z/OS Communications Server

IP CICS Sockets Guide

Version 1 Release 5

SC31-8807-02

z/OS Communications Server

IP CICS Sockets Guide

Version 1 Release 5

SC31-8807-02

Note:
Before using this information and the product it supports, be sure to read the general information under "Notices" on page 503.

Third edition (March 2004)

This edition applies to Version 1 Release 5 of z/OS (5694-A01) and Version 1 Release 5 of z/OS.e (5655-G52) and to all subsequent releases and modifications until otherwise indicated in new editions.

Publications are not stocked at the address given below. If you want more IBM publications, ask your IBM representative or write to the IBM branch office serving your locality.

A form for your comments is provided at the back of this document. If the form has been removed, you may address comments to:
IBM Corporation
G7IA 7G Globalization Services
Department QSJA/500/2
P.O. Box 12195, 3039 Cornwallis Road
Research Triangle Park, NC 27709-2195
U.S.A.

If you prefer to send comments electronically, use one of the following methods:

Fax (USA and Canada):
1-800-254-0206

Internet e-mail:
usib2hpd@vnet.ibm.com

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

IBMLink™:
CIBMORCF at RALVM17

IBM Mail Exchange:
tkinlaw@us.ibm.com

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.

This document supports z/OS.e™.

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 or COBOL programming languages. Since the CICS transaction processing system is a prerequisite for CICS TCP/IP, the document assumes the reader is also familiar with CICS.

How this document is organized

This book contains the following chapters and appendixes:

- 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 21 describes the steps required to configure CICS TCP/IP.
- Chapter 3, "Configuring the CICS Domain Name System cache", on page 71 describes how to configure the CICS domain name server cache.
- Chapter 4, "Starting and stopping CICS sockets", on page 81 explains how to start and stop (enable and disable) CICS TCP/IP.
- Chapter 5, "Writing your own Listener", on page 89 discusses writing your own Listener.
- Chapter 6, "Application programming guide", on page 95 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 119 describes the C language API provided by CICS TCP/IP.
- Chapter 8, "Sockets extended application programming interface (API)", on page 173 describes the sockets extended API.
- Appendix A, "Original COBOL application programming interface (EZACICAL)", on page 307 describes the EZACICAL API.
- Appendix B, "Return codes", on page 337 describes system-wide message numbers and codes set by the system calls.

- Appendix C, “GETSOCKOPT/SETSOCKOPT command values”, on page 353 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 355 contains CICS socket interface messages.
- Appendix E, “Sample programs”, on page 387 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 (RFCs)”, on page 487 lists the related protocol specifications for TCP/IP.
- Appendix G, “Information APARs”, on page 497 lists information APARs for IP and SNA documents.
- Appendix H, “Accessibility”, on page 501 describes accessibility features to help users with physical disabilities.
- “Notices” on page 503 contains notices and trademarks used in this document.
- “Bibliography” on page 513 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 if 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 filename (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 527.

Conventions and terminology used in this document

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

Recommendation
Something you should do; an endorsement

Guideline
Customary way to perform a procedure; stronger request than recommendation

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

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 513, 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"
- "Redbooks" on page xix
- "Where to find related information on the Internet" on page xix
- "Accessing z/OS licensed documents on the Internet" on page xx
- "Using LookAt to look up message explanations" on page xxi

Softcopy information: Softcopy publications are available in the following collections:

Titles	Order Number	Description
z/OS V1R5 Collection	SK3T-4269	This is the CD collection shipped with the z/OS product. It includes the libraries for z/OS V1R5, 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 V1R5 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.
System Center Publication IBM S/390® Redbooks™ Collection	SK2T-2177	This collection contains over 300 ITSO redbooks that apply to the S/390 platform and to host networking arranged into subject bookshelves.

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 may be helpful to readers.

Title	Number
z/OS Integrated Security Services Firewall Technologies	SC24-5922
S/390: OSA-Express Customer's Guide and Reference	SA22-7403
z/OS JES2 Initialization and Tuning Guide	SA22-7532
z/OS MVS Diagnosis: Procedures	GA22-7587
z/OS MVS Diagnosis: Reference	GA22-7588
z/OS MVS Diagnosis: Tools and Service Aids	GA22-7589
z/OS Integrated Security Services LDAP Client Programming	SC24-5924

Title	**Number**
z/OS Integrated Security Services LDAP Server Administration and Use	SC24-5923
Understanding LDAP	SG24-4986
z/OS UNIX System Services Programming: Assembler Callable Services Reference	SA22-7803
z/OS UNIX System Services Command Reference	SA22-7802
z/OS UNIX System Services User's Guide	SA22-7801
z/OS UNIX System Services Planning	GA22-7800
z/OS MVS Using the Subsystem Interface	SA22-7642
z/OS C/C++ Run-Time Library Reference	SA22-7821
z/OS Program Directory	GI10-0670
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
TCP/IP Tutorial and Technical Overview	GG24-3376
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
z/OS System Secure Sockets Layer Programming	SC24-5901
SNA Formats	GA27-3136

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

Title	**Number**
TCP/IP Tutorial and Technical Overview	GG24–3376
SNA and TCP/IP Integration	SG24–5291
IBM Communications Server for OS/390® V2R10 TCP/IP Implementation Guide: Volume 1: Configuration and Routing	SG24–5227
IBM Communications Server for OS/390 V2R10 TCP/IP Implementation Guide: Volume 2: UNIX Applications	SG24–5228
IBM Communications Server for OS/390 V2R7 TCP/IP Implementation Guide: Volume 3: MVS Applications	SG24–5229
Secureway Communications Server for OS/390 V2R8 TCP/IP: Guide to Enhancements	SG24–5631
TCP/IP in a Sysplex	SG24–5235
Managing OS/390 TCP/IP with SNMP	SG24–5866
Security in OS/390–based TCP/IP Networks	SG24–5383
IP Network Design Guide	SG24–2580
Migrating Subarea Networks to an IP Infrastructure	SG24–5957
IBM Communication Controller Migration Guide	SG24–6298

Where to find related information on the Internet:

z/OS

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

z/OS Internet Library
- http://www.ibm.com/servers/eserver/zseries/zos/bkserv/

IBM Communications Server product
- http://www.software.ibm.com/network/commserver/

IBM Communications Server product support
- http://www.software.ibm.com/network/commserver/support/

IBM Systems Center publications
- http://www.redbooks.ibm.com/

IBM Systems Center flashes
- http://www-1.ibm.com/support/techdocs/atsmastr.nsf

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

Internet drafts
- http://www.ietf.org/ID.html

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

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

USENET news groups:
comp.protocols.dns.bind

For BIND mailing lists, see:

- 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 (Note: This list may 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.

For definitions of the terms and abbreviations used in this document, you can view or download the latest *IBM Glossary of Computing Terms* at the following Web address:

http://www.ibm.com/ibm/terminology

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.

Accessing z/OS licensed documents on the Internet: z/OS licensed documentation is available on the Internet in PDF format at the IBM Resource Link™ Web site at:

http://www.ibm.com/servers/resourcelink

Licensed documents are available only to customers with a z/OS license. Access to these documents requires an IBM Resource Link user ID and password, and a key code. With your z/OS order you received a Memo to Licensees, (GI10-0671), that includes this key code.

To obtain your IBM Resource Link user ID and password, log on to:

http://www.ibm.com/servers/resourcelink

To register for access to the z/OS licensed documents:

1. Sign in to Resource Link using your Resource Link user ID and password.
2. Select **User Profiles** located on the left-hand navigation bar.

Note: You cannot access the z/OS licensed documents unless you have registered for access to them and received an e-mail confirmation informing you that your request has been processed.

Printed licensed documents are not available from IBM.

You can use the PDF format on either **z/OS Licensed Product Library CD-ROM** or IBM Resource Link to print licensed documents.

Using LookAt to look up message explanations: LookAt is an online facility that allows you to look up explanations for most 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 access LookAt from the Internet at:

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

or from anywhere in z/OS where you can access a TSO/E command line (for example, TSO/E prompt, ISPF, z/OS UNIX System Services running OMVS). You can also download code from the *z/OS Collection* (SK3T-4269) and the LookAt Web site that will allow you to access LookAt from a handheld computer (Palm Pilot VIIx suggested).

To use LookAt as a TSO/E command, you must have LookAt installed on your host system. You can obtain the LookAt code for TSO/E from a disk on your *z/OS Collection* (SK3T-4269) or from the **News** section on the LookAt Web site.

Some messages have information in more than one document. For those messages, LookAt displays a list of documents in which the message appears.

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 usib2hpd@vnet.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 page number or table number).
- Fill out one of the forms at the back of this document and return it by mail, by fax, or by giving it to an IBM representative.

Summary of changes

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 117, "EZACIC14" on page 303, and "EZACIC15" on page 305)
- A section 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 27)
- File definitions for EZACACHE (see "File definitions" on page 30)
- IPv6 examples for EZACICD (see "Building the configuration data set with EZACICD" on page 41 and "JCL for the configuration macro" on page 48)
- Recommendations for CICS DNS Caching and DNS/WLM support (see Chapter 3, "Configuring the CICS Domain Name System cache", on page 71)
- IPv6 information throughout the section on socket addresses (see "Socket addresses" on page 102)
- IPv6 information and examples throughout the section on Listener output format (see "Listener output format" on page 108)
- IPv6 information on Listener configuration (see "Writing your own security/transaction link module for the Listener" on page 114)
- Information on C structures (see Table 13 on page 122)
- IPv6 information throughout the section on C socket calls (see "C socket calls" on page 124)
- A section on address testing macros (see "Address Testing Macros" on page 171)
- IPv6 information throughout the section on code call instructions (see "Code CALL instructions" on page 176)
- EZACIC09 for TCP/IP bit string processing (see "Bit string processing" on page 291 and "EZACIC09" on page 299)
- EZACICAC, EZACICAS, EZACIC6C, and EZACIC6S sample programs (see "EZACICAC" on page 456, "EZACICAS" on page 466, "EZACIC6C" on page 415, and "EZACIC6S" on page 428)
- CICS sockets messages (see "EZY1218—EZY1352" on page 355)

Changed information

- Information on "Using IBM's environmental support" on page 89
- Information throughout the section on code call instructions (see "Code CALL instructions" on page 176)

- CICS resource definition information and examples (see “CICS — Defining CICS TCP/IP resources” on page 22)
- Information about Monitor Control Table entries (see “CICS monitoring” on page 33)
- EZAC and EZAO transaction screens (see “Configuration transaction (EZAC)” on page 52 and “Starting/stopping CICS TCP/IP manually” on page 82)
- Information on automatically starting and stopping CICS TCP/IP (see “Starting/stopping CICS TCP/IP automatically” on page 81 and “CICS program list table (PLT)” on page 37)
- CICS Sockets environment configuration file information throughout “Configuring the CICS TCP/IP environment” on page 41
- The description of TERMAPI has been updated at “TERMAPI” on page 286
- The description of the *max_sock*, MAXSOC, and MAX-SOCK parameters (see “Parameters” on page 157, “Parameter values set by the application” on page 227, and “Parameter values to be set by the application” on page 321)
- 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 section on GIVESOCKET and TAKESOCKET calls (see “GIVESOCKET and TAKESOCKET calls” on page 16)
- The section on conversion routines (see “Conversion routines” on page 19)
- Call for the client application (see Table 5 on page 97)
- IPv6 information about EZACICAL (see Appendix A, “Original COBOL application programming interface (EZACICAL)”, on page 307)
- CICS sockets messages (see “EZY1218—EZY1352” on page 355)
- EZACICSC and EZACICSS sample programs (see “EZACICSC” on page 387 and “EZACICSS” on page 395)

Deleted information:

- The SIOCADDRT, SIOCDELRT, SIOCGIFFLAGS, SIOCGIFMETRIC, SIOCGIFNETMASK, SIOCSIFDSTADDR, SIOCSIFFLAGS, and SIOCSIFMETRIC parameters (see “ioctl()” on page 157)

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 may 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 SC31-8807-01
z/OS Version 1 Release 4

This document contains information previously presented in SC31-8807-00, which supports z/OS Version 1 Release 2.

New information

- Call instructions GETSOCKOPT and SETSOCKOPT have been updated. For details, see "GETSOCKOPT" on page 213 and "SETSOCKOPT" on page 271.
- Call instruction INITAPI has been updated to include INITAPIX. For details, see "INITAPI and INITAPIX" on page 226.
- CICS sockets message EZY1348E has been added. For details, see Appendix D, "CICS sockets messages", on page 355.

An appendix with z/OS product accessibility information has been added.

Changed information

- The modifications required in the CICS startup job have been updated. For details, see Figure 8 on page 22.
- The Monitor Control Table for TRUE has been updated. For details, see Figure 34 on page 34.
- The Monitor Control Table for Listener has been updated. For details, see Figure 35 on page 36.
- The call instruction examples have changed for the following call instructions. For details, see their specific sections in "Code CALL instructions" on page 176.
  - IOCTL
  - RECV
  - RECVFROM
  - RECVMSG
  - SEND
  - SENDMSG
  - SENDTO
  - SHUTDOWN
  - SOCKET

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 may 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.

This document supports z/OS.e.

Summary of changes
for SC31-8807-00
z/OS Version 1 Release 2

This document contains information previously presented in *OS/390 V2R8 SecureWay® Communications Server: IP CICS Sockets Guide*.

New information

- The CICS Sockets Interface has been updated to allow configuration of an enhanced version of the CICS Listener, as well as the standard version previously supplied. For details, see Chapter 2, "Setting up and configuring CICS TCP/IP", on page 21.

- The TCP_NODELAY option is now available to disable the Nagle algorithm to improve response time. For details, see “getsockopt(), setsockopt()” on page 146, “GETSOCKOPT” on page 213, and “SETSOCKOPT” on page 271.

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.

