-IF
       END-IF.
   TALK-CLIENT-EXIT.
       EXIT.
  *---------------------------------------------------------------*
  *                                                               *
  *   CLOSE ORIGINAL SOCKET DESCRIPTOR                            *
  *                                                               *
  *---------------------------------------------------------------*
   CLOSE-SOCKET.
       CALL 'EZASOKET' USING SOKET-CLOSE
                             SRV-SOCKID
                             ERRNO
                             RETCODE.
       IF RETCODE < 0 THEN
          MOVE ERRNO             TO CLOSE-ERRNO
          MOVE CLOSE-ERR         TO MSG-AREA
          PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
   CLOSE-SOCKET-EXIT.
       EXIT.
  *---------------------------------------------------------------*
  *                                                               *
  *  SEND TCP/IP ERROR MESSAGE                                    *
  *                                                               *
  *---------------------------------------------------------------*
   HANDLE-TCPCICS.
       MOVE LENGTH OF TCPCICS-MSG-AREA TO LENG.
       EXEC CICS ASKTIME
            ABSTIME (TSTAMP)
            NOHANDLE
       END-EXEC.
       EXEC CICS FORMATTIME
            ABSTIME (TSTAMP)
            MMDDYY  (MSGDATE)
            TIME    (MSGTIME)
            DATESEP ('/')
            TIMESEP (':')
            NOHANDLE
       END-EXEC.
       EXEC CICS WRITEQ TD
            QUEUE  ('CSMT')
            FROM   (TCPCICS-MSG-AREA)
            RESP   (RESPONSE)
            LENGTH (LENG)
       END-EXEC.
       IF RESPONSE = DFHRESP(NORMAL)
          THEN NEXT SENTENCE
          ELSE
            IF RESPONSE = DFHRESP(INVREQ)
               THEN MOVE TS-INVREQ-ERR          TO MSG-AREA
               ELSE
                 IF RESPONSE = DFHRESP(NOTAUTH)
                    THEN MOVE TS-NOTAUTH-ERR    TO MSG-AREA
```

Figure 177. EZACICSS IPv4 iterative server sample (Part 20 of 22)

```
                  ELSE
                    IF RESPONSE = DFHRESP(IOERR)
                       THEN MOVE TS-IOERR-ERR TO MSG-AREA
                       ELSE MOVE WRITETS-ERR  TO MSG-AREA
                    END-IF
               END-IF
          END-IF
     END-IF.
     IF TCP-INDICATOR = 'T' THEN
        MOVE BUFFER-LENG            TO TCPLENG
        MOVE LOW-VALUES             TO TCP-BUF
        MOVE TCPCICS-MSG-2          TO TCP-BUF
        CALL 'EZACIC04' USING TCP-BUF TCPLENG
        MOVE ' '                    TO TCP-INDICATOR
        CALL 'EZASOKET' USING SOKET-WRITE
                              CLI-SOCKID
                              TCPLENG
                              TCP-BUF
                              ERRNO
                              RETCODE
        IF RETCODE < 0
           THEN
             MOVE ERRNO             TO WRITE-ERRNO
             MOVE WRITE-ERR         TO MSG-AREA
             EXEC CICS WRITEQ TD
                  QUEUE  ('CSMT')
                  FROM   (TCPCICS-MSG-AREA)
                  LENGTH (LENG)
                  NOHANDLE
             END-EXEC
             IF TASK-TERM OR TASK-END
                THEN NEXT SENTENCE
                ELSE MOVE '1'       TO  TASK-FLAG
             END-IF
        END-IF.
     MOVE SPACES                    TO MSG-AREA.
 HANDLE-TCPCICS-EXIT.
     EXIT.
*---------------------------------------------------------------*
*                                                               *
*  SEND DB2    ERROR MESSAGE                                    *
*                                                               *
*---------------------------------------------------------------*
 SQL-ERROR-ROU.
*    MOVE SQLCODE         TO SQL-ERR-CODE.
     MOVE SPACES          TO MSG-AREA.
*    MOVE SQL-ERROR       TO MSG-AREA.
```

Figure 177. EZACICSS IPv4 iterative server sample (Part 21 of 22)

```
           EXEC CICS WRITEQ TD
                QUEUE  ('CSMT')
                FROM   (TCPCICS-MSG-AREA)
                RESP   (RESPONSE)
                LENGTH (LENG)
           END-EXEC.
           MOVE LOW-VALUES      TO TCP-BUF.
           MOVE TCPCICS-MSG-2   TO TCP-BUF.
|          CALL 'EZACIC04' USING TCP-BUF TCPLENG.
|          CALL 'EZASOKET' USING SOKET-WRITE
|                                CLI-SOCKID
|                                TCPLENG
|                                TCP-BUF
|                                ERRNO
|                                RETCODE.
           IF RETCODE < 0 THEN
              MOVE ERRNO        TO WRITE-ERRNO
              MOVE WRITE-ERR    TO MSG-AREA
              PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
           GO TO PGM-EXIT.
       SQL-ERROR-ROU-EXIT.
           EXIT.
      *----------------------------------------------------------------*
      *                                                                *
      *  OTHER ERRORS (HANDLE CONDITION)                               *
      *                                                                *
      *----------------------------------------------------------------*
       INVREQ-ERR-SEC.
           MOVE TCP-EXIT-ERR      TO MSG-AREA.
           PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
           GO TO PGM-EXIT.
       IOERR-SEC.
           MOVE IOERR-ERR         TO MSG-AREA.
           PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
           GO TO PGM-EXIT.
       LENGERR-SEC.
           MOVE LENGERR-ERR       TO MSG-AREA.
           PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
           GO TO PGM-EXIT.
       NOSPACE-ERR-SEC.
           MOVE NOSPACE-ERR       TO MSG-AREA.
           PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
           GO TO PGM-EXIT.
       QIDERR-SEC.
           MOVE QIDERR-ERR        TO MSG-AREA.
           PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
           GO TO PGM-EXIT.
       ITEMERR-SEC.
           MOVE ITEMERR-ERR       TO MSG-AREA.
           PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
           GO TO PGM-EXIT.
       ENDDATA-SEC.
           MOVE ENDDATA-ERR       TO MSG-AREA.
           PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
           GO TO PGM-EXIT.
```

Figure 177. EZACICSS IPv4 iterative server sample (Part 22 of 22)

EZACIC6C

The following COBOL socket program is in the SEZAINST data set.

```
***************************************************************
*                                                             *
* Communications Server for z/OS,   Version 1, Release 9      *
*                                                             *
*                                                             *
* Copyright:    Licensed Materials - Property of IBM          *
*                                                             *
*               "Restricted Materials of IBM"                 *
*                                                             *
*               5694-A01                                      *
*                                                             *
*               Copyright IBM Corp. 2003, 2007                *
*                                                             *
*               US Government Users Restricted Rights -       *
*               Use, duplication or disclosure restricted by  *
*               GSA ADP Schedule Contract with IBM Corp.      *
*                                                             *
* Status:       CSV1R9                                        *
*                                                             *
* $MOD(EZACIC6C),COMP(CICS),PROD(TCPIP):                      *
*                                                             *
***************************************************************
* $SEG(EZACIC6C)
*--------------------------------------------------------------*
*                                                              *
*   Module Name : EZACIC6C                                     *
*                                                              *
*   Description :                                              *
*                                                              *
*      This is a sample CICS/TCP application program. It issues*
*      TAKESOCKET to obtain the socket passed from MASTER      *
*      SERVER and perform dialog function with CLIENT program. *
*                                                              *
*--------------------------------------------------------------*
*
 IDENTIFICATION DIVISION.
 PROGRAM-ID. EZACIC6C.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
*
 WORKING-STORAGE SECTION.
 77  TASK-START                     PIC X(40)
     VALUE IS 'TASK STARTING THRU CICS/TCPIP INTERFACE '.
 77  GNI-ERR                       PIC X(24)
     VALUE IS ' GETNAMEINFO FAIL      '.
 77  GNI-SUCCESS                   PIC X(24)
     VALUE IS ' GETNAMEINFO SUCCESSFUL'.
 77  GPN-ERR                       PIC X(24)
     VALUE IS ' GETPEERNAME FAIL      '.
```

Figure 178. EZACIC6C IPv6 child server sample (Part 1 of 12)

```
       77  GPN-SUCCESS                   PIC X(24)
            VALUE IS ' GETPEERNAME SUCCESSFUL'.
       77  TAKE-ERR                       PIC X(24)
            VALUE IS ' TAKESOCKET FAIL       '.
       77  TAKE-SUCCESS                    PIC X(24)
            VALUE IS ' TAKESOCKET SUCCESSFUL '.
       77  READ-ERR                        PIC X(24)
            VALUE IS ' READ SOCKET FAIL       '.
       77  READ-SUCCESS                    PIC X(24)
            VALUE IS ' READ SOCKET SUCCESSFUL '.
       77  WRITE-ERR                       PIC X(24)
            VALUE IS ' WRITE SOCKET FAIL      '.
       77  WRITE-END-ERR                       PIC X(32)
            VALUE IS ' WRITE SOCKET FAIL - PGM END MSG'.
       77  WRITE-SUCCESS                   PIC X(25)
            VALUE IS ' WRITE SOCKET SUCCESSFUL '.
       77  CLOS-ERR                        PIC X(24)
            VALUE IS ' CLOSE SOCKET FAIL      '.
       77  CLOS-SUCCESS                   PIC X(24)
            VALUE IS 'CLOSE SOCKET SUCCESSFUL '.
       77  INVREQ-ERR                     PIC X(24)
            VALUE IS 'INTERFACE IS NOT ACTIVE '.
       77  IOERR-ERR                     PIC X(24)
            VALUE IS 'IOERR OCCURRS           '.
       77  LENGERR-ERR                   PIC X(24)
            VALUE IS 'LENGERR ERROR           '.
       77  ITEMERR-ERR                   PIC X(24)
            VALUE IS 'ITEMERR ERROR           '.
       77  NOSPACE-ERR                   PIC X(24)
            VALUE IS 'NOSPACE CONDITION       '.
       77  QIDERR-ERR                   PIC X(24)
            VALUE IS 'QIDERR  CONDITION       '.
       77  ENDDATA-ERR                  PIC X(30)
            VALUE IS 'RETRIEVE DATA CAN NOT BE FOUND'.
       77  WRKEND                       PIC X(20)
            VALUE 'CONNECTION END      '.
       77  WRITE-SW                     PIC X(1)
            VALUE 'N'.
       77  FORCE-ERROR-MSG              PIC X(1)
            VALUE 'N'.
       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-GETCLIENTID     PIC X(16) VALUE 'GETCLIENTID     '.
           02 SOKET-GETHOSTBYADDR   PIC X(16) VALUE 'GETHOSTBYADDR   '.
           02 SOKET-GETHOSTBYNAME   PIC X(16) VALUE 'GETHOSTBYNAME   '.
           02 SOKET-GETHOSTID       PIC X(16) VALUE 'GETHOSTID       '.
           02 SOKET-GETHOSTNAME     PIC X(16) VALUE 'GETHOSTNAME     '.
           02 SOKET-GETPEERNAME     PIC X(16) VALUE 'GETPEERNAME     '.
           02 SOKET-GETNAMEINFO     PIC X(16) VALUE 'GETNAMEINFO     '.
           02 SOKET-GETSOCKNAME     PIC X(16) VALUE 'GETSOCKNAME     '.
           02 SOKET-GETSOCKOPT      PIC X(16) VALUE 'GETSOCKOPT      '.
```

Figure 178. EZACIC6C IPv6 child server sample (Part 2 of 12)

```
    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-READ            PIC X(16) VALUE 'READ            '.
    02 SOKET-RECV            PIC X(16) VALUE 'RECV            '.
    02 SOKET-RECVFROM        PIC X(16) VALUE 'RECVFROM        '.
    02 SOKET-SELECT          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           '.
01  WRKMSG.
    02 WRKM                         PIC X(14)
       VALUE IS 'DATA RECEIVED '.
*----------------------------------------------------------------*
*    program's variables                                         *
*----------------------------------------------------------------*
77  SUBTRACE                PIC X(8)  VALUE 'CONTRACE'.
77  RESPONSE                       PIC 9(9) COMP.
77  TASK-FLAG                      PIC X(1) VALUE '0'.
77  TAKE-SOCKET                    PIC 9(8) COMP.
77  DATA2-LENGTH                   PIC 9(04).
77  NTOP-FAMILY                    PIC 9(8) COMP.
77  NTOP-LENGTH                    PIC 9(4) COMP.
77  SOCKID                         PIC 9(4) COMP.
77  SOCKID-FWD                     PIC 9(8) COMP.
77  ERRNO                          PIC 9(8) COMP.
77  RETCODE                        PIC S9(8) COMP.
01  TCP-BUF.
    05 TCP-BUF-H                   PIC X(3) VALUE IS SPACES.
    05 TCP-BUF-DATA                PIC X(197) VALUE IS SPACES.
77  TCPLENG                        PIC 9(8) COMP.
77  RECV-FLAG                      PIC 9(8) COMP.
77  CLENG                          PIC 9(4) COMP.
77  CPTRREF                        PIC 9(8) COMP.
77  CNT                            PIC 9(4) COMP.
77  MSGLENG                        PIC 9(4) COMP.
01  ZERO-PARM                  PIC X(16) VALUE LOW-VALUES.
01  DUMMY-MASK REDEFINES ZERO-PARM.
    05 DUMYMASK                PIC X(8).
    05 ZERO-FLD-8              PIC X(8).
01  ZERO-FLD REDEFINES ZERO-PARM.
    05 ZERO-FWRD               PIC 9(8)  COMP.
    05 ZERO-HWRD               PIC 9(4)  COMP.
    05 ZERO-DUM                PIC X(10).
01  TD-MSG.
    03 TASK-LABEL              PIC X(07) VALUE 'TASK # '.
```

Figure 178. EZACIC6C IPv6 child server sample (Part 3 of 12)

```
           03 TASK-NUMBER              PIC 9(07).
           03 TASK-SEP                 PIC X     VALUE ' '.
           03  CICS-MSG-AREA           PIC X(70).
       01  CICS-DETAIL-AREA.
           03  DETAIL-FIELD        PIC X(20).
           03  DETAIL-EQUALS       PIC X(02) VALUE '= '.
           03  DETAIL-DATA         PIC X(48) VALUE SPACES.
       01  CICS-ERR-AREA.
           03  ERR-MSG             PIC X(24).
           03  SOCK-HEADER         PIC X(08) VALUE ' SOCKET='.
           03  ERR-SOCKET          PIC 9(05).
           03  RETC-HEADER         PIC X(09) VALUE ' RETCDE=-'.
           03  ERR-RETCODE         PIC 9(05).
           03  ERRN-HEADER         PIC X(07) VALUE ' ERRNO='.
           03  ERR-ERRNO           PIC 9(05).
       01  CICS-DATA2-AREA.
           05 DATA-2-FOR-MSG       PIC X(48) VALUE SPACES.
           05 FILLER               PIC X(951).
      *
       01  CLIENTID-LSTN.
           05 CID-DOMAIN-LSTN                PIC 9(8) COMP.
           05 CID-NAME-LSTN                  PIC X(8).
           05 CID-SUBTASKNAME-LSTN           PIC X(8).
           05 CID-RES-LSTN                   PIC X(20).
       01  CLIENTID-APPL.
           05 CID-DOMAIN-APPL                PIC 9(8) COMP.
           05 CID-NAME-APPL                  PIC X(8).
           05 CID-SUBTASKNAME-APPL           PIC X(8).
           05 CID-RES-APPL                   PIC X(20).
      *
      * GETNAMEINFO Call variables.
      *
       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.
      *
      * GETNAMEINFO FLAG VALUES
      *
       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-NUMERICSERV                  PIC 9(8) BINARY VALUE 8.
       01  NI-DGRAM                        PIC 9(8) BINARY VALUE 16.
      *
      * GETPEERNAME SOCKET ADDRESS STRUCTURE
      *
       01 PEER-NAME.
          05 PEER-FAMILY                 PIC 9(4) BINARY.
             88 PEER-FAMILY-IS-AFINET    VALUE 2.
             88 PEER-FAMILY-IS-AFINET6   VALUE 19.
          05 PEER-DATA                   PIC X(26).
          05 PEER-SIN REDEFINES PEER-DATA.
```

Figure 178. EZACIC6C IPv6 child server sample (Part 4 of 12)

```
          10 PEER-SIN-PORT           PIC 9(4) BINARY.
          10 PEER-SIN-ADDR           PIC 9(8) BINARY.
          10 FILLER                  PIC X(8).
          10 FILLER                  PIC X(12).
       05 PEER-SIN6 REDEFINES PEER-DATA.
          10 PEER-SIN6-PORT          PIC 9(4) BINARY.
          10 PEER-SIN6-FLOWINFO      PIC 9(8) BINARY.
          10 PEER-SIN6-ADDR.
             15 FILLER               PIC 9(16) BINARY.
             15 FILLER               PIC 9(16) BINARY.
          10 PEER-SIN6-SCOPEID       PIC 9(8) BINARY.
      *
      * TRANSACTION INPUT MESSAGE FROMT THE LISTENER
      *
       01  TCPSOCKET-PARM.
           05 GIVE-TAKE-SOCKET             PIC 9(8) COMP.
           05 LSTN-NAME                    PIC X(8).
           05 LSTN-SUBTASKNAME             PIC X(8).
           05 CLIENT-IN-DATA               PIC X(35).
           05 THREADSAFE-INDICATOR         PIC X(1).
              88 INTERFACE-IS-THREADSAFE            VALUE '1'.
           05 SOCKADDR-IN.
              10 SOCK-FAMILY               PIC 9(4) BINARY.
                 88 SOCK-FAMILY-IS-AFINET   VALUE 2.
                 88 SOCK-FAMILY-IS-AFINET6  VALUE 19.
              10 SOCK-DATA                 PIC X(26).
              10 SOCK-SIN REDEFINES SOCK-DATA.
                 15 SOCK-SIN-PORT          PIC 9(4) BINARY.
                 15 SOCK-SIN-ADDR          PIC 9(8) BINARY.
                 15 FILLER                 PIC X(8).
                 15 FILLER                 PIC X(12).
              10 SOCK-SIN6 REDEFINES SOCK-DATA.
                 15 SOCK-SIN6-PORT         PIC 9(4) BINARY.
                 15 SOCK-SIN6-FLOWINFO     PIC 9(8) BINARY.
                 15 SOCK-SIN6-ADDR.
                    20 FILLER              PIC 9(16) BINARY.
                    20 FILLER              PIC 9(16) BINARY.
                 15 SOCK-SIN6-SCOPEID      PIC 9(8) BINARY.
           05 FILLER                       PIC X(68).
           05 CLIENT-IN-DATA-LENGTH        PIC 9(4) COMP.
           05 CLIENT-IN-DATA-2             PIC X(999).
       PROCEDURE DIVISION.
           MOVE 'Y' TO WRITE-SW.
           EXEC CICS HANDLE CONDITION INVREQ  (INVREQ-ERR-SEC)
                                      IOERR   (IOERR-SEC)
                                      ENDDATA (ENDDATA-SEC)
                                      NOSPACE (NOSPACE-ERR-SEC)
                                      QIDERR  (QIDERR-SEC)
                                      ITEMERR (ITEMERR-SEC)
                END-EXEC.
           EXEC CICS IGNORE CONDITION LENGERR
                END-EXEC.
           PERFORM INITIAL-SEC     THRU   INITIAL-SEC-EXIT.
           PERFORM TAKESOCKET-SEC  THRU   TAKESOCKET-SEC-EXIT.
           PERFORM GET-PEER-NAME   THRU   GET-PEER-NAME-EXIT.
```

Figure 178. EZACIC6C IPv6 child server sample (Part 5 of 12)

```
     PERFORM GET-NAME-INFO   THRU   GET-NAME-INFO-EXIT.
     MOVE '0' TO TASK-FLAG.
     PERFORM CLIENT-TASK     THRU   CLIENT-TASK-EXIT
         VARYING CNT FROM 1 BY 1  UNTIL TASK-FLAG = '1'.
 CLOSE-SOCK.
*---------------------------------------------------------------*
*                                                               *
*   CLOSE 'accept descriptor'                                   *
*                                                               *
*---------------------------------------------------------------*
     CALL 'EZASOKET' USING SOKET-CLOSE SOCKID
           ERRNO RETCODE.
     IF RETCODE <  0 THEN
        MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
        MOVE CLOS-ERR TO ERR-MSG
        MOVE SOCKID TO ERR-SOCKET
        MOVE RETCODE TO ERR-RETCODE
        MOVE ERRNO TO ERR-ERRNO
        MOVE CICS-ERR-AREA TO CICS-MSG-AREA
     ELSE
        MOVE CLOS-SUCCESS TO CICS-MSG-AREA.
     PERFORM WRITE-CICS  THRU WRITE-CICS-EXIT.
 PGM-EXIT.
     IF RETCODE < 0 THEN
        EXEC CICS ABEND ABCODE('SRV6') END-EXEC.
     MOVE SPACES TO CICS-MSG-AREA.
     MOVE 'END OF EZACIC6C PROGRAM' TO CICS-MSG-AREA.
     PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
     EXEC CICS RETURN  END-EXEC.
     GOBACK.
*---------------------------------------------------------------*
*
*  RECEIVE PASSED PARAMETER WHICH ARE CID                       *
*
*---------------------------------------------------------------*
 INITIAL-SEC.
     MOVE SPACES TO CICS-MSG-AREA.
     MOVE 50 TO MSGLENG.
     MOVE 'SRV6 TRANSACTION START UP     ' TO CICS-MSG-AREA.
     PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
*
*  PREPARE TO RECEIVE AND ENHANCED TIM
*
     MOVE 1153 TO CLENG.
     INITIALIZE TCPSOCKET-PARM.
     EXEC CICS RETRIEVE INTO(TCPSOCKET-PARM)
                        LENGTH(CLENG)
                        END-EXEC.
     MOVE 'LISTENER ADDR SPACE ' TO DETAIL-FIELD.
     MOVE SPACES TO DETAIL-DATA.
     MOVE LSTN-NAME TO DETAIL-DATA.
     MOVE CICS-DETAIL-AREA TO CICS-MSG-AREA.
     PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
     MOVE 'LISTENER TASK ID    ' TO DETAIL-FIELD.
     MOVE SPACES TO DETAIL-DATA.
```

Figure 178. EZACIC6C IPv6 child server sample (Part 6 of 12)

```
     MOVE LSTN-SUBTASKNAME TO DETAIL-DATA.
     MOVE CICS-DETAIL-AREA TO CICS-MSG-AREA.
     PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
     IF CLIENT-IN-DATA-LENGTH <= 0
         MOVE 'TIM IS STANDARD' TO CICS-MSG-AREA
         PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
         MOVE 'CLIENT IN DATA      ' TO DETAIL-FIELD
         MOVE SPACES TO DETAIL-DATA
         MOVE CLIENT-IN-DATA TO DETAIL-DATA
         MOVE CICS-DETAIL-AREA TO CICS-MSG-AREA
         PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
     ELSE
         MOVE 'TIM IS ENHANCED' TO CICS-MSG-AREA
         PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
         MOVE 'CLIENT IN DATA      ' TO DETAIL-FIELD
         MOVE SPACES TO DETAIL-DATA
         MOVE CLIENT-IN-DATA TO DETAIL-DATA
         MOVE CICS-DETAIL-AREA TO CICS-MSG-AREA
         PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
         MOVE 'CLIENT IN DATA 2 LEN' TO DETAIL-FIELD
         MOVE SPACES TO DETAIL-DATA
         MOVE CLIENT-IN-DATA-LENGTH TO DATA2-LENGTH
         MOVE DATA2-LENGTH TO DETAIL-DATA
         MOVE CICS-DETAIL-AREA TO CICS-MSG-AREA
         PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
         MOVE 'CLIENT IN DATA 2    ' TO DETAIL-FIELD
         MOVE SPACES TO DETAIL-DATA
         MOVE CLIENT-IN-DATA-2 TO CICS-DATA2-AREA
         MOVE DATA-2-FOR-MSG TO DETAIL-DATA
         MOVE CICS-DETAIL-AREA TO CICS-MSG-AREA
         PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
 INITIAL-SEC-EXIT.
     EXIT.
*---------------------------------------------------------------*
*                                                               *
*  Perform TCP SOCKET functions by passing socket command to    *
*  EZASOKET routine.  SOCKET command are translated to pre-     *
*  define integer.                                              *
*                                                               *
*---------------------------------------------------------------*
 TAKESOCKET-SEC.
*---------------------------------------------------------------*
*                                                               *
*   Issue 'TAKESOCKET' call to acquire a socket which was       *
*   given by the LISTENER program.                              *
*                                                               *
*---------------------------------------------------------------*
*    MOVE AF-INET TO CID-DOMAIN-LSTN CID-DOMAIN-APPL.
     MOVE SOCK-FAMILY TO CID-DOMAIN-LSTN CID-DOMAIN-APPL.
     MOVE LSTN-NAME TO CID-NAME-LSTN.
     MOVE LSTN-SUBTASKNAME TO CID-SUBTASKNAME-LSTN.
     MOVE GIVE-TAKE-SOCKET TO TAKE-SOCKET SOCKID SOCKID-FWD.
     CALL 'EZASOKET' USING SOKET-TAKESOCKET SOCKID
          CLIENTID-LSTN ERRNO RETCODE.
     IF RETCODE <  0 THEN
```

Figure 178. EZACIC6C IPv6 child server sample (Part 7 of 12)

```
               MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
               MOVE TAKE-ERR TO ERR-MSG
               MOVE SOCKID TO ERR-SOCKET
               MOVE RETCODE TO ERR-RETCODE
               MOVE ERRNO TO ERR-ERRNO
               MOVE CICS-ERR-AREA TO CICS-MSG-AREA
               PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
               GO TO PGM-EXIT
           ELSE
               MOVE SPACES TO CICS-MSG-AREA
               MOVE TAKE-SUCCESS TO CICS-MSG-AREA
               PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
           MOVE SPACES TO CICS-MSG-AREA.
           IF SOCK-FAMILY-IS-AFINET
               MOVE 'TOOK AN AF_INET SOCKET' TO CICS-MSG-AREA
               PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
               MOVE SPACES TO DETAIL-DATA
               MOVE 'AF_INET ADDRESS IS ' TO DETAIL-FIELD
               MOVE SOCK-FAMILY TO NTOP-FAMILY
               MOVE 16 TO NTOP-LENGTH
               CALL 'EZASOKET' USING SOKET-NTOP
                                     NTOP-FAMILY
                                     SOCK-SIN-ADDR
                                     DETAIL-DATA
                                     NTOP-LENGTH
                                     ERRNO
                                     RETCODE
           ELSE
               MOVE 'TOOK AN AF_INET6 SOCKET' TO CICS-MSG-AREA
               PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
               MOVE 'AF_INET6 ADDRESS IS ' TO DETAIL-FIELD
               MOVE SPACES TO DETAIL-DATA
               MOVE SOCK-FAMILY TO NTOP-FAMILY
               MOVE 45 TO NTOP-LENGTH
               CALL 'EZASOKET' USING SOKET-NTOP
                                     NTOP-FAMILY
                                     SOCK-SIN6-ADDR
                                     DETAIL-DATA
                                     NTOP-LENGTH
                                     ERRNO
                                     RETCODE.
           MOVE CICS-DETAIL-AREA TO CICS-MSG-AREA.
           PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
           MOVE RETCODE TO SOCKID.
           MOVE SPACES TO TCP-BUF.
           MOVE TASK-START TO TCP-BUF.
           MOVE 50  TO TCPLENG.
      *
      *    REMOVE FOLLOWING STATEMENT IF USING EBCDIC CLIENT
      *
|          CALL 'EZACIC04' USING TCP-BUF TCPLENG.
           CALL 'EZASOKET' USING SOKET-WRITE SOCKID TCPLENG
                 TCP-BUF ERRNO RETCODE.
```

Figure 178. EZACIC6C IPv6 child server sample (Part 8 of 12)

```
    IF RETCODE <  0 THEN
       MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
       MOVE WRITE-ERR TO ERR-MSG
       MOVE SOCKID TO ERR-SOCKET
       MOVE RETCODE TO ERR-RETCODE
       MOVE ERRNO TO ERR-ERRNO
       MOVE CICS-ERR-AREA TO CICS-MSG-AREA
       PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
       GO TO PGM-EXIT
    ELSE
       MOVE WRITE-SUCCESS TO CICS-MSG-AREA
       PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
TAKESOCKET-SEC-EXIT.
    EXIT.
GET-PEER-NAME.
    CALL 'EZASOKET' USING SOKET-GETPEERNAME
       SOCKID PEER-NAME ERRNO RETCODE.
    IF RETCODE <  0 THEN
       MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
       MOVE GPN-ERR TO ERR-MSG
       MOVE SOCKID TO ERR-SOCKET
       MOVE RETCODE TO ERR-RETCODE
       MOVE ERRNO TO ERR-ERRNO
       MOVE CICS-ERR-AREA TO CICS-MSG-AREA
       PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
       GO TO PGM-EXIT
    ELSE
       MOVE GPN-SUCCESS TO CICS-MSG-AREA
       PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
GET-PEER-NAME-EXIT.
    EXIT.
GET-NAME-INFO.
    IF PEER-FAMILY-IS-AFINET
       MOVE 16 TO NAME-LEN
    ELSE
       MOVE 28 TO NAME-LEN.
    MOVE SPACES TO HOST-NAME.
    MOVE 256 TO HOST-NAME-LEN.
    MOVE SPACES TO SERVICE-NAME.
    MOVE 32 TO SERVICE-NAME-LEN.
    CALL 'EZASOKET' USING SOKET-GETNAMEINFO
       PEER-NAME NAME-LEN
       HOST-NAME HOST-NAME-LEN
       SERVICE-NAME SERVICE-NAME-LEN
       NAME-INFO-FLAGS
       ERRNO RETCODE.
    IF RETCODE <  0 THEN
       MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
       MOVE GNI-ERR TO ERR-MSG
       MOVE SOCKID TO ERR-SOCKET
       MOVE RETCODE TO ERR-RETCODE
       MOVE ERRNO TO ERR-ERRNO
       MOVE CICS-ERR-AREA TO CICS-MSG-AREA
       PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
       GO TO PGM-EXIT
```

Figure 178. EZACIC6C IPv6 child server sample (Part 9 of 12)

```
     ELSE
        MOVE GNI-SUCCESS TO CICS-MSG-AREA
        PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
 GET-NAME-INFO-EXIT.
     EXIT.
 CLIENT-TASK.
*---------------------------------------------------------------*
*                                                               *
*  Issue 'RECV' socket to receive input data from client       *
*                                                               *
*---------------------------------------------------------------*
     MOVE LOW-VALUES TO TCP-BUF.
     MOVE 200 TO TCPLENG.
     MOVE ZEROS TO RECV-FLAG.
     CALL 'EZASOKET' USING SOKET-RECV SOCKID
          RECV-FLAG TCPLENG TCP-BUF ERRNO RETCODE.
     IF RETCODE <  0 THEN
        MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
        MOVE READ-ERR TO ERR-MSG
        MOVE SOCKID TO ERR-SOCKET
        MOVE RETCODE TO ERR-RETCODE
        MOVE ERRNO TO ERR-ERRNO
        MOVE CICS-ERR-AREA TO CICS-MSG-AREA
        PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
        GO TO PGM-EXIT
     ELSE
        MOVE READ-SUCCESS TO CICS-MSG-AREA
        PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
*
*    REMOVE FOLLOWING STATEMENT IF USING EBCDIC CLIENT
*
     CALL 'EZACIC05' USING TCP-BUF TCPLENG.
*
*    DETERMINE WHETHER THE CLIENT IS FINISHED SENDING DATA
*
     IF TCP-BUF-H = 'END' OR TCP-BUF-H = 'end' THEN
        MOVE '1' TO TASK-FLAG
        PERFORM CLIENT-TALK-END THRU CLIENT-TALK-END-EXIT
        GO TO CLIENT-TASK-EXIT.
     IF RETCODE = 0  THEN
        MOVE '1' TO TASK-FLAG
        GO TO CLIENT-TASK-EXIT.
*---------------------------------------------------------------*
**  ECHO RECEIVING DATA
*---------------------------------------------------------------*
     MOVE TCP-BUF TO CICS-MSG-AREA.
     PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
     MOVE RETCODE TO TCPLENG.
*
*    REMOVE FOLLOWING STATEMENT IF USING EBCDIC CLIENT
*
     CALL 'EZACIC04' USING TCP-BUF TCPLENG.
```

Figure 178. EZACIC6C IPv6 child server sample (Part 10 of 12)

```
           CALL 'EZASOKET' USING SOKET-WRITE SOCKID TCPLENG
                TCP-BUF ERRNO RETCODE.
           IF RETCODE <  0 THEN
              MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
              MOVE WRITE-ERR TO ERR-MSG
              MOVE SOCKID TO ERR-SOCKET
              MOVE RETCODE TO ERR-RETCODE
              MOVE ERRNO TO ERR-ERRNO
              MOVE CICS-ERR-AREA TO CICS-MSG-AREA
              PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
              GO TO PGM-EXIT
           ELSE
              MOVE WRITE-SUCCESS TO CICS-MSG-AREA
              PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
       CLIENT-TASK-EXIT.
           EXIT.
       WRITE-CICS.
           MOVE 78 TO CLENG.
           MOVE EIBTASKN TO TASK-NUMBER.
           IF WRITE-SW = 'Y' THEN
               IF INTERFACE-IS-THREADSAFE THEN
                   IF FORCE-ERROR-MSG = 'Y' THEN
                       EXEC CICS WRITEQ TD QUEUE('CSMT') FROM(TD-MSG)
                            LENGTH(CLENG) NOHANDLE
                       END-EXEC
                   ELSE
                       NEXT SENTENCE
               ELSE
                   EXEC CICS WRITEQ TD QUEUE('CSMT') FROM(TD-MSG)
                        LENGTH(CLENG) NOHANDLE
                   END-EXEC
           ELSE
               NEXT SENTENCE.
           MOVE SPACES TO CICS-MSG-AREA.
       WRITE-CICS-EXIT.
           EXIT.
       CLIENT-TALK-END.
              MOVE LOW-VALUES TO TCP-BUF.
              MOVE WRKEND TO TCP-BUF CICS-MSG-AREA.
              MOVE 50 TO TCPLENG.
      *
      *    REMOVE FOLLOWING STATEMENT IF USING EBCDIC CLIENT
      *
              CALL 'EZACIC04' USING TCP-BUF TCPLENG.
              CALL 'EZASOKET' USING SOKET-WRITE SOCKID TCPLENG
                   TCP-BUF ERRNO RETCODE.
              IF RETCODE <  0 THEN
                 MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
                 MOVE WRITE-END-ERR TO ERR-MSG
                 MOVE SOCKID TO ERR-SOCKET
                 MOVE RETCODE TO ERR-RETCODE
                 MOVE ERRNO TO ERR-ERRNO
```

Figure 178. EZACIC6C IPv6 child server sample (Part 11 of 12)

```
            MOVE CICS-ERR-AREA TO CICS-MSG-AREA
            PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
            GO TO PGM-EXIT.
       CLIENT-TALK-END-EXIT.
           EXIT.
       INVREQ-ERR-SEC.
           MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
           MOVE INVREQ-ERR TO CICS-MSG-AREA.
           PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
           GO TO PGM-EXIT.
       IOERR-SEC.
           MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
           MOVE IOERR-ERR TO CICS-MSG-AREA.
           PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
           GO TO PGM-EXIT.
       LENGERR-SEC.
           MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
           MOVE LENGERR-ERR TO CICS-MSG-AREA.
           PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
           GO TO PGM-EXIT.
       NOSPACE-ERR-SEC.
           MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
           MOVE NOSPACE-ERR TO CICS-MSG-AREA.
           PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
           GO TO PGM-EXIT.
       QIDERR-SEC.
           MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
           MOVE QIDERR-ERR TO CICS-MSG-AREA.
           PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
           GO TO PGM-EXIT.
       ITEMERR-SEC.
           MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
           MOVE ITEMERR-ERR TO CICS-MSG-AREA.
           PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
           GO TO PGM-EXIT.
       ENDDATA-SEC.
           MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
           MOVE ENDDATA-ERR TO CICS-MSG-AREA.
           PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
           GO TO PGM-EXIT.
```

Figure 178. EZACIC6C IPv6 child server sample (Part 12 of 12)

EZACIC6S

The following COBOL socket program is in the SEZAINST data set.

```
***************************************************************
*                                                             *
* Communications Server for z/OS     Version 1, Release 9     *
*                                                             *
*                                                             *
* Copyright:    Licensed Materials - Property of IBM          *
*                                                             *
*               "Restricted Materials of IBM"                 *
*                                                             *
*               5694-A01                                      *
*                                                             *
*               Copyright IBM Corp. 2003, 2007                *
*                                                             *
*               US Government Users Restricted Rights -       *
*               Use, duplication or disclosure restricted by  *
*               GSA ADP Schedule Contract with IBM Corp.      *
*                                                             *
* Status:       CSV1R9                                        *
*                                                             *
* $MOD(EZACIC6S),COMP(CICS),PROD(TCPIP):                      *
*                                                             *
***************************************************************
* $SEG(EZACIC6S)
*--------------------------------------------------------------*
*                                                              *
*   Module Name :  EZACIC6S                                    *
*                                                              *
*   Description :  This is a sample server program.  It        *
*                  establishes a connection between            *
*                  CICS & TCPIP to process client requests.    *
*                  The server expects the data received        *
*                  from a host / workstation in ASCII.         *
*                  All responses sent by the server to the     *
*                  CLIENT are in ASCII.  This server is        *
*                  started using CECI or via the LISTENER.     *
*                                                              *
*                    CECI START TRANS(xxxx) from(yyyy)         *
*                       where xxxx is this servers CICS        *
*                       transaction id and yyyy is the         *
*                       port this server will listen on.       *
*                                                              *
*                  It processes request received from          *
*                  clients for updates to a hypothetical       *
*                  DB2 database.  Any and all references to    *
*                  DB2 or SQL are commented out as this        *
*                  sample is to illustrate CICS Sockets.       *
*                                                              *
```

Figure 179. EZACIC6S IPv6 iterative server sample (Part 1 of 24)

```
*                 A client connection is broken when the      *
*                 client transmits and 'END' token to the     *
*                 server.  All processing is terminated       *
*                 when an 'TRM' token is received from a      *
*                 client.                                     *
*                                                             *
*                                                             *
*-------------------------------------------------------------*
*                                                             *
*   LOGIC       :  1.  Establish server setup                 *
*                      a).  TRUE Active                       *
*                      b).  CAF Active                        *
*                  2.  Assign user specified port at          *
*                      start up or use the program            *
*                      declared default.                      *
*                  3.  Initialize the AF_INET6 socket.        *
*                  4.  Bind the port and in6addr_any.         *
*                  5.  Set Bit Mask to accept incoming        *
*                      read request.                          *
*                  6.  Process request from clients.          *
*                      a).  Wait for connection               *
*                      b).  Process request until 'END'       *
*                           token is receive from client.     *
*                      c).  Close connection.                 *
*                      note:  The current client request      *
*                             ends when the client closes     *
*                             the connection or sends an      *
*                             'END' token to the server.      *
*                      d).  If the last request received by   *
*                           the current client is not a       *
*                           request to the server to          *
*                           terminate processing ('TRM'),     *
*                           continue at step 6A.              *
*                  7.  Close the server's connection.         *
*                                                             *
*-------------------------------------------------------------*
 IDENTIFICATION DIVISION.
 PROGRAM-ID. EZACIC6S.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*--------------------------------------------------------------*
*   MESSAGES                                                   *
*--------------------------------------------------------------*
 77  BITMASK-ERR                     PIC X(30)
      VALUE IS 'BITMASK CONVERSION - FAILED   '.
 77  ENDDATA-ERR                     PIC X(30)
      VALUE IS 'RETRIEVE DATA CAN NOT BE FOUND'.
 77  INIT-MSG                        PIC X(30)
      VALUE IS 'INITAPI COMPLETE              '.
 77  IOERR-ERR                       PIC X(30)
      VALUE IS 'IOERR OCCURRS                 '.
 77  ITEMERR-ERR                     PIC X(30)
      VALUE IS 'ITEMERR ERROR                 '.
 77  KEYWORD-ERR                     PIC X(30)
```

Figure 179. EZACIC6S IPv6 iterative server sample (Part 2 of 24)

```
     VALUE IS 'INPUT KEYWORD ERROR           '.
77  LENGERR-ERR                     PIC X(30)
     VALUE IS 'LENGERR ERROR                 '.
77  NOSPACE-ERR                     PIC X(30)
     VALUE IS 'NOSPACE CONDITION             '.
77  NULL-DATA                       PIC X(30)
     VALUE IS 'READ NULL DATA                '.
77  QIDERR-ERR                      PIC X(30)
     VALUE IS 'TRANSIENT DATA QUEUE NOT FOUND'.
77  START-MSG                       PIC X(30)
     VALUE IS 'SERVER PROGRAM IS STARTING    '.
77  TCP-EXIT-ERR                    PIC X(30)
     VALUE IS 'SERVER STOPPED:TRUE NOT ACTIVE'.
77  TCP-SERVER-OFF                  PIC X(30)
     VALUE IS 'SERVER IS ENDING              '.
77  TS-INVREQ-ERR                   PIC X(30)
     VALUE IS 'WRITE TS FAILED  - INVREQ     '.
77  TS-NOTAUTH-ERR                  PIC X(30)
     VALUE IS 'WRITE TS FAILED  - NOTAUTH    '.
77  TS-IOERR-ERR                    PIC X(30)
     VALUE IS 'WRITE TS FAILED  - IOERR      '.
77  WRITETS-ERR                     PIC X(30)
     VALUE IS 'WRITE TS FAILED               '.
01  ACCEPT-ERR.
    05  ACCEPT-ERR-M                PIC X(25)
         VALUE IS 'SOCKET CALL FAIL - ACCEPT'.
    05  FILLER                      PIC X(9)
         VALUE IS ' ERRNO = '.
    05  ACCEPT-ERRNO                PIC 9(8) DISPLAY.
    05  FILLER                      PIC X(13)
         VALUE IS SPACES.
01  NTOP-ERR.
    05  NTOP-ERR-M                  PIC X(23)
         VALUE IS 'SOCKET CALL FAIL - NTOP'.
    05  FILLER                      PIC X(9)
         VALUE IS ' ERRNO = '.
    05  NTOP-ERRNO                  PIC 9(8) DISPLAY.
    05  FILLER                      PIC X(13)
         VALUE IS SPACES.
01  NTOP-OK.
    05  NTOP-OK-M                   PIC X(21)
         VALUE IS 'ACCEPTED IP ADDRESS: '.
    05  NTOP-PRESENTABLE-ADDR       PIC X(45) DISPLAY
         VALUE IS SPACES.
01  GNI-ERR.
    05  GNI-ERR-M                   PIC X(30)
         VALUE IS 'SOCKET CALL FAIL - GETNAMEINFO'.
    05  FILLER                      PIC X(9)
         VALUE IS ' ERRNO = '.
    05  GNI-ERRNO                   PIC 9(8) DISPLAY.
    05  FILLER                      PIC X(13)
         VALUE IS SPACES.
01  GNI-HOST-NAME-OK.
    05  FILLER                      PIC X(19)
         VALUE IS 'CLIENTS HOST NAME: '.
```

Figure 179. EZACIC6S IPv6 iterative server sample (Part 3 of 24)

```
    05  GNI-HOST-NAME              PIC X(255) DISPLAY
         VALUE IS SPACES.
01  GNI-SERVICE-NAME-OK.
    05  FILLER                     PIC X(22)
         VALUE IS 'CLIENTS SERVICE NAME: '.
    05  GNI-SERVICE-NAME           PIC X(32) DISPLAY
         VALUE IS SPACES.
01  GPN-ERR.
    05  GPN-ERR-M                  PIC X(30)
         VALUE IS 'SOCKET CALL FAIL - GETPEERNAME'.
    05  FILLER                     PIC X(9)
         VALUE IS ' ERRNO = '.
    05  GPN-ERRNO                  PIC 9(8) DISPLAY.
    05  FILLER                     PIC X(13)
         VALUE IS SPACES.
01  BIND-ERR.
    05  BIND-ERR-M                 PIC X(25)
         VALUE IS 'SOCKET CALL FAIL  -  BIND'.
    05  FILLER                     PIC X(9)
         VALUE IS ' ERRNO = '.
    05  BIND-ERRNO                 PIC 9(8) DISPLAY.
    05  FILLER                     PIC X(13)
         VALUE IS SPACES.
01  CLOSE-ERR.
    05  CLOSE-ERR-M                PIC X(30)
         VALUE IS 'CLOSE SOCKET DESCRIPTOR FAILED'.
    05  FILLER                     PIC X(9)
         VALUE IS ' ERRNO = '.
    05  CLOSE-ERRNO                PIC 9(8)  DISPLAY.
    05  FILLER                     PIC X(8)
         VALUE IS SPACES.
01  DB2END.
    05  FILLER                     PIC X(16)
         VALUE IS 'DB2 PROCESS ENDS'.
    05  FILLER                     PIC X(39)
         VALUE IS SPACES.
01  DB2-CAF-ERR.
    05  FILLER                     PIC X(24)
         VALUE IS 'CONNECT NOT ESTABLISHED '.
    05  FILLER                     PIC X(30)
         VALUE IS 'ATTACHMENT FACILITY NOT ACTIVE'.
    05  FILLER                     PIC X(1)
         VALUE IS SPACES.
01  DB2MSG.
    05  DB2-ACT                    PIC X(6)  VALUE SPACES.
        88 DAINSERT                          VALUE 'INSERT'.
        88 DADELETE                          VALUE 'DELETE'.
        88 DAUPDATE                          VALUE 'UPDATE'.
    05  DB2M                       PIC X(18)
         VALUE IS ' COMPLETE - #ROWS '.
    05  DB2M-VAR                   PIC X(10).
    05  FILLER                     PIC X(2)  VALUE SPACES.
    05  DB2CODE                    PIC -(9)9.
    05  FILLER                     PIC X(11)
         VALUE IS SPACES.
```

Figure 179. EZACIC6S IPv6 iterative server sample (Part 4 of 24)

```
       01  INITAPI-ERR.
           05  INITAPI-ERR-M               PIC X(35)
               VALUE IS 'INITAPI FAILED - SERVER NOT STARTED'.
           05  FILLER                      PIC X(9)
               VALUE IS ' ERRNO = '.
           05  INIT-ERRNO                  PIC 9(8) DISPLAY.
           05  FILLER                      PIC X(3)
               VALUE IS SPACES.
       01  LISTEN-ERR.
           05  LISTEN-ERR-M                PIC X(25)
               VALUE IS 'SOCKET CALL FAIL - LISTEN'.
           05  FILLER                      PIC X(9)
               VALUE IS ' ERRNO = '.
           05  LISTEN-ERRNO                PIC 9(8) DISPLAY.
           05  FILLER                      PIC X(13)
               VALUE IS SPACES.
       01  LISTEN-SUCC.
           05  FILLER                      PIC X(34)
               VALUE IS 'READY TO ACCEPT REQUEST ON PORT:  '.
           05  BIND-PORT                   PIC X(4).
           05  FILLER                      PIC X(10)  VALUE SPACES.
           05  FILLER                      PIC X(7)
               VALUE IS SPACES.
       01  PORTNUM-ERR.
           05  INVALID-PORT                PIC X(33)
               VALUE IS 'SERVER NOT STARTED - INVALID PORT'.
           05  FILLER                      PIC X(10)
               VALUE IS ' NUMBER = '.
           05  PORT-ERRNUM                 PIC X(4).
           05  FILLER                      PIC X(8)
               VALUE IS SPACES.
       01  RECVFROM-ERR.
           05  RECVFROM-ERR-M              PIC X(24)
               VALUE IS 'RECEIVE SOCKET CALL FAIL'.
           05  FILLER                      PIC X(9)
               VALUE IS ' ERRNO = '.
           05  RECVFROM-ERRNO              PIC 9(8) DISPLAY.
           05  FILLER                      PIC X(14)
               VALUE IS SPACES.
       01  SELECT-ERR.
           05  SELECT-ERR-M                PIC X(24)
               VALUE IS 'SELECT CALL FAIL        '.
           05  FILLER                      PIC X(9)
               VALUE IS ' ERRNO = '.
           05  SELECT-ERRNO                PIC 9(8) DISPLAY.
           05  FILLER                      PIC X(14)
               VALUE IS SPACES.
       01  SQL-ERROR.
           05  FILLER                      PIC X(35)
               VALUE IS 'SQLERR -PROG TERMINATION,SQLCODE = '.
           05  SQL-ERR-CODE                PIC -(9)9.
           05  FILLER                      PIC X(11)
               VALUE IS SPACES.
       01  SOCKET-ERR.
           05  SOCKET-ERR-M                PIC X(25)
```

Figure 179. EZACIC6S IPv6 iterative server sample (Part 5 of 24)

```
                VALUE IS 'SOCKET CALL FAIL - SOCKET'.
           05  FILLER                      PIC X(9)
                VALUE IS ' ERRNO = '.
           05  SOCKET-ERRNO                PIC 9(8) DISPLAY.
           05  FILLER                      PIC X(13)
                VALUE IS SPACES.
       01  TAKE-ERR.
           05  TAKE-ERR-M                  PIC X(17)
                VALUE IS 'TAKESOCKET FAILED'.
           05  FILLER                      PIC X(9)
                VALUE IS ' ERRNO = '.
           05  TAKE-ERRNO                  PIC 9(8) DISPLAY.
           05  FILLER                      PIC X(21)
                VALUE IS SPACES.
       01  WRITE-ERR.
           05  WRITE-ERR-M                 PIC X(33)
                VALUE IS 'WRITE SOCKET FAIL'.
           05  FILLER                      PIC X(9)
                VALUE IS ' ERRNO = '.
           05  WRITE-ERRNO                 PIC 9(8) DISPLAY.
           05  FILLER                      PIC X(21)
                VALUE IS SPACES.
      *----------------------------------------------------------------*
      *    PROGRAM'S CONSTANTS                                         *
      *----------------------------------------------------------------*
|      77  CTOB                     PIC X(4)  VALUE 'CTOB'.
       77  DEL-ID                   PIC X(1)  VALUE ','.
       77  BACKLOG                  PIC 9(8)  COMP VALUE 5.
       77  NONZERO-FWRD             PIC 9(8)  VALUE 256.
       77  TCP-FLAG                 PIC 9(8)  COMP VALUE 0.
       77  SOCK-TYPE                PIC 9(8)  COMP VALUE 1.
       77  AF-INET6                 PIC 9(8)  COMP VALUE 19.
       77  NUM-FDS                  PIC 9(8)  COMP VALUE 5.
       77  LOM                      PIC 9(4)  COMP VALUE 4.
       77  CECI-LENG                PIC 9(8)  COMP VALUE 5.
       77  BUFFER-LENG              PIC 9(8)  COMP VALUE 55.
       77  GWLENG                   PIC 9(4)  COMP VALUE 256.
       77  DEFAULT-PORT             PIC X(4)  VALUE '????'.
           88  DEFAULT-SPECIFIED              VALUE '1950'.
       01  IN6ADDR-ANY.
           05 FILLER                PIC 9(16) BINARY VALUE 0.
           05 FILLER                PIC 9(16) BINARY VALUE 0.
       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-GETCLIENTID     PIC X(16) VALUE 'GETCLIENTID     '.
           02 SOKET-GETHOSTBYADDR   PIC X(16) VALUE 'GETHOSTBYADDR   '.
           02 SOKET-GETHOSTBYNAME   PIC X(16) VALUE 'GETHOSTBYNAME   '.
           02 SOKET-GETHOSTID       PIC X(16) VALUE 'GETHOSTID       '.
           02 SOKET-GETHOSTNAME     PIC X(16) VALUE 'GETHOSTNAME     '.
```

Figure 179. EZACIC6S IPv6 iterative server sample (Part 6 of 24)

```
     02 SOKET-GETPEERNAME     PIC X(16) VALUE 'GETPEERNAME     '.
     02 SOKET-GETNAMEINFO     PIC X(16) VALUE 'GETNAMEINFO     '.
     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-IOCTL           PIC X(16) VALUE 'IOCTL           '.
     02 SOKET-LISTEN          PIC X(16) VALUE 'LISTEN          '.
     02 SOKET-NTOP            PIC X(16) VALUE 'NTOP            '.
     02 SOKET-READ            PIC X(16) VALUE 'READ            '.
     02 SOKET-RECV            PIC X(16) VALUE 'RECV            '.
     02 SOKET-RECVFROM        PIC X(16) VALUE 'RECVFROM        '.
     02 SOKET-SELECT          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           '.
*----------------------------------------------------------------*
*    PROGRAM'S VARIABLES                                         *
*----------------------------------------------------------------*
 77  PROTOCOL                 PIC 9(8)  COMP VALUE 0.
 77  SRV-SOCKID               PIC 9(4)  COMP VALUE 0.
 77  SRV-SOCKID-FWD           PIC 9(8)  COMP VALUE 0.
 77  CLI-SOCKID               PIC 9(4)  COMP VALUE 0.
 77  CLI-SOCKID-FWD           PIC S9(8) COMP VALUE 0.
 77  LENG                     PIC 9(4)  COMP.
 77  WSLENG                   PIC 9(4)  COMP.
 77  RESPONSE                 PIC 9(9)  COMP.
 77  TSTAMP                   PIC 9(8).
 77  TASK-FLAG                PIC X(1)  VALUE '0'.
     88  TASK-END             VALUE '1'.
     88  TASK-TERM            VALUE '2'.
 77  GWPTR                    PIC S9(8) COMP.
 77  WSPTR                    PIC S9(8) COMP.
 77  TCP-INDICATOR            PIC X(1)  VALUE IS SPACE.
 77  TAKESOCKET-SWITCH        PIC X(1)  VALUE IS SPACE.
     88  DOTAKESOCKET         VALUE '1'.
 77  TCPLENG                  PIC 9(8)  COMP VALUE 0.
 77  ERRNO                    PIC 9(8)  COMP.
 77  RETCODE                  PIC S9(8) COMP.
 77  TRANS                    PIC X(4).
 01  CLIENTID-LSTN.
     05  CID-DOMAIN-LSTN      PIC 9(8)  COMP VALUE 19.
     05  CID-LSTN-INFO.
         10  CID-NAME-LSTN    PIC X(8).
         10  CID-SUBTNAM-LSTN PIC X(8).
     05  CID-RES-LSTN         PIC X(20) VALUE LOW-VALUES.
 01  INIT-SUBTASKID.
     05  SUBTASKNO            PIC X(7)  VALUE LOW-VALUES.
```

Figure 179. EZACIC6S IPv6 iterative server sample (Part 7 of 24)