Chapter 1. Introduction to CICS TCP/IP

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

CICS is an online transaction processing system. This means that 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.

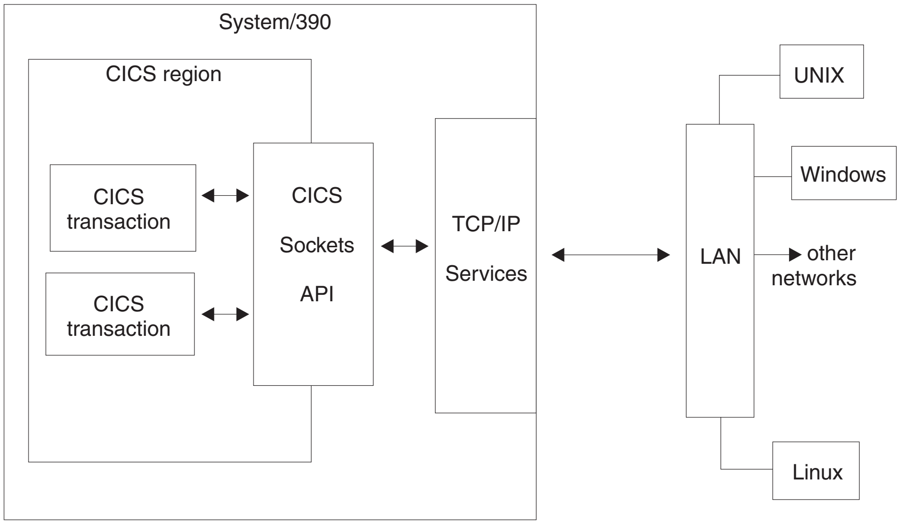


Figure 1. The use of CICS sockets

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 section describes some of the basic ideas behind the TCP/IP family of protocols. For more detailed and comprehensive treatments of this subject, refer to the documents on TCP/IP listed in “z/OS Communications Server information” on page 513.

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 will be 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.

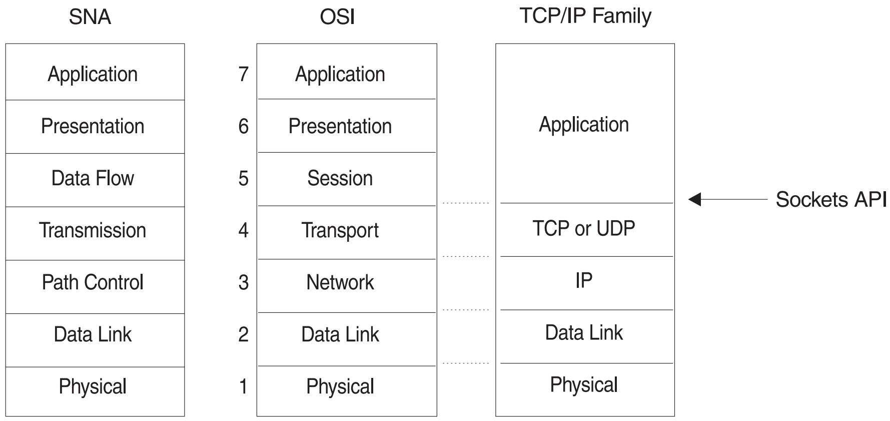


Figure 2. TCP/IP protocols compared to the OSI model and SNA

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 119 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 application programming interface (API)", on page 173 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 307.

Programming with sockets

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

Socket types

The MVS socket APIs provide a standard interface to the transport and Internetwork layer interfaces of TCP/IP. They support three socket types: stream, datagram, and raw. Stream and datagram sockets interface to the transport layer protocols, and raw sockets interface to the network layer protocols. All three socket types are discussed here for background purposes. 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 section 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. Refer to the API section of *z/OS Communications Server: IPv6 Network and Application Design Guide* for more information on 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.
             88 SOCK-FAMILY-IS-AFINET6   VALUE 19.
```

[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*.

```
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.
```

You will find the IPv4 or IPv6 socket address structure in every call that addresses another TCP/IP host.

This structure contains the followinng 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 will be 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 *fec0:129:5:25::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 may 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 FEC0::206:2AFF:FE66:C800 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 FEC0::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.

Note that 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 it can be done using the SO_REUSADDR 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 will help you convert from one notation to another.

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

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

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

A typical client server program flow chart

Stream-oriented socket programs generally follow a prescribed sequence. See Figure 3 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.

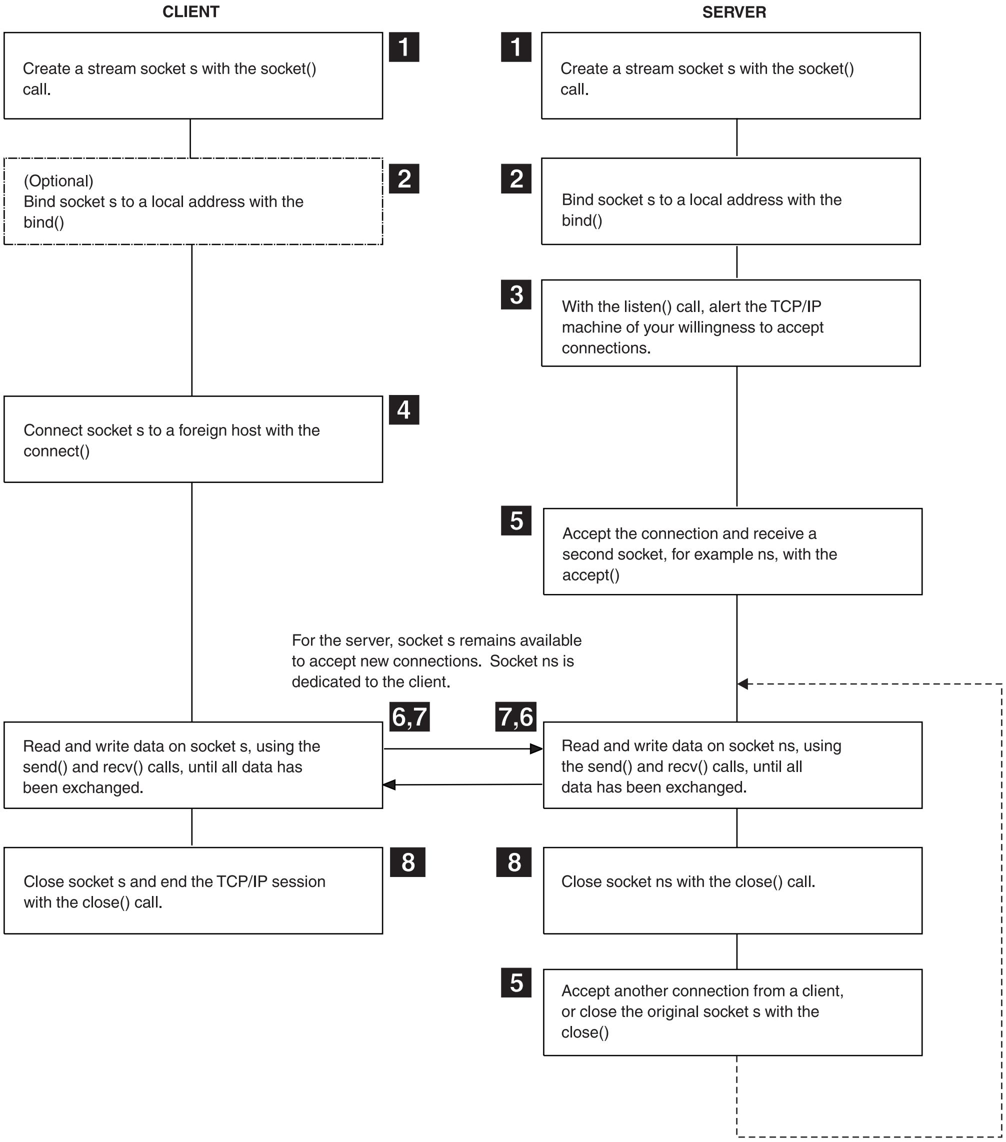

CLIENT
SERVER
1
Create a stream socket s with the socket() call.
1
Create a stream socket s with the socket() call.
2
(Optional) Bind socket s to a local address with the bind()
2
Bind socket s to a local address with the bind()
3
With the listen() call, alert the TCP/IP machine of your willingness to accept connections.
4
Connect socket s to a foreign host with the connect()
5
Accept the connection and receive a second socket, for example ns, with the accept()
For the server, socket s remains available to accept new connections. Socket ns is dedicated to the client.
6,7
7,6
Read and write data on socket s, using the send() and recv() calls, until all data has been exchanged.
Read and write data on socket ns, using the send() and recv() calls, until all data has been exchanged.
8
Close socket s and end the TCP/IP session with the close() call.
8
Close socket ns with the close() call.
5
Accept another connection from a client, or close the original socket s with the close()

Figure 3. A typical client server session

Concurrent and iterative servers

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

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

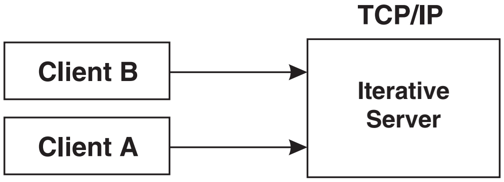


Figure 4. An iterative server

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 Listener" on page 106.

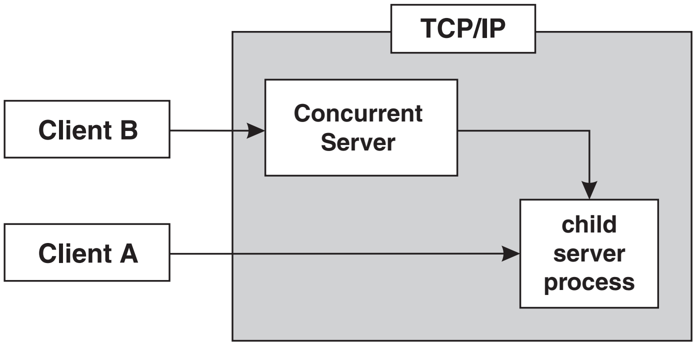


Figure 5. A concurrent server

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 application programming interface (API)", on page 173.

Server TCP/IP calls

To understand Socket programming, the client program and the server program must be considered separately. In this section the call sequence for the *server* is described; the next section discusses the typical call sequence for a *client*. This is the logical presentation sequence because the server is usually already in execution before the client is started. The step numbers (such as 5) in this section 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 283.

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 wishes to provide its service.

- The IP address of the network connection on which the server is listening. If the application wants to receive connection requests from any network interface, the IP address should be set to zeros specifying inaddr_any for IPv4 or in6addr_any for IPv6.

For an example of the BIND call, see “BIND” on page 179.

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 will be queued.

For an example of the LISTEN call, see “LISTEN” on page 235.

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 176.

[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

Once a client has been connected with the server, and the socket has been transferred from the main task (parent) to the subtask (child), the client and server exchange application data, using various forms of READ/WRITE calls. See "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 will communicate.

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

See "SOCKET" on page 283 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 183 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, or will contain, the data (BUF), and the amount of data transferred (NBYTE). The server uses the socket that is returned from the ACCEPT call.

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

- For an example of the READ call, see “READ” on page 240.
- For an example of the WRITE call, see “WRITE” on page 287.

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 182.

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 will be tested, and for what. When the Select call is issued, it blocks until the specified sockets are ready to be serviced (or, optionally) until a timer expires. When the select call returns, the program must check to see which sockets require service, and then process them.

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

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

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

The first rule is that the length of a bit string is always expressed as a number of fullwords. If the highest socket descriptor you want to test is socket descriptor 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

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 5	55	54	53	52	51	50	49	48
Byte 6	47	46	45	44	43	42	41	40
Byte 7	39	38	37	36	35	34	33	32

If you develop your program in COBOL or PL/I, you may find that the EZACIC06 routine, which is provided as part of TCP/IP Services, will make it easier for you to build and test these bit strings. This routine translates between a character string mask (one byte per socket) and a bit string mask (one bit per socket).

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

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

IOCTL and FCNTL calls

In addition to SELECT, applications can use the IOCTL or FCNTL calls to help perform asynchronous (nonblocking) socket operations. An example of the use of the IOCTL call is shown in "IOCTL" on page 228.

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

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

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

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 will use. When TAKESOCKET accepts the socket that has been given, the TAKESOCKET call assigns a new socket number for the subtask to use. This new socket number represents the same connection as the parent's socket. (The transferred socket might be referred to as socket number 54 by the parent task and as socket number 3 by the subtask; however, both socket descriptors represent the same connection.)

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 will be 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 Listener" on page 106. Figure 5 on page 9 shows a concurrent server.

What you must have to run CICS TCP/IP

In order to use the updates described in this document, you must have OS/390 V2R5 or later.

TCP/IP Services is not described in this document since 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. For more information about TCP/IP Services, see the documents listed in “z/OS Communications Server information” on page 513.

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 OS/2® 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** enables programs to pass calls to the subtask and to the TCP/IP address space.
- 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. It also handles the MVS waits incurred during socket calls.
- 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`
Read/Write calls:	`READ`, `READV`, `RECV`, `RECVFROM`, `RECVMSG`, `SEND`, `SENDMSG`, `SENDTO`, `WRITE`, `WRITEV`

Call type	IP CICS TCP API function
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

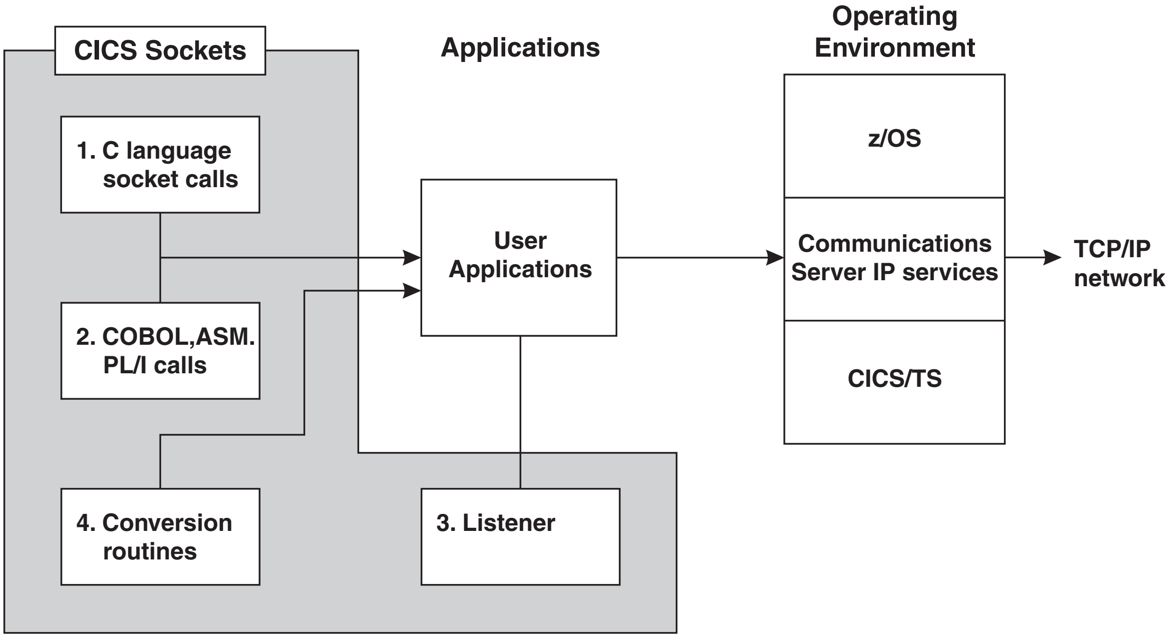


Figure 7. How user applications access TCP/IP networks with CICS TCP/IP (run-time environment)

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 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. Refer to *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, used to convert EBCDIC data to the ASCII format used in TCP/IP networks and workstations. It 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 303.
- A corresponding alternate ASCII-to-EBCDIC conversion routine, EZACIC15, which uses the translation table listed in "EZACIC15" on page 305.
- A module that converts COBOL character arrays into bit-mask arrays used in TCP/IP. This module, which is run by calling EZACIC06, is used with the socket SELECT call.
- A special 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 special 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. Refer to *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 119 or Chapter 8, "Sockets extended application programming interface (API)", on page 173 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 will be issued if the child server's TAKESOCKET is AF_INET. If the Listener is defined as an INET Listener, the EBADF errno will be 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
AF_INET6	AF_INET6	AF_INET6	zeros	IPv6 addr	zeros	IPv6 addr

Chapter 2. Setting up and configuring CICS TCP/IP

This chapter 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 RDO and the CICS resource management utility DFHCSDUP commands.	"CICS — Defining CICS TCP/IP resources" on page 22
Modify TCP/IP Services data sets.	"TCP/IP services — Modifying data sets" on page 39
Use the configuration macro (EZACICD), to build the TCP Configuration data set.	"Building the configuration data set with EZACICD" on page 41
Use the configuration transaction (EZAC) to customize the Configuration data set.	"Customizing the configuration data set" on page 52
Note: You can modify the data set while CICS is running by using EZAC. See "Configuration transaction (EZAC)" on page 52.	

MVS JCL — Modifying CICS startup

Figure 8 on page 22 illustrates the modifications required in the CICS startup job stream to enable CICS TCP/IP startup. The modifications are highlighted.