```
           05  SUBT-CHAR            PIC A(1)  VALUE 'L'.
       01  IDENT.
           05  TCPNAME              PIC X(8) VALUE 'TCPCS   '.
           05  ADSNAME              PIC X(8) VALUE 'EZACIC6S'.
       01  MAXSOC                   PIC 9(4) BINARY VALUE 0.
       01  MAXSNO                   PIC 9(8) BINARY VALUE 0.
       01  NFDS                     PIC 9(8) BINARY.
       01  PORT-RECORD.
           05  PORT                 PIC X(4).
           05  FILLER               PIC X(36).
       01  SELECT-CSOCKET.
           05  READMASK             PIC X(4)  VALUE LOW-VALUES.
           05  DUMYMASK             PIC X(4)  VALUE LOW-VALUES.
           05  REPLY-RDMASK         PIC X(4)  VALUE LOW-VALUES.
           05  REPLY-RDMASK-FF      PIC X(4).
       01 SOCKADDR-IN.
          05 SAIN-FAMILY                PIC 9(4) BINARY.
             88 SAIN-FAMILY-IS-AFINET   VALUE 2.
             88 SAIN-FAMILY-IS-AFINET6  VALUE 19.
          05 SAIN-DATA                  PIC X(26).
          05 SAIN-SIN REDEFINES SAIN-DATA.
             10 SAIN-SIN-PORT           PIC 9(4) BINARY.
             10 SAIN-SIN-ADDR           PIC 9(8) BINARY.
             10 FILLER                  PIC X(8).
             10 FILLER                  PIC X(12).
          05 SAIN-SIN6 REDEFINES SAIN-DATA.
             10 SAIN-SIN6-PORT          PIC 9(4) BINARY.
             10 SAIN-SIN6-FLOWINFO      PIC 9(8) BINARY.
             10 SAIN-SIN6-ADDR.
                15 FILLER               PIC 9(16) BINARY.
                15 FILLER               PIC 9(16) BINARY.
             10 SAIN-SIN6-SCOPEID       PIC 9(8) BINARY.
       01 SOCKADDR-PEER.
          05 PEER-FAMILY                PIC 9(4) BINARY.
             88 PEER-FAMILY-IS-AFINET   VALUE 2.
             88 PEER-FAMILY-IS-AFINET6  VALUE 19.
          05 PEER-DATA                  PIC X(26).
          05 PEER-SIN REDEFINES PEER-DATA.
             10 PEER-SIN-PORT           PIC 9(4) BINARY.
             10 PEER-SIN-ADDR           PIC 9(8) BINARY.
             10 FILLER                  PIC X(8).
             10 FILLER                  PIC X(12).
          05 PEER-SIN6 REDEFINES PEER-DATA.
             10 PEER-SIN6-PORT          PIC 9(4) BINARY.
             10 PEER-SIN6-FLOWINFO      PIC 9(8) BINARY.
             10 PEER-SIN6-ADDR.
                15 FILLER               PIC 9(16) BINARY.
                15 FILLER               PIC 9(16) BINARY.
             10 PEER-SIN6-SCOPEID       PIC 9(8) BINARY.
       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.
```

Figure 179. EZACIC6S IPv6 iterative server sample (Part 8 of 24)

```
    05 FILLER                      PIC 9(16) BINARY VALUE 0.
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-NUMERICSERV                 PIC 9(8) BINARY VALUE 8.
01  NI-DGRAM                       PIC 9(8) BINARY VALUE 16.
01  HOST-NAME-CHAR-COUNT       PIC 9(4) COMP.
01  HOST-NAME-UNSTRUNG         PIC X(255) VALUE SPACES.
01  SERVICE-NAME-CHAR-COUNT    PIC 9(4) COMP.
01  SERVICE-NAME-UNSTRUNG      PIC X(32) VALUE SPACES.
01  SOCKET-CONV.
    05  SOCKET-TBL  OCCURS 6 TIMES.
        10  SOCK-CHAR        PIC X(1)  VALUE '0'.
01  TCP-BUF.
    05  TCP-BUF-H            PIC X(3).
    05  TCP-BUF-DATA         PIC X(52).
01  TCPCICS-MSG-AREA.
    02  TCPCICS-MSG-1.
        05  MSGDATE          PIC 9(8).
        05  FILLER           PIC X(2)  VALUE SPACES.
        05  MSGTIME          PIC 9(8).
        05  FILLER           PIC X(2)  VALUE SPACES.
        05  MODULE           PIC X(10) VALUE 'EZACIC6S: '.
    02  TCPCICS-MSG-2.
        05  MSG-AREA         PIC X(55) VALUE SPACES.
01  TCP-INPUT-DATA              PIC X(85) VALUE LOW-VALUES.
01  TCPSOCKET-PARM REDEFINES TCP-INPUT-DATA.
    05 GIVE-TAKE-SOCKET            PIC 9(8) COMP.
    05 CLIENTID-PARM.
       10 LSTN-NAME                PIC X(8).
       10 LSTN-SUBTASKNAME         PIC X(8).
    05 CLIENT-DATA-FLD.
       10 CLIENT-IN-DATA           PIC X(35).
       10 FILLER                   PIC X(1).
    05 TCPSOCKADDR-IN.
       10 SOCK-FAMILY              PIC 9(4) BINARY.
          88 SOCK-FAMILY-IS-AFINET   VALUE 2.
          88 SOCK-FAMILY-IS-AFINET6  VALUE 19.
       10 SOCK-DATA                PIC X(26).
       10 SOCK-SIN REDEFINES SOCK-DATA.
          15 SOCK-SIN-PORT         PIC 9(4) BINARY.
          15 SOCK-SIN-ADDR         PIC 9(8) BINARY.
          15 FILLER                PIC X(8).
          15 FILLER                PIC X(12).
       10 SOCK-SIN6 REDEFINES SOCK-DATA.
          15 SOCK-SIN6-PORT        PIC 9(4) BINARY.
          15 SOCK-SIN6-FLOWINFO    PIC 9(8) BINARY.
          15 SOCK-SIN6-ADDR.
             20 FILLER             PIC 9(16) BINARY.
```

Figure 179. EZACIC6S IPv6 iterative server sample (Part 9 of 24)

```
                20 FILLER               PIC 9(16) BINARY.
             15 SOCK-SIN6-SCOPEID       PIC 9(8) BINARY.
       05 FILLER                        PIC X(68).
       05 CLIENT-IN-DATA-LENGTH         PIC 9(4) COMP.
       05 CLIENT-IN-DATA-2              PIC X(999).
|   01  SOCK-TO-RECV-FWD.
|      02 FILLER                PIC 9(4) BINARY.
|      02 SOCK-TO-RECV          PIC 9(4) BINARY.
    01  TIMEVAL.
       02 TVSEC                 PIC 9(8)  COMP VALUE 180.
       02 TVUSEC                PIC 9(8)  COMP VALUE 0.
    01  ZERO-PARM               PIC X(16) VALUE LOW-VALUES.
    01  ZERO-FLD REDEFINES ZERO-PARM.
       02  ZERO-8               PIC X(8).
       02  ZERO-DUM             PIC X(2).
       02  ZERO-HWRD            PIC 9(4)  COMP.
       02  ZERO-FWRD            PIC 9(8)  COMP.
   * ********************************************** *
   *  INPUT FORMAT FOR UPDATING THE SAMPLE DB2 TABLE *
   * ********************************************** *
    01  INPUT-DEPT.
       05  IN-ACT               PIC X(3).
       05  IN-DEPTNO            PIC X(3).
       05  IN-DEPTN             PIC X(36).
       05  IN-MGRNO             PIC X(6).
       05  IN-ADMRDEPT          PIC X(3).
   *---------------------------------------------------------------*
   *    SQL STATEMENTS:  SQL COMMUNICATION AREA                    *
   *---------------------------------------------------------------*
   ***  EXEC SQL INCLUDE SQLCA    END-EXEC.
   *---------------------------------------------------------------*
   *    SQL STATEMENTS:  DEPARTMENT TABLE CREATE STATEMENT FOR DB2 *
   *                                                               *
   *             CREATE TABLE TCPCICS.DEPT                         *
   *                    (DEPTNO       CHAR(03),                    *
   *                     DEPTNAME     CHAR(36),                    *
   *                     MGRNO        CHAR(06),                    *
   *                     ADMRDEPT     CHAR(03));                   *
   *                                                               *
   *---------------------------------------------------------------*
   *    DCLGEN GENERATED FROM DB2 FOR THE DEPARTMENT TABLE.        *
   *---------------------------------------------------------------*
   * ***EXEC SQL INCLUDE DCLDEPT  END-EXEC.
   ****************************************************************
   * DCLGEN TABLE(TCPCICS.DEPT)                                    *
   *        LIBRARY(SYSADM.CICS.SPUFI(DCLDEPT))                    *
   *        LANGUAGE(COBOL)                                        *
   *        QUOTE                                                  *
   * ... IS THE DCLGEN COMMAND THAT MADE THE FOLLOWING STATEMENTS   *
   ****************************************************************
   ***  EXEC SQL DECLARE TCPCICS.DEPT TABLE
   ***  ( DEPTNO                        CHAR(3),
   ***    DEPTNAME                      CHAR(36),
```

Figure 179. EZACIC6S IPv6 iterative server sample (Part 10 of 24)

```
***    MGRNO                         CHAR(6),
***    ADMRDEPT                      CHAR(3)
***  ) END-EXEC.
******************************************************************
* COBOL DECLARATION FOR TABLE TCPCICS.DEPT                       *
******************************************************************
 01  DCLDEPT.
     10 DEPTNO               PIC X(3).
     10 DEPTNAME             PIC X(36).
     10 MGRNO                PIC X(6).
     10 ADMRDEPT             PIC X(3).
******************************************************************
* THE NUMBER OF COLUMNS DESCRIBED BY THIS DECLARATION IS 4       *
******************************************************************
 PROCEDURE DIVISION.
***  EXEC SQL WHENEVER SQLERROR    GO TO SQL-ERROR-ROU END-EXEC.
***  EXEC SQL WHENEVER SQLWARNING  GO TO SQL-ERROR-ROU END-EXEC.
     EXEC CICS IGNORE CONDITION TERMERR
                                EOC
                                SIGNAL
     END-EXEC.
     EXEC CICS HANDLE CONDITION ENDDATA   (ENDDATA-SEC)
                                IOERR     (IOERR-SEC)
                                LENGERR   (LENGERR-SEC)
                                NOSPACE   (NOSPACE-ERR-SEC)
                                QIDERR    (QIDERR-SEC)
     END-EXEC.
     MOVE START-MSG                   TO MSG-AREA.
     PERFORM HANDLE-TCPCICS           THRU HANDLE-TCPCICS-EXIT.
*----------------------------------------------------------------*
*                                                                *
*  BEFORE SERVER  STARTS, TRUE MUST BE ACTIVE.  ISSUE 'EXTRACT   *
*  EXIT' COMMAND TO CHECK IF TRUE IS ACTIVE OR NOT               *
*                                                                *
*----------------------------------------------------------------*
     EXEC CICS PUSH HANDLE END-EXEC.
     EXEC CICS HANDLE CONDITION
          INVEXITREQ(TCP-TRUE-REQ)
     END-EXEC.
     EXEC CICS EXTRACT EXIT
          PROGRAM ('EZACIC01')
          GASET   (GWPTR)
          GALENGTH(GWLENG)
     END-EXEC.
     EXEC CICS POP HANDLE END-EXEC.
*----------------------------------------------------------------*
*                                                                *
*  CICS ATTACH FACILITY MUST BE STARTED FOR THE APPROPRIATE DB2 *
*  SUBSYSTEM BEFORE YOU EXECUTE CICS TRANSACTIONS REQUIRING      *
*  ACCESS TO DB2 DATABASES.                                      *
*                                                                *
*----------------------------------------------------------------*
*    EXEC CICS PUSH HANDLE END-EXEC.
*
*    EXEC CICS HANDLE CONDITION
```

Figure 179. EZACIC6S IPv6 iterative server sample (Part 11 of 24)

```
*        INVEXITREQ(DB2-TRUE-REQ)
*    END-EXEC.
*
*    EXEC CICS EXTRACT EXIT
*         PROGRAM   ('DSNCEXT1')
*         ENTRYNAME ('DSNCSQL')
*         GASET     (WSPTR)
*         GALENGTH  (WSLENG)
*    END-EXEC.
*
*    EXEC CICS POP HANDLE END-EXEC.
*
*
*-------------------------------------------------------------*
*                                                             *
*  AT START UP THE SERVER REQUIRES THE PORT NUMBER FOR TCP/IP *
*  IT WILL USE.  THE PORT NUMBER SUPPORTED BY THIS SAMPLE IS  *
*  4 DIGITS IN LENGTH.                                        *
*                                                             *
*  INVOCATION:  <server>,<port number>                        *
*   LISTENER => SRV2,4000  - OR -  SRV2,4    -                *
*   CECI     => CECI START TR(SRV2) FROM(4000)                *
*                                                             *
*  THE LEADING SPACES ARE SIGNIFICANT.                        *
*                                                             *
*-------------------------------------------------------------*
     MOVE EIBTRNID                    TO TRANS.
     EXEC CICS RETRIEVE
          INTO   (TCP-INPUT-DATA)
          LENGTH (LENG)
     END-EXEC.
* *********************************************************** *
* THE PORT CAN SPECIFIED IN THE FROM(????) OPTION OF THE CECI *
* COMMAND OR THE DEFAULT PORT IS USED.                        *
* THE PORT FOR THE LISTENER STARTED SERVER IS THE PORT        *
* SPECIFIED IN THE CLIENT-DATA-FLD OR THE DEFAULT PORT        *
* IS USED.                                                    *
* *********************************************************** *
*       THE DEFAULT PORT MUST BE SET, BY THE PROGRAMMER.      *
* *********************************************************** *
     IF LENG < CECI-LENG
        THEN MOVE TCP-INPUT-DATA      TO PORT
        ELSE
          MOVE CLIENT-DATA-FLD        TO PORT-RECORD
          MOVE '1'                    TO TAKESOCKET-SWITCH
     END-IF.
     INSPECT PORT REPLACING LEADING SPACES BY '0'.
     IF PORT IS NUMERIC
        THEN MOVE PORT                TO BIND-PORT
        ELSE
          IF DEFAULT-SPECIFIED
             THEN  MOVE DEFAULT-PORT  TO PORT
                                         BIND-PORT
             ELSE
               MOVE PORT              TO PORT-ERRNUM
```

Figure 179. EZACIC6S IPv6 iterative server sample (Part 12 of 24)

```
                MOVE PORTNUM-ERR       TO MSG-AREA
                PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT
                GO TO PGM-EXIT
           END-IF
      END-IF.
      IF DOTAKESOCKET
         THEN PERFORM LISTENER-STARTED-TASK THRU
                 LISTENER-STARTED-TASK-EXIT
         ELSE PERFORM INIT-SOCKET           THRU
                 INIT-SOCKET-EXIT
      END-IF.
      PERFORM SCKET-BIND-LSTN          THRU SCKET-BIND-LSTN-EXIT.
      MOVE 2                           TO CLI-SOCKID
                                          CLI-SOCKID-FWD.
      MOVE LISTEN-SUCC                 TO MSG-AREA.
      PERFORM HANDLE-TCPCICS           THRU HANDLE-TCPCICS-EXIT.
      COMPUTE NFDS = NUM-FDS + 1.
      MOVE LOW-VALUES                  TO READMASK.
      MOVE 6                           TO TCPLENG.
      CALL 'EZACIC06' USING CTOB
                            READMASK
                            SOCKET-CONV
                            TCPLENG
                            RETCODE.
      IF RETCODE = -1
         THEN
           MOVE BITMASK-ERR            TO MSG-AREA
           PERFORM HANDLE-TCPCICS      THRU HANDLE-TCPCICS-EXIT
         ELSE
           PERFORM ACCEPT-CLIENT-REQ   THRU
                   ACCEPT-CLIENT-REQ-EXIT
                   UNTIL TASK-TERM
      END-IF.
      PERFORM CLOSE-SOCKET             THRU CLOSE-SOCKET-EXIT.
      MOVE TCP-SERVER-OFF              TO MSG-AREA.
      PERFORM HANDLE-TCPCICS           THRU HANDLE-TCPCICS-EXIT.
 *---------------------------------------------------------------*
 *                                                               *
 *    END OF PROGRAM                                             *
 *                                                               *
 *---------------------------------------------------------------*
  PGM-EXIT.
      EXEC CICS
           RETURN
      END-EXEC.
      GOBACK.
 *---------------------------------------------------------------*
 *                                                               *
 *          TRUE IS NOT ENABLED                                  *
 *                                                               *
 *---------------------------------------------------------------*
  TCP-TRUE-REQ.
      MOVE TCP-EXIT-ERR      TO MSG-AREA.
```

Figure 179. EZACIC6S IPv6 iterative server sample (Part 13 of 24)

```
         PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
         GO TO PGM-EXIT.
     *---------------------------------------------------------------*
     *                                                               *
     *          DB2 CALL ATTACH FACILITY IS NOT ENABLED              *
     *                                                               *
     *---------------------------------------------------------------*
      DB2-TRUE-REQ.
         MOVE DB2-CAF-ERR       TO MSG-AREA.
         PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
         GO TO PGM-EXIT.
     *---------------------------------------------------------------*
     *                                                               *
     *  LISTENER STARTED TASK                                        *
     *                                                               *
     *---------------------------------------------------------------*
      LISTENER-STARTED-TASK.
         MOVE CLIENTID-PARM               TO CID-LSTN-INFO.
         MOVE GIVE-TAKE-SOCKET            TO SOCK-TO-RECV-FWD.
         CALL 'EZASOKET' USING SOKET-TAKESOCKET
                               SOCK-TO-RECV
                               CLIENTID-LSTN
                               ERRNO
                               RETCODE.
         IF RETCODE < 0
            THEN
              MOVE ERRNO                  TO TAKE-ERRNO
              MOVE TAKE-ERR               TO MSG-AREA
              PERFORM HANDLE-TCPCICS      THRU HANDLE-TCPCICS-EXIT
              GO TO PGM-EXIT
            ELSE
              MOVE BUFFER-LENG            TO TCPLENG
              MOVE START-MSG              TO TCP-BUF
              MOVE RETCODE                TO SRV-SOCKID
              CALL 'EZACIC04' USING TCP-BUF TCPLENG
              CALL 'EZASOKET' USING SOKET-WRITE
                                    SRV-SOCKID
                                    TCPLENG
                                    TCP-BUF
                                    ERRNO
                                    RETCODE
              IF RETCODE < 0
                 THEN
                   MOVE ERRNO             TO WRITE-ERRNO
                   MOVE WRITE-ERR         TO MSG-AREA
                   PERFORM HANDLE-TCPCICS THRU
                           HANDLE-TCPCICS-EXIT
                   GO TO PGM-EXIT
                 ELSE
```

Figure 179. EZACIC6S IPv6 iterative server sample (Part 14 of 24)

```
                CALL 'EZASOKET' USING SOKET-CLOSE
                                      SRV-SOCKID
                                      ERRNO
                                      RETCODE
                IF RETCODE < 0
                   THEN
                     MOVE ERRNO        TO CLOSE-ERRNO
                     MOVE CLOSE-ERR    TO MSG-AREA
                     PERFORM HANDLE-TCPCICS  THRU
                             HANDLE-TCPCICS-EXIT
                     GO TO PGM-EXIT
                   ELSE NEXT SENTENCE
                END-IF
            END-IF
      END-IF.
      MOVE LOW-VALUES                  TO TCP-BUF.
  LISTENER-STARTED-TASK-EXIT.
      EXIT.
*----------------------------------------------------------------*
*                                                                *
*  START SERVER  PROGRAM                                         *
*                                                                *
*----------------------------------------------------------------*
  INIT-SOCKET.
      MOVE EIBTASKN                TO SUBTASKNO.
      CALL 'EZASOKET' USING SOKET-INITAPI
                            MAXSOC
                            IDENT
                            INIT-SUBTASKID
                            MAXSNO
                            ERRNO
                            RETCODE.
      IF RETCODE < 0
         THEN
           MOVE ERRNO              TO INIT-ERRNO
           MOVE INITAPI-ERR        TO MSG-AREA
           PERFORM HANDLE-TCPCICS  THRU HANDLE-TCPCICS-EXIT
           GO TO PGM-EXIT
         ELSE
           MOVE INIT-MSG           TO MSG-AREA
           PERFORM HANDLE-TCPCICS  THRU HANDLE-TCPCICS-EXIT
         END-IF.
  INIT-SOCKET-EXIT.
      EXIT.
  SCKET-BIND-LSTN.
      MOVE  -1                    TO SRV-SOCKID-FWD.
*---------------------------------------------------------------*
*                                                               *
*   CREATING A SOCKET TO ALLOCATE                               *
*   AN OPEN SOCKET FOR INCOMING CONNECTIONS                     *
*                                                               *
*---------------------------------------------------------------*
      CALL 'EZASOKET' USING SOKET-SOCKET
                            AF-INET6
                            SOCK-TYPE
```

Figure 179. EZACIC6S IPv6 iterative server sample (Part 15 of 24)

```
                      PROTOCOL
                      ERRNO
                      RETCODE.
      IF RETCODE < 0
         THEN
           MOVE ERRNO             TO SOCKET-ERRNO
           MOVE SOCKET-ERR        TO MSG-AREA
           PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT
           GO TO PGM-EXIT
         ELSE MOVE RETCODE        TO SRV-SOCKID
              MOVE  '1' TO SOCK-CHAR(RETCODE + 1)
      END-IF.
*---------------------------------------------------------------*
*                                                               *
*  BIND THE SOCKET TO THE SERVICE PORT                          *
*  TO ESTABLISH A LOCAL ADDRESS FOR PROCESSING INCOMING         *
*  CONNECTIONS.                                                 *
*                                                               *
*---------------------------------------------------------------*
      MOVE AF-INET6               TO SAIN-FAMILY.
      MOVE ZEROS                  TO SAIN-SIN6-FLOWINFO.
      MOVE IN6ADDR-ANY            TO SAIN-SIN6-ADDR.
      MOVE ZEROS                  TO SAIN-SIN6-SCOPEID.
      MOVE PORT                   TO SAIN-SIN6-PORT.
      CALL 'EZASOKET' USING SOKET-BIND
                      SRV-SOCKID
                      SOCKADDR-IN
                      ERRNO
                      RETCODE.
      IF RETCODE < 0 THEN
         MOVE ERRNO               TO BIND-ERRNO
         MOVE BIND-ERR            TO MSG-AREA
         PERFORM HANDLE-TCPCICS   THRU HANDLE-TCPCICS-EXIT
         GO TO PGM-EXIT.
*---------------------------------------------------------------*
*                                                               *
*  CALL THE LISTEN COMMAND TO ALLOWS SERVERS TO                 *
*  PREPARE A SOCKET FOR INCOMING CONNECTIONS AND SET MAXIMUM    *
*  CONNECTIONS.                                                 *
*                                                               *
*---------------------------------------------------------------*
      CALL 'EZASOKET' USING SOKET-LISTEN
                      SRV-SOCKID
                      BACKLOG
                      ERRNO
                      RETCODE.
      IF RETCODE < 0 THEN
         MOVE ERRNO               TO LISTEN-ERRNO
         MOVE LISTEN-ERR          TO MSG-AREA
         PERFORM HANDLE-TCPCICS   THRU HANDLE-TCPCICS-EXIT
         GO TO PGM-EXIT.
 SCKET-BIND-LSTN-EXIT.
      EXIT.
*---------------------------------------------------------------*
*                                                               *
```

Figure 179. EZACIC6S IPv6 iterative server sample (Part 16 of 24)

```
*  SOCKET HAS BEEN SET UP, THEN CALL 'ACCEPT' TO              *
*  ACCEPT A REQUEST WHEN A CONNECTION ARRIVES.                *
*                                                             *
*  THIS SAMPLE PROGRAM WILL ONLY USE 5 SOCKETS.               *
*                                                             *
*-------------------------------------------------------------*
 ACCEPT-CLIENT-REQ.
     CALL 'EZASOKET' USING SOKET-SELECT
                           NFDS
                           TIMEVAL
                           READMASK
                           DUMYMASK
                           DUMYMASK
                           REPLY-RDMASK
                           DUMYMASK
                           DUMYMASK
                           ERRNO
                           RETCODE.
     IF RETCODE < 0
        THEN
          MOVE ERRNO             TO SELECT-ERRNO
          MOVE SELECT-ERR        TO MSG-AREA
          PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT
          GO TO PGM-EXIT.
     IF RETCODE = 0
        THEN GO TO ACCEPT-CLIENT-REQ-EXIT.
*-------------------------------------------------------------*
*                                                             *
*  ACCEPT REQUEST                                             *
*                                                             *
*-------------------------------------------------------------*
     CALL 'EZASOKET' USING SOKET-ACCEPT
                           SRV-SOCKID
                           SOCKADDR-IN
                           ERRNO
                           RETCODE.
     IF RETCODE < 0 THEN
        MOVE ERRNO               TO ACCEPT-ERRNO
        MOVE ACCEPT-ERR          TO MSG-AREA
        PERFORM HANDLE-TCPCICS   THRU HANDLE-TCPCICS-EXIT
        GO TO PGM-EXIT.
     MOVE RETCODE TO CLI-SOCKID.
     PERFORM GET-NAME-INFO       THRU GET-NAME-INFO-EXIT.
     PERFORM ACCEPT-RECV         THRU ACCEPT-RECV-EXIT
             UNTIL TASK-END OR TASK-TERM.
     MOVE DB2END                 TO MSG-AREA.
     PERFORM HANDLE-TCPCICS      THRU HANDLE-TCPCICS-EXIT.
     CALL 'EZASOKET' USING SOKET-CLOSE
                           CLI-SOCKID
                           ERRNO
                           RETCODE.
     IF RETCODE < 0 THEN
        MOVE ERRNO               TO CLOSE-ERRNO
        MOVE CLOSE-ERR           TO MSG-AREA
        PERFORM HANDLE-TCPCICS   THRU HANDLE-TCPCICS-EXIT.
```

Figure 179. EZACIC6S IPv6 iterative server sample (Part 17 of 24)

```
     IF NOT TASK-TERM
        MOVE '0'                 TO TASK-FLAG.
 ACCEPT-CLIENT-REQ-EXIT.
     EXIT.
*---------------------------------------------------------------*
*                                                               *
*  DETERMINE THE CONNECTED HOST NAME BY ISSUING THE             *
*  GETNAMEINFO COMMAND.                                         *
*                                                               *
*---------------------------------------------------------------*
 GET-NAME-INFO.
     MOVE SAIN-SIN6-ADDR TO NUMERIC-ADDR.
     MOVE 45 TO PRESENTABLE-ADDR-LEN.
     MOVE SPACES TO PRESENTABLE-ADDR.
     CALL 'EZASOKET' USING SOKET-NTOP AF-INET6
        NUMERIC-ADDR
        PRESENTABLE-ADDR PRESENTABLE-ADDR-LEN
        ERRNO RETCODE.
     IF RETCODE < 0 THEN
        MOVE ERRNO               TO NTOP-ERRNO
        MOVE NTOP-ERR            TO MSG-AREA
        PERFORM HANDLE-TCPCICS   THRU HANDLE-TCPCICS-EXIT.
     MOVE PRESENTABLE-ADDR       TO NTOP-PRESENTABLE-ADDR.
     MOVE NTOP-OK                TO MSG-AREA.
     PERFORM HANDLE-TCPCICS   THRU HANDLE-TCPCICS-EXIT.
     CALL 'EZASOKET' USING SOKET-GETPEERNAME
                           CLI-SOCKID
                           SOCKADDR-PEER
                           ERRNO
                           RETCODE.
     IF RETCODE < 0 THEN
        MOVE ERRNO               TO GPN-ERRNO
        MOVE GPN-ERR             TO MSG-AREA
        PERFORM HANDLE-TCPCICS   THRU HANDLE-TCPCICS-EXIT
        GO TO PGM-EXIT.
     MOVE 28 TO NAME-LEN.
     MOVE 255 TO HOST-NAME-LEN.
     MOVE 32 TO SERVICE-NAME-LEN.
     MOVE ZEROS TO NAME-INFO-FLAGS.
     CALL 'EZASOKET' USING SOKET-GETNAMEINFO
                           SOCKADDR-PEER
                           NAME-LEN
                           HOST-NAME
                           HOST-NAME-LEN
                           SERVICE-NAME
                           SERVICE-NAME-LEN
                           NAME-INFO-FLAGS
                           ERRNO
                           RETCODE.
     IF RETCODE < 0 THEN
        MOVE ERRNO               TO GNI-ERRNO
        MOVE GNI-ERR             TO MSG-AREA
        PERFORM HANDLE-TCPCICS   THRU HANDLE-TCPCICS-EXIT.
     MOVE 0 TO HOST-NAME-CHAR-COUNT.
     INSPECT HOST-NAME TALLYING HOST-NAME-CHAR-COUNT
```

Figure 179. EZACIC6S IPv6 iterative server sample (Part 18 of 24)