```
//SERVA JOB (999,POK),'JOHN DOE',CLASS=A,MSGCLASS=T,
//       NOTIFY=&SYSUID,MSGLEVEL=(1,1)
//CICS    EXEC PGM=DFHSIP,REGION=32M,TIME=1440,
//       PARM=SYSIN
//SYSIN     DD *
SIT=6$,
START=AUTO,
DCT=IP,
GRPLIST=TCPLIST,
GMTEXT='   WELCOME TO CICS TRANSACTION SERVER WITH TCP/IP SOCKETS INTERFACE',
APPLID=SCMCICSA
.END
//DFHXRCTL  DD DISP=SHR,DSN=hlq.CNTL.CICS.DFHXRCTL
//STEPLIB   DD DISP=SHR,DSN=hlq.SDFHAUTH
//          DD DISP=SHR,DSN=SYS1.CSSLIB
//          DD DISP=SHR,DSN=SYS1.COBOL.V1R3M2.COB2CICS
//          DD DISP=SHR,DSN=COBOL.V1R3M2.COB2LIB
//          DD DISP=SHR,DSN=hlq.SEZALOAD   1
//DFHRPL    DD DISP=SHR,DSN=hlq.SDFHLOAD
//          DD DISP=SHR,DSN=hlq.SEZATCP    2
//          DD DISP=SHR,DSN=SYS1.CSSLIB
//          DD DISP=SHR,DSN=SYS1.COBOL.V1R3M2.COB2CICS
//          DD DISP=SHR,DSN=COBOL.V1R3M2.COB2LIB
//DFHINTRA  DD DISP=SHR,DSN=hlq.CNTL.CICS.DFHINTRA
//LOGUSR    DD SYSOUT=*,DCB=(DSORG=PS,RECFM=V,BLKSIZE=136)
//MSGUSR    DD SYSOUT=*,DCB=(DSORG=PS,RECFM=V,BLKSIZE=136)
//TCPDATA   DD SYSOUT=*,DCB=(DSORG=PS,RECFM=V,BLKSIZE=136)  3
//SYSTCPD   DD DSN=hlq.SEZAINST(TCPDATA),DISP=SHR  4
⋮
```

Figure 8. JCL for CICS startup with the TCP/IP socket interface

These are the required alterations to the startup of CICS:

1. You must concatenate the data set *hlq*.SEZALOAD to STEPLIB. This data set contains CICS TCP/IP module EZACIC03.
2. You must concatenate the data set *hlq*.SEZATCP to DFHRPL. This data set contains all the other CICS TCP/IP modules. [6]
3. You can add a TCPDATA entry for the output messages from CICS TCP/IP (see "Transient data definition" on page 32).
4. SYSTCPD explicitly identifies which data set is to be used to obtain the parameters defined by TCPIP.DATA, which describes the stack you want to use if there are multiple TCP/IP stacks running.[6]

CICS — Defining CICS TCP/IP resources

The following definitions must be made in CICS:

- Transactions
- Programs (see "Program definitions" on page 24)
- BMS mapset (EZACICM, shown in Figure 23 on page 27)
- Files (see "File definitions" on page 30)
- Transient data queues (see "Transient data definition" on page 32)

In order to ensure that the CICS CSD contains all necessary socket-related resource definitions, a CSD upgrade (DFHCSDUP) using member EZACICCT in *hlq*.SEZAINST should be executed. Refer to *CICS Resource Definition Guide* for information on DFHCSDUP.

[6] TCP/IP Services 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.

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.

Requirement: The user ID invoking the EZAO transaction to activate or deactivate the IP CICS Sockets Interface requires the UPDATE access to the EXITPROGRAM security class. The user ID invoking the EZAC transaction requires the UPDATE access to the EXITPROGRAM security class to allow the EZAC transaction to perform an IPv6 runtime check when the AF is changed to INET6. Failure to have at least the UPDATE access to the EXITPROGRAM security class will cause the IP CICS Sockets Interface and Listener to not start or not stop.

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 24, Figure 11 on page 24, and Figure 12 on page 24 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 may 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 Sockets Interface requires the UPDATE access to the EXITPROGRAM security class. The user ID invoking the EZAC transaction requires the UPDATE access to the EXITPROGRAM security class to allow the EZAC transaction to perform an IPv6 runtime check when the AF is changed to INET6. Failure to have at least the UPDATE access to the EXITPROGRAM security class will cause the IP CICS Sockets 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 12 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/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)
```

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)
```

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)
```

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 *hlq*.SEZAINST:

EZACICSC
: A sample IPv4 child server that works with the IPv4 Listener (EZACIC02). See "EZACICSC" on page 387.

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 395.

EZACIC6C

A sample IPv6 child server that works with either a standard or enhanced IPv6 Listener (EZACIC02). See “EZACIC6C” on page 415.

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 428.

EZACICAC

A sample assembler child server that works with either a standard or enhanced, IPv4 or IPv6 Listener (EZACIC02). See “EZACICAC” on page 456.

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 466.

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

```
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)
```

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)
```

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)
```

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 *hlq*.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 *hlq*.SEZALOAD.

EZACIC07

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

EZACIC17

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

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:

```
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) 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 31. DFHCSDUP commands to define EZACONFG

Notes:

1. Choose a DSName to fit installation standards.
2. If it is desired 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 will not be accessible from the AOR.
3. If it is desired 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 YES. The FOR will therefore be the only CICS region that can open EZACONFG. Thus, no sharing of EZACONFG between different CICS regions will be possible.

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 will be 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 may outweigh the benefits of connection balancing.

  Refer to *z/OS Communications Server: IP Configuration Reference* for information on caching issues.
- DNS/WLM will continue to support CICS Listeners wanting to participate in work load balancing for IPv4 clients. IPv6 enabled Listeners will still be 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 will 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 32. 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 datatables, see *CICS Resource Definition Guide*.
6. Maxnumrecs equals the maximum number of destinations queried.

Transient data definition

Figure 33 shows the DFHCSDUP commands required to define the TCPM transient data queue for CICS TCP/IP. For more information on DFHCSDUP commands, refer to *CICS Resource Definition Guide*.

Note that the destination TCPM can be changed. If so, it must match the name specified in the ERRORTD parameter of the EZAC DEFINE CICS, the EZACICD TYPE=CICS, or both (refer to “Building the configuration data set with EZACICD” on page 41).

```
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 33. 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 33.

The CICS startup JCL must include a DD statement for the extrapartition transient data queue being defined (as in Figure 8 on page 22, line **3**).

The Listener transaction can start a server using a transient data queue, as described in “Listener input format” on page 107. The intrapartition transient data queue definition in Figure 33 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
```

Event monitoring points for the TRUE

The TRUE monitors call activity plus use of reusable or attached 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 and use of Attached 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

The following Monitor Control Table (MCT) entries make use of the event-monitoring points in the performance class used by the TRUE.

```
DFHMCT TYPE=INITIAL,SUFFIX=SO
DFHMCT TYPE=EMP,ID=(EZA01.01),CLASS=PERFORM,                  X
      PERFORM=SCLOCK(1),CLOCK=(1,INIT)
DFHMCT TYPE=EMP,ID=(EZA01.02),CLASS=PERFORM,                  X
      PERFORM=PCLOCK(1),CLOCK=(1,INIT)
DFHMCT TYPE=EMP,ID=(EZA01.03),CLASS=PERFORM,                  X
      PERFORM=SCLOCK(2),CLOCK=(2,READ)
DFHMCT TYPE=EMP,ID=(EZA01.04),CLASS=PERFORM,                  X
      PERFORM=PCLOCK(2),CLOCK=(2,READ)
DFHMCT TYPE=EMP,ID=(EZA01.05),CLASS=PERFORM,                  X
      PERFORM=SCLOCK(3),CLOCK=(3,WRITE)
DFHMCT TYPE=EMP,ID=(EZA01.06),CLASS=PERFORM,                  X
      PERFORM=PCLOCK(3),CLOCK=(3,WRITE)
DFHMCT TYPE=EMP,ID=(EZA01.07),CLASS=PERFORM,                  X
      PERFORM=SCLOCK(4),CLOCK=(4,SELECT)
DFHMCT TYPE=EMP,ID=(EZA01.08),CLASS=PERFORM,                  X
      PERFORM=PCLOCK(4),CLOCK=(4,SELECT)
DFHMCT TYPE=EMP,ID=(EZA01.09),CLASS=PERFORM,                  X
      PERFORM=SCLOCK(5),CLOCK=(5,OTHER)
DFHMCT TYPE=EMP,ID=(EZA01.10),CLASS=PERFORM,                  X
      PERFORM=PCLOCK(5),CLOCK=(6,OTHER)
DFHMCT TYPE=EMP,ID=(EZA01.11),CLASS=PERFORM,                  X
      PERFORM=ADDCNT(1,1),COUNT=(6,REUSABLE)
DFHMCT TYPE=EMP,ID=(EZA01.12),CLASS=PERFORM,                  X
      PERFORM=ADDCNT(2,1),COUNT=(7,ATTACHED)
DFHMCT TYPE=EMP,ID=(EZA01.13),CLASS=PERFORM,                  X
      PERFORM=(MLTCNT(1,5)),                                  X
      CLOCK=(1,INIT,READ,WRITE,SELECT,OTHER)
DFHMCT TYPE=EMP,ID=(EZA01.14),CLASS=PERFORM,                  X
      PERFORM=(MLTCNT(6,2)),                                  X
      COUNT=(6,REUSABLE,ATTACHED)
```

Figure 34. The Monitor Control Table (MCT) for TRUE

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 Sockets Interface Termination

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 make use of the event-monitoring points in the performance class used by the Listener.

```
DFHMCT TYPE=EMP,ID=(EZA02.01),CLASS=PERFORM,                  X
      PERFORM=ADDCNT(1,1),COUNT=(1,CONN)
DFHMCT TYPE=EMP,ID=(EZA02.02),CLASS=PERFORM,                  X
      PERFORM=ADDCNT(2,1),COUNT=(2,STARTED)
DFHMCT TYPE=EMP,ID=(EZA02.03),CLASS=PERFORM,                  X
      PERFORM=ADDCNT(3,1),COUNT=(3,INVALID)
DFHMCT TYPE=EMP,ID=(EZA02.04),CLASS=PERFORM,                  X
      PERFORM=ADDCNT(4,1),COUNT=(4,DISTRAN)
DFHMCT TYPE=EMP,ID=(EZA02.05),CLASS=PERFORM,                  X
      PERFORM=ADDCNT(5,1),COUNT=(5,DISPROG)
DFHMCT TYPE=EMP,ID=(EZA02.06),CLASS=PERFORM,                  X
      PERFORM=ADDCNT(6,1),COUNT=(6,GIVESOKT)
DFHMCT TYPE=EMP,ID=(EZA02.07),CLASS=PERFORM,                  X
      PERFORM=ADDCNT(7,1),COUNT=(7,SECEXIT)
DFHMCT TYPE=EMP,ID=(EZA02.08),CLASS=PERFORM,                  X
      PERFORM=ADDCNT(8,1),COUNT=(8,NOTAUTH)
DFHMCT TYPE=EMP,ID=(EZA02.09),CLASS=PERFORM,                  X
      PERFORM=ADDCNT(9,1),COUNT=(9,IOERR)
DFHMCT TYPE=EMP,ID=(EZA02.10),CLASS=PERFORM,                  X
      PERFORM=ADDCNT(10,1),COUNT=(10,NOSPACE)
DFHMCT TYPE=EMP,ID=(EZA02.11),CLASS=PERFORM,                  X
      PERFORM=ADDCNT(11,1),COUNT=(11,LENERR)
DFHMCT TYPE=EMP,ID=(EZA02.12),CLASS=PERFORM,                  X
      PERFORM=(MLTCNT(1,11)),                                 X
      COUNT=(1,CONN,STARTED,INVALID,DISTRAN,DISPROG,GIVESOKT,SX
      ECEXIT,NOTAUTH,IOERR,NOSPACE,LENERR)
DFHMCT TYPE=FINAL
END
```

Figure 35. The Monitor Control Table (MCT) for Listener

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

CICS program list table (PLT)

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

To start the IP CICS Sockets 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 Sockets 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, refer to 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, refer to 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, refer to 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 (refer to *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 will try 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
```

TCP/IP services — Modifying data sets

To run CICS TCP/IP, you need to make entries in the *hlq*.PROFILE.TCPIP configuration data set. [7]

The *hlq*.PROFILE.TCPIP data set

You define the CICS region to TCP/IP on MVS 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 36 on page 40.

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 36 on page 40 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 MVS are prevented from using these ports.

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

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

```
;
; 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 36. Definition of the hlq.TCP/IP profile

Two different CICS Listeners running on the same host can share a port. Refer to the discussion on port descriptions in *z/OS Communications Server: IP Configuration Reference* for more information on 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 will need it when you initialize CICS TCP/IP (see Chapter 4, "Starting and stopping CICS sockets", on page 81). 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 37. The TCPIPJOBNAME parameter in the hlq.TCPIP.DATA data set

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 48.

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 52.[8]

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 RDO and modified using the configuration transactions (see "Configuration transaction (EZAC)" on page 52). The macro is keyword-driven with the TYPE keyword controlling the specific function request. The data set contains one record for each instance of CICS it supports, and one record for each Listener. The following is an example of the macros required to create a configuration file for one instance of the CICS Sockets Interface using one Listener:

```
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                 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
      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
      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
```

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

```
      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
      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=CICSPROD,  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=CICSPROD,  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 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
```

TYPE parameter

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

Value Meaning

INITIAL
Initialize the generation environment. This value should only be used once per generation and it should be in the first invocation of the macro. For subparameters, refer to “TYPE=INITIAL” on page 43.

CICS Identify a CICS object. This corresponds to a specific instance of CICS and will create a configuration record. For subparameters, refer to "TYPE=CICS".

LISTENER
Identify a Listener object. This will create a Listener record. For subparameters, refer to "TYPE=LISTENER" on page 44.

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

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

TCPADDR
The name of the TCP/IP address space.

NTASKS
The number of reusable MVS subtasks that will be 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.

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 sockets 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.

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.

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.

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 will not happen unless their age is greater than the CACHMAX value. The default value is 10.

ERRORTD
The name of a Transient Data destination to which error messages will be written. The default value is CSMT.

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.

Note: For detailed information on CICS sockets messages, see Appendix D, "CICS sockets messages", on page 355.

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

Value Meaning

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.

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

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.

PORT The port number this Listener will use for accepting connections. This parameter is mandatory. The value should be between 2049 and 65535. The ports may be shared. See *z/OS Communications Server: IP Configuration Reference* for more information on port sharing.

| **AF** Determines if the Listener being defined will support IPv6 partners and be able to give an IPv6 socket descriptor to an IPv6 child server program. YES indicates the Listener will give an IPv6 socket to the child server program. NO, the default, indicates the Listener will give 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.

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

ACCTIME
The time in seconds this Listener will wait 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 will reduce the resources used to support the listener on a lightly loaded system and will consequently lengthen shutdown processing. Conversely, setting this to a low value will increase resources used to support the listener but facilitate shutdown processing.

GIVTIME
The time in seconds this Listener will wait for a response to a GIVESOCKET. If this time expires, the Listener will assume that either the server transaction did not start or the TAKESOCKET failed. At this time, the Listener will send 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.

REATIME
The time in seconds this Listener will wait for a response to a RECV request. If this time expires, the Listener will assume that the client has failed and will terminate the connection by closing the socket. If this parameter is not specified, no checking for read timeout is done.

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.

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.

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.

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).

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.

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.

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*.

WLMGN1

The group name this Listener will use 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 will continue to support CICS Listeners desiring to participate in work load balancing for IPv4 clients. IPv6 enabled Listeners will be able to participate in work load balancing for their IPv4 and IPv6 clients.

IPv6 clients should use unique hostnames and you should enable DNS entries to allow unique host names to exist in different DNS zones. This will enable 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 will 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 in the range of 1–12 characters. The name is padded to the right with blanks to meet the 18 character name required by the Workload Manager.

The default is no registration.

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

WLMGN2

See WLMGN1 for information.

WLMGN3

See WLMGN1 for information.

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 will continue to read on the connection until this length of data has been received. FASTRD handles blocking.

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.

FASTRD
This parameter is obsolete and has been removed from the EZAC screens. If specified in the EZACICD macro, it is ignored and a warning note is generated. The Listener always issues a SELECT between ACCEPT and READ.

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. With NO specified for TRANTRN, the user data is translated if and only if TRANUSR is YES. The default value for TRANTRN is YES.

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 may be translated depending on TRANTRN and whether the transaction code is uppercase EBCDIC. The default value for TRANUSR is YES.

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 4 shows how the Listener handles translation with different combinations of TRANTRN, TRANSUSR, and character format of the transaction code:

Table 4. 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

Table 4. Conditions for translation of tranid and user data (continued)

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

SECEXIT

The name of the security exit used by this Listener. The default is no security exit.

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 38 on page 49 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) )
 /*
//*
```

Figure 38. Example of JCL to define a configuration file (Part 1 of 4)

```
//* 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 *
         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                 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
               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
               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
```

Figure 38. Example of JCL to define a configuration file (Part 2 of 4)

```
      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
      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=CICSPROD,  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=CICSPROD,  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 38. 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 38. 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 Sockets 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 may have no Listener, but this is not common practice. A CICS may have multiple Listeners that are either multiple instances of the supplied Listener with different specifications, multiple user-written Listeners, or some combination.

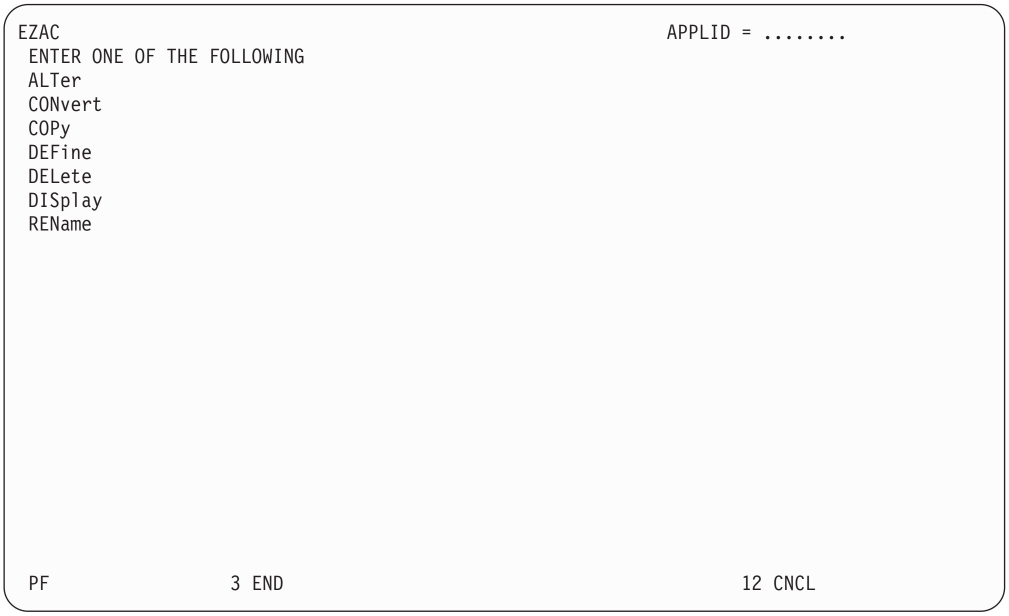
Configuration transaction (EZAC)

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.

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 will fail if the new CICS object already exists. • Listener - Copies the Listener object to create another Listener object. COPY will fail 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.

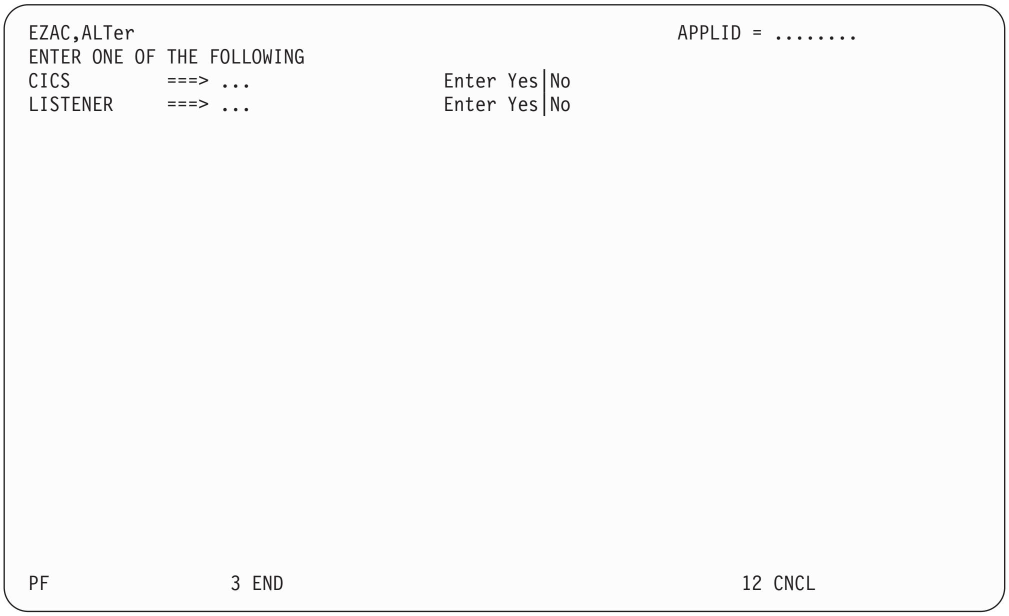
If you enter EZAC, the following screen is displayed:



```
EZAC                                                        APPLID = ........
 ENTER ONE OF THE FOLLOWING
 ALTer
 CONvert
 COPy
 DEFine
 DELete
 DISplay
 REName





















 PF                 3 END                                             12 CNCL
```

Figure 39. 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:

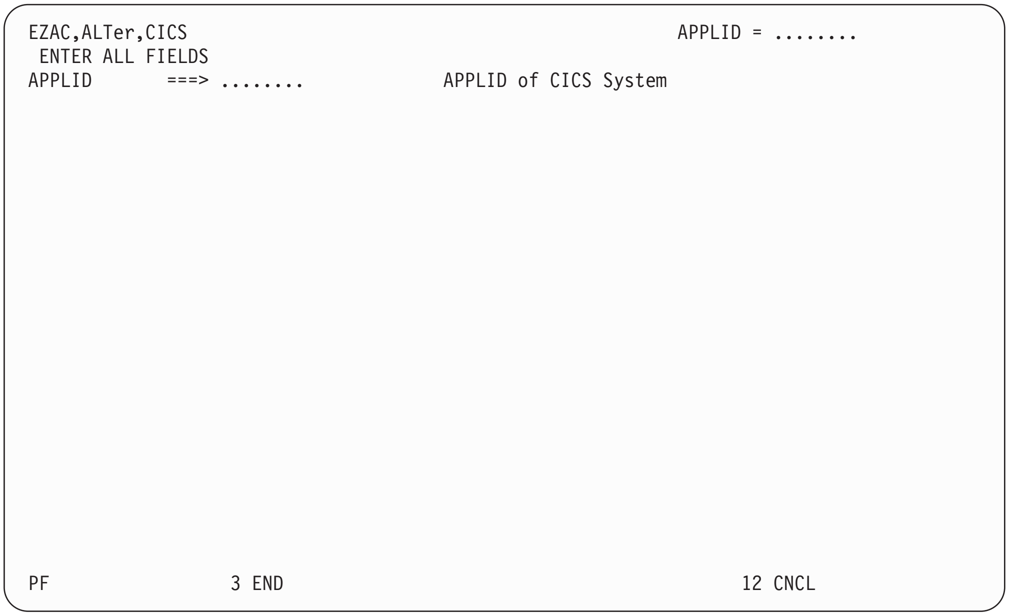


```
EZAC,ALTer                                              APPLID = ........
ENTER ONE OF THE FOLLOWING
CICS        ===> ...                 Enter Yes|No
LISTENER    ===> ...                 Enter Yes|No