```
        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 GNI-HOST-NAME.
     MOVE GNI-HOST-NAME-OK       TO MSG-AREA.
     PERFORM HANDLE-TCPCICS      THRU HANDLE-TCPCICS-EXIT.
     MOVE 0 TO SERVICE-NAME-CHAR-COUNT.
     INSPECT SERVICE-NAME TALLYING SERVICE-NAME-CHAR-COUNT
        FOR CHARACTERS BEFORE X'00'.
     UNSTRING SERVICE-NAME DELIMITED BY X'00'
        INTO SERVICE-NAME-UNSTRUNG
        COUNT IN SERVICE-NAME-CHAR-COUNT.
     STRING SERVICE-NAME-UNSTRUNG DELIMITED BY ' '
        INTO GNI-SERVICE-NAME.
     MOVE GNI-SERVICE-NAME-OK    TO MSG-AREA.
     PERFORM HANDLE-TCPCICS      THRU HANDLE-TCPCICS-EXIT.
     DISPLAY 'HOST NAME = ' HOST-NAME.
     DISPLAY 'SERVICE = ' SERVICE-NAME.
 GET-NAME-INFO-EXIT.
     EXIT.
*---------------------------------------------------------------*
*                                                               *
*  RECEIVING DATA THROUGH A SOCKET BY ISSUING 'RECVFROM'        *
*  COMMAND.                                                     *
*                                                               *
*---------------------------------------------------------------*
 ACCEPT-RECV.
     MOVE 'T'                                TO TCP-INDICATOR.
     MOVE BUFFER-LENG                        TO TCPLENG.
     MOVE LOW-VALUES                         TO TCP-BUF.
     CALL 'EZASOKET' USING SOKET-RECVFROM
                           CLI-SOCKID
                           TCP-FLAG
                           TCPLENG
                           TCP-BUF
                           SOCKADDR-IN
                           ERRNO
                           RETCODE.
     IF RETCODE EQUAL 0 AND TCPLENG EQUAL 0
        THEN NEXT SENTENCE
        ELSE
          IF RETCODE < 0
             THEN
               MOVE ERRNO                    TO RECVFROM-ERRNO
               MOVE RECVFROM-ERR             TO MSG-AREA
               PERFORM HANDLE-TCPCICS        THRU
                       HANDLE-TCPCICS-EXIT
               MOVE '1'                      TO TASK-FLAG
             ELSE
               CALL 'EZACIC05' USING TCP-BUF TCPLENG
               IF TCP-BUF-H = LOW-VALUES OR SPACES
```

Figure 179. EZACIC6S IPv6 iterative server sample (Part 19 of 24)

```
                  THEN
                    MOVE NULL-DATA          TO MSG-AREA
                    PERFORM HANDLE-TCPCICS   THRU
                            HANDLE-TCPCICS-EXIT
                  ELSE
                    IF TCP-BUF-H =  'END'
                       THEN MOVE '1'         TO TASK-FLAG
                       ELSE IF TCP-BUF-H = 'TRM'
                               THEN MOVE '2' TO TASK-FLAG
                               ELSE PERFORM TALK-CLIENT THRU
                                            TALK-CLIENT-EXIT
                            END-IF
                    END-IF
               END-IF
          END-IF
     END-IF.
 ACCEPT-RECV-EXIT.
     EXIT.
**********************************************************
**    PROCESSES TALKING TO CLIENT THAT WILL UPDATE DB2  **
**    TABLES.                                           **
**********************************************************
**    DATA PROCESS:                                     **
**                                                      **
**    INSERT REC -  INS,X81,TEST DEPT,A0213B,Y94        **
**    UPDATE REC -  UPD,X81,,A1234C,                    **
**    DELETE REC -  DEL,X81,,,                          **
**    END CLIENT -  END,{end client connection     }    **
**    END SERVER -  TRM,{terminate server          }    **
**                                                      **
**********************************************************
 TALK-CLIENT.
     UNSTRING TCP-BUF DELIMITED BY DEL-ID OR ALL '*'
         INTO IN-ACT
              IN-DEPTNO
              IN-DEPTN
              IN-MGRNO
              IN-ADMRDEPT.
     IF IN-ACT EQUAL 'END'
        THEN
          MOVE '1'                             TO TASK-FLAG
        ELSE
          IF IN-ACT EQUAL 'U' OR EQUAL 'UPD'
             THEN
***            EXEC SQL UPDATE TCPCICS.DEPT
***              SET    MGRNO  = :IN-MGRNO
***              WHERE  DEPTNO = :IN-DEPTNO
***            END-EXEC
               MOVE 'UPDATE'                   TO DB2-ACT
               MOVE 'UPDATED:  '               TO DB2M-VAR
             ELSE
               IF IN-ACT EQUAL 'I' OR EQUAL 'INS'
                  THEN
***                 EXEC SQL INSERT
***                   INTO TCPCICS.DEPT (DEPTNO,     DEPTNAME,
```

Figure 179. EZACIC6S IPv6 iterative server sample (Part 20 of 24)

```
***                                      MGRNO,      ADMRDEPT)
***                 VALUES           (:IN-DEPTNO, :IN-DEPTN,
***                                   :IN-MGRNO,  :IN-ADMRDEPT)
***               END-EXEC
                  MOVE 'INSERT'               TO DB2-ACT
                  MOVE 'INSERTED: '           TO DB2M-VAR
                ELSE
                  IF IN-ACT EQUAL 'D' OR EQUAL 'DEL'
                     THEN
***                    EXEC SQL DELETE
***                      FROM  TCPCICS.DEPT
***                      WHERE DEPTNO = :IN-DEPTNO
***                    END-EXEC
                       MOVE 'DELETE'          TO DB2-ACT
                       MOVE 'DELETED: '       TO DB2M-VAR
                     ELSE
                       MOVE KEYWORD-ERR       TO MSG-AREA
                       PERFORM HANDLE-TCPCICS THRU
                               HANDLE-TCPCICS-EXIT
                  END-IF
             END-IF
        END-IF
     END-IF.
     IF DADELETE OR DAINSERT OR DAUPDATE
        THEN
*         MOVE SQLERRD(3)                     TO DB2CODE
          MOVE DB2MSG                         TO MSG-AREA
          MOVE LENGTH OF TCPCICS-MSG-AREA     TO LENG
          EXEC CICS SYNCPOINT END-EXEC
          EXEC CICS WRITEQ TD
               QUEUE    ('CSMT')
               FROM     (TCPCICS-MSG-AREA)
               LENGTH   (LENG)
               NOHANDLE
          END-EXEC
**********************************************************
**          WRITE THE DB2 MESSAGE TO CLIENT.            **
**********************************************************
          MOVE TCPCICS-MSG-2                  TO TCP-BUF
          CALL 'EZACIC04' USING TCP-BUF TCPLENG
          CALL 'EZASOKET' USING SOKET-WRITE
                                CLI-SOCKID
                                TCPLENG
                                TCP-BUF
                                ERRNO
                                RETCODE
          MOVE LOW-VALUES                     TO TCP-BUF
                                                 TCP-INDICATOR
                                                 DB2-ACT
          IF RETCODE < 0
             THEN
               MOVE ERRNO                     TO WRITE-ERRNO
               MOVE WRITE-ERR                 TO MSG-AREA
```

Figure 179. EZACIC6S IPv6 iterative server sample (Part 21 of 24)

```
              PERFORM HANDLE-TCPCICS           THRU
                      HANDLE-TCPCICS-EXIT
              MOVE '1'                         TO TASK-FLAG
         END-IF
    END-IF.
TALK-CLIENT-EXIT.
    EXIT.
*---------------------------------------------------------------*
*                                                               *
*   CLOSE ORIGINAL SOCKET DESCRIPTOR                            *
*                                                               *
*---------------------------------------------------------------*
CLOSE-SOCKET.
    CALL 'EZASOKET' USING SOKET-CLOSE
                          SRV-SOCKID
                          ERRNO
                          RETCODE.
    IF RETCODE < 0 THEN
       MOVE ERRNO             TO CLOSE-ERRNO
       MOVE CLOSE-ERR         TO MSG-AREA
       PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
CLOSE-SOCKET-EXIT.
    EXIT.
*---------------------------------------------------------------*
*                                                               *
*  SEND TCP/IP ERROR MESSAGE                                    *
*                                                               *
*---------------------------------------------------------------*
HANDLE-TCPCICS.
    MOVE LENGTH OF TCPCICS-MSG-AREA TO LENG.
    EXEC CICS ASKTIME
         ABSTIME (TSTAMP)
         NOHANDLE
    END-EXEC.
    EXEC CICS FORMATTIME
         ABSTIME (TSTAMP)
         MMDDYY  (MSGDATE)
         TIME    (MSGTIME)
         DATESEP ('/')
         TIMESEP (':')
         NOHANDLE
    END-EXEC.
    EXEC CICS WRITEQ TD
         QUEUE  ('CSMT')
         FROM   (TCPCICS-MSG-AREA)
         RESP   (RESPONSE)
         LENGTH (LENG)
    END-EXEC.
    IF RESPONSE = DFHRESP(NORMAL)
       THEN NEXT SENTENCE
       ELSE
         IF RESPONSE = DFHRESP(INVREQ)
            THEN MOVE TS-INVREQ-ERR          TO MSG-AREA
            ELSE
              IF RESPONSE = DFHRESP(NOTAUTH)
```

Figure 179. EZACIC6S IPv6 iterative server sample (Part 22 of 24)

```
                THEN MOVE TS-NOTAUTH-ERR    TO MSG-AREA
                ELSE
                  IF RESPONSE = DFHRESP(IOERR)
                     THEN MOVE TS-IOERR-ERR TO MSG-AREA
                     ELSE MOVE WRITETS-ERR  TO MSG-AREA
                  END-IF
              END-IF
          END-IF
      END-IF.
      IF TCP-INDICATOR = 'T' THEN
         MOVE BUFFER-LENG            TO TCPLENG
         MOVE LOW-VALUES             TO TCP-BUF
         MOVE TCPCICS-MSG-2          TO TCP-BUF
         CALL 'EZACIC04' USING TCP-BUF TCPLENG
         MOVE ' '                    TO TCP-INDICATOR
         CALL 'EZASOKET' USING SOKET-WRITE
                               CLI-SOCKID
                               TCPLENG
                               TCP-BUF
                               ERRNO
                               RETCODE
         IF RETCODE < 0
            THEN
              MOVE ERRNO             TO WRITE-ERRNO
              MOVE WRITE-ERR         TO MSG-AREA
              EXEC CICS WRITEQ TD
                   QUEUE  ('CSMT')
                   FROM   (TCPCICS-MSG-AREA)
                   LENGTH (LENG)
                   NOHANDLE
              END-EXEC
              IF TASK-TERM OR TASK-END
                 THEN NEXT SENTENCE
                 ELSE MOVE '1'       TO  TASK-FLAG
              END-IF
         END-IF.
      MOVE SPACES                    TO MSG-AREA.
  HANDLE-TCPCICS-EXIT.
      EXIT.
 *---------------------------------------------------------------*
 *                                                               *
 *  SEND DB2    ERROR MESSAGE                                    *
 *                                                               *
 *---------------------------------------------------------------*
  SQL-ERROR-ROU.
 *    MOVE SQLCODE         TO SQL-ERR-CODE.
      MOVE SPACES          TO MSG-AREA.
 *    MOVE SQL-ERROR       TO MSG-AREA.
      EXEC CICS WRITEQ TD
           QUEUE  ('CSMT')
           FROM   (TCPCICS-MSG-AREA)
           RESP   (RESPONSE)
           LENGTH (LENG)
```

Figure 179. EZACIC6S IPv6 iterative server sample (Part 23 of 24)

```
           END-EXEC.
           MOVE LOW-VALUES      TO TCP-BUF.
           MOVE TCPCICS-MSG-2   TO TCP-BUF.
           CALL 'EZACIC04' USING TCP-BUF TCPLENG.
           CALL 'EZASOKET' USING SOKET-WRITE
                                 CLI-SOCKID
                                 TCPLENG
                                 TCP-BUF
                                 ERRNO
                                 RETCODE.
           IF RETCODE < 0 THEN
              MOVE ERRNO        TO WRITE-ERRNO
              MOVE WRITE-ERR    TO MSG-AREA
              PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
           GO TO PGM-EXIT.
       SQL-ERROR-ROU-EXIT.
           EXIT.
      *----------------------------------------------------------------*
      *                                                                *
      *  OTHER ERRORS (HANDLE CONDITION)                               *
      *                                                                *
      *----------------------------------------------------------------*
       INVREQ-ERR-SEC.
           MOVE TCP-EXIT-ERR      TO MSG-AREA.
           PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
           GO TO PGM-EXIT.
       IOERR-SEC.
           MOVE IOERR-ERR         TO MSG-AREA.
           PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
           GO TO PGM-EXIT.
       LENGERR-SEC.
           MOVE LENGERR-ERR       TO MSG-AREA.
           PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
           GO TO PGM-EXIT.
       NOSPACE-ERR-SEC.
           MOVE NOSPACE-ERR       TO MSG-AREA.
           PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
           GO TO PGM-EXIT.
       QIDERR-SEC.
           MOVE QIDERR-ERR        TO MSG-AREA.
           PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
           GO TO PGM-EXIT.
       ITEMERR-SEC.
           MOVE ITEMERR-ERR       TO MSG-AREA.
           PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
           GO TO PGM-EXIT.
       ENDDATA-SEC.
           MOVE ENDDATA-ERR       TO MSG-AREA.
           PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
           GO TO PGM-EXIT.
```

Figure 179. EZACIC6S IPv6 iterative server sample (Part 24 of 24)

EZACICAC

The following Assembler socket program is in the SEZAINST data set.

```
***********************************************************************
*                                                                     *
* Module Name:  EZACICAC - This is a very simple child server         *
*                                                                     *
* Copyright:    Licensed Materials - Property of IBM                  *
*                                                                     *
*               "Restricted Materials of IBM"                         *
*                                                                     *
*               5694-A01                                              *
*                                                                     *
*               Copyright IBM Corp. 2003, 2007                        *
*                                                                     *
*               US Government Users Restricted Rights -               *
*               Use, duplication or disclosure restricted by          *
*               GSA ADP Schedule Contract with IBM Corp.              *
*                                                                     *
* Status:       CSV1R9                                                *
*                                                                     *
*                                                                     *
*   LANGUAGE:  ASSEMBLER                                              *
*                                                                     *
*   ATTRIBUTES: NON-REUSEABLE                                         *
*                                                                     *
*   REGISTER USAGE:                                                   *
*        R1  =                                                        *
*        R2  =                                                        *
*        R3  =                                                        *
*        R4  =                                                        *
*        R5  =                                                        *
*        R6  =                                                        *
*        R7  =                                                        *
*        R8  =                                                        *
*        R9  =                                                        *
*        R10 =                                                        *
*        R11 =                                                        *
*        R12 =                                                        *
*        R13 =                                                        *
*        R14 =                                                        *
*        R15 =                                                        *
*                                                                     *
*   INPUT:                                                            *
*                                                                     *
*   OUTPUT:                                                           *
*                                                                     *
* $MOD(EZACICAC),COMP(CICS),PROD(TCPIP):                              *
*                                                                     *
*                                                                     *
***********************************************************************
DFHEISTG DSECT
```

Figure 180. EZACICAC assembler child server sample (Part 1 of 10)

```
SOCSTG   DS    0F                  PROGRAM STORAGE
*
* Storage to format messages
*
TDMSG    DS    0F                  WRITEQ TD Message area
TDDATE   DS    CL8                 MM/DD/YY
TDFILL1  DS    CL2
TDTIME   DS    CL8                 HH:MM:SS
TDFILL2  DS    CL2
TDTEXT   DS    CL40                TDTEXT
*
         ORG   TDTEXT
TDTEXT0  DS    0CL40
TDCMD    DS    CL16                COMMAND ISSUED
TDRESULT DS    CL24                SUCCESSFUL/UNSUCCESSFUL
TDMSGE   EQU   *                   End of message
TDMSGL   EQU   TDMSGE-TDMSG        Length of TD message text
*
* Message to display the clients host name
*
         ORG   TDTEXT
TDHOSTMSG DS    0CL40
TDHOSTLIT DS   CL9
TDHOST   DS    CL31
*
* Message to display the clients service name
*
         ORG   TDTEXT
TDSERVMSG DS    0CL40
TDSERVLIT DS   CL8
TDSERV   DS    CL32
*
TDLEN    DS    H                   Length of TD message text
*
* Working storage fields
*
CLENG    DS    H                   Length of data to RETRIEVE
UTIME    DS    PL8                 ABSTIME data area
DWORK    DS    D                   Double work work area
UNPKWRK  DS    CL15                For packing/unpacking
PARMLIST DS    20F                 Parm list for EZASOKET calls
*
SOCDESC  DS    H                   Socket Descriptor
*
ERRNO    DS    F                   ERRNO
RETCODE  DS    F                   Return code
*
* Storage to map the clientid structure.
*
CLIENTID DS 0CL40
GIVE_DOM DS F                      Domain of socket given/taken
AS_NAME  DS CL8                    Address space name
TASK_ID  DS CL8                    Task identifier
         DS CL20                   Reserved
*
```

Figure 180. EZACICAC assembler child server sample (Part 2 of 10)

```
* Storage to address the Transaction Input Message from the Listener.
*
SOKTIM   DS    0CL1153
SOKDESC  DS    F                  Socket descriptor given
SOKLASID DS    CL8                Listener address space name
SOKLTID  DS    CL8                Listener task identifier
SOKDATA1 DS    CL35               Client input data
SOKTSI   DS    CL1                Threadsafe inidicator
SOKADDR  DS    0F                 Clients socket address
SOKFAM   DS    H                  Address family
SOK_DATA DS    0C                 Protocol specific area
SOK#LEN  EQU   *-SOKADDR
         ORG   SOK_DATA           Start of AF_INET unique area
SOK_SIN  DS    0C
SOK_SIN_PORT DS H                 Clients port number
SOK_SIN_CIPAD DS F                Clients INET address (netid)
         DS    CL8                Reserved area not used
         DS    20F
SOK_SIN#LEN EQU *-SOK_SIN         Length of AF_INET area
         ORG   SOK_DATA           Start of AF_INET6 unique area
SOK_SIN6 DS    0C
SOK_SIN6_PORT DS H                Clients port number
SOK_SIN6_FLOWINFO DS CL4          Flow information
SOK_SIN6_CIPAD DS CL16            Clients INET address (netid)
SOK_SIN6_SCOPE_ID DS CL4          Scope Id
SOK_SIN6#LEN EQU *-SOK_SIN6       Length of AF_INET6 area
         ORG
         DS    CL68               Reserved
SOKDATAL DS    H                  Length of data area 2
SOKDATA2 DS    CL999              Data area 2
*
* Program storage marker
*
SOCSTGE  EQU   *                  End of Program Storage
SOCSTGL  EQU   SOCSTGE-SOCSTG     Length of Program Storage
*
* Beginning of program
*
EZACICAC CSECT
EZACICAC AMODE ANY                Addressing mode ...
EZACICAC RMODE ANY                Residency mode ...
SOC0000  DS    0H
         B     SOC00100           Branch to startup address
         DC    CL17'EZACICAC-EYECATCH'
SOC00100 DS    0H                 Beginning of program
         LA    R10,SOCSTG         Address Pgm Dynamic Stg
         USING SOCSTG,R10         Tell Assembler about storage
         MVC   TDTEXT(40),STARTED_MSG Move STARTED message to TD area
         BAL   R7,WRITEQ          Write to TD Queue
         MVC   CLENG,=H'72'       Length for standard listener
         MVC   CLENG,=H'1153'     Length for enhanced listener
*
* Retrieve the Task Input Message(TIM) from the Listener
*
         EXEC CICS RETRIEVE INTO(SOKTIM) LENGTH(CLENG)
```

Figure 180. EZACICAC assembler child server sample (Part 3 of 10)

```
*
* Issue the 'TAKESOCKET' call to acquire the socket which was
* given by the listener program.
*
         XC    CLIENTID,CLIENTID  Clear the clientid structure
         MVC   GIVE_DOM+2,SOKFAM  Based on the AF in the TIM
         MVC   AS_NAME,SOKLASID   Set the address space name
         MVC   TASK_ID,SOKLTID      and the subtask identifier
         MVC   SOCDESC,SOKDESC+2      and the socket descriptor.
*
         CALL  EZASOKET,(SOCTSOCK,SOCDESC,CLIENTID,                    X
               ERRNO,RETCODE),VL,MF=(E,PARMLIST)
         L     R5,ERRNO           Capture the ERRNO and
         L     R6,RETCODE            the return code.
         C     R6,=F'0'           Is the call successful?
         BL    SOCERR             No!  Go display error and terminate
         MVC   SOCDESC,RETCODE+2  Yes, format the return code and
         MVC   TDCMD,SOCTSOCK        the API function performed.
         MVC   TDRESULT(24),SUCC  Move SUCCESSFUL msg to TD area
         MVC   TDTEXT(40),TDTEXT0 Move message to TD area
         BAL   R7,WRITEQ          Write to TD Queue
*
         XC    TCP_BUF,TCP_BUF    Clear the buffer storage
         MVC   TCP_BUF(L'TASK_START),TASK_START Set the message
         L     R8,=F'50'          Set the
         ST    R8,TCPLENG           message length.
*
* Remove the following call to EZACIC04 if using an EBCDIC client.
*
|        CALL  EZACIC04,(TCP_BUF,TCPLENG),VL,MF=(E,PARMLIST)
*
* Notify client the the child subtask has started.
*
         CALL  EZASOKET,(SOCWRITE,SOCDESC,TCPLENG,TCP_BUF,             X
               ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
         L     R5,ERRNO           Capture the ERRNO and
         L     R6,RETCODE            the return code.
         C     R6,=F'0'           Is the call successful?
         BL    SOCERR             No!  Go display error and terminate
         MVC   TDCMD,SOCWRITE        the API function performed.
         MVC   TDRESULT(24),SUCC  Move SUCCESSFUL msg to TD area
         MVC   TDTEXT(40),TDTEXT0 Move message to TD area
         BAL   R7,WRITEQ          Write to TD Queue
*
* Get our peers' socket address
*
         CALL  EZASOKET,(SOCGPNA,SOCDESC,PEERADDR,                     X
               ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
         L     R5,ERRNO           Capture the ERRNO and
         L     R6,RETCODE            the return code.
         C     R6,=F'0'           Is the call successful?
```

Figure 180. EZACICAC assembler child server sample (Part 4 of 10)

```
         BL    SOCERR             No!  Go display error and terminate
         MVC   TDCMD,SOCGPNA         the API function performed.
         MVC   TDRESULT(24),SUCC  Move SUCCESSFUL msg to TD area
         MVC   TDTEXT(40),TDTEXT0 Move message to TD area
         BAL   R7,WRITEQ          Write to TD Queue
*
* Get our client's host name and service name
*
         L     R8,=F'16'          Set the sockaddr length to IPv4
         CLC   SOKFAM,=AL2(AF_INET) Is the client AF_INET ?
         BE    SET_SOCKADDR_LEN   Yes.  Go store the length.
         L     R8,=F'28'          Set the sockaddr length to IPv6
SET_SOCKADDR_LEN DS 0H
         ST    R8,PEERADDR_LEN    Save the value of the sockaddr length
         L     R8,=F'0'           Clear the
         ST    R8,GNI_FLAGS          flags
         XC    PEER_HOSTNAME,PEER_HOSTNAME Clear the host name storage
         L     R8,=F'255'         Set the length of
         ST    R8,PEER_HOSTNAMELEN   the host name storage
         XC    PEER_SERVICENAME,PEER_SERVICENAME Clear the service     X
                                                 name storage
         L     R8,=F'32'          Set the length of
         ST    R8,PEER_SERVICENAMELEN the service name storage
*
         CALL  EZASOKET,(SOCGNI,PEERADDR,PEERADDR_LEN,                 X
               PEER_HOSTNAME,PEER_HOSTNAMELEN,                         X
               PEER_SERVICENAME,PEER_SERVICENAMELEN,                   X
               GNI_FLAGS,                                              X
               ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
         L     R5,ERRNO           Capture the ERRNO and
         L     R6,RETCODE            the return code.
         C     R6,=F'0'           Is the call successful?
         BL    SOCERR             No!  Go display error and terminate
         MVC   TDCMD,SOCGNI          the API function performed.
         MVC   TDRESULT(24),SUCC  Move SUCCESSFUL msg to TD area
         MVC   TDTEXT(40),TDTEXT0 Move message to TD area
         BAL   R7,WRITEQ          Write to TD Queue
*
* Display the host name
*
         MVC   TDHOSTLIT,=C'HOSTNAME='
         MVC   TDHOST(L'TDHOST),PEER_HOSTNAME
         MVC   TDTEXT(40),TDHOSTMSG Move message to TD area
         BAL   R7,WRITEQ          Write to TD Queue
*
* Display the service name
*
         MVC   TDHOSTLIT,=C'SERVICE='
         MVC   TDSERV(L'TDSERV),PEER_SERVICENAME
         MVC   TDTEXT(40),TDSERVMSG Move message to TD area
         BAL   R7,WRITEQ          Write to TD Queue
*
* Receive data from the client
*
```

Figure 180. EZACICAC assembler child server sample (Part 5 of 10)

```
AGAIN1   DS    0H
*
         XC    TCP_BUF,TCP_BUF    Clear the buffer storage
*
         CALL  EZASOKET,(SOCRECV,SOCDESC,RECV_FLAG,TCPLENG,TCP_BUF,    X
               ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
         L     R5,ERRNO           Capture the ERRNO and
         L     R6,RETCODE            the return code.
         C     R6,=F'0'           Is the call successful?
         BL    SOCERR             No!  Go display error and terminate
         MVC   TDCMD,SOCRECV         the API function performed.
         MVC   TDRESULT(24),SUCC  Move SUCCESSFUL msg to TD area
         MVC   TDTEXT(40),TDTEXT0 Move message to TD area
         BAL   R7,WRITEQ          Write to TD Queue
*
* Remove the following call to EZACIC05 if using an EBCDIC client.
*
         CALL  EZACIC05,(TCP_BUF,TCPLENG),VL,MF=(E,PARMLIST)
*
* Determine whether the client is finished sending data
*
         CLC   TCP_BUF_H,=C'END'
         BE    SIGNAL_CLOSING
         CLC   TCP_BUF_H,=C'end'
         BE    SIGNAL_CLOSING
*
* Remove the following call to EZACIC04 if using an EBCDIC client.
*
         CALL  EZACIC04,(TCP_BUF,TCPLENG),VL,MF=(E,PARMLIST)
*
* Echo the data received back to the client
*
         CALL  EZASOKET,(SOCWRITE,SOCDESC,TCPLENG,TCP_BUF,             X
               ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
         L     R5,ERRNO           Capture the ERRNO and
         L     R6,RETCODE            the return code.
         C     R6,=F'0'           Is the call successful?
         BL    SOCERR             No!  Go display error and terminate
         MVC   TDCMD,SOCWRITE        the API function performed.
         MVC   TDRESULT(24),SUCC  Move SUCCESSFUL msg to TD area
         MVC   TDTEXT(40),TDTEXT0 Move message to TD area
         BAL   R7,WRITEQ          Write to TD Queue
*
* Go receive another message
*
         B     AGAIN1
*
```

Figure 180. EZACICAC assembler child server sample (Part 6 of 10)

```
* Tell client the connection will close.
*
SIGNAL_CLOSING DS 0H
         XC    TCP_BUF,TCP_BUF    Clear the buffer storage
         MVC   TCP_BUF(L'WRKEND),WRKEND Set the message
         L     R8,=F'50'          Set the
         ST    R8,TCPLENG           message length.
*
* Remove the following call to EZACIC04 if using an EBCDIC client.
*
         CALL  EZACIC04,(TCP_BUF,TCPLENG),VL,MF=(E,PARMLIST)
*
* Notify the client that the connection will end.
*
         CALL  EZASOKET,(SOCWRITE,SOCDESC,TCPLENG,TCP_BUF,             X
               ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
         L     R5,ERRNO           Capture the ERRNO and
         L     R6,RETCODE            the return code.
         C     R6,=F'0'           Is the call successful?
         BL    SOCERR             No!  Go display error and terminate
         MVC   TDCMD,SOCWRITE        the API function performed.
         MVC   TDRESULT(24),SUCC  Move SUCCESSFUL msg to TD area
         MVC   TDTEXT(40),TDTEXT0 Move message to TD area
         BAL   R7,WRITEQ          Write to TD Queue
*
* Close the socket
*
         CALL  EZASOKET,(SOCCLOSE,SOCDESC,                             X
               ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
         L     R5,ERRNO           Capture the ERRNO and
         L     R6,RETCODE            the return code.
         C     R6,=F'0'           Is the call successful?
         BL    SOCERR             No!  Go display error and terminate
         MVC   TDCMD,SOCCLOSE    Yes, format the API function performed
         MVC   TDRESULT(24),SUCC  Move SUCCESSFUL msg to TD area
         MVC   TDTEXT(40),TDTEXT0 Move message to TD area
         BAL   R7,WRITEQ          Write to TD Queue
         B     SOCRET             Go return to CICS
*
* Error routine for all socket calls
*
SOCERR   DS    0H
         MVI   FORCEMSG,C'Y'      Indicate message should be forced
         MVC   TDTEXT(40),=C'SOCKET ERROR                            '
         BAL   R7,WRITEQ          Write to TD Queue
         L     R6,RETCODE         Pick up the return code value
         L     R5,ERRNO           Pick up the ERRNO value
*
         CVD   R6,DWORK           Format the return code
         UNPK  TDRETC,DWORK+4(4)    for printing to the
         OI    TDRETC+6,X'F0'          TD queue
```

Figure 180. EZACICAC assembler child server sample (Part 7 of 10)