PF                3 END                                       12 CNCL
```

Figure 40. EZAC,ALTER screen

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

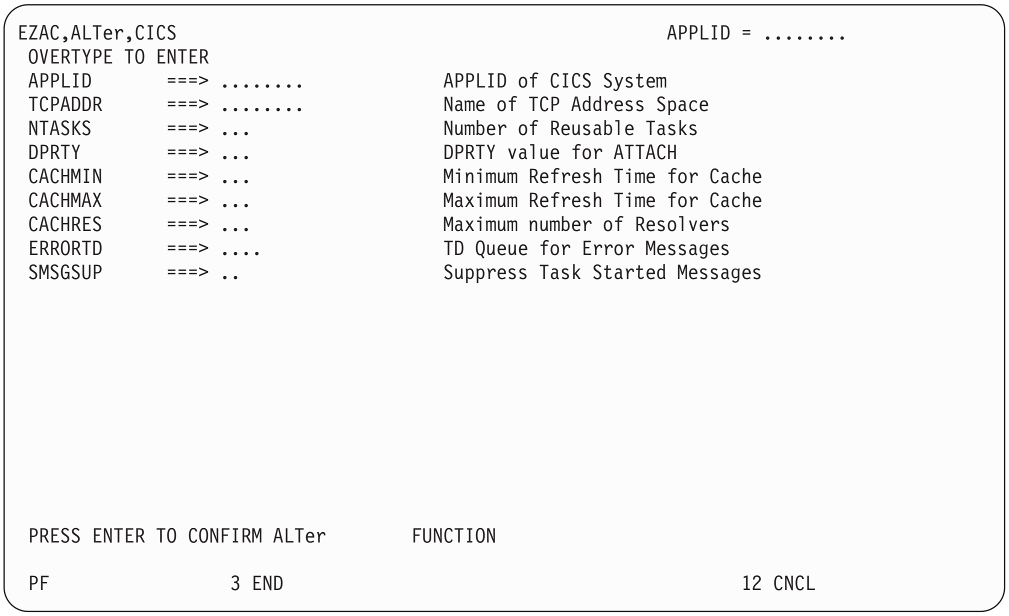
ALTER,CICS: For alteration of a CICS object, the following screen is displayed:



```
EZAC,ALTer,CICS                                         APPLID = ........
 ENTER ALL FIELDS
APPLID      ===> ........            APPLID of CICS System
















PF                3 END                                       12 CNCL
```

Figure 41. EZAC,ALTER,CICS screen

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

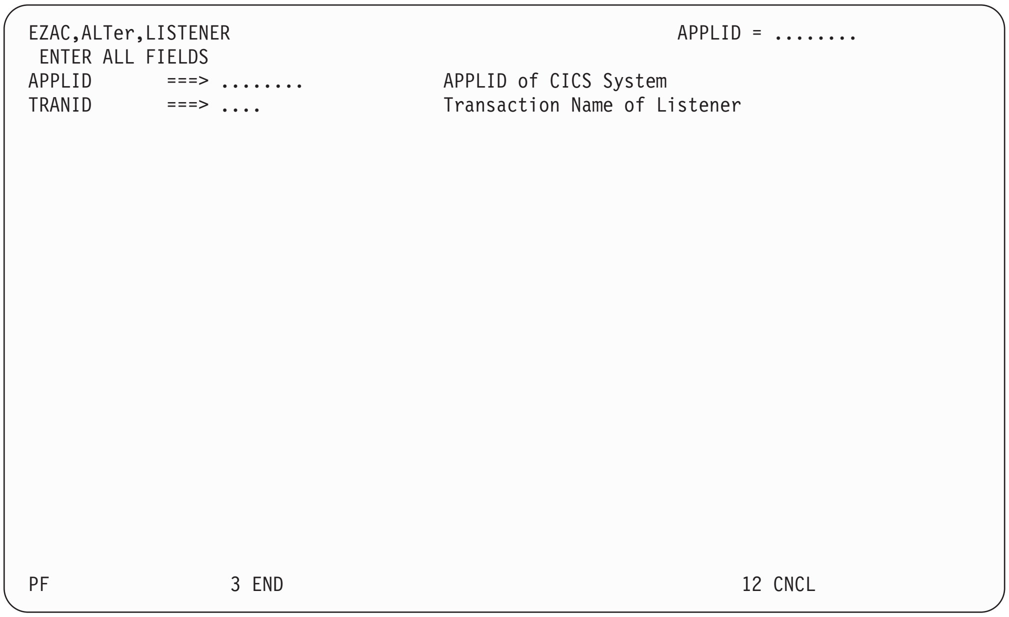


```
EZAC,ALTer,CICS                                                  APPLID = ........
 OVERTYPE TO ENTER
 APPLID      ===> ........            APPLID of CICS System
 TCPADDR     ===> ........            Name of TCP Address Space
 NTASKS      ===> ...                 Number of Reusable Tasks
 DPRTY       ===> ...                 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

 PRESS ENTER TO CONFIRM ALTer        FUNCTION

 PF                3 END                                        12 CNCL
```

Figure 42. EZAC,ALTER,CICS detail screen

The system will request a confirmation of the values displayed. After the changes are confirmed, the changed values will be in effect for the next initialization of the CICS Sockets interface.

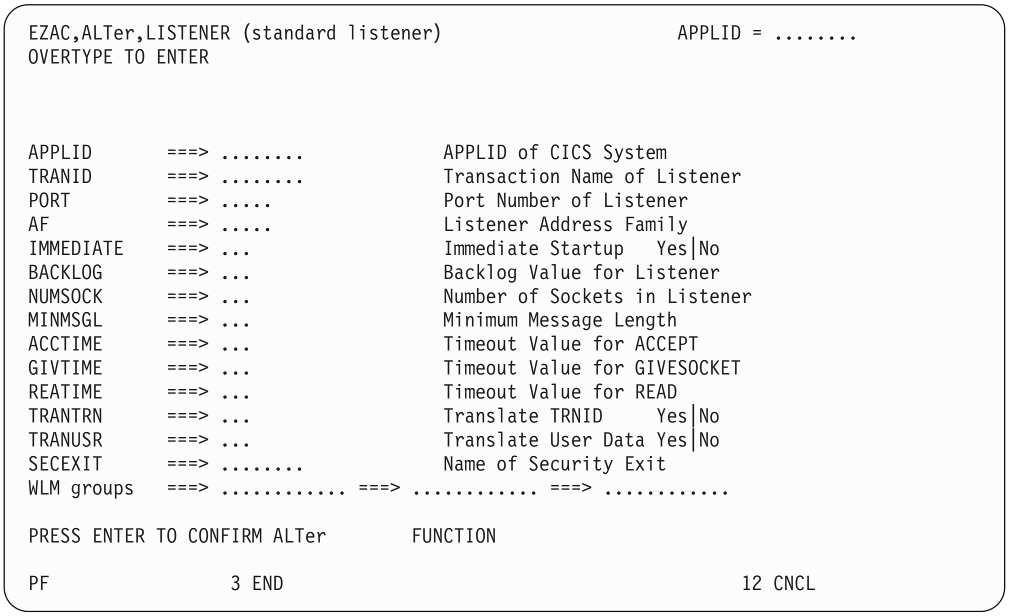
ALTER,LISTENER: For alteration of a Listener, the following screen is displayed:



```
 EZAC,ALTer,LISTENER                                              APPLID = ........
  ENTER ALL FIELDS
 APPLID      ===> ........            APPLID of CICS System
 TRANID      ===> ....                Transaction Name of Listener

 PF                3 END                                        12 CNCL
```

Figure 43. ALTER,LISTENER screen

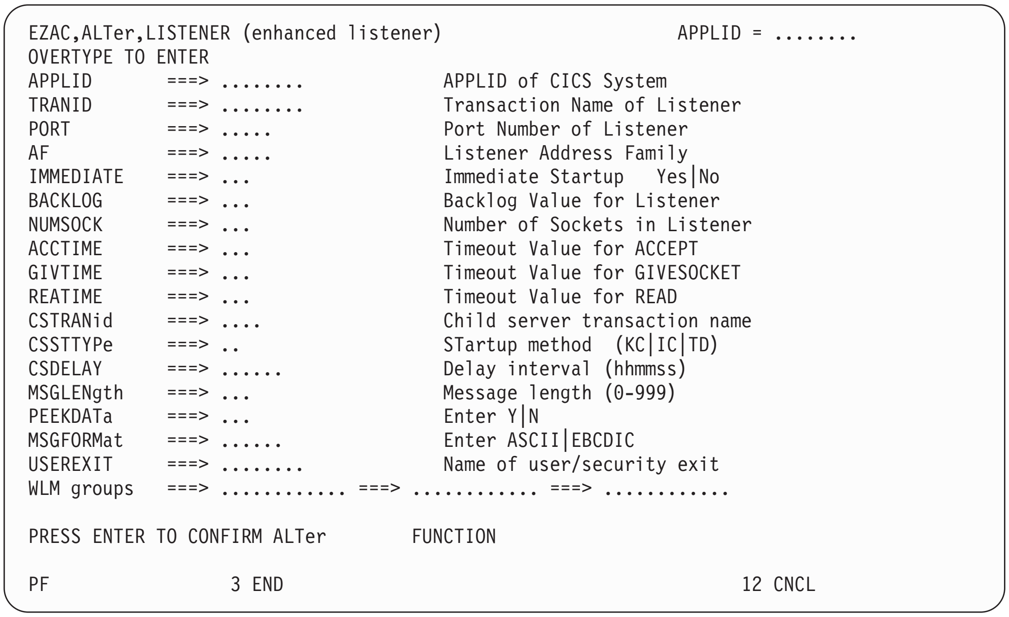
After the names are entered, one of the following two screens is displayed. The first screen is displayed for the standard version:



```
EZAC,ALTer,LISTENER (standard listener)                 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
MINMSGL      ===> ...                 Minimum Message Length
ACCTIME      ===> ...                 Timeout Value for ACCEPT
GIVTIME      ===> ...                 Timeout Value for GIVESOCKET
REATIME      ===> ...                 Timeout Value for READ
TRANTRN      ===> ...                 Translate TRNID     Yes|No
TRANUSR      ===> ...                 Translate User Data Yes|No
SECEXIT      ===> ........            Name of Security Exit
WLM groups   ===> ........... ===> ........... ===> ...........

PRESS ENTER TO CONFIRM ALTer       FUNCTION

PF              3 END                                       12 CNCL
```

Figure 44. EZAC,ALTER,LISTENER detail screen - Standard version

The following screen is displayed for the enhanced version:

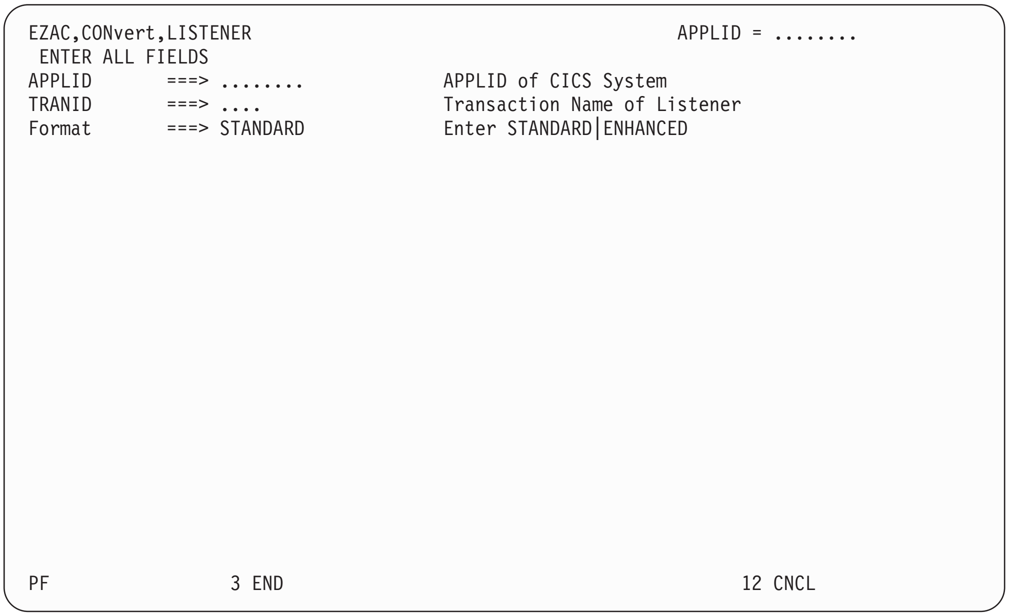


```
EZAC,ALTer,LISTENER (enhanced listener)                 APPLID = ........
OVERTYPE TO ENTER
APPLID       ===> ........            APPLID of CICS System
TRANID       ===> ........            Transaction Name of Listener
PORT         ===> .....               Port Number of Listener
AF           ===> .....               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
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
WLM groups   ===> ........... ===> ........... ===> ...........

PRESS ENTER TO CONFIRM ALTer       FUNCTION

PF              3 END                                       12 CNCL
```

Figure 45. EZAC,ALTER,LISTENER detail screen - Enhanced version

The system will request a confirmation of the values displayed. After the changes are confirmed, the changed values will be in effect for the next initialization of the CICS Sockets Interface.

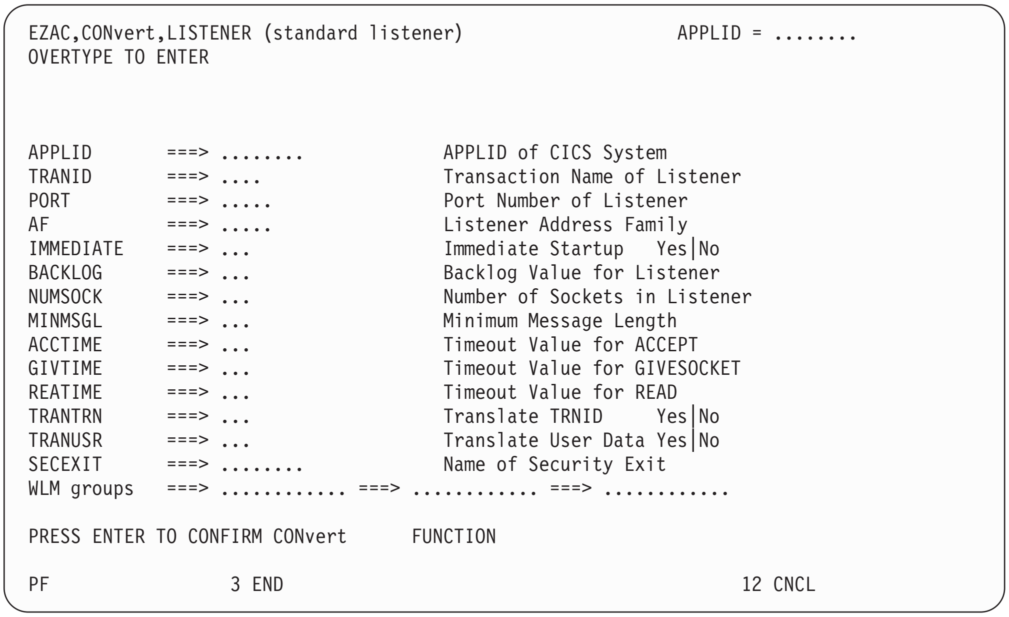
CONVERT function: The CONVERT function is used to convert between standard and enhanced versions of the Listener. If you specify CONvert on the EZAC Initial Screen or enter EZAC,CON on a blank screen, the following screen is displayed:



```
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 46. EZAC,CONVERT,LISTENER screen

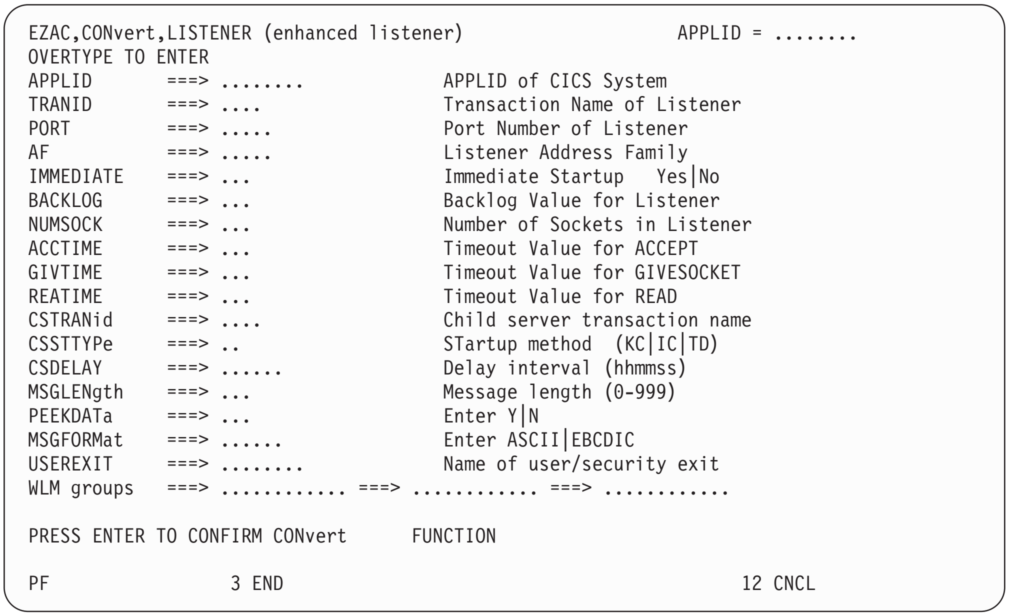
After the names and format type are entered, one of the following two screens is displayed. The first screen is displayed for the standard version:



```
EZAC,CONvert,LISTENER (standard listener)                    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
MINMSGL      ===> ...                  Minimum Message Length
ACCTIME      ===> ...                  Timeout Value for ACCEPT
GIVTIME      ===> ...                  Timeout Value for GIVESOCKET
REATIME      ===> ...                  Timeout Value for READ
TRANTRN      ===> ...                  Translate TRNID     Yes|No
TRANUSR      ===> ...                  Translate User Data Yes|No
SECEXIT      ===> ........             Name of Security Exit
WLM groups   ===> ........... ===> ........... ===> ...........

PRESS ENTER TO CONFIRM CONvert      FUNCTION

PF               3 END                                            12 CNCL
```

Figure 47. EZAC,CONVERT,LISTENER detail screen - Standard version

The following screen is displayed for the enhanced version:

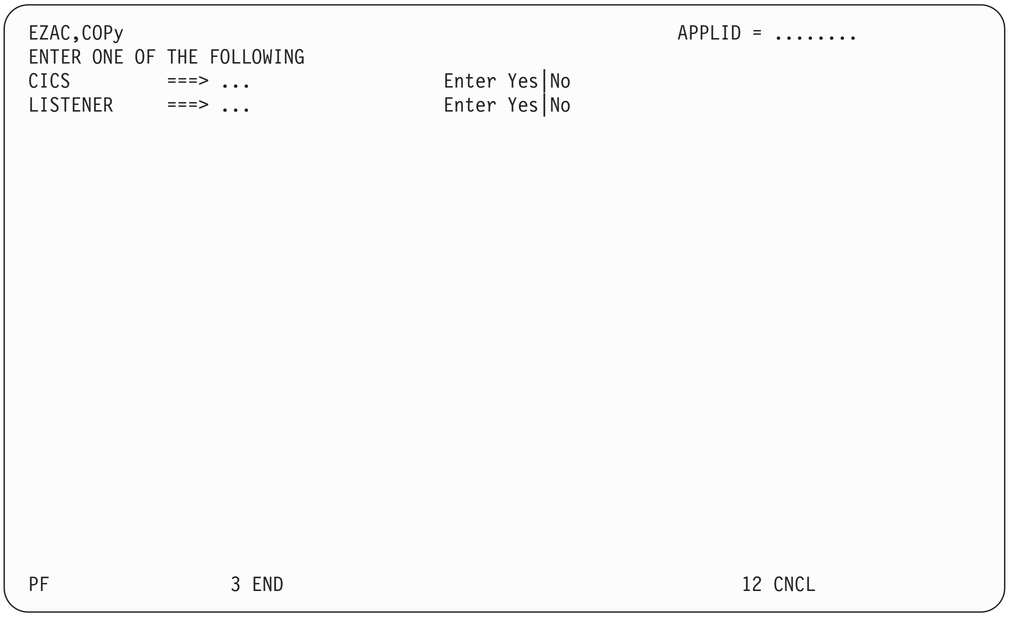


```
EZAC,CONvert,LISTENER (enhanced listener)                    APPLID = ........
OVERTYPE TO ENTER
APPLID       ===> ........             APPLID of CICS System
TRANID       ===> ....                 Transaction Name of Listener
PORT         ===> .....                Port Number of Listener
AF           ===> .....                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
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
WLM groups   ===> ........... ===> ........... ===> ...........

PRESS ENTER TO CONFIRM CONvert      FUNCTION

PF                 3 END                                       12 CNCL
```

Figure 48. EZAC,CONVERT,LISTENER detail screen - Enhanced version

The system will request a confirmation of the values displayed. After the changes are confirmed, the changed values will be 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:

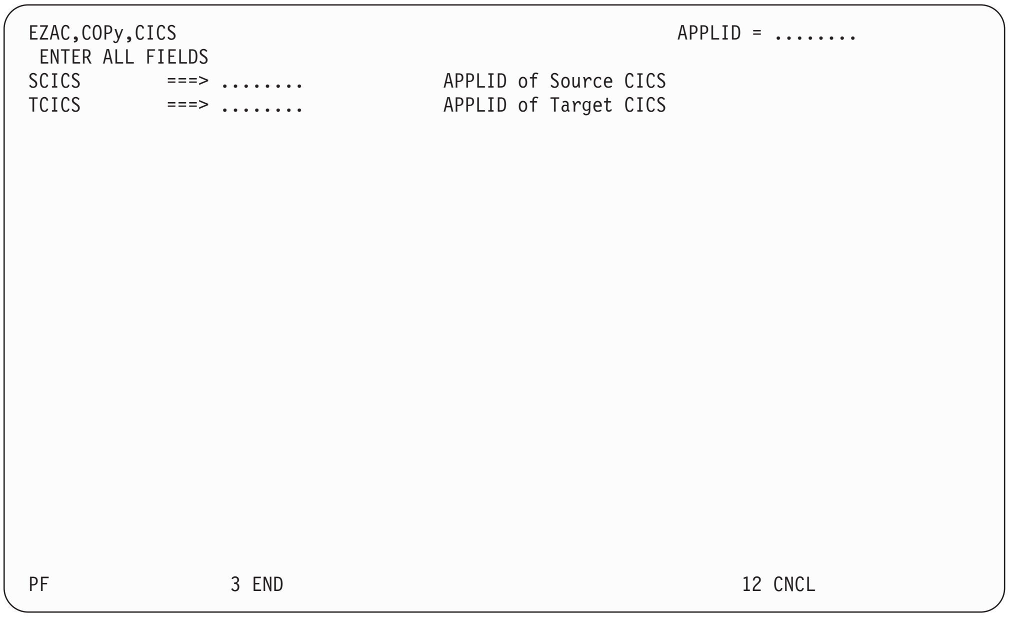


```
EZAC,COPy                                                    APPLID = ........
ENTER ONE OF THE FOLLOWING
CICS         ===> ...                  Enter Yes|No
LISTENER     ===> ...                  Enter Yes|No















PF                 3 END                                       12 CNCL
```

Figure 49. 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:

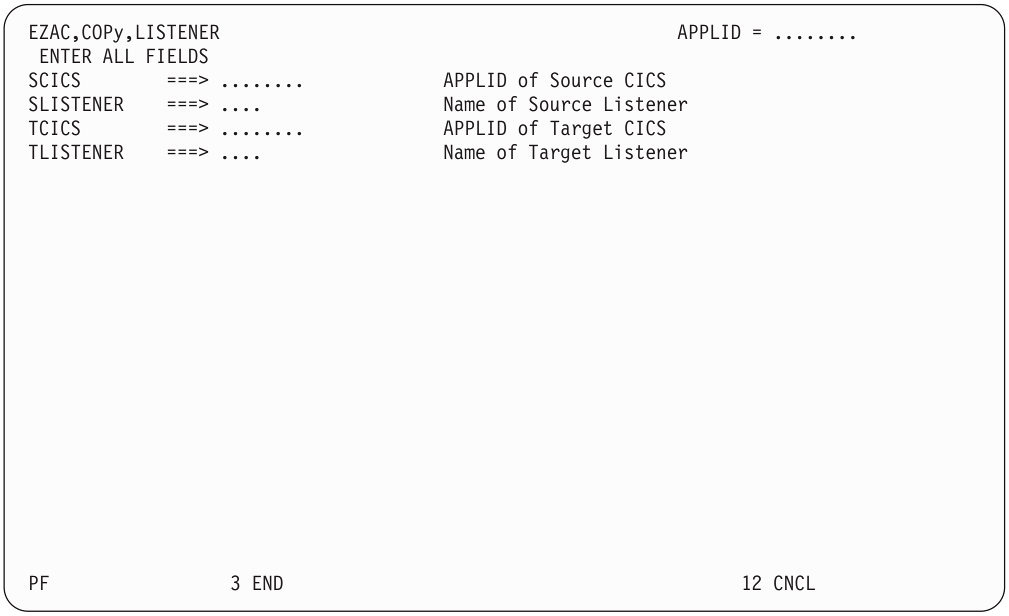


```
EZAC,COPy,CICS                                        APPLID = ........
 ENTER ALL FIELDS
SCICS       ===> ........           APPLID of Source CICS
TCICS       ===> ........           APPLID of Target CICS
















PF                 3 END                                    12 CNCL
```

Figure 50. 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:

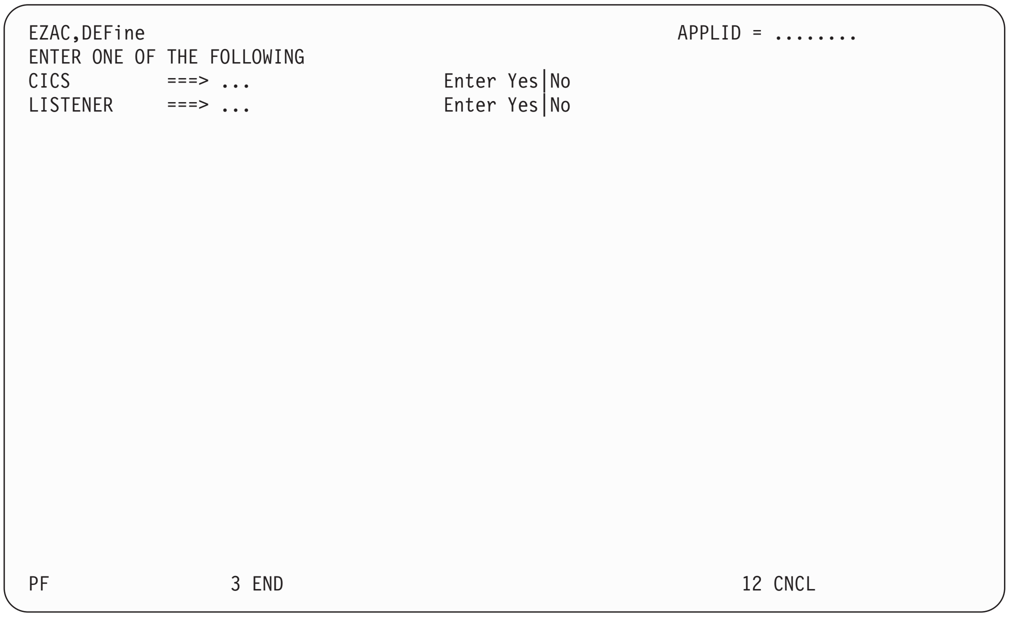


```
EZAC,COPy,LISTENER                                    APPLID = ........
 ENTER ALL FIELDS
SCICS       ===> ........           APPLID of Source CICS
SLISTENER   ===> ....               Name of Source Listener
TCICS       ===> ........           APPLID of Target CICS
TLISTENER   ===> ....               Name of Target Listener














PF                 3 END                                    12 CNCL
```