```
*
         CVD   R5,DWORK           Format the ERRNO
         UNPK  TDERRNO,DWORK+4(4)   for printing to the
         OI    TDERRNO+6,X'F0'         TD queue
*
         MVC   TDTEXT(40),TDTEXT5 Move the return code and ERRNO to
         BAL   R7,WRITEQ          the TD queue.  Write to the TD queue
*
         B     SOCRET             Go return to CICS
*
* Subroutine to write messages to the destination "CSMT" for logging
*
WRITEQ   DS    0H
         CLI   SOKTSI,C'1'        Is interface using OTE ?
         BNE   WRITEQ01           No, write message.
         CLI   FORCEMSG,C'Y'      Is this an error message ?
         BNE   WRITEQ02           Yes, bypass writing message.
WRITEQ01 DS    0H
         EXEC CICS ASKTIME ABSTIME(UTIME)
         EXEC CICS FORMATTIME ABSTIME(UTIME)                           X
              DATESEP('/') DDMMYY(TDDATE)                              X
              TIME(TDTIME) TIMESEP
         LA    R6,TDMSGL
         STH   R6,TDLEN
         EXEC CICS WRITEQ TD QUEUE('CSMT')                             X
               FROM(TDMSG)                                             X
               LENGTH(TDLEN)
WRITEQ02 DS    0H
         XC    TDMSG,TDMSG
         BR    R7                 Return to caller
*
* Socket family values
*
AFINET   DC    F'2'               AF_INET
AFINET6  DC    F'19'              AF_INET6
AF_INET  EQU   2
AF_INET6 EQU   19
*
* Socket protocol values
*
SSTREAM  DC    F'1'               socket type stream
SDATAGRM DC    F'2'               socket type datagram
SRAW     DC    F'3'               socket type raw
*
* IP CICS Socket API functions
*
SOCACCT  DC    CL16'ACCEPT          '
SOCBIND  DC    CL16'BIND            '
SOCCLOSE DC    CL16'CLOSE           '
SOCCONNT DC    CL16'CONNECT         '
SOCFCNTL DC    CL16'FCNTL           '
SOCGCLID DC    CL16'GETCLIENTID     '
SOCGTHBA DC    CL16'GETHOSTBYADDR   '
SOCGTHBN DC    CL16'GETHOSTBYNAME   '
SOCGTHID DC    CL16'GETHOSTID       '
```

Figure 180. EZACICAC assembler child server sample (Part 8 of 10)

```
SOCGTHN  DC    CL16'GETHOSTNAME     '
SOCGPNA  DC    CL16'GETPEERNAME     '
SOCGNI   DC    CL16'GETNAMEINFO     '
SOCFAI   DC    CL16'FREEADDRINFO    '
SOCGAI   DC    CL16'GETADDRINFO     '
SOCGTSN  DC    CL16'GETSOCKNAME     '
SOCGSOPT DC    CL16'GETSOCKOPT      '
SOCGSOCK DC    CL16'GIVESOCKET      '
SOCINIT  DC    CL16'INITAPI         '
SOCIOCTL DC    CL16'IOCTL           '
SOCLISTN DC    CL16'LISTEN          '
SOCNTOP  DC    CL16'NTOP            '
SOCPTON  DC    CL16'PTON            '
SOCREAD  DC    CL16'READ            '
SOCREADV DC    CL16'READV           '
SOCRECV  DC    CL16'RECV            '
SOCRECVF DC    CL16'RECVFROM        '
SOCRECVM DC    CL16'RECVMSG         '
SOCSELCT DC    CL16'SELECT          '
SOCSELX  DC    CL16'SELECTEX        '
SOCSEND  DC    CL16'SEND            '
SOCSENDM DC    CL16'SENDMSG         '
SOCSENDT DC    CL16'SENDTO          '
SOCSSOPT DC    CL16'SETSOCKOPT      '
SOCSHUTD DC    CL16'SHUTDOWN        '
SOCSOKET DC    CL16'SOCKET          '
SOCTSOCK DC    CL16'TAKESOCKET      '
SOCTERM  DC    CL16'TERMAPI         '
SOCWRITE DC    CL16'WRITE           '
SOCWRITV DC    CL16'WRITEV          '
ZERO     DC    F'0'
*
* Message(s) written to the transient data queue
*
STARTED_MSG DC CL40'EZACICAC Started successfully           '
STOPPED_MSG DC CL40'EZACICAC Stopped successfully           '
NOCOMMAREA DC  CL40'EZACICAC ***ERROR*** NO COMMAREA PASSED!'
TASK_START DC  CL40'TASK STARTING THRU CICS/TCPIP INTERFACE '
WRKEND   DC    CL20'CONNECTION END      '
*
* Message buffer for data from/to client
*
TCP_BUF  DS    0CL200             Buffer
TCP_BUF_H DC   CL3' '
TCP_BUF_DATA DC CL197' '
TCPLENG  DC    F'200'             Length of buffer
*
* Peers sockaddr
*
PEERADDR DS    0F                 Clients socket address
PEERFAM  DS    H                  Address family
PEER_DATA DS   0C                 Protocol specific area
PEER#LEN EQU   *-PEERADDR
```

Figure 180. EZACICAC assembler child server sample (Part 9 of 10)

```
         ORG   PEER_DATA          Start of AF_INET unique area
PEER_SIN DS    0C
PEER_SIN_PORT DS H                Clients port number
PEER_SIN_ADDR DS F                Clients INET address (netid)
         DS    CL8                Reserved area not used
         DS    20F
PEER_SIN#LEN EQU *-PEER_SIN       Length of AF_INET area
         ORG   PEER_DATA          Start of AF_INET6 unique area
PEER_SIN6 DS   0C
PEER_SIN6_PORT DS H               Clients port number
PEER_SIN6_FLOWINFO DS CL4         Flow information
PEER_SIN6_ADDR DS CL16            Clients INET address (netid)
PEER_SIN6_SCOPE_ID DS CL4         Scope Id
PEER_SIN6#LEN EQU *-PEER_SIN6     Length of AF_INET6 area
*
PEERADDR_LEN DS F
*
* Peers HOST/SERVICE NAME/LEN
*
PEER_HOSTNAME DS CL255
PEER_HOSTNAMELEN DS F
PEER_SERVICENAME DS CL32
PEER_SERVICENAMELEN DS F
*
* Receive Flag
*
GNI_FLAGS DS   F                  GETNAMEINFO flags
*
* Receive Flag
*
RECV_FLAG DS   F                  RECEIVE flags
*
*
*
TDTEXT5  DS    0CL40
         DC    CL10'Retcode = '
TDRETC   DC    CL7' '             Printable RETCODE
         DC    CL3' '
         DC    CL9'ERRNO = '
TDERRNO  DC    CL7' '             Printable ERRNO
         DC    CL4' '
*
*
*
SUCC     DC    CL24'Successful              '
NOTSUCC  DC    CL24'Not successful          '
FORCEMSG DS    CL1            Used to force the message when threadsafe
         LTORG
         YREGS
*
* All done.  Return to CICS...
*
SOCRET   DS    0H
         MVC   TDTEXT(40),STOPPED_MSG Move STOPPED message to TD area
         BAL   R7,WRITEQ          Write to TD Queue
         EXEC CICS RETURN
         END
```

Figure 180. EZACICAC assembler child server sample (Part 10 of 10)

EZACICAS

The following Assembler socket program is in the SEZAINST data set.

```
*ASM XOPTS(NOPROLOG)
*********************************************************************
*                                                                   *
* Module Name:  EZACICAS - This is a sample iterative server        *
*                                                                   *
* Copyright:    Licensed Materials - Property of IBM                *
*                                                                   *
*               "Restricted Materials of IBM"                       *
*                                                                   *
*               5694-A01                                            *
*                                                                   *
*               Copyright IBM Corp. 2003, 2007                      *
*                                                                   *
*               US Government Users Restricted Rights -             *
*               Use, duplication or disclosure restricted by        *
*               GSA ADP Schedule Contract with IBM Corp.            *
*                                                                   *
* Status:       CSV1R9                                              *
*                                                                   *
*                                                                   *
*   LANGUAGE:  ASSEMBLER                                            *
*                                                                   *
*   ATTRIBUTES: NON-REUSEABLE                                       *
*                                                                   *
*   REGISTER USAGE:                                                 *
*        R1  =                                                      *
*        R2  =                                                      *
*        R3  = BASE REGISTER                                        *
*        R4  = BASE REGISTER                                        *
*        R5  =                                                      *
*        R6  = WORK                                                 *
*        R7  = SUBROUTINE                                           *
*        R8  = WORK                                                 *
*        R9  = GWA REGISTER                                         *
*        R10 =                                                      *
*        R11 = EIB REGISTER                                         *
*        R12 =                                                      *
*        R13 = DATA REGISTER                                        *
*        R14 =                                                      *
*        R15 =                                                      *
*                                                                   *
*   INPUT:                                                          *
*                                                                   *
*   OUTPUT:                                                         *
*                                                                   *
* $MOD(EZACICAS),COMP(CICS),PROD(TCPIP):                            *
*                                                                   *
*                                                                   *
*********************************************************************
```

Figure 181. EZACICAS assembler iterative server sample (Part 1 of 20)

```
EZACICAS CSECT
         DFHEIENT CODEREG=(3,4),  Base registers for the program       X
               DATAREG=(13),      Base register for data               X
               EIBREG=(11)        Base register for CICS EIB
EZACICAS AMODE ANY  ADDRESSING MODE ...
EZACICAS RMODE ANY  RESIDENCY MODE ...
         B     SRV60000           Branch to startup address
         DC    CL17'EZACICAS-EYECATCH'
SRV60000 DS    0H                 Beginning of program
         USING GWA0000,R9         Address GWA storage
         MVC   MODULE,=C'EZACICAS: '
*
* Establish conditions to be ignored
*
         EXEC CICS IGNORE CONDITION TERMERR EOC SIGNAL NOTALLOC
*
* Establish conditions to be handled
*
         EXEC CICS HANDLE CONDITION ENDDATA(ENDDATA_ERR),              X
              IOERR(IOERR_ERR),                                        X
              LENGERR(LENGERR_ERR),                                    X
              NOSPACE(NOSPACE_ERR),                                    X
              QIDERR(QIDERR_ERR)
*
* Send message that server has started.
*
*        XC    MSGAREA,MSGAREA    Clear the message buffer
         MVC   MSGAREA(L'STARTOK),STARTOK Move STARTED message
         BAL   R7,HANDLE_TCPCICS  Write to TD Queue
*
* Determine the CICS Applid
*
         EXEC CICS ASSIGN APPLID(APPLID)
*
* Before the server can start, determine whether the IP CICS Sockets
* interface is active.
*
         EXEC CICS PUSH HANDLE
         EXEC CICS HANDLE CONDITION INVEXITREQ(TCP_TRUE_REQ),          X
               NOTAUTH(NOTAUTH_ERR)
         EXEC CICS EXTRACT EXIT PROGRAM('EZACIC01'),                   X
              GASET(R9) GALENGTH(GWALEN)
*
         EXEC CICS POP HANDLE
*
* At startup , the server requires the port number which it will use
* for its passive socket.
*
*  Invocation:  <server>,<port number>
*    where server is the CICS Transaction name assigned to EZACICAS
*    and port number is a port to which EZACICA will bind as its
*    passive socket.
*   TERMINAL => SRV6 04000
*   LISTENER => SRV6,04000
*   CECI     => CECI START TR(SRV6) FROM(04000)
```

Figure 181. EZACICAS assembler iterative server sample (Part 2 of 20)

```
*
*  THE LEADING SPACES ARE SIGNIFICANT.
*
         XC    TCP_INPUT_DATA,TCP_INPUT_DATA Clear input data area
         L     R8,ZERO
         STH   R8,TRMNL_LEN
         L     R8,TEN              Look for up to ten bytes data
         STH   R8,TRMNL_MAXLEN      from the terminal
*
         EXEC CICS RECEIVE INTO(TCP_INPUT_DATA) LENGTH(TRMNL_LEN)      X
              MAXLENGTH(TRMNL_MAXLEN)
*
         LH    R8,TRMNL_LEN        Check the amount of data received
         C     R8,TEN              from the terminal.  Was it 10?
         BE    USE_RECEIVED_PORT  Yes, go determine the port number
*
         XC    TCP_INPUT_DATA,TCP_INPUT_DATA Clear input data area
         L     R8,=F'1153'
         STH   R8,RETRIEVE_LEN      from The Listener
         MVC   TRANS,EIBTRNID      Copy the passed trans
*
         EXEC CICS RETRIEVE INTO(TCP_INPUT_DATA) LENGTH(RETRIEVE_LEN)
*
* Determine if the server was started by CECI or a listener.
*
         LH    R8,RETRIEVE_LEN    Load the RETRIEVED length
         C     R8,CECI_LEN        Is it less than 5?
         BNH   USE_RETRIEVED_PORT Yes.  Go use the RETRIEVE'd port
         OI    TAKESOCKET_SWITCH,X'01' Otherwise indicate the server   X
                                   was started by the Listener
         MVC   BIND_PORT(5),CLIENT_IN_DATA For the LISTEN message
         PACK  DWORK(8),CLIENT_IN_DATA(5) Use port from TIM
         B     CONVERT_PORT        Go convert it to binary format
USE_RECEIVED_PORT DS 0H
         MVC   BIND_PORT(5),TCP_INPUT_DATA+5 For the LISTEN message
         PACK  DWORK(8),TCP_INPUT_DATA+5(5) Use the port RECEIVE'd
         B     CONVERT_PORT
USE_RETRIEVED_PORT DS 0H
         MVC   BIND_PORT(5),TCP_INPUT_DATA For the LISTEN message
         PACK  DWORK(8),TCP_INPUT_DATA(5) Use the port RETRIEVE'd
CONVERT_PORT DS 0H
         CVB   R8,DWORK            Convert user supplied port to binary
         STH   R8,PORT              and save it for the passive socket
*
* If the server was started by a listener, then we must take the socket
* given.  Otherwise, we should proceed with an INITAPI.
*
         TM    TAKESOCKET_SWITCH,X'01' Do we need to use TAKESOCKET ?
         BO    LISTENER_STARTED_TASK Yes.  Go issue TAKESOCKET
*
* Since the server was not started by a listener, we should initialize
* the IP CICS Sockets interface.
*
INIT_SOCKETS DS 0H
         MVC   SUBTASKNO,EIBTASKN Use the CICS task number
```

Figure 181. EZACICAS assembler iterative server sample (Part 3 of 20)

```
*
         CALL  EZASOKET,(SOCINIT,MAXSOC,IDENT,INIT_SUBTASKID,MAXSNO,   X
               ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
         L     R5,ERRNO           Check for successful call
         L     R6,RETCODE         Check for successful call
         MVC   MSGCMD,SOCINIT     Show the API command
         C     R6,ZERO            Is it less than zero
         BL    SOCERR             Yes, go display error and terminate
         MVC   MSGRESULT(L'SUCC),SUCC Move SUCCESSFUL msg to TD area
         BAL   R7,HANDLE_TCPCICS  Write to TD Queue
         MVI   TERMAPI_REQUIRED_SW,C'Y' Since we did an INITAPI.
*
* Get an AF_INET6 socket.  If unsuccessful, then get an AF_INET socket.
*
SOCKET_BIND_LISTEN DS 0H
*
         CALL  EZASOKET,(SOCSOKET,AFINET6,SSTREAM,ZERO,                X
               ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
         L     R5,ERRNO           Check for successful call
         L     R6,RETCODE         Check for successful call
         MVC   MSGCMD,SOCSOKET     Show the API command
         C     R6,ZERO            Is it less than zero
         BL    GET_IPV4_SOCKET    Yes, go get an IPv4 socket
         STH   R6,SRV_SOCKID      Save the new socket descriptor
         MVC   MSGRESULT(L'SUCC),SUCC Move SUCCESSFUL msg to TD area
         BAL   R7,HANDLE_TCPCICS  Write to TD Queue
*
* Setup an IPv6 sockaddr.
*
         MVC   SAIN_SOCK_FAMILY,=AL2(AF_INET6) Set family to AF_INET6
         XC    SAIN_SOCK_SIN6_FLOWINFO,SAIN_SOCK_SIN6_FLOWINFO         X
                                  Flow info is zeros
         MVC   SAIN_SOCK_SIN6_ADDR,IN6ADDR_ANY Use IN6ADDR_ANY
         XC    SAIN_SOCK_SIN6_SCOPE_ID,SAIN_SOCK_SIN6_SCOPE_ID         X
                                  Scope ID is zeros
         MVC   SAIN_SOCK_SIN6_PORT,PORT Use the user specified port
         B     BIND_SERVER_SOCKET  Now go issue a BIND
*
GET_IPV4_SOCKET DS 0H
         CALL  EZASOKET,(SOCSOKET,AFINET,SSTREAM,ZERO,                 X
               ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
         L     R5,ERRNO           Check for successful call
         L     R6,RETCODE         Check for successful call
         MVC   MSGCMD,SOCSOKET
         C     R6,ZERO            Is it less than zero
         BL    SOCERR             Yes, go display error and terminate
         STH   R6,SRV_SOCKID      Save the new socket descriptor
         MVC   MSGRESULT(L'SUCC),SUCC Move SUCCESSFUL msg to TD area
         BAL   R7,HANDLE_TCPCICS  Write to TD Queue
*
* Setup an IPv4 sockaddr
*
```

Figure 181. EZACICAS assembler iterative server sample (Part 4 of 20)

```
         XC    SOCKADDR_IN(28),SOCKADDR_IN  Clear the sockaddr storage
         MVC   SAIN_SOCK_FAMILY,=AL2(AF_INET) Set family to AF_INET
         MVC   SAIN_SOCK_SIN_ADDR,INADDR_ANY Use INADDR_ANY
         MVC   SAIN_SOCK_SIN_PORT,PORT Use the user specified port
*
* Bind the socket to the service port to establish a local address for
* processing incoming connections.
*
BIND_SERVER_SOCKET DS 0H
*
         CALL  EZASOKET,(SOCBIND,SRV_SOCKID,SOCKADDR_IN,               X
               ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
         L     R5,ERRNO           Check for successful call
         L     R6,RETCODE         Check for successful call
         MVC   MSGCMD,SOCBIND
         C     R6,ZERO            Is it less than zero
         BL    SOCERR             Yes, go dispay error and terminate
         MVC   MSGRESULT(L'SUCC),SUCC Move SUCCESSFUL msg to TD area
         BAL   R7,HANDLE_TCPCICS  Write to TD Queue
*
* Call the LISTEN command to allow server to prepare a socket for
* incomming connections and set the maximum number of connections.
*
         MVC   BACKLOG,TEN        Set backlog to 10
*
         CALL  EZASOKET,(SOCLISTN,SRV_SOCKID,BACKLOG,                  X
               ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
         L     R5,ERRNO           Check for successful call
         L     R6,RETCODE         Check for successful call
         MVC   MSGCMD,SOCLISTN
         C     R6,ZERO            Is it less than zero
         BL    SOCERR             Yes, go dispay error and terminate
         MVC   MSGRESULT(L'SUCC),SUCC Move SUCCESSFUL msg to TD area
         BAL   R7,HANDLE_TCPCICS  Write to TD Queue
*
* Show server is ready to process client connections.
*
         L     R6,TWO             Force client socket desctiptor
         STH   R6,CLI_SOCKID        to be 2.
         MVC   MSGAREA(L'LISTEN_SUCC),LISTEN_SUCC
         BAL   R7,HANDLE_TCPCICS  Write to TD Queue
*
* Create a read mask for the SELECT command
*
         L     R8,NUM_FDS         Get the number of allowed FD's
         A     R8,ONE               and add one
         ST    R8,NFDS                for the SELECT call.
*
* Determine status IP CICS Sockets Interface
*
         CLI   GWATSTAT,GWATIMED  Are we in immediate termination
         BE    SOCRET             Return if so
         CLI   GWATSTAT,GWATQUIE  Are we in quiesceent termination
```

Figure 181. EZACICAS assembler iterative server sample (Part 5 of 20)

```
         BNE   SET_SELECT_BIT_MASK No, continue with SELECT
         B     CLOSEDOWN
*
* Create the read bitmask
*
SET_SELECT_BIT_MASK DS 0H
         LH    R6,SRV_SOCKID      Get the servers socket desciptor
         SRDL  R6,5               Compute the word number
         SRL   R7,27              Compute the socket number within the X
                                  mask word.
         SLR   R8,R8              Clear work register
         LA    R8,1               Set high-order bit
         SLL   R8,0(R7)           Create mask word
         ST    R8,SAVER8          Save mask word
         SLL   R6,2               Compute the offset
         LA    R7,READMASK        Address the read mask storage
         LA    R7,0(R6,R7)        Point to the word
         OC    0(4,R7),SAVER8     Turn on bits
*
* SELECT client connections
*
ACCEPT_CLIENT_REQ DS 0H
*
         CALL  EZASOKET,(SOCSELCT,NFDS,TIMEVAL,                        X
               READMASK,DUMYMASK,DUMYMASK,                             X
               REPLY_RDMASK,DUMYMASK,DUMYMASK,                         X
               ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
         L     R5,ERRNO           Check for successful call
         L     R6,RETCODE         Check for successful call
         ST    R6,SELECT_RETCODE  Save the SELECT return code
         MVC   MSGCMD,SOCSELCT
         C     R6,ZERO            Is it less than zero
         BL    SOCERR             Yes, go display error and terminate
         MVC   MSGRESULT(L'SUCC),SUCC Move SUCCESSFUL msg to TD area
         BAL   R7,HANDLE_TCPCICS  Write to TD Queue
*
* Check the return code to determine if any sockets are ready to be
* accepted.  If RETCODE is zero then there are no sockets ready.
*
         L     R6,SELECT_RETCODE  Retrieve the SELECT return code
         C     R6,ZERO            Any sockets ready ?
         BE    ACCEPT_CLIENT_REQ  No.  Go back and SELECT again
*
* Accept the client request.
*
         CALL  EZASOKET,(SOCACCT,SRV_SOCKID,SOCKADDR_IN,               X
               ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
         L     R5,ERRNO           Check for successful call
         L     R6,RETCODE         Check for successful call
         MVC   MSGCMD,SOCACCT
         C     R6,ZERO            Is it less than zero
         BL    SOCERR             Yes, go display error and terminate
         STH   R6,CLI_SOCKID      Save the new socket descriptor
```

Figure 181. EZACICAS assembler iterative server sample (Part 6 of 20)

```
         MVC   MSGRESULT(L'SUCC),SUCC Move SUCCESSFUL msg to TD area
         BAL   R7,HANDLE_TCPCICS  Write to TD Queue
*
* Get our peers' socket address
*
         CALL  EZASOKET,(SOCGPEER,CLI_SOCKID,SOCKADDR_PEER,            X
               ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
         L     R5,ERRNO           Capture the ERRNO and
         L     R6,RETCODE            the return code.
         MVC   MSGCMD,SOCGPEER       the API function performed.
         C     R6,ZERO            Is the call successful?
         BL    SOCERR             No!  Go display error and terminate
         MVC   MSGRESULT(L'SUCC),SUCC Move SUCCESSFUL msg to TD area
         BAL   R7,HANDLE_TCPCICS  Write to TD Queue
*
* Get our client's host name and service name
*
         L     R8,=F'16'          Set the sockaddr length to IPv4
         CLC   PEER_SOCK_FAMILY,=AL2(AF_INET) Is the client AF_INET ?
         BE    SET_SOCKADDR_LEN   Yes.  Go store the length.
         L     R8,=F'28'          Set the sockaddr length to IPv6
SET_SOCKADDR_LEN DS 0H
         ST    R8,PEERADDR_LEN    Save the value of the sockaddr length
         L     R8,ZERO            Clear the
         ST    R8,GNI_FLAGS          GETNAMEINFO flags
         XC    PEER_HOSTNAME,PEER_HOSTNAME Clear the host name storage
         L     R8,=F'255'         Set the length of
         ST    R8,PEER_HOSTNAMELEN   the host name storage
         XC    PEER_SERVICENAME,PEER_SERVICENAME Clear the service     X
                                                 name storage
         L     R8,=F'32'          Set the length of
         ST    R8,PEER_SERVICENAMELEN the service name storage
*
         CALL  EZASOKET,(SOCGNI,SOCKADDR_PEER,PEERADDR_LEN,            X
               PEER_HOSTNAME,PEER_HOSTNAMELEN,                         X
               PEER_SERVICENAME,PEER_SERVICENAMELEN,                   X
               GNI_FLAGS,                                              X
               ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
         L     R5,ERRNO           Capture the ERRNO and
         L     R6,RETCODE            the return code.
         MVC   MSGCMD,SOCGNI         the API function performed.
         C     R6,ZERO            Is the call successful?
         BL    SOCERR             No!  Go display error and terminate
         MVC   MSGRESULT(L'SUCC),SUCC Move SUCCESSFUL msg to TD area
         BAL   R7,HANDLE_TCPCICS  Write to TD Queue
*
* Display the host name
*
         MVC   TDHOST(L'TDHOST),PEER_HOSTNAME
         MVC   MSGAREA(L'TDHOSTMSG),TDHOSTMSG Move message to TD area
         BAL   R7,HANDLE_TCPCICS  Write to TD Queue
*
* Display the service name
```

Figure 181. EZACICAS assembler iterative server sample (Part 7 of 20)

```
*
         MVC   TDSERV(L'TDSERV),PEER_SERVICENAME
         MVC   MSGAREA(L'TDSERVMSG),TDSERVMSG Move message to TD area
         BAL   R7,HANDLE_TCPCICS  Write to TD Queue
*
* Receiving data through a socket by issuing  the RECVFROM command.
*
ACCEPT_RECEIVE DS 0H
         MVI   TCP_INDICATOR,C'T'
         MVC   TCPLENG,BUFFER_LENG
         XC    TCP_BUF,TCP_BUF    Clear the buffer storage
*
         CALL  EZASOKET,(SOCRECVF,CLI_SOCKID,RCVFM_FLAG,TCPLENG,       X
               TCP_BUF,SOCKADDR_IN,                                    X
               ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
         L     R5,ERRNO           Capture the ERRNO and
         L     R6,RETCODE            the return code.
         ST    R6,RECVFROM_RETCODE Save the RECVFROM return code
         C     R6,ZERO            Is the call successful?
         BL    RECVFROM_ERROR     No!
*
* If the RECVFROM return code is zero and the number of bytes received
* is also zero, then there is nothing further to process.
*
         BE    CHECK_NBYTES       Yes. Go check number bytes received
         B     RECVFROM_OK        NO. Go interpret clients data
CHECK_NBYTES DS 0H
         L     R6,TCPLENG         Check number of bytes received
         C     R6,ZERO            Is it zero ?
         BE    ACCEPT_RECEIVE     Yes.  Go issue RECVFROM again.
         B     RECVFROM_OK        No.  Must have received something.
RECVFROM_ERROR DS 0H
         MVC   MSGAREA(L'RECVFROM_ERR),RECVFROM_ERR
         BAL   R7,HANDLE_TCPCICS  Write to TD Queue
         MVI   TASK_FLAG,C'1'     Force the Client connection to end
         B     CLOSE_CLIENT       Go close clients socket
RECVFROM_OK DS 0H
*
* Interpret the clients request.
*
* Remove the following call to EZACIC05 if using an EBCDIC client.
*
*        CALL  EZACIC05,(TCP_BUF,TCPLENG),VL,MF=(E,PARMLIST)
*
         CLC   TCP_BUF_H,TCP_BUF_H_LOW_VALUES Display data received
         BE    COMMAND_IS_LOW_VALUES from the client as blanks.
         CLC   TCP_BUF_H,TCP_BUF_H_SPACES Display data received from
         BE    COMMAND_IS_SPACES  the client as blanks
         CLC   TCP_BUF_H,TCP_BUF_H_END End client connection?
         BE    SET_END            Yes.
         CLC   TCP_BUF_H,TCP_BUF_H_TRM Terminate server?
         BE    SET_TERM           Yes.
```

Figure 181. EZACICAS assembler iterative server sample (Part 8 of 20)

```
*
* Inform the cleint that the server has process the message
*
         XC    MSGAREA,MSGAREA
         MVC   MSGAREA(L'SERVER_PROC_MSG),SERVER_PROC_MSG
*
         EXEC CICS SYNCPOINT
*
         EXEC CICS ASKTIME ABSTIME(UTIME) NOHANDLE
         EXEC CICS FORMATTIME ABSTIME(UTIME)                           X
              DATESEP('/') MMDDYY(MSGDATE)                             X
              TIME(MSGTIME) TIMESEP(':') NOHANDLE
         LA    R6,TCPCICS_MSG_AREA_LEN
         STH   R6,TDLEN
         EXEC CICS WRITEQ TD QUEUE('CSMT')                             X
              FROM(TCPCICS_MSG_AREA)                                   X
              LENGTH(TDLEN)
*
         MVC   TCP_BUF,TCPCICS_MSG_AREA_2
*
* Remove the following call to EZACIC04 if using an EBCDIC client.
*
*        CALL  EZACIC04,(TCP_BUF,TCPLENG),VL,MF=(E,PARMLIST)
*
* Write the server process message back to the client
*
         CALL  EZASOKET,(SOCWRITE,CLI_SOCKID,TCPLENG,TCP_BUF,          X
               ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
         L     R5,ERRNO           Capture the ERRNO and
         L     R6,RETCODE            the return code.
         MVC   MSGCMD,SOCWRITE        the API function performed.
         C     R6,ZERO            Is the call successful?
         BL    TALK_CLIENT_BAD    No!  Go display error
         MVC   MSGRESULT(L'SUCC),SUCC Move SUCCESSFUL msg to TD area
*
         XC    TCP_BUF,TCP_BUF
         MVI   TCP_INDICATOR,X'00'
         B     ACCEPT_RECEIVE     Go receive more client data
TALK_CLIENT_BAD DS 0H
         MVI   TASK_FLAG,C'1'     Force client connection to end.
         B     CLOSE_CLIENT
*
* Process command from client
*
COMMAND_IS_LOW_VALUES DS 0H
COMMAND_IS_SPACES DS 0H
         XC    MSGRESULT,MSGRESULT
         MVC   MSGCMD,SOCRECVF
         MVC   MSGRESULT(37),=C'CLIENT COMMAND IS BLANKS OR LOWVALUES'
         BAL   R7,HANDLE_TCPCICS  Write to TD Queue
         B     ACCEPT_RECEIVE     Go receive more data from client
SET_END DS 0H
```

Figure 181. EZACICAS assembler iterative server sample (Part 9 of 20)

```
         MVI   TASK_FLAG,C'1'
         B     CLOSE_CLIENT
SET_TERM DS 0H
         MVI   TASK_FLAG,C'2'
         B     CLOSE_CLIENT
*
*   CLOSE CLIENT SOCKET DESCRIPTOR
*
CLOSE_CLIENT DS 0H
         CALL  EZASOKET,(SOCCLOSE,CLI_SOCKID,                          X
               ERRNO,RETCODE),VL,MF=(E,PARMLIST)
         L     R5,ERRNO           Check for successful call
         L     R6,RETCODE         Check for successful call
         MVC   MSGCMD,SOCCLOSE
         C     R6,ZERO            Is it less than zero
         BL    SOCERR             Yes, go display error and terminat
         MVC   MSGRESULT(L'SUCC),SUCC Move SUCCESSFUL msg to TD area
         BAL   R7,HANDLE_TCPCICS  Write to TD Queue
*
* Determine whether we should select another socket
*
         CLI   TASK_FLAG,C'2'     Terminate server?
         BE    CLOSEDOWN          Yes. Go close passive socket
         MVI   TASK_FLAG,C'0'     Reset the task flag for next client
         B     ACCEPT_CLIENT_REQ  Go select new connection.
*
CLOSEDOWN DS   0H
*
* CLOSE SOCKET DESCRIPTOR
*
* SET THE SERVER SOCKET TO NOT LINGER ON THE CLOSE
*
         CALL  EZASOKET,(SOCSETSO,SRV_SOCKID,SOCK#SO_LINGER,ON_ZERO,   X
               EIGHT,ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
* CLOSE THE SERVER PASSIVE SOCKET
*
         CALL  EZASOKET,(SOCCLOSE,SRV_SOCKID,                          X
               ERRNO,RETCODE),VL,MF=(E,PARMLIST)
         L     R5,ERRNO           Check for successful call
         L     R6,RETCODE         Check for successful call
         MVC   MSGCMD,SOCCLOSE
         C     R6,ZERO            Is it less than zero
         BL    SOCERR             Yes, go display error and terminat
         MVC   MSGRESULT(L'SUCC),SUCC Move SUCCESSFUL msg to TD area
         BAL   R7,HANDLE_TCPCICS  Write to TD Queue
         CLI   TERMAPI_REQUIRED_SW,C'Y' A TERMAPI needed ?
         BE    TERM_API           Yes, go issue TERMAPI
         B     SOCRET             No, return to CICS
*
* Terminate IP CICS Sockets API
*
TERM_API DS    0H
         CALL  EZASOKET,(SOCTERM),VL,MF=(E,PARMLIST)
         MVC   MSGCMD,SOCTERM
```

Figure 181. EZACICAS assembler iterative server sample (Part 10 of 20)

```
         MVC   MSGRESULT(L'SUCC),SUCC Move SUCCESSFUL msg to TD area
         BAL   R7,HANDLE_TCPCICS  Write to TD Queue
*
         B     SOCRET
*
* Listener Started Task routine.
*
LISTENER_STARTED_TASK DS 0H
*
* Take the socket which was given by the listener.
*
         L     R8,GIVE_TAKE_SOCKET Use the socket descriptor from the
         STH   R8,SOCKET_TO_TAKE      TIM for the TAKESOCKET
         XC    CLIENTID_LSTN,CLIENTID_LSTN Clear the clientid
         LH    R8,STIM_FAMILY     Get the domain from the TIM
         ST    R8,CID_DOMAIN_LSTN Set the domain
         MVC   CID_LSTN_INFO,CLIENTID_PARM Set the Address space and   X
                                  subtask name.
*
         CALL  EZASOKET,(SOCTSOCK,SOCKET_TO_TAKE,CLIENTID_LSTN,        X
               ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
         L     R5,ERRNO           Check for successful call
         L     R6,RETCODE         Check for successful call
         MVC   MSGCMD,SOCTSOCK     Set the API name
         C     R6,ZERO            Is it less than zero
         BL    SOCERR             Yes, go display error and terminate
         STH   R6,SRV_SOCKID      Save the taken socket descriptor
         MVC   MSGRESULT(L'SUCC),SUCC Move SUCCESSFUL msg to TD area
         BAL   R7,HANDLE_TCPCICS  Write to TD Queue
*
* Inform the client that the server has started.
*
         MVC   TCPLENG,BUFFER_LENG Set the message length
         XC    TCP_BUF,TCP_BUF    Clear the buffer
         MVC   TCP_BUF(L'STARTOK),STARTOK Move STARTED message
*
* Remove the following call to EZACIC04 if using an EBCDIC client.
*
| *      CALL  EZACIC04,(TCP_BUF,TCPLENG),VL,MF=(E,PARMLIST)
*
* Notify client the the child subtask has started.
*
         CALL  EZASOKET,(SOCWRITE,SRV_SOCKID,TCPLENG,TCP_BUF,          X
               ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
         L     R5,ERRNO           Capture the ERRNO and
         L     R6,RETCODE            the return code.
         MVC   MSGCMD,SOCWRITE        the API function performed.
         C     R6,ZERO            Is the call successful?
         BL    SOCERR             No!  Go display error and terminate
         MVC   MSGRESULT(L'SUCC),SUCC Move SUCCESSFUL msg to TD area
         BAL   R7,HANDLE_TCPCICS  Write to TD Queue
```

Figure 181. EZACICAS assembler iterative server sample (Part 11 of 20)

```
*
* Close the taken socket descriptor
*
         CALL  EZASOKET,(SOCCLOSE,SRV_SOCKID,                          X
               ERRNO,RETCODE),VL,MF=(E,PARMLIST)
         L     R5,ERRNO           Check for successful call
         L     R6,RETCODE         Check for successful call
         MVC   MSGCMD,SOCCLOSE
         C     R6,ZERO            Is it less than zero
         BL    SOCERR             Yes, go display error and terminat
         MVC   MSGRESULT(L'SUCC),SUCC Move SUCCESSFUL msg to TD area
         BAL   R7,HANDLE_TCPCICS  Write to TD Queue
*
* Continue with server startup
*
         B     SOCKET_BIND_LISTEN Go continue the server startup
*
* Various routines to process error conditions
*
TCP_TRUE_REQ DS 0H
         MVC   MSGAREA(L'TCP_EXIT_MSG),TCP_EXIT_MSG
         B     SEND_ERR_MSG
NOTAUTH_ERR DS 0H
         MVC   MSGAREA(L'NOTAUTH_MSG),NOTAUTH_MSG
         B     SEND_ERR_MSG
INVREQ_ERR DS  0H
         MVC   MSGAREA(L'TCP_EXIT_MSG),TCP_EXIT_MSG
         B     SEND_ERR_MSG
IOERR_ERR DS   0H
         MVC   MSGAREA(L'IOERR_MSG),IOERR_MSG
         B     SEND_ERR_MSG
LENGERR_ERR DS 0H
         MVC   MSGAREA(L'LENGERR_MSG),LENGERR_MSG
         B     SEND_ERR_MSG
NOSPACE_ERR DS 0H
         MVC   MSGAREA(L'NOSPACE_MSG),NOSPACE_MSG
         B     SEND_ERR_MSG
QIDERR_ERR DS  0H
         MVC   MSGAREA(L'QIDERR_MSG),QIDERR_MSG
         B     SEND_ERR_MSG
ITEMERR_ERR DS 0H
         MVC   MSGAREA(L'ITEMERR_MSG),ITEMERR_MSG
         B     SEND_ERR_MSG
ENDDATA_ERR DS 0H
         MVC   MSGAREA(L'ENDDATA_MSG),ENDDATA_MSG
         B     SEND_ERR_MSG
SEND_ERR_MSG DS 0H
         BAL   R7,HANDLE_TCPCICS  Write to TD Queue
         B     SOCRET             Return to CICS!
*
* Error on EZASOKET call
*
SOCERR   DS    0H
         MVC   MSGAREA(L'MSGCMD),MSGCMD
         MVC   MSGAREA+16(L'SOCKET_ERR),SOCKET_ERR
```

Figure 181. EZACICAS assembler iterative server sample (Part 12 of 20)

```
         BAL   R7,HANDLE_TCPCICS  Write to TD Queue
*
         L     R6,RETCODE         Pick up the RETCODE value
         L     R5,ERRNO           Pick up the ERRNO value
         CVD   R6,DWORK           Format the RETCODE
         UNPK  TDRETC,DWORK+4(4)    for printing to the
         OI    TDRETC+6,X'F0'          TD queue
*
         CVD   R5,DWORK           Format the ERRNO
         UNPK  TDERRNO,DWORK+4(4)   for printing to the
         OI    TDERRNO+6,X'F0'         TD queue
*
         MVC   MSGAREA(L'TDTEXT5),TDTEXT5 Move the RETCODE and ERRNO   X
                                  to the TD queue area
         BAL   R7,HANDLE_TCPCICS  Write the message to the TD queue
*
         B     SOCRET             Return to CICS
*
* Write a message to the "CSMT" destination queue for logging
*
HANDLE_TCPCICS DS 0H
         EXEC CICS ASKTIME ABSTIME(UTIME) NOHANDLE
         EXEC CICS FORMATTIME ABSTIME(UTIME)                           X
              DATESEP('/') MMDDYY(MSGDATE)                             X
              TIME(MSGTIME) TIMESEP(':') NOHANDLE
         LA    R6,TCPCICS_MSG_AREA_LEN
         STH   R6,TDLEN
         EXEC CICS WRITEQ TD QUEUE('CSMT')                             X
              FROM(TCPCICS_MSG_AREA)                                   X
              LENGTH(TDLEN)
*
* Tell the client?
*
         CLI   TCP_INDICATOR,C'T'
         BNE   HANDLE_TCPCICS_RETURN
         MVC   TCPLENG,BUFFER_LENG
         XC    TCP_BUF,TCP_BUF
         MVC   TCP_BUF,TCPCICS_MSG_AREA_2
*
* Remove the following call to EZACIC04 if using an EBCDIC client.
*
*        CALL  EZACIC04,(TCP_BUF,TCPLENG),VL,MF=(E,PARMLIST)
         MVI   TCP_INDICATOR,C' '
*
* Notify client the the child subtask has started.
*
         CALL  EZASOKET,(SOCWRITE,CLI_SOCKID,TCPLENG,TCP_BUF,          X
               ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
         L     R5,ERRNO           Capture the ERRNO and
         L     R6,RETCODE            the return code.
         MVC   MSGCMD,SOCWRITE        the API function performed.
         C     R6,ZERO            Is the call successful?
```

Figure 181. EZACICAS assembler iterative server sample (Part 13 of 20)

```
         BL    HANDLE_TCPCICS_RETURN
         MVC   MSGRESULT(L'SUCC),SUCC Move SUCCESSFUL msg to TD area
*
         EXEC CICS ASKTIME ABSTIME(UTIME) NOHANDLE
         EXEC CICS FORMATTIME ABSTIME(UTIME)                          X
              DATESEP('/') MMDDYY(MSGDATE)                            X
              TIME(MSGTIME) TIMESEP(':') NOHANDLE
         LA    R6,TCPCICS_MSG_AREA_LEN
         STH   R6,TDLEN
         EXEC CICS WRITEQ TD QUEUE('CSMT')                            X
              FROM(TCPCICS_MSG_AREA)                                  X
              LENGTH(TDLEN)
*
HANDLE_TCPCICS_RETURN DS 0H
         XC    MSGAREA,MSGAREA
         BR    R7                  Return to caller
*
* ALL DONE.
*
SOCRET   DS    0H
         MVC   MSGAREA(L'STOPOK),STOPOK Move STOPPED msg to TD area
         BAL   R7,HANDLE_TCPCICS  Write to TD Queue
         EXEC CICS RETURN
*
* INITAPI parameters
*
MAXSOC   DC    H'0'                MAXSOC value, use the default
IDENT    DC    0CL16' '
TCPNAME  DC    CL8'TCPCS   '       Name of the TCP
APPLID   DC    CL8'CICS    '       Address space name
INIT_SUBTASKID DS 0CL8             Subtask for INITAPI
SUBTASKNO DC   CL7'       '          from EIBTASKN
SUBT_CHAR DC   CL1'L'             Make server use a non-reusable subtask
MAXSNO   DC    F'0'                Highest socket descriptor available
*
* Sockets address family
*
AFINET   DC    F'2'                AF_INET
AFINET6  DC    F'19'               AF_INET6
*
* SOCKET FUNCTIONS
*
SOCACCT  DC    CL16'ACCEPT          '
SOCBIND  DC    CL16'BIND            '
SOCCLOSE DC    CL16'CLOSE           '
SOCCONNT DC    CL16'CONNECT         '
SOCFCNTL DC    CL16'FCNTL           '
SOCFAI   DC    CL16'FREEADDRINFO    '
SOCGCLID DC    CL16'GETCLIENTID     '
SOCGAI   DC    CL16'GETADDRINFO     '
SOCGNI   DC    CL16'GETNAMEINFO     '
SOCGTHID DC    CL16'GETHOSTID       '
SOCGTHN  DC    CL16'GETHOSTNAME     '
SOCGPEER DC    CL16'GETPEERNAME     '
SOCGTSN  DC    CL16'GETSOCKNAME     '
```

Figure 181. EZACICAS assembler iterative server sample (Part 14 of 20)

```
SOCGETSO DC    CL16'GETSOCKOPT      '
SOCGSOCK DC    CL16'GIVESOCKET      '
SOCINIT  DC    CL16'INITAPI         '
SOCIOCTL DC    CL16'IOCTL           '
SOCLISTN DC    CL16'LISTEN          '
SOCNTOP  DC    CL16'NTOP            '
SOCPTON  DC    CL16'PTON            '
SOCREAD  DC    CL16'READ            '
SOCREADV DC    CL16'READV           '
SOCRECV  DC    CL16'RECV            '
SOCRECVF DC    CL16'RECVFROM        '
SOCRECVM DC    CL16'RECVMSG         '
SOCSELCT DC    CL16'SELECT          '
SOCSELX  DC    CL16'SELECTEX        '
SOCSEND  DC    CL16'SEND            '
SOCSENDM DC    CL16'SENDMSG         '
SOCSENDT DC    CL16'SENDTO          '
SOCSETSO DC    CL16'SETSOCKOPT      '
SOCSOKET DC    CL16'SOCKET          '
SOCTSOCK DC    CL16'TAKESOCKET      '
SOCTERM  DC    CL16'TERMAPI         '
SOCWRITE DC    CL16'WRITE           '
SOCWRITV DC    CL16'WRITEV          '
*
* SELECT parms
*
NUM_FDS  DC    F'5'                Number of file descriptors
NFDS     DS    F
TIMEVAL  DC    AL4(180),AL4(0)
SELECT_CSOCKET DS 0CL12
READMASK DC    XL4'00'             SELECT read mask
DUMYMASK DC    XL4'00'             mask set to binary zeros
REPLY_RDMASK DC XL4'00'            SELECT reply read mask
REPLY_RDMASK_FF DS XL4
SELECT_RETCODE DS F                Sum of all ready sockets in masks
*
TCPLENG  DC    F'0'
*
SSTREAM  DC    F'1'                socket type stream
ZERO     DC    F'0'
ONE      DC    F'1'
TWO      DC    F'2'
SIX      DC    F'6'
EIGHT    DC    F'8'
TEN      DC    F'10'
*
* Data for RETRIEVE
*
TRANS    DS    CL4                 Transaction retrieved
LENG     DS    H                   Length of data retreived
CECI_LEN DC    F'5'                Length of Port from CICS Start
TAKESOCKET_SWITCH DC X'00'         Used to drive a TAKESOCKET
TCP_INDICATOR DC CL1' '
```

Figure 181. EZACICAS assembler iterative server sample (Part 15 of 20)

```
TASK_FLAG DC   CL1'0'             Server task flag
*
TCP_BUF  DS    0CL55              Buffer
TCP_BUF_H DC   CL3' '             Used to pass the server commands
TCP_BUF_DATA DC CL52' '
TCP_BUF_H_END DC CL3'END'         Command to end the client connection
TCP_BUF_H_LOW_VALUES DC XL3'000000' Client sent command=low values
TCP_BUF_H_SPACES DC CL3'   '      Client sent command=spaces
TCP_BUF_H_TRM DC CL3'TRM'         Command to terminate the server
BUFFER_LENG  DC F'55'             Length of buffer
*
* LISTEN parms
*
BACKLOG  DC    F'0'               Backlog for LISTEN
*
* RECVFROM parms
*
RCVFM_FLAG DC  F'0'               RECVFROM flag
*
* MESSAGE(S) WRITTEN TO TRANSIENT DATA QUEUE
*
BITMASK_ERR  DC CL36'BITMASK CONVERSION - FAILED'
LISTEN_SUCC DS 0CL46
         DC    CL34'READY TO ACCEPT REQUESTS ON PORT: '
BIND_PORT DC   CL5' '
         DC    CL7' '
ENDDATA_MSG DC CL30'RETRIEVE DATA CAN NOT BE FOUND'
IOERR_MSG DC   CL12'IOERR OCCURS'
ITEMERR_MSG DC CL13'ITEMERR ERROR'
LENGERR_MSG DC CL13'LENGERR ERROR'
NOSPACE_MSG DC CL17'NOSPACE CONDITION'
RECVFROM_ERR DC CL36'RECVFROM SOCKET CALL FAILED'
QIDERR_MSG DC  CL30'TRANSIENT DATA QUEUE NOT FOUND'
SERVER_PROC_MSG DC CL55'SERVER PROCESSED MESSAGE'
SOCKET_ERR DC  CL15'EZASOKET ERROR!'
STARTOK  DC    CL27'SERVER STARTED SUCCESSFULLY'
STOPOK   DC    CL27'SERVER STOPPED SUCCESSFULLY'
TCP_EXIT_MSG DC CL31'SERVER STOPPED:TRUE NOT ACTIVE'
NOTAUTH_MSG DC CL31'SERVER STOPPED: NOT AUTHORIZED'
*
* Message to display the clients host name
*
TDHOSTMSG DS   0CL55
TDHOSTLIT DC   CL9'HOSTNAME='
TDHOST   DC    CL46' '
*
* Message to display the clients service name
*
TDSERVMSG DS   0CL55
TDSERVLIT DC   CL8'SERVICE='
TDSERV   DC    CL32' '
         DC    CL15' '
*
* Message to display EZASOKET RETCODE and ERRNO
*
```

Figure 181. EZACICAS assembler iterative server sample (Part 16 of 20)

```
TDTEXT5  DS    0CL40
         DC    CL10'RETCODE = '
TDRETC   DC    CL7' '              Printable RETCODE
         DC    CL3' '
         DC    CL9'ERRNO = '
TDERRNO  DC    CL7' '              Printable ERRNO
         DC    CL4' '
*
* Misc
*
SUCC     DC    CL10'SUCCESSFUL'
NOTSUCC  DC    CL14'NOT SUCCESSFUL'
TERMAPI_REQUIRED_SW DC CL1'N'
ON_ZERO  DS    0C
LINGERON DC    F'1'                On/Off
LINGERTIME DC  F'0'                Linger time
         LTORG
*
* DSECTs
*
         EZACICA TYPE=DSECT,AREA=GWA
         EZACICA TYPE=DSECT,AREA=TIE
         DFHEISTG
SRV6SAVE DS    18F                 Register Save Area
SRV6STRSV DS   F                   Save area for start subroutine
*
* Socket address structure
*
         CNOP  0,8                 DOUBLEWORD BOUNDARY
SOCKADDR_IN              DS 0F    Socket address structure
SAIN_SOCK_FAMILY        DS H      Address Family
SAIN_SOCK_DATA          DS 0C     Protocol specific area
         ORG SAIN_SOCK_DATA       Start of AF_INET unique area
SAIN_SOCK_SIN           DS 0C
SAIN_SOCK_SIN_PORT      DS H      Port number
SAIN_SOCK_SIN_ADDR      DS CL4    IPv4 address
                         DS CL8   Reserved area not used
         ORG SAIN_SOCK_DATA       Start of AF_INET6 area
SAIN_SOCK_SIN6          DS 0C
SAIN_SOCK_SIN6_PORT     DS H      Port number
SAIN_SOCK_SIN6_FLOWINFO DS CL4    Flow Information
SAIN_SOCK_SIN6_ADDR     DS CL16   IPv6 address
SAIN_SOCK_SIN6_SCOPE_ID DS CL4    Scope id
*
* Peers address structure
*
         CNOP  0,8                 DOUBLEWORD BOUNDARY
SOCKADDR_PEER            DS 0F    Socket address structure
PEER_SOCK_FAMILY        DS H      Address Family
PEER_SOCK_DATA          DS 0C     Protocol specific area
         ORG PEER_SOCK_DATA       Start of AF_INET unique area
PEER_SOCK_SIN           DS 0C
PEER_SOCK_SIN_PORT      DS H      Port number
PEER_SOCK_SIN_ADDR      DS CL4    IPv4 address
                         DS CL8   Reserved area not used
```

Figure 181. EZACICAS assembler iterative server sample (Part 17 of 20)

```
         ORG PEER_SOCK_DATA       Start of AF_INET6 area
PEER_SOCK_SIN6          DS 0C
PEER_SOCK_SIN6_PORT     DS H      Port number
PEER_SOCK_SIN6_FLOWINFO DS CL4    Flow Information
PEER_SOCK_SIN6_ADDR     DS CL16   IPv6 address
PEER_SOCK_SIN6_SCOPE_ID DS CL4    Scope id
*
PEERADDR_LEN DS F                 Length of Peers sockaddr
*
* Peers HOST/SERVICE NAME/LEN
*
PEER_HOSTNAME DS CL255            Peers Host name
PEER_HOSTNAMELEN DS F             Peers Host name length
PEER_SERVICENAME DS CL32          Peers Service name
PEER_SERVICENAMELEN DS F          Peers Service name length
*
* Receive Flag
*
GNI_FLAGS DS   F                  GETNAMEINFO flags
*
* User supplied port to listen on
*
PORT     DS    H                  User supplied port
*
* Storage used to create a message to be written to the CSMT TD Queue
*
TCPCICS_MSG_AREA DS 0F            TD Message area
TCPCICS_MSG_AREA_1 DS 0C
MSGDATE  DS    CL8                MM/DD/YY
MSGFILR1 DS    CL2
MSGTIME  DS    CL8                HH:MM:SS
MSGFILR2 DS    CL2
MODULE   DS    CL10               "EZACICAS: "
TCPCICS_MSG_AREA_2 DS 0C
MSGAREA  DS    CL55
         ORG   MSGAREA
MSGCMD   DS    CL16               EZASOKET command issued
MSGRESULT DS   CL39               Outcome of the command issued
TCPCICS_MSG_AREA_END EQU *        End of message
TCPCICS_MSG_AREA_LEN EQU TCPCICS_MSG_AREA_END-TCPCICS_MSG_AREA         X
                                  Length of TD message text
*
TDLEN    DS    H                  Length of TD message text
*
* Various other working storage areas
*
UTIME    DS    PL8                ABSTIME data area
DWORK    DS    D                  Double word work area
UNPKWRK  DS    CL15               Unpack work area
PARMLIST DS    20F
*
* Error numbers and return codes
*
ERRNO    DS    F                  ERRNO
RETCODE  DS    F                  Return Code
```

Figure 181. EZACICAS assembler iterative server sample (Part 18 of 20)

```
RECVFROM_RETCODE DS F
*
* Client ID from Listener to be used by the TAKESOKET command
*
CLIENTID_LSTN DS 0CL40
CID_DOMAIN_LSTN DS F               Domain
CID_LSTN_INFO DS 0CL16
CID_NAME_LSTN DS CL8               Address space name
CID_SUBTNAM_LSTN DS CL8            Subtask name
CID_RES_LSTN DS CL20
*
SOCKET_TO_TAKE DS H                Socket descriptor to take
*
* Data from the CICS RECIEVE command
*
TRMNL_LEN DS   H                   Length of data RECEIVE'd
TRMNL_MAXLEN DS   H
*
* Data from the CICS RETRIEVE command
*
RETRIEVE_LEN DS H                  Length of data RETRIEVE'd
*
* Socket descriptors
*
SRV_SOCKID DS  H                   Server socket descriptor
CLI_SOCKID DS  H                   Client socket descriptor
*
* For saving R8
*
SAVER8   DS    F
*
* Server data
*
         CNOP  0,8                 DOUBLEWORD BOUNDARY
TCP_INPUT_DATA DS CL85             Data retrieved
         ORG   TCP_INPUT_DATA
*
* The Listeners Task Input Message (TIM)
*
TCPSOCKET_PARM DS 0C
GIVE_TAKE_SOCKET DS F
CLIENTID_PARM DS 0CL16
LSTN_NAME DS   CL8
LSTN_SUBNAME DS CL8
CLIENT_IN_DATA DS CL35
         DS    CL1
SOCKADDR_TIM DS    0F
STIM_FAMILY DS H
STIM_DATA DS   0C
STIM#LEN EQU   *-SOCKADDR_TIM
         ORG   STIM_DATA
STIM_SIN DS    0C
STIM_SIN_PORT DS H
STIM_SIN_ADDR DS CL4
         DS    CL8
```

Figure 181. EZACICAS assembler iterative server sample (Part 19 of 20)

```
         DS    20F
STIM_SIN#LEN EQU *-STIM_SIN
          ORG  STIM_DATA
STIM_SIN6 DS 0C
STIM_SIN6_PORT DS H
STIM_SIN6_FLOWINFO DS CL4
STIM_SIN6_ADDR DS CL16
STIM_SIN6_SCOPE_ID DS CL4
STIM_SIN6#LEN EQU *-STIM_SIN6
         ORG
         DS    CL68
CLIENT_IN_DATA_LENGTH DS H
CLIENT_IN_DATA_2 DS 0C
*
* Fields for EXTRACT EXIT to determine if IP CICS Sockets interface
* is active.
*
GWALEN   DS    H
*
         EZBREHST DSECT=NO,LIST=YES,HOSTENT=NO,ADRINFO=NO
         BPXYSOCK DSECT=NO,LIST=YES
         DFHEIEND TERMINATE EXECUTE INTERFACE DYNAMIC STORAGE
         YREGS
         END   EZACICAS
```

Figure 181. EZACICAS assembler iterative server sample (Part 20 of 20)

SELECTEX

The following sample displays COBOL code issuing the SELECTEX socket call:

```
  This is sample COBOL code issuing the SELECTEX socket call:

*-------------------------------------------------------------------*
* Here is a anotated SAMPLE code from a test tool used to test     *
* the SELECTEX:                                                     *
*-------------------------------------------------------------------*
       WORKING-STORAGE SECTION.
       01  SELECT-BITMASK                PIC 9(16) BINARY VALUE 0.
       01  SELECT-BITMASK-LEN            PIC 9(8) BINARY VALUE 0.
       01  SELECT-CHAR-STRING            PIC X(64).
       01  SELECT-MAXSOC                 PIC 9(8) BINARY VALUE 0.
       01  SELECT-TIMEOUT.
           03 SELECT-TIMEOUT-SECONDS     PIC S9(8) BINARY VALUE 0.
           03 SELECT-TIMEOUT-MICROSEC    PIC S9(8) BINARY VALUE 0.
       01  SELECT-RSNDMSK                PIC 9(16) BINARY.
       01  SELECT-WSNDMSK                PIC 9(16) BINARY.
       01  SELECT-ESNDMSK                PIC 9(16) BINARY.
       01  SELECT-RRETMSK                PIC 9(16) BINARY.
       01  SELECT-WRETMSK                PIC 9(16) BINARY.
       01  SELECT-ERETMSK                PIC 9(16) BINARY.
       77  SELECT-ECB-PTR                USAGE IS POINTER.

       LINKAGE SECTION.
       01  SELECT-ECB                    PIC 9(8) BINARY.

       PROCEDURE DIVISION USING L1.

       PROCESS-SELECTEX.
      *
      * GET SHARED STORAGE FOR ECB.
      *
           EXEC CICS GETMAIN SHARED
                SET (SELECT-ECB-PTR)
                FLENGTH (4)
```

```
           INITIMG ('00')
           END-EXEC.
      SET ADDRESS OF SELECT-ECB TO SELECT-ECB-PTR.
      INITIALIZE SELECT-ECB.

*
* WRITE ECB ADDRESS TO TS QUEUE
*
      EXEC CICS WRITEQ TS
           QUEUE ('POSTECB@')
           FROM (SELECT-ECB-PTR)
           LENGTH (4)
           END-EXEC.

*
* SOCKET CALL SELECTEX
*

      MOVE 10 TO SELECT-MAXSOC.

      MOVE -1 TO SELECT-TIMEOUT-SECONDS.
      MOVE -1 TO SELECT-TIMEOUT-MICROSEC.

      MOVE read-send-maskTO SELECT-CHAR-STRING.
      MOVE 64 TO SELECT-BITMASK-LEN.
      CALL 'EZACIC06' USING CTOB
                            SELECT-BITMASK
                            SELECT-CHAR-STRING
                            SELECT-BITMASK-LEN
                            RETCODE.
      MOVE SELECT-BITMASK TO SELECT-RSNDMSK.

      MOVE write-send-maskTO SELECT-CHAR-STRING.
      MOVE 64 TO SELECT-BITMASK-LEN.
      CALL 'EZACIC06' USING CTOB
                            SELECT-BITMASK
                            SELECT-CHAR-STRING
                            SELECT-BITMASK-LEN
                            RETCODE.
      MOVE SELECT-BITMASK TO SELECT-WSNDMSK.

      MOVE exception-send-maskTO SELECT-CHAR-STRING.
      MOVE 64 TO SELECT-BITMASK-LEN.
      CALL 'EZACIC06' USING CTOB
                            SELECT-BITMASK
                            SELECT-CHAR-STRING
                            SELECT-BITMASK-LEN
                            RETCODE.
      MOVE SELECT-BITMASK TO SELECT-ESNDMSK.

      CALL 'EZASOKET' USING SOKET-SELECTEX
                            SELECT-MAXSOC
                            SELECT-TIMEOUT
                            SELECT-RSNDMSK
                            SELECT-WSNDMSK
                            SELECT-ESNDMSK
                            SELECT-RRETMSK
                            SELECT-WRETMSK
                            SELECT-ERETMSK
                            SELECT-ECB
                            ERRNO
                            RETCODE.
```

```
      *
      * FREE THE STORAGE FOR THE ECB
      *
           EXEC CICS FREEMAIN
                DATAPOINTER(SELECT-ECB-PTR)
                END-EXEC.

      *
      * DELETE THE TS QUEUE
      *
           EXEC CICS DELETEQ TS
                QUEUE ('POSTECB@')
                END-EXEC.

           IF RETCODE < 0 THEN
               MOVE 'SELECTEX FAILED' TO MSG1
           ELSE
               MOVE 'SELECTEX PROCESSED' TO MSG1.

           MOVE SELECT-RRETMSK TO SELECT-BITMASK.
           CALL 'EZACIC06' USING BTOC
                                 SELECT-BITMASK
                                 SELECT-CHAR-STRING
                                 SELECT-BITMASK-LEN
                                 RETCODE.
           MOVE SELECT-CHAR-STRING TO read-returned-mask.

           MOVE SELECT-WRETMSK TO SELECT-BITMASK.
           CALL 'EZACIC06' USING BTOC
                                 SELECT-BITMASK
                                 SELECT-CHAR-STRING
                                 SELECT-BITMASK-LEN
                                 RETCODE.
           MOVE SELECT-CHAR-STRING TO write-returned-mask.

           MOVE SELECT-ERETMSK TO SELECT-BITMASK.
           CALL 'EZACIC06' USING BTOC
                                 SELECT-BITMASK
                                 SELECT-CHAR-STRING
                                 SELECT-BITMASK-LEN
                                 RETCODE.
           MOVE SELECT-CHAR-STRING TO exception-returned-mask.


       PROCESS-SELECTEX-EXIT.
           EXIT.

*-------------------------------------------------------------------*
* Here is the anotated SAMPLE code from a test tool used to         *
* call the subroutine used to post the ECB:                         *
*-------------------------------------------------------------------*
       WORKING-STORAGE SECTION.
       01  POST-ECB-ADDRESS               PIC 9(8) BINARY.
       01  POST-ECB-LEN                   PIC 9(4) BINARY.

       PROCEDURE DIVISION USING L1.

        PROCESS-POSTECB.

       *
       * LOOK FOR THE ADDRESS OF THE ECB IN TEMP STORAGE
       *
            MOVE 4 TO POST-ECB-LEN.
            EXEC CICS READQ TS
```

```
                          ITEM (1)
                          QUEUE ('POSTECB@')
                          INTO (POST-ECB-ADDRESS)
                          LENGTH (POST-ECB-LEN)
                    END-EXEC.

               CALL 'POSTECB' USING POST-ECB-ADDRESS
                                    RETCODE.

               IF RETCODE < 0 THEN
                   MOVE 'POSTECB FAILED'
                       TO MSG1
               ELSE
                   MOVE 'POSTECB PROCESSED'
                       TO MSG.

           PROCESS-POSTECB-EXIT.
               EXIT.

*-------------------------------------------------------------------*
* Here is a sample assembler program that can be used to post the  *
* SELECTEX ECB:                                                     *
*-------------------------------------------------------------------*
         TITLE 'POSTECB'
POSTECB  CSECT    ,            ENTRY POINT OF THIS CONTROL SECTION
POSTECB  AMODE ANY             ADDRESSING MODE...
POSTECB  RMODE ANY             RESIDENCY MODE...
         USING POSTECB,R15     USE ENTRY REGISTER AS BASE
POSTECB  MODID                 EYECATCHER INFO
         SAVE  (14,12)         SAVE THE CALLERS REGISTERS
         LR    R9,R15
         DROP  R15
         USING POSTECB,R9      USE R90 AS BASE REGISTER
         L     R12,0(R1)       LOAD ECB ADDRESS
         L     R10,0(0,R12)    LOAD CONTENTS OF ECB
         L     R12,0(0,R12)    LOAD CONTENTS OF ECB
         L     R11,NEWECB      LOAD CONTENTS OF NEW ECB
         TM    0(R12),X'80'    CHECK IF WAIT ISSUED
         BO    POST0100        IF YES, ISSUE POST MACRO
         CS    R10,R11,0(R12) IF NO, TRY QUICK POST
         BC    4,POST0100      IF UNSUCCESSFUL, ISSUE POST MACRO
         B     POST9999        RETURN TO CALLER
POST0100 DS    0H
         POST  (R12),255
POST9999 DS    0H
         RETURN (14,12)        RETURN TO CALLER
ECBADDR  DS    F
NEWECB   DC    X'400000FF'     ECB WITH POST BIT ON AND CC=255
         LTORG
         YREGS
         END
```

Appendix F. 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 578 for draft RFCs implemented in this and previous Communications Server releases.

Many features of TCP/IP Services are based on the following RFCs:

RFC	Title and Author
RFC 652	*Telnet output carriage-return disposition option* D. Crocker
RFC 653	*Telnet output horizontal tabstops option* D. Crocker
RFC 654	*Telnet output horizontal tab disposition option* D. Crocker
RFC 655	*Telnet output formfeed disposition option* D. Crocker
RFC 657	*Telnet output vertical tab disposition option* D. Crocker
RFC 658	*Telnet output linefeed disposition* D. Crocker
RFC 698	*Telnet extended ASCII option* T. Mock
RFC 726	*Remote Controlled Transmission and Echoing Telnet option* J. Postel, D. Crocker
RFC 727	*Telnet logout option* M.R. Crispin
RFC 732	*Telnet Data Entry Terminal option* J.D. Day
RFC 733	*Standard for the format of ARPA network text messages* D. Crocker, J. Vittal, K.T. Pogran, D.A. Henderson

RFC 734 *SUPDUP Protocol* M.R. Crispin

RFC 735 *Revised Telnet byte macro option* D. Crocker, R.H. Gumpertz

RFC 736 *Telnet SUPDUP option* M.R. Crispin

RFC 749 *Telnet SUPDUP—Output option* B. Greenberg

RFC 765 *File Transfer Protocol specification* J. Postel

RFC 768 *User Datagram Protocol* J. Postel

RFC 779 *Telnet send-location option* E. Killian

RFC 783 *TFTP Protocol (revision 2)* K.R. Sollins

RFC 791 *Internet Protocol* J. Postel

RFC 792 *Internet Control Message Protocol* J. Postel

RFC 793 *Transmission Control Protocol* J. Postel

RFC 820 *Assigned numbers* J. Postel

RFC 821 *Simple Mail Transfer Protocol* J. Postel

RFC 822 *Standard for the format of ARPA Internet text messages* D. Crocker

RFC 823 *DARPA Internet gateway* R. Hinden, A. Sheltzer

RFC 826 *Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware* D. Plummer

RFC 854 *Telnet Protocol Specification* J. Postel, J. Reynolds

RFC 855 *Telnet Option Specification* J. Postel, J. Reynolds

RFC 856 *Telnet Binary Transmission* J. Postel, J. Reynolds

RFC 857 *Telnet Echo Option* J. Postel, J. Reynolds

RFC 858 *Telnet Suppress Go Ahead Option* J. Postel, J. Reynolds

RFC 859 *Telnet Status Option* J. Postel, J. Reynolds

RFC 860 *Telnet Timing Mark Option* J. Postel, J. Reynolds

RFC 861 *Telnet Extended Options: List Option* J. Postel, J. Reynolds

RFC 862 *Echo Protocol* J. Postel

RFC 863 *Discard Protocol* J. Postel

RFC 864 *Character Generator Protocol* J. Postel

RFC 865 *Quote of the Day Protocol* J. Postel

RFC 868 *Time Protocol* J. Postel, K. Harrenstien

RFC 877 *Standard for the transmission of IP datagrams over public data networks* J.T. Korb

RFC 883 *Domain names: Implementation specification* P.V. Mockapetris

RFC 884 *Telnet terminal type option* M. Solomon, E. Wimmers

RFC 885 *Telnet end of record option* J. Postel

RFC 894 *Standard for the transmission of IP datagrams over Ethernet networks* C. Hornig

RFC 896 *Congestion control in IP/TCP internetworks* J. Nagle

RFC 903 *Reverse Address Resolution Protocol* R. Finlayson, T. Mann, J. Mogul, M. Theimer

RFC 904 *Exterior Gateway Protocol formal specification* D. Mills

RFC 919 *Broadcasting Internet Datagrams* J. Mogul

RFC 922 *Broadcasting Internet datagrams in the presence of subnets* J. Mogul

RFC 927 *TACACS user identification Telnet option* B.A. Anderson

RFC 933 *Output marking Telnet option* S. Silverman

RFC 946 *Telnet terminal location number option* R. Nedved

RFC 950 *Internet Standard Subnetting Procedure* J. Mogul, J. Postel

RFC 951 *Bootstrap Protocol* W.J. Croft, J. Gilmore

RFC 952 *DoD Internet host table specification* K. Harrenstien, M. Stahl, E. Feinler

RFC 959 *File Transfer Protocol* J. Postel, J.K. Reynolds

RFC 961 *Official ARPA-Internet protocols* J.K. Reynolds, J. Postel

RFC 974 *Mail routing and the domain system* C. Partridge

RFC 1001 *Protocol standard for a NetBIOS service on a TCP/UDP transport: Concepts and methods* NetBios Working Group in the Defense Advanced Research Projects Agency, Internet Activities Board, End-to-End Services Task Force

RFC 1002 *Protocol Standard for a NetBIOS service on a TCP/UDP transport: Detailed specifications* NetBios Working Group in the Defense Advanced Research Projects Agency, Internet Activities Board, End-to-End Services Task Force

RFC 1006 *ISO transport services on top of the TCP: Version 3* M.T. Rose, D.E. Cass

RFC 1009 *Requirements for Internet gateways* R. Braden, J. Postel

RFC 1011 *Official Internet protocols* J. Reynolds, J. Postel

RFC 1013 *X Window System Protocol, version 11: Alpha update April 1987* R. Scheifler

RFC 1014 *XDR: External Data Representation standard* Sun Microsystems

RFC 1027 *Using ARP to implement transparent subnet gateways* S. Carl-Mitchell, J. Quarterman

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

RFC 1044 *Internet Protocol on Network System's HYPERchannel: Protocol specification* K. Hardwick, J. Lekashman

RFC 1053 *Telnet X.3 PAD option* S. Levy, T. Jacobson

RFC 1055 *Nonstandard for transmission of IP datagrams over serial lines: SLIP* J. Romkey

RFC 1057 *RPC: Remote Procedure Call Protocol Specification: Version 2* Sun Microsystems

RFC 1058 *Routing Information Protocol* C. Hedrick

RFC 1060 *Assigned numbers* J. Reynolds, J. Postel

RFC 1067 *Simple Network Management Protocol* J.D. Case, M. Fedor, M.L. Schoffstall, J. Davin

RFC 1071 *Computing the Internet checksum* R.T. Braden, D.A. Borman, C. Partridge

RFC 1072 *TCP extensions for long-delay paths* V. Jacobson, R.T. Braden

RFC 1073 *Telnet window size option* D. Waitzman

RFC 1079 *Telnet terminal speed option* C. Hedrick

RFC 1085 *ISO presentation services on top of TCP/IP based internets* M.T. Rose

RFC 1091 *Telnet terminal-type option* J. VanBokkelen

RFC 1094 *NFS: Network File System Protocol specification* Sun Microsystems

RFC 1096 *Telnet X display location option* G. Marcy

RFC 1101 *DNS encoding of network names and other types* P. Mockapetris

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 *Hitchhikers Guide to the Internet* E. Krol

RFC 1122 *Requirements for Internet Hosts—Communication Layers* R. Braden, Ed.

RFC 1123 *Requirements for Internet Hosts—Application and Support* R. Braden, Ed.

RFC 1146 *TCP alternate checksum options* J. Zweig, C. Partridge

RFC 1155 *Structure and identification of management information for TCP/IP-based internets* M. Rose, K. McCloghrie

RFC 1156 *Management Information Base for network management of TCP/IP-based internets* K. McCloghrie, M. Rose

RFC 1157 *Simple Network Management Protocol (SNMP)* J. Case, M. Fedor, M. Schoffstall, J. Davin

RFC 1158 *Management Information Base for network management of TCP/IP-based internets: MIB-II* M. Rose

RFC 1166 *Internet numbers* S. Kirkpatrick, M.K. Stahl, M. Recker

RFC 1179 *Line printer daemon protocol* L. McLaughlin

RFC 1180 *TCP/IP tutorial* T. Socolofsky, C. Kale

RFC 1183 *New DNS RR Definitions* C.F. Everhart, L.A. Mamakos, R. Ullmann, P.V. Mockapetris

RFC 1184 *Telnet Linemode Option* D. Borman

RFC 1186 *MD4 Message Digest Algorithm* R.L. Rivest

RFC 1187 *Bulk Table Retrieval with the SNMP* M. Rose, K. McCloghrie, J. Davin

RFC 1188 *Proposed Standard for the Transmission of IP Datagrams over FDDI Networks* D. Katz

RFC 1190 *Experimental Internet Stream Protocol: Version 2 (ST-II)* C. Topolcic

RFC 1191 *Path MTU discovery* J. Mogul, S. Deering

RFC 1198 *FYI on the X window system* R. Scheifler

RFC 1207 *FYI on Questions and Answers: Answers to commonly asked "experienced Internet user" questions* G. Malkin, A. Marine, J. Reynolds

RFC 1208 *Glossary of networking terms* O. Jacobsen, D. Lynch

RFC 1213 *Management Information Base for Network Management of TCP/IP-based internets: MIB-II* K. McCloghrie, M.T. Rose

RFC 1215 *Convention for defining traps for use with the SNMP* M. Rose

RFC 1227 *SNMP MUX protocol and MIB* M.T. Rose

RFC 1228 *SNMP-DPI: Simple Network Management Protocol Distributed Program Interface* G. Carpenter, B. Wijnen

RFC 1229 *Extensions to the generic-interface MIB* K. McCloghrie

RFC 1230 *IEEE 802.4 Token Bus MIB* K. McCloghrie, R. Fox

RFC 1231 *IEEE 802.5 Token Ring MIB* K. McCloghrie, R. Fox, E. Decker

RFC 1236 *IP to X.121 address mapping for DDN* L. Morales, P. Hasse

RFC 1256 *ICMP Router Discovery Messages* S. Deering, Ed.

RFC 1267 *Border Gateway Protocol 3 (BGP-3)* K. Lougheed, Y. Rekhter

RFC 1268 *Application of the Border Gateway Protocol in the Internet* Y. Rekhter, P. Gross

RFC 1269 *Definitions of Managed Objects for the Border Gateway Protocol: Version 3* S. Willis, J. Burruss

RFC 1270 *SNMP Communications Services* F. Kastenholz, ed.

RFC 1285 *FDDI Management Information Base* J. Case

RFC 1315 *Management Information Base for Frame Relay DTEs* C. Brown, F. Baker, C. Carvalho

RFC 1321 *The MD5 Message-Digest Algorithm* R. Rivest

RFC 1323 *TCP Extensions for High Performance* V. Jacobson, R. Braden, D. Borman

RFC 1325 *FYI on Questions and Answers: Answers to Commonly Asked "New Internet User" Questions* G. Malkin, A. Marine

RFC 1327 *Mapping between X.400 (1988)/ISO 10021 and RFC 822* S. Hardcastle-Kille

RFC 1340 *Assigned Numbers* J. Reynolds, J. Postel

RFC 1344 *Implications of MIME for Internet Mail Gateways* N. Bornstein

RFC 1349 *Type of Service in the Internet Protocol Suite* P. Almquist

RFC 1350 *The TFTP Protocol (Revision 2)* K.R. Sollins

RFC 1351 *SNMP Administrative Model* J. Davin, J. Galvin, K. McCloghrie

RFC 1352 *SNMP Security Protocols* J. Galvin, K. McCloghrie, J. Davin

RFC 1353 *Definitions of Managed Objects for Administration of SNMP Parties* K. McCloghrie, J. Davin, J. Galvin

RFC 1354 *IP Forwarding Table MIB* F. Baker

RFC 1356 *Multiprotocol Interconnect on X.25 and ISDN in the Packet Mode* A. Malis, D. Robinson, R. Ullmann

RFC 1358 *Charter of the Internet Architecture Board (IAB)* L. Chapin

RFC 1363 *A Proposed Flow Specification* C. Partridge

RFC 1368 *Definition of Managed Objects for IEEE 802.3 Repeater Devices* D. McMaster, K. McCloghrie

RFC 1372 *Telnet Remote Flow Control Option* C. L. Hedrick, D. Borman

RFC 1374 *IP and ARP on HIPPI* J. Renwick, A. Nicholson

RFC 1381 *SNMP MIB Extension for X.25 LAPB* D. Throop, F. Baker

RFC 1382 *SNMP MIB Extension for the X.25 Packet Layer* D. Throop

RFC 1387 *RIP Version 2 Protocol Analysis* G. Malkin

RFC 1388 *RIP Version 2 Carrying Additional Information* G. Malkin

RFC 1389 *RIP Version 2 MIB Extensions* G. Malkin, F. Baker

RFC 1390 *Transmission of IP and ARP over FDDI Networks* D. Katz

RFC 1393 *Traceroute Using an IP Option* G. Malkin

RFC 1398 *Definitions of Managed Objects for the Ethernet-Like Interface Types* F. Kastenholz

RFC 1408 *Telnet Environment Option* D. Borman, Ed.

RFC 1413 *Identification Protocol* M. St. Johns

RFC 1416 *Telnet Authentication Option* D. Borman, ed.

RFC 1420 *SNMP over IPX* S. Bostock

RFC 1428 *Transition of Internet Mail from Just-Send-8 to 8bit-SMTP/MIME* G. Vaudreuil

RFC 1442 *Structure of Management Information for version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1443 *Textual Conventions for version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1445 *Administrative Model for version 2 of the Simple Network Management Protocol (SNMPv2)* J. Galvin, K. McCloghrie

RFC 1447 *Party MIB for version 2 of the Simple Network Management Protocol (SNMPv2)* K. McCloghrie, J. Galvin

RFC 1448 *Protocol Operations for version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1464 *Using the Domain Name System to Store Arbitrary String Attributes* R. Rosenbaum

RFC 1469 *IP Multicast over Token-Ring Local Area Networks* T. Pusateri

RFC 1483 *Multiprotocol Encapsulation over ATM Adaptation Layer 5* Juha Heinanen

RFC 1497 *BOOTP Vendor Information Extensions* J. Reynolds

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 1533 *DHCP Options and BOOTP Vendor Extensions* S. Alexander, R. Droms

RFC 1534 *Interoperation Between DHCP and BOOTP* R. Droms

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 1541 *Dynamic Host Configuration Protocol* R. Droms

RFC 1542 *Clarifications and Extensions for the Bootstrap Protocol* W. Wimer

RFC 1571 *Telnet Environment Option Interoperability Issues* D. Borman

RFC 1572 *Telnet Environment Option* S. Alexander

RFC 1573 *Evolution of the Interfaces Group of MIB-II* K. McCloghrie, F. Kastenholz

RFC 1577 *Classical IP and ARP over ATM* M. Laubach

RFC 1583 *OSPF Version 2* J. Moy

RFC 1591 *Domain Name System Structure and Delegation* J. Postel

RFC 1592 *Simple Network Management Protocol Distributed Protocol Interface Version 2.0* B. Wijnen, G. Carpenter, K. Curran, A. Sehgal, G. Waters

RFC 1594 *FYI on Questions and Answers— Answers to Commonly Asked "New Internet User" Questions* A. Marine, J. Reynolds, G. Malkin

RFC 1644 *T/TCP — TCP Extensions for Transactions Functional Specification* R. Braden

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-MIMEtransport* J. Klensin, N. Freed, M. Rose, E. Stefferud, D. Crocker

RFC 1664 *Using the Internet DNS to Distribute RFC1327 Mail Address Mapping Tables* C. Allochio, A. Bonito, B. Cole, S. Giordano, R. Hagens

RFC 1693 *An Extension to TCP: Partial Order Service* T. Connolly, P. Amer, P. Conrad

RFC 1695 *Definitions of Managed Objects for ATM Management Version 8.0 using SMIv2* M. Ahmed, K. Tesink

RFC 1701 *Generic Routing Encapsulation (GRE)* S. Hanks, T. Li, D. Farinacci, P. Traina

RFC 1702 *Generic Routing Encapsulation over IPv4 networks* S. Hanks, T. Li, D. Farinacci, P. Traina

RFC 1706 *DNS NSAP Resource Records* B. Manning, R. Colella

RFC 1712 *DNS Encoding of Geographical Location* C. Farrell, M. Schulze, S. Pleitner D. Baldoni

RFC 1713 *Tools for DNS debugging* A. Romao

RFC 1723 *RIP Version 2—Carrying Additional Information* G. Malkin

RFC 1752 *The Recommendation for the IP Next Generation Protocol* S. Bradner, A. Mankin

RFC 1766 *Tags for the Identification of Languages* H. Alvestrand

RFC 1771 *A Border Gateway Protocol 4 (BGP-4)* Y. Rekhter, T. Li

RFC 1794 *DNS Support for Load Balancing* T. Brisco

RFC 1819 *Internet Stream Protocol Version 2 (ST2) Protocol Specification—Version ST2+* L. Delgrossi, L. Berger Eds.

RFC 1826 *IP Authentication Header* R. Atkinson

RFC 1828 *IP Authentication using Keyed MD5* P. Metzger, W. Simpson

RFC 1829 *The ESP DES-CBC Transform* P. Karn, P. Metzger, W. Simpson

RFC 1830 *SMTP Service Extensions for Transmission of Large and Binary MIME Messages* G. Vaudreuil

RFC 1831 *RPC: Remote Procedure Call Protocol Specification Version 2* R. Srinivasan

RFC 1832 *XDR: External Data Representation Standard* R. Srinivasan

RFC 1833 *Binding Protocols for ONC RPC Version 2* R. Srinivasan

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 1884 *IP Version 6 Addressing Architecture* R. Hinden, S. Deering, Eds.

RFC 1886 *DNS Extensions to support IP version 6* S. Thomson, C. Huitema

RFC 1888 *OSI NSAPs and IPv6* J. Bound, B. Carpenter, D. Harrington, J. Houldsworth, A. Lloyd

RFC 1891 *SMTP Service Extension for Delivery Status Notifications* K. Moore

RFC 1892 *The Multipart/Report Content Type for the Reporting of Mail System Administrative Messages* G. Vaudreuil

RFC 1894 *An Extensible Message Format for Delivery Status Notifications*K. Moore, G. Vaudreuil

RFC 1901 *Introduction to Community-based SNMPv2* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1902 *Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1903 *Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1904 *Conformance Statements for Version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1905 *Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1906 *Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1907 *Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1908 *Coexistence between Version 1 and Version 2 of the Internet-standard Network Management Framework* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1912 *Common DNS Operational and Configuration Errors* D. Barr

RFC 1918 *Address Allocation for Private Internets* Y. Rekhter, B. Moskowitz, D. Karrenberg, G.J. de Groot, E. Lear

RFC 1928 *SOCKS Protocol Version 5* M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, L. Jones

RFC 1930 *Guidelines for creation, selection, and registration of an Autonomous System (AS)* J. Hawkinson, T. Bates

RFC 1939 *Post Office Protocol-Version 3* J. Myers, M. Rose

RFC 1981 *Path MTU Discovery for IP version 6* J. McCann, S. Deering, J. Mogul

RFC 1982 *Serial Number Arithmetic* R. Elz, R. Bush

RFC 1985 *SMTP Service Extension for Remote Message Queue Starting* J. De Winter

RFC 1995 *Incremental Zone Transfer in DNS* M. Ohta

RFC 1996 *A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)* P. Vixie

RFC 2010 *Operational Criteria for Root Name Servers* B. Manning, P. Vixie

RFC 2011 *SNMPv2 Management Information Base for the Internet Protocol using SMIv2* K. McCloghrie, Ed.

RFC 2012 *SNMPv2 Management Information Base for the Transmission Control Protocol using SMIv2* K. McCloghrie, Ed.

RFC 2013 *SNMPv2 Management Information Base for the User Datagram Protocol using SMIv2* K. McCloghrie, Ed.

RFC 2018 *TCP Selective Acknowledgement Options* M. Mathis, J. Mahdavi, S. Floyd, A. Romanow

RFC 2026 *The Internet Standards Process — Revision 3* S. Bradner

RFC 2030 *Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI* D. Mills

RFC 2033 *Local Mail Transfer Protocol* J. Myers

RFC 2034 *SMTP Service Extension for Returning Enhanced Error Codes*N. Freed

RFC 2040 *The RC5, RC5–CBC, RC-5–CBC-Pad, and RC5–CTS Algorithms*R. Baldwin, R. Rivest

RFC 2045 *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies* N. Freed, N. Borenstein

RFC 2052 *A DNS RR for specifying the location of services (DNS SRV)* A. Gulbrandsen, P. Vixie

RFC 2065 *Domain Name System Security Extensions* D. Eastlake 3rd, C. Kaufman

RFC 2066 *TELNET CHARSET Option* R. Gellens

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 2132 *DHCP Options and BOOTP Vendor Extensions* S. Alexander, R. Droms

RFC 2133 *Basic Socket Interface Extensions for IPv6* R. Gilligan, S. Thomson, J. Bound, W. Stevens

RFC 2136 *Dynamic Updates in the Domain Name System (DNS UPDATE)* P. Vixie, Ed., S. Thomson, Y. Rekhter, J. Bound

RFC 2137 *Secure Domain Name System Dynamic Update* D. Eastlake 3rd

RFC 2163 *Using the Internet DNS to Distribute MIXER Conformant Global Address Mapping (MCGAM)* C. Allocchio

RFC 2168 *Resolution of Uniform Resource Identifiers using the Domain Name System* R. Daniel, M. Mealling

RFC 2178 *OSPF Version 2* J. Moy

RFC 2181 *Clarifications to the DNS Specification* R. Elz, R. Bush

RFC 2205	*Resource ReSerVation Protocol (RSVP)—Version 1 Functional Specification* R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, S. Jamin
RFC 2210	*The Use of RSVP with IETF Integrated Services* J. Wroclawski
RFC 2211	*Specification of the Controlled-Load Network Element Service* J. Wroclawski
RFC 2212	*Specification of Guaranteed Quality of Service* S. Shenker, C. Partridge, R. Guerin
RFC 2215	*General Characterization Parameters for Integrated Service Network Elements* S. Shenker, J. Wroclawski
RFC 2217	*Telnet Com Port Control Option* G. Clarke
RFC 2219	*Use of DNS Aliases for Network Services* M. Hamilton, R. Wright
RFC 2228	*FTP Security Extensions* M. Horowitz, S. Lunt
RFC 2230	*Key Exchange Delegation Record for the DNS* R. Atkinson
RFC 2233	*The Interfaces Group MIB using SMIv2* K. McCloghrie, F. Kastenholz
RFC 2240	*A Legal Basis for Domain Name Allocation* O. Vaughn
RFC 2246	*The TLS Protocol Version 1.0* T. Dierks, C. Allen
RFC 2251	*Lightweight Directory Access Protocol (v3)* M. Wahl, T. Howes, S. Kille
RFC 2253	*Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names* M. Wahl, S. Kille, T. Howes
RFC 2254	*The String Representation of LDAP Search Filters* T. Howes
RFC 2261	*An Architecture for Describing SNMP Management Frameworks* D. Harrington, R. Presuhn, B. Wijnen
RFC 2262	*Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)* J. Case, D. Harrington, R. Presuhn, B. Wijnen
RFC 2271	*An Architecture for Describing SNMP Management Frameworks* D. Harrington, R. Presuhn, B. Wijnen
RFC 2273	*SNMPv3 Applications* D. Levi, P. Meyer, B. Stewartz
RFC 2274	*User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)* U. Blumenthal, B. Wijnen
RFC 2275	*View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)* B. Wijnen, R. Presuhn, K. McCloghrie
RFC 2279	*UTF-8, a transformation format of ISO 10646* F. Yergeau
RFC 2292	*Advanced Sockets API for IPv6* W. Stevens, M. Thomas
RFC 2308	*Negative Caching of DNS Queries (DNS NCACHE)* M. Andrews
RFC 2317	*Classless IN-ADDR.ARPA delegation* H. Eidnes, G. de Groot, P. Vixie
RFC 2320	*Definitions of Managed Objects for Classical IP and ARP Over ATM Using SMIv2 (IPOA-MIB)* M. Greene, J. Luciani, K. White, T. Kuo
RFC 2328	*OSPF Version 2* J. Moy
RFC 2345	*Domain Names and Company Name Retrieval* J. Klensin, T. Wolf, G. Oglesby

RFC 2352 *A Convention for Using Legal Names as Domain Names* O. Vaughn

RFC 2355 *TN3270 Enhancements* B. Kelly

RFC 2358 *Definitions of Managed Objects for the Ethernet-like Interface Types* J. Flick, J. Johnson

RFC 2373 *IP Version 6 Addressing Architecture* R. Hinden, S. Deering

RFC 2374 *An IPv6 Aggregatable Global Unicast Address Format* R. Hinden, M. O'Dell, S. Deering

RFC 2375 *IPv6 Multicast Address Assignments* R. Hinden, S. Deering

RFC 2385 *Protection of BGP Sessions via the TCP MD5 Signature Option*A. Hefferman

RFC 2389 *Feature negotiation mechanism for the File Transfer Protocol* P. Hethmon, R. Elz

RFC 2401 *Security Architecture for Internet Protocol* S. Kent, R. Atkinson

RFC 2402 *IP Authentication Header* S. Kent, R. Atkinson

RFC 2403 *The Use of HMAC-MD5–96 within ESP and AH* C. Madson, R. Glenn

RFC 2404 *The Use of HMAC-SHA–1–96 within ESP and AH* C. Madson, R. Glenn

RFC 2405 *The ESP DES-CBC Cipher Algorithm With Explicit IV* C. Madson, N. Doraswamy

RFC 2406 *IP Encapsulating Security Payload (ESP)* S. Kent, R. Atkinson

RFC 2407 *The Internet IP Security Domain of Interpretation for ISAKMP*D. Piper

RFC 2408 *Internet Security Association and Key Management Protocol (ISAKMP)* D. Maughan, M. Schertler, M. Schneider, J. Turner

RFC 2409 *The Internet Key Exchange (IKE)* D. Harkins, D. Carrel

RFC 2410 *The NULL Encryption Algorithm and Its Use With IPsec* R. Glenn, S. Kent,

RFC 2428 *FTP Extensions for IPv6 and NATs* M. Allman, S. Ostermann, C. Metz

RFC 2445 *Internet Calendaring and Scheduling Core Object Specification (iCalendar)* F. Dawson, D. Stenerson

RFC 2459 *Internet X.509 Public Key Infrastructure Certificate and CRL Profile* R. Housley, W. Ford, W. Polk, D. Solo

RFC 2460 *Internet Protocol, Version 6 (IPv6) Specification* S. Deering, R. Hinden

RFC 2461 *Neighbor Discovery for IP Version 6 (IPv6)* T. Narten, E. Nordmark, W. Simpson

RFC 2462 *IPv6 Stateless Address Autoconfiguration* S. Thomson, T. Narten

RFC 2463 *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification* A. Conta, S. Deering

RFC 2464 *Transmission of IPv6 Packets over Ethernet Networks* M. Crawford

RFC 2466 *Management Information Base for IP Version 6: ICMPv6 Group* D. Haskin, S. Onishi

RFC 2476 *Message Submission* R. Gellens, J. Klensin

RFC 2487 *SMTP Service Extension for Secure SMTP over TLS* P. Hoffman

RFC 2505 *Anti-Spam Recommendations for SMTP MTAs* G. Lindberg

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

RFC 2570 *Introduction to Version 3 of the Internet-standard Network Management Framework* J. Case, R. Mundy, D. Partain, B. Stewart

RFC 2571 *An Architecture for Describing SNMP Management Frameworks* B. Wijnen, D. Harrington, R. Presuhn

RFC 2572 *Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)* J. Case, D. Harrington, R. Presuhn, B. Wijnen

RFC 2573 *SNMP Applications* D. Levi, P. Meyer, B. Stewart

RFC 2574 *User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)* U. Blumenthal, B. Wijnen

RFC 2575 *View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)* B. Wijnen, R. Presuhn, K. McCloghrie

RFC 2576 *Co-Existence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework* R. Frye, D. Levi, S. Routhier, B. Wijnen

RFC 2578 *Structure of Management Information Version 2 (SMIv2)* K. McCloghrie, D. Perkins, J. Schoenwaelder

RFC 2579 *Textual Conventions for SMIv2* K. McCloghrie, D. Perkins, J. Schoenwaelder

RFC 2580 *Conformance Statements for SMIv2* K. McCloghrie, D. Perkins, J. Schoenwaelder

RFC 2581 *TCP Congestion Control* M. Allman, V. Paxson, W. Stevens

RFC 2583 *Guidelines for Next Hop Client (NHC) Developers* R. Carlson, L. Winkler

RFC 2591 *Definitions of Managed Objects for Scheduling Management Operations* D. Levi, J. Schoenwaelder

RFC 2625 *IP and ARP over Fibre Channel* M. Rajagopal, R. Bhagwat, W. Rickard

RFC 2635 *Don't SPEW A Set of Guidelines for Mass Unsolicited Mailings and Postings (spam*)* S. Hambridge, A. Lunde

RFC 2637 *Point-to-Point Tunneling Protocol* K. Hamzeh, G. Pall, W. Verthein, J. Taarud, W. Little, G. Zorn

RFC 2640 *Internationalization of the File Transfer Protocol* B. Curtin

RFC 2665 *Definitions of Managed Objects for the Ethernet-like Interface Types* J. Flick, J. Johnson

RFC 2671 *Extension Mechanisms for DNS (EDNS0)* P. Vixie

RFC 2672 *Non-Terminal DNS Name Redirection* M. Crawford

RFC 2675 *IPv6 Jumbograms* D. Borman, S. Deering, R. Hinden

RFC 2710 *Multicast Listener Discovery (MLD) for IPv6* S. Deering, W. Fenner, B. Haberman

RFC 2711 *IPv6 Router Alert Option* C. Partridge, A. Jackson

RFC 2740 *OSPF for IPv6* R. Coltun, D. Ferguson, J. Moy

RFC 2753 *A Framework for Policy-based Admission Control* R. Yavatkar, D. Pendarakis, R. Guerin

RFC 2782 *A DNS RR for specifying the location of services (DNS SRV)* A. Gubrandsen, P. Vixix, L. Esibov

RFC 2821 *Simple Mail Transfer Protocol* J. Klensin, Ed.

RFC 2822 *Internet Message Format* P. Resnick, Ed.

RFC 2840 *TELNET KERMIT OPTION* J. Altman, F. da Cruz

RFC 2845 *Secret Key Transaction Authentication for DNS (TSIG)* P. Vixie, O. Gudmundsson, D. Eastlake 3rd, B. Wellington

RFC 2851 *Textual Conventions for Internet Network Addresses* M. Daniele, B. Haberman, S. Routhier, J. Schoenwaelder

RFC 2852 *Deliver By SMTP Service Extension* D. Newman

RFC 2874 *DNS Extensions to Support IPv6 Address Aggregation and Renumbering* M. Crawford, C. Huitema

RFC 2915 *The Naming Authority Pointer (NAPTR) DNS Resource Record* M. Mealling, R. Daniel

RFC 2920 *SMTP Service Extension for Command Pipelining* N. Freed

RFC 2930 *Secret Key Establishment for DNS (TKEY RR)* D. Eastlake, 3rd

RFC 2941 *Telnet Authentication Option* T. Ts'o, ed., J. Altman

RFC 2942 *Telnet Authentication: Kerberos Version 5* T. Ts'o

RFC 2946 *Telnet Data Encryption Option* T. Ts'o

RFC 2952 *Telnet Encryption: DES 64 bit Cipher Feedback* T. Ts'o

RFC 2953 *Telnet Encryption: DES 64 bit Output Feedback* T. Ts'o

RFC 2992 *Analysis of an Equal-Cost Multi-Path Algorithm* C. Hopps

RFC 3019 *IP Version 6 Management Information Base for The Multicast Listener Discovery Protocol* B. Haberman, R. Worzella

RFC 3060 *Policy Core Information Model—Version 1 Specification* B. Moore, E. Ellesson, J. Strassner, A. Westerinen

RFC 3152 *Delegation of IP6.ARPA* R. Bush

RFC 3164 *The BSD Syslog Protocol* C. Lonvick

RFC 3291 *Textual Conventions for Internet Network Addresses* M. Daniele, B. Haberman, S. Routhier, J. Schoenwaelder

RFC 3363 *Representing Internet Protocol version 6 (IPv6) Addresses in the Domain Name System* R. Bush, A. Durand, B. Fink, O. Gudmundsson, T. Hain

| **RFC 3376** *Internet Group Management Protocol, Version 3* B. Cain, S. Deering, I. Kouvelas, B. Fenner, A. Thyagarajan

RFC 3390 *Increasing TCP's Initial Window* M. Allman, S. Floyd, C. Partridge

RFC 3410 *Introduction and Applicability Statements for Internet-Standard Management Framework* J. Case, R. Mundy, D. Partain, B. Stewart

RFC 3411 *An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks* D. Harrington, R. Presuhn, B. Wijnen

RFC 3412 *Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)* J. Case, D. Harrington, R. Presuhn, B. Wijnen

RFC 3413 *Simple Network Management Protocol (SNMP) Applications* D. Levi, P. Meyer, B. Stewart

RFC 3414 *User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)* U. Blumenthal, B. Wijnen

RFC 3415 *View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)* B. Wijnen, R. Presuhn, K. McCloghrie

RFC 3419 *Textual Conventions for Transport Addresses* M. Daniele, J. Schoenwaelder

RFC 3484 *Default Address Selection for Internet Protocol version 6 (IPv6)* R. Draves

RFC 3493 *Basic Socket Interface Extensions for IPv6* R. Gilligan, S. Thomson, J. Bound, J. McCann, W. Stevens

RFC 3513 *Internet Protocol Version 6 (IPv6) Addressing Architecture* R. Hinden, S. Deering

RFC 3526 *More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)* T. Kivinen, M. Kojo

RFC 3542 *Advanced Sockets Application Programming Interface (API) for IPv6* W. Richard Stevens, M. Thomas, E. Nordmark, T. Jinmei

| **RFC 3569** *An Overview of Source-Specific Multicast (SSM)* S. Bhattacharyya, Ed.

RFC 3584 *Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework* R. Frye, D. Levi, S. Routhier, B. Wijnen

RFC 3602 *The AES-CBC Cipher Algorithm and Its Use with IPsec* S. Frankel, R. Glenn, S. Kelly

RFC 3629 *UTF-8, a transformation format of ISO 10646* R. Kermode, C. Vicisano

RFC 3658 *Delegation Signer (DS) Resource Record (RR)* O. Gudmundsson

| **RFC 3678** *Socket Interface Extensions for Multicast Source Filters* D. Thaler, B. Fenner, B. Quinn

RFC 3715 *IPsec-Network Address Translation (NAT) Compatibility Requirements* B. Aboba, W. Dixon

RFC 3810	*Multicast Listener Discovery Version 2 (MLDv2) for IPv6* R. Vida, Ed., L. Costa, Ed.
RFC 3947	*Negotiation of NAT-Traversal in the IKE* T. Kivinen, B. Swander, A. Huttunen, V. Volpe
RFC 3948	*UDP Encapsulation of IPsec ESP Packets* A. Huttunen, B. Swander, V. Volpe, L. DiBurro, M. Stenberg
RFC 4007	*IPv6 Scoped Address Architecture* S. Deering, B. Haberman, T. Jinmei, E. Nordmark, B. Zill
RFC 4217	*Securing FTP with TLS* P. Ford-Hutchinson

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.

Draft **Title and Author**

draft-bivens-sasp-02
Server/Application State Protocol v1 A. Bivens

draft-ietf-ipngwg-icmp-v3-07
Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification A. Conta, S. Deering

draft-ietf-ipsec-esp-v3-10
IP Encapsulating Security Payload (ESP) S. Kent

draft-ietf-ipsec-rfc2402bis-11
IP Authentication Header S. Kent

draft-ietf-ipsec-rfc2401bis-06
Security Architecture for the Internet Protocol S. Kent, K. Seo

draft-ietf-ospf-ospfv3-auth-07
Authentication/Confidentiality for OSPFv3 M. Gupta, N. Melam

Appendix G. Information APARs and technotes

This appendix lists information APARs for IP and SNA documents.

Note:

1. Information APARs contain updates to previous editions of the documents listed in Table 28 and Table 29 on page 580. Documents updated for V1R9 are complete except for the updates contained in the information APARs that might be issued after V1R9 documents went to press.
2. Information APARs are predefined for z/OS V1R9 Communications Server and might not contain updates.
3. Information APARs for z/OS documents are in the document called *z/OS and z/OS.e DOC APAR and PTF ++HOLD Documentation*, which can be found at http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/ZIDOCMST/CCONTENTS.

Information APARs for IP documents

Table 28 lists information APARs for V1R6 IP documents. For releases V1R7 and later, updates are available as technotes, which can be found at http://www.ibm.com/support/docview.wss?uid=swg21178966.

Table 28. IP information APARs for z/OS Communications Server

Title	Information APAR for V1R6
New Function Summary (both IP and SNA)	II13824
Quick Reference (both IP and SNA)	II13831
IP and SNA Codes	II13842
IP Sockets API Guide	II13844
IP Configuration Guide	II13826
IP Configuration Reference	II13827
IP Diagnosis	II13836
IP Messages Volume 1	II13838
IP Messages Volume 2	II13839
IP Messages Volume 3	II13840
IP Messages Volume 4	II13841
IPv6 Network and Application Design Guide	II13825
IP Programmer’s Guide and Reference	II13843
IP User’s Guide and Commands	II13832
IP System Admininstrator’s Commands	II13833

Information APARs for SNA documents

Table 29 lists information APARs for V1R6 SNA documents. For releases V1R7 and later, updates are available as technotes, which can be found at http://www.ibm.com/support/docview.wss?uid=swg21178966.

Table 29. SNA information APARs for z/OS Communications Server

Title	Information APAR for V1R6
New Function Summary (both IP and SNA)	II13824
Quick Reference (both IP and SNA)	II13831
IP and SNA Codes	II13842
SNA Customization	II13857
SNA Diagnosis, Vol. 1: Techniques and Procedures	II13852
SNA Diagnosis, Vol. 2: FFST Dumps and the VIT	II13853
SNA Messages	II13854
SNA Network Implementation Guide	II13849
SNA Operation	II13851
SNA Programming	II13858
SNA Resource Definition Reference	II13850
SNA Data Areas Volume 1	II13855
SNA Data Areas Volume 2	II13856

Other information APARs

Table 30 lists information APARs not related to documents.

Table 30. Non-document information APARs

Content	Number
Index to APARs that list recommended VTAM maintenance	II11220
Index to APARs that list trace and dump requests for VTAM problems	II13202
Index of Communication Server IP information APARs	II12028
Collecting TCPIP CTRACEs	II12014
CSM for VTAM	II13442
CSM for TCP/IP	II13951
DLUR/DLUS	II12986, II13456, and II13783
Documentation required for FTP server problems	II12925
Documentation required for OSA/2, OSA Express and OSA QDIO	II13016
DNS — common problems and solutions	II13453
Enterprise Extender	II12223
FTP client and FTP server TLS support	II13516
FTP problems	II12079
FTPing doc to z/OS Ssupport	II12030
Generic resources	II10986
HPR	II10953

Table 30. Non-document information APARs (continued)

Content	Number
iQDIO	II13142
LPR problems	II12022
MNPS	II10370
MPC and CTC	II01501
NCPROUTE problems	II12025
OMPROUTE	II12026
PASCAL API	II11814
Performance	II11710 II11711 II11712
Resolver	II13398 II13399 II13452
Socket API	II11996 II12020
SMTP problems	II12023
SNMP	II13477 II13478
SYSLOGD howto	II12021
TCPIP connection states	II12449
TN3270E Telnet server	II11574 II13135
TN3270E Telnet server SSL common problems	II13369

Appendix H. 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/servers/eserver/zseries/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 © 1983, 1995-1997 Eric P. Allman

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.

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 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.

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

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.

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.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

Advanced Peer-to-Peer Networking	NetView
AFS	Network Station
AD/Cycle	Nways
AIX	OfficeVision
AIX/ESA	OS/2
AnyNet	OS/390
APL2	Parallel Sysplex
AS/400	PROFS
BookManager	pSeries
C/370	RACF
CICS	Redbooks
CICS/ESA	RETAIN
C Set ++	REXX
Common User Access	RISC System/6000
CUA	RMF
DB2	RS/6000
DFSMS	S/370
DFSMSdfp	S/390
DFSMShsm	S/390 Parallel Enterprise Server
DPI	SAA
ESCON	SecureWay
eServer	SET
ES/9000	SiteCheck
FFST	SP
FICON	System/360
First Failure Support Technology	System/370
GDDM	System/390
IBM	System z
ibm.com	System z9
IBMLink	Tivoli
IMS	Tivoli Enterprise Console
IMS/ESA	VM/ESA
HiperSockets	VSE/ESA
Language Environment	VTAM
Micro Channel	WebSphere
Multiprise	z9
MVS	z/Architecture
MVS/DFP	z/OS
MVS/ESA	z/VM
MVS/SP	zSeries
	400

The following terms are trademarks of other companies:

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Microsoft, Windows, and Windows NT are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

PostScript is a registered trademark of Adobe Systems Incorporated in the United States, other countries, or both.

Other company, product or service names may be trademarks or service marks of others.

Bibliography

z/OS Communications Server information

This section contains descriptions of the documents in the z/OS Communications Server library.

z/OS Communications Server documentation is available:

- Online at the z/OS Internet Library web page at http://www.ibm.com/servers/eserver/zseries/zos/bkserv
- In softcopy on CD-ROM collections. See "Softcopy information" on page xxiv.

z/OS Communications Server library

z/OS Communications Server documents are available on the CD-ROM accompanying z/OS (SK3T-4269 or SK3T-4307). Unlicensed documents can be viewed at the z/OS Internet library site.

Updates to documents are available on RETAIN® and in information APARs (info APARs). See Appendix G, "Information APARs and technotes," on page 579 for a list of the documents and the info APARs associated with them.

Info APARs for z/OS documents are in the document called *z/OS and z/OS.e DOC APAR and PTF ++HOLD Documentation* which can be found at http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/ BOOKS/ZIDOCMST/CCONTENTS.

Planning

Title	Number	Description
z/OS Communications Server: New Function Summary	GC31-8771	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.
z/OS Communications Server: IPv6 Network and Application Design Guide	SC31-8885	This document is a high-level introduction to IPv6. It describes concepts of z/OS Communications Server's support of IPv6, coexistence with IPv4, and migration issues.

Resource definition, configuration, and tuning

Title	Number	Description
z/OS Communications Server: IP Configuration Guide	SC31-8775	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*.

Title	Number	Description
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

Title	Number	Description
z/OS Communications Server: IP User's Guide and Commands	SC31-8780	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.
z/OS Communications Server: IP System Administrator's Commands	SC31-8781	This document describes the functions and commands helpful in configuring or monitoring your system. It contains system administrator's commands, such as TSO NETSTAT, PING, TRACERTE and their UNIX counterparts. It also includes TSO and MVS commands commonly used during the IP configuration process.
z/OS Communications Server: SNA Operation	SC31-8779	This document serves as a reference for programmers and operators requiring detailed information about specific operator commands.
z/OS Communications Server: Quick Reference	SX75-0124	This document contains essential information about SNA and IP commands.

Customization

Title	Number	Description
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

Title	Number	Description
z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference	SC31-8788	This document describes the syntax and semantics of program source code necessary to write your own application programming interface (API) into TCP/IP. You can use this interface as the communication base for writing your own client or server application. You can also use this document to adapt your existing applications to communicate with each other using sockets over TCP/IP.
z/OS Communications Server: IP CICS Sockets Guide	SC31-8807	This document is for programmers who want to set up, write application programs for, and diagnose problems with the socket interface for CICS using z/OS TCP/IP.
z/OS Communications Server: IP IMS Sockets Guide	SC31-8830	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.
z/OS Communications Server: IP Programmer's Guide and Reference	SC31-8787	This document describes the syntax and semantics of a set of high-level application functions that you can use to program your own applications in a TCP/IP environment. These functions provide support for application facilities, such as user authentication, distributed databases, distributed processing, network management, and device sharing. Familiarity with the z/OS operating system, TCP/IP protocols, and IBM Time Sharing Option (TSO) is recommended.
z/OS Communications Server: SNA Programming	SC31-8829	This document describes how to use SNA macroinstructions to send data to and receive data from (1) a terminal in either the same or a different domain, or (2) another application program in either the same or a different domain.
z/OS Communications Server: SNA Programmer's LU 6.2 Guide	SC31-8811	This document describes how to use the SNA LU 6.2 application programming interface for host application programs. This document applies to programs that use only LU 6.2 sessions or that use LU 6.2 sessions along with other session types. (Only LU 6.2 sessions are covered in this document.)
z/OS Communications Server: SNA Programmer's LU 6.2 Reference	SC31-8810	This document provides reference material for the SNA LU 6.2 programming interface for host application programs.
z/OS Communications Server: CSM Guide	SC31-8808	This document describes how applications use the communications storage manager.

Title	Number	Description
z/OS Communications Server: CMIP Services and Topology Agent Guide	SC31-8828	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.

Diagnosis

Title	Number	Description
z/OS Communications Server: IP Diagnosis Guide	GC31-8782	This document explains how to diagnose TCP/IP problems and how to determine whether a specific problem is in the TCP/IP product code. It explains how to gather information for and describe problems to the IBM Software Support Center.
z/OS Communications Server: SNA Diagnosis Vol 1, Techniques and Procedures and *z/OS Communications Server: SNA Diagnosis Vol 2, FFST Dumps and the VIT*	GC31-6850 GC31-6851	These documents help you identify an SNA problem, classify it, and collect information about it before you call the IBM Support Center. The information collected includes traces, dumps, and other problem documentation.
z/OS Communications Server: SNA Data Areas Volume 1 and *z/OS Communications Server: SNA Data Areas Volume 2*	GC31-6852 GC31-6853	These documents describe SNA data areas and can be used to read an SNA dump. They are intended for IBM programming service representatives and customer personnel who are diagnosing problems with SNA.

Messages and codes

Title	Number	Description
z/OS Communications Server: SNA Messages	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

Special characters

A

B

C

D

E

F

G

H

I

J

K

L

M

N

T

U

V

W

Z

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.

Program Number: 5694–A01

Printed in USA

SC31-8807-04

Spine information:

IBM z/OS Communications Server

z/OS V1R9.0 Comm Svr: IP CICS Sockets Guide

Version 1
Release 9