Figure 51. 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:

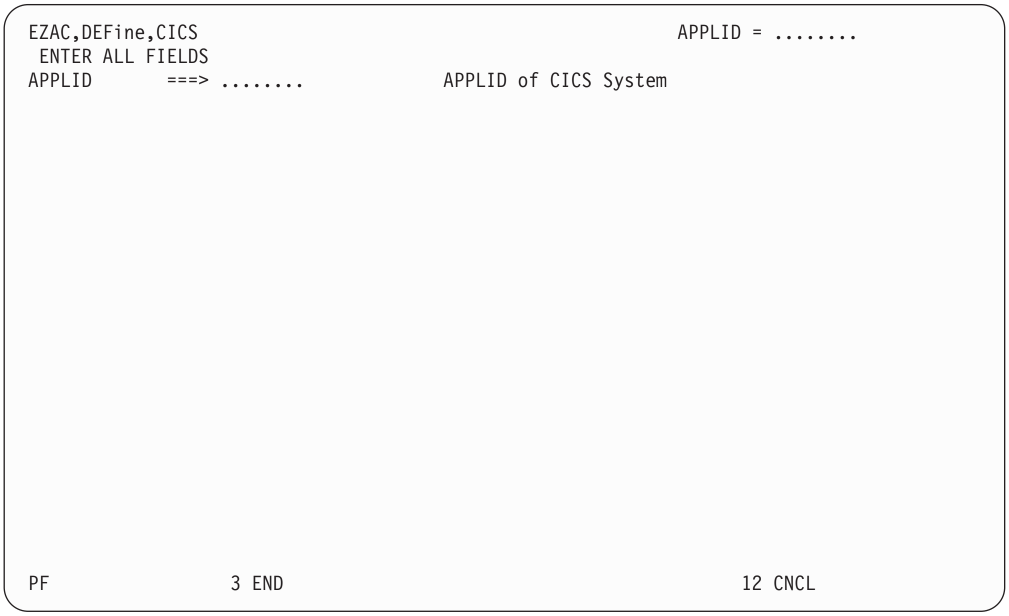


```
EZAC,DEFine                                                  APPLID = ........
ENTER ONE OF THE FOLLOWING
CICS         ===> ...                 Enter Yes|No
LISTENER     ===> ...                 Enter Yes|No
















PF                 3 END                                           12 CNCL
```

Figure 52. EZAC,DEFINE screen

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

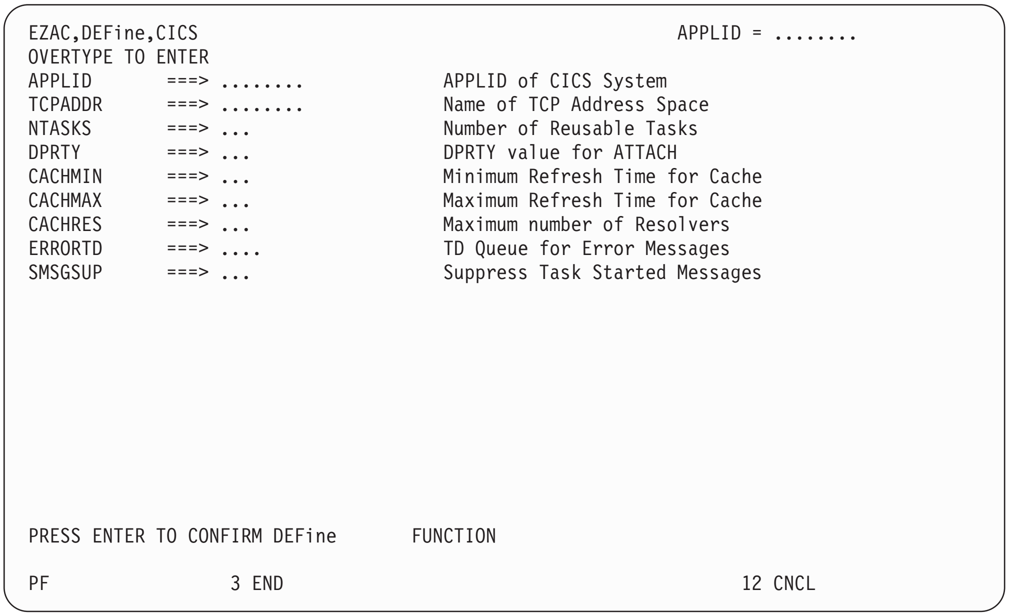
DEFINE,CICS: For definition of a CICS object, the following screen is displayed:



```
EZAC,DEFine,CICS                                             APPLID = ........
 ENTER ALL FIELDS
APPLID       ===> ........            APPLID of CICS System



















PF                 3 END                                           12 CNCL
```

Figure 53. EZAC,DEFINE,CICS screen

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

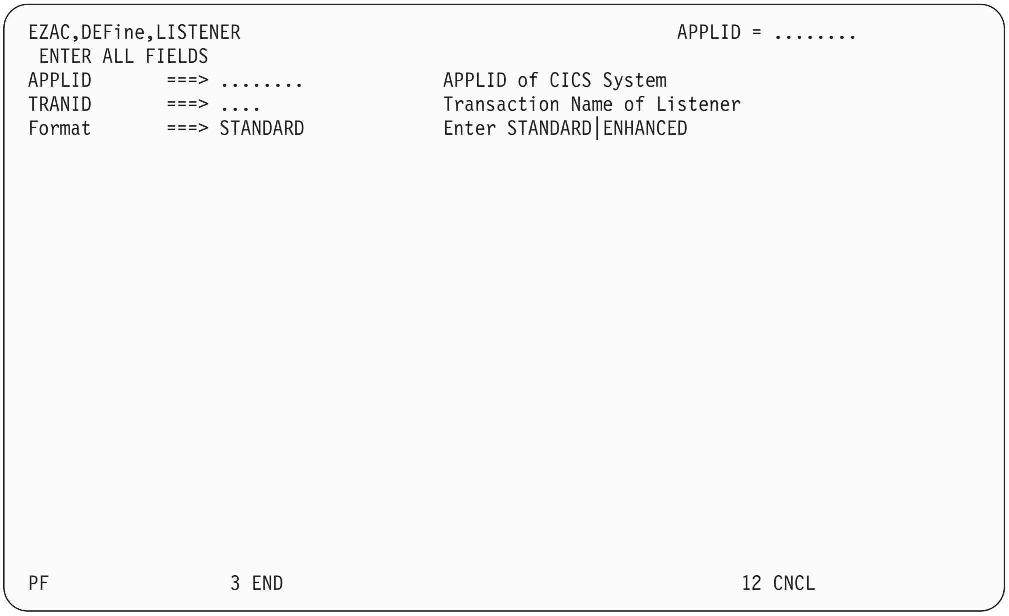


```
EZAC,DEFine,CICS                                             APPLID = ........
OVERTYPE TO ENTER
APPLID       ===> ........             APPLID of CICS System
TCPADDR      ===> ........             Name of TCP Address Space
NTASKS       ===> ...                  Number of Reusable Tasks
DPRTY        ===> ...                  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











PRESS ENTER TO CONFIRM DEFine       FUNCTION

PF               3 END                                          12 CNCL
```

Figure 54. 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.

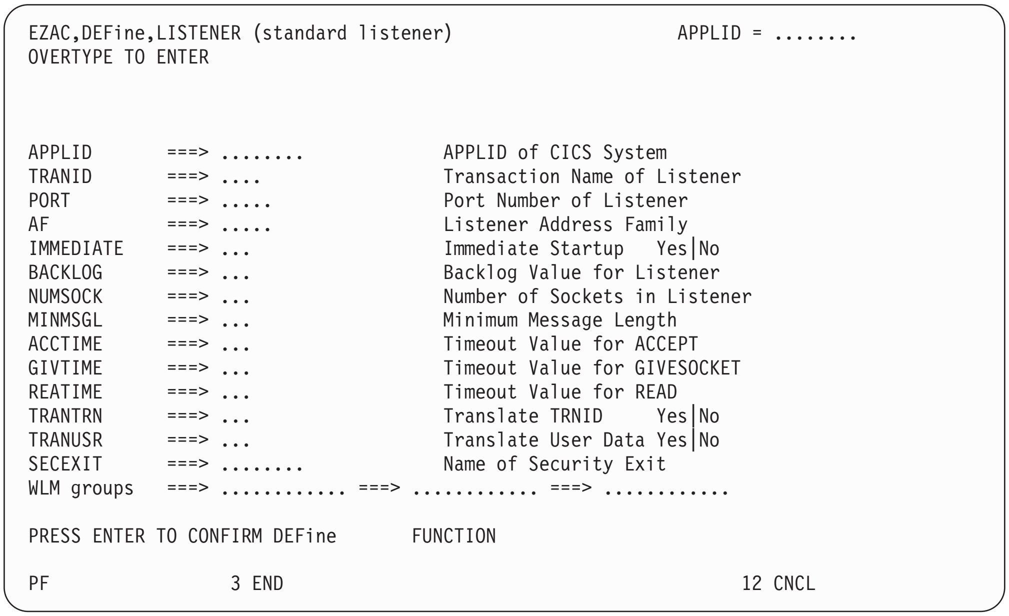
DEFINE,LISTENER: For definition of a Listener, the following screen is displayed:



```
EZAC,DEFine,LISTENER                                         APPLID = ........
 ENTER ALL FIELDS
APPLID       ===> ........             APPLID of CICS System
TRANID       ===> ....                 Transaction Name of Listener
Format       ===> STANDARD             Enter STANDARD|ENHANCED



















PF               3 END                                          12 CNCL
```

Figure 55. EZAC,DEFINE,LISTENER screen

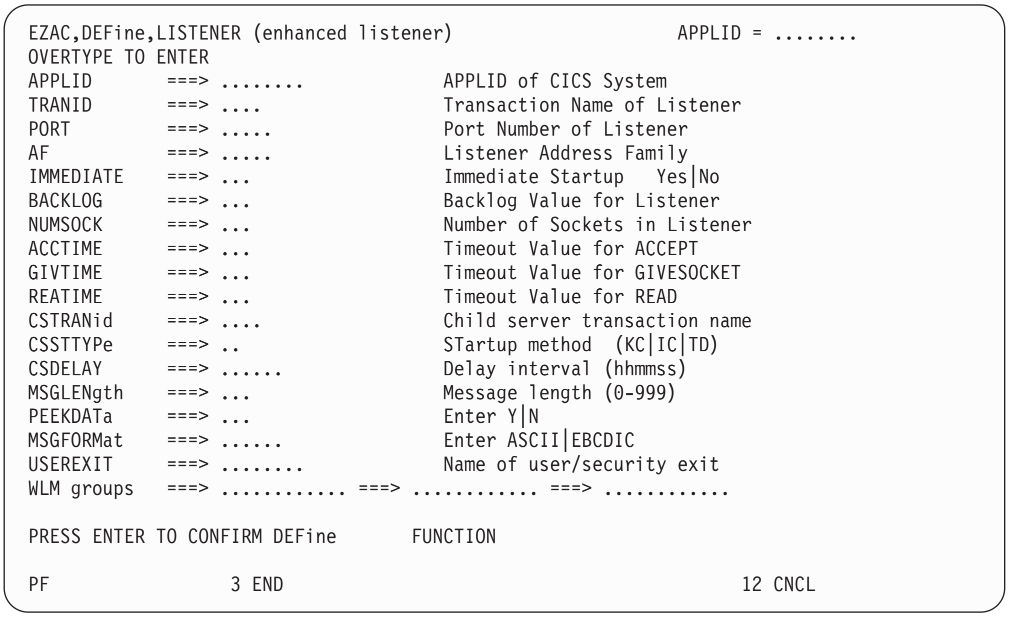
After the names are entered, one of the two following screens is displayed. The first screen is displayed for the standard version:



```
EZAC,DEFine,LISTENER (standard listener)                 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
MINMSGL      ===> ...                  Minimum Message Length
ACCTIME      ===> ...                  Timeout Value for ACCEPT
GIVTIME      ===> ...                  Timeout Value for GIVESOCKET
REATIME      ===> ...                  Timeout Value for READ
TRANTRN      ===> ...                  Translate TRNID     Yes|No
TRANUSR      ===> ...                  Translate User Data Yes|No
SECEXIT      ===> ........             Name of Security Exit
WLM groups   ===> ........... ===> ........... ===> ...........

PRESS ENTER TO CONFIRM DEFine       FUNCTION

PF               3 END                                         12 CNCL
```

Figure 56. EZAC,DEFINE,LISTENER detail screen - Standard version

The following screen is displayed for the enhanced version:

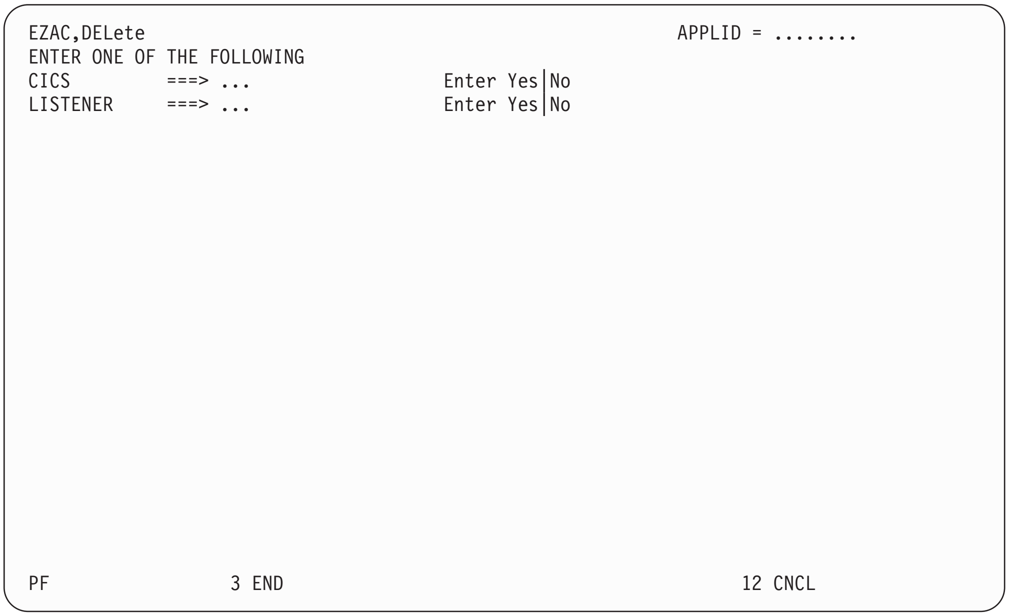


```
EZAC,DEFine,LISTENER (enhanced listener)                 APPLID = ........
OVERTYPE TO ENTER
APPLID       ===> ........             APPLID of CICS System
TRANID       ===> ....                 Transaction Name of Listener
PORT         ===> .....                Port Number of Listener
AF           ===> .....                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
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
WLM groups   ===> ........... ===> ........... ===> ...........

PRESS ENTER TO CONFIRM DEFine       FUNCTION

PF               3 END                                         12 CNCL
```

Figure 57. EZAC,DEFINE,LISTENER detail screen - Enhanced version

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

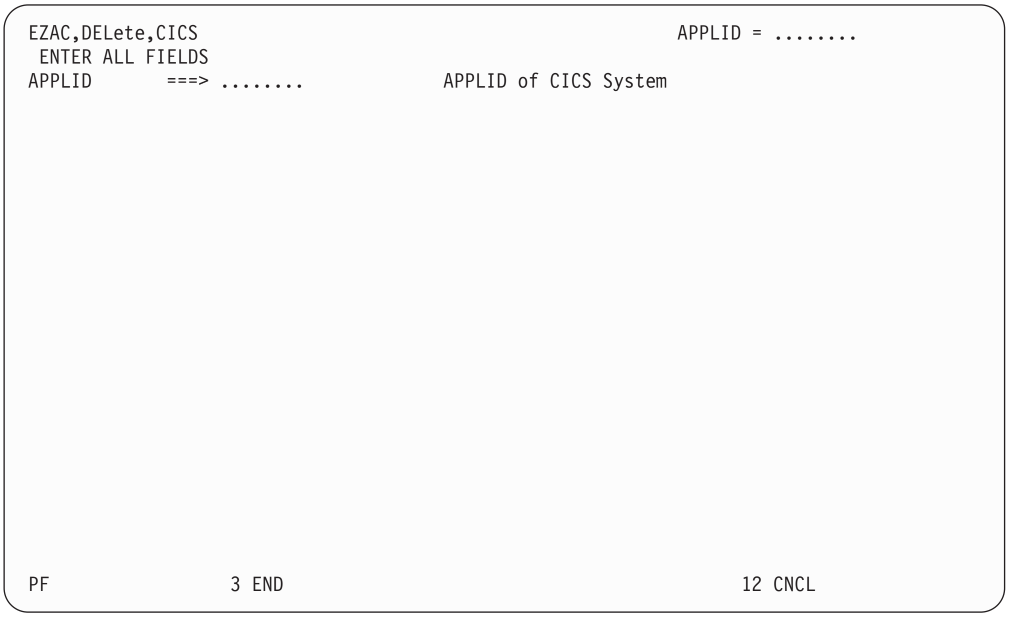
DELETE function: The DELETE function is used to delete a CICS object or a Listener object. Deleting a CICS object deletes all Listener objects within that CICS object. If you specify DELete on the EZAC initial screen or enter EZAC,DEL on a blank screen, the following screen is displayed:



```
EZAC,DELete                                               APPLID = ........
ENTER ONE OF THE FOLLOWING
CICS         ===> ...                 Enter Yes|No
LISTENER     ===> ...                 Enter Yes|No














PF                3 END                                         12 CNCL
```

Figure 58. EZAC,DELETE screen

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

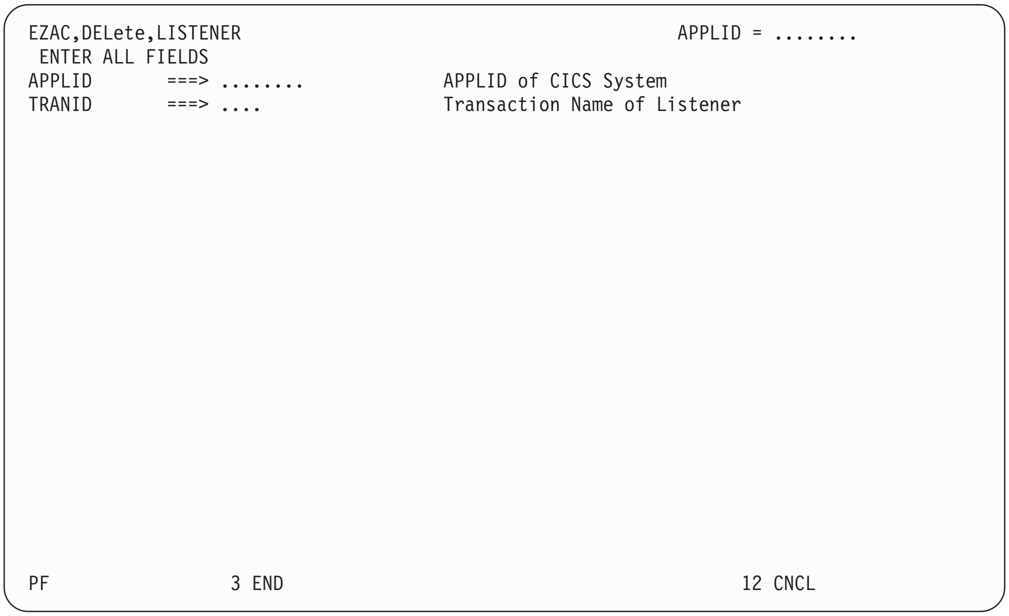


```
EZAC,DELete,CICS                                          APPLID = ........
 ENTER ALL FIELDS
APPLID       ===> ........            APPLID of CICS System















PF                3 END                                         12 CNCL
```

Figure 59. 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:

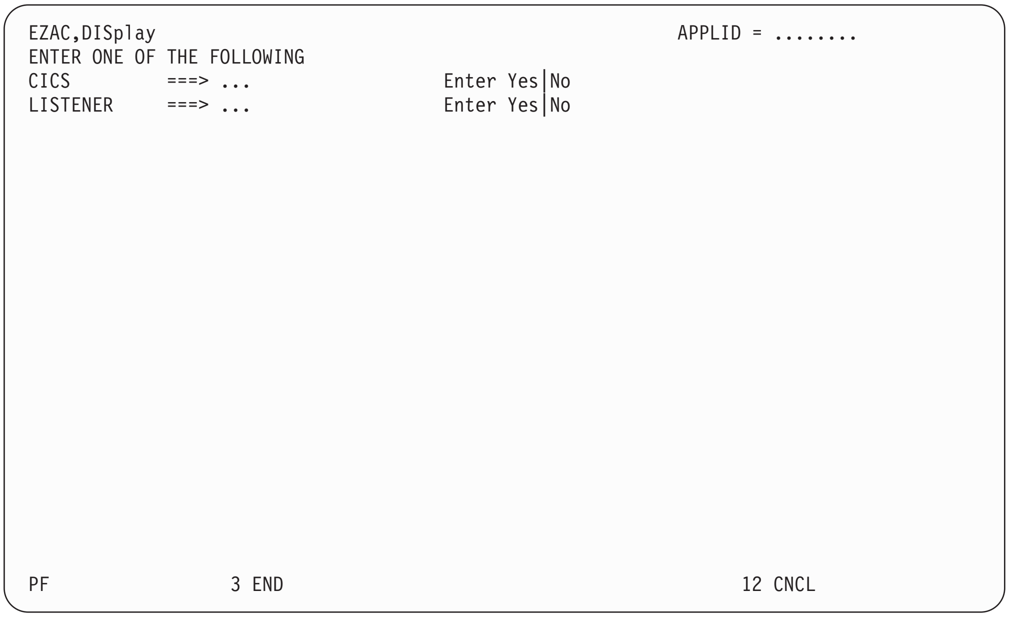


```
EZAC,DELete,LISTENER                                        APPLID = ........
 ENTER ALL FIELDS
APPLID       ===> ........             APPLID of CICS System
TRANID       ===> ....                 Transaction Name of Listener




















PF               3 END                                            12 CNCL
```

Figure 60. 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:

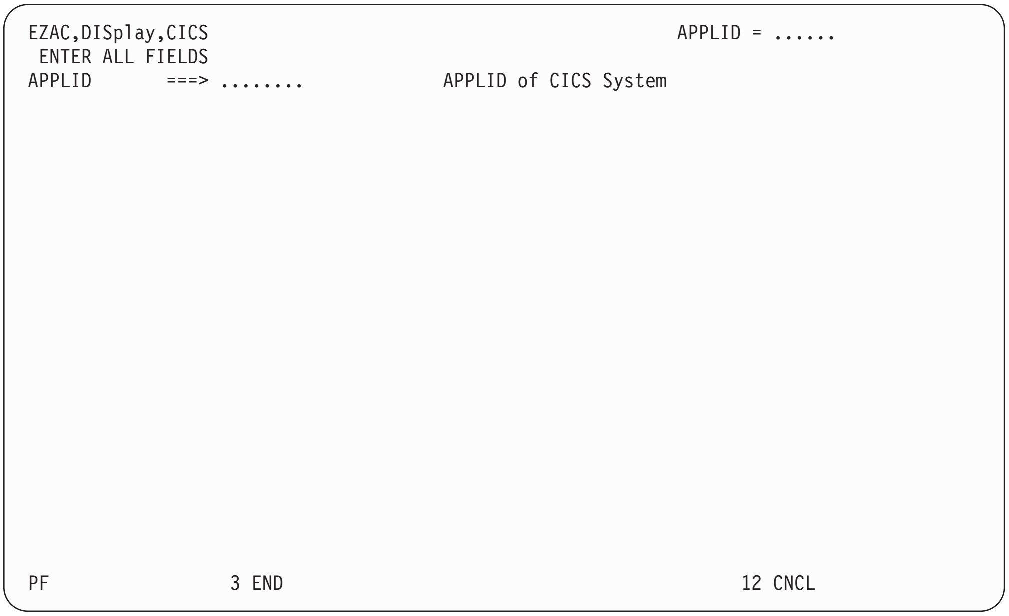


```
EZAC,DISplay                                                APPLID = ........
ENTER ONE OF THE FOLLOWING
CICS         ===> ...                  Enter Yes|No
LISTENER     ===> ...                  Enter Yes|No




















PF               3 END                                            12 CNCL
```

Figure 61. EZAC,DISPLAY screen

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

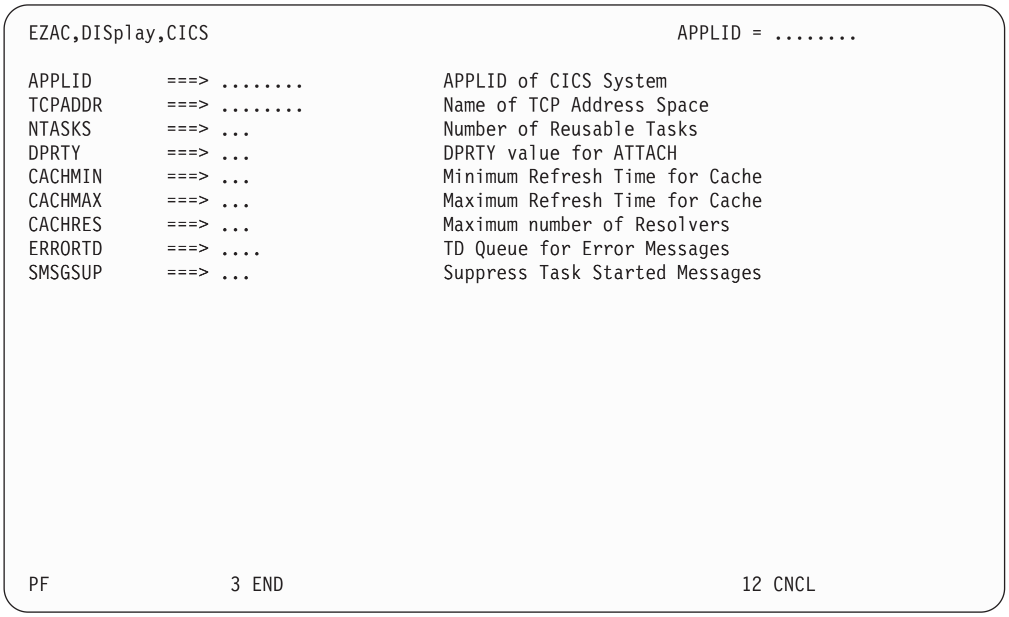
DISPLAY,CICS: If you specify DISPLAY,CICS, the following screen is displayed:



```
EZAC,DISplay,CICS                                          APPLID = ......
 ENTER ALL FIELDS
APPLID       ===> ........            APPLID of CICS System




















PF                3 END                                          12 CNCL
```

Figure 62. EZAC,DISPLAY,CICS screen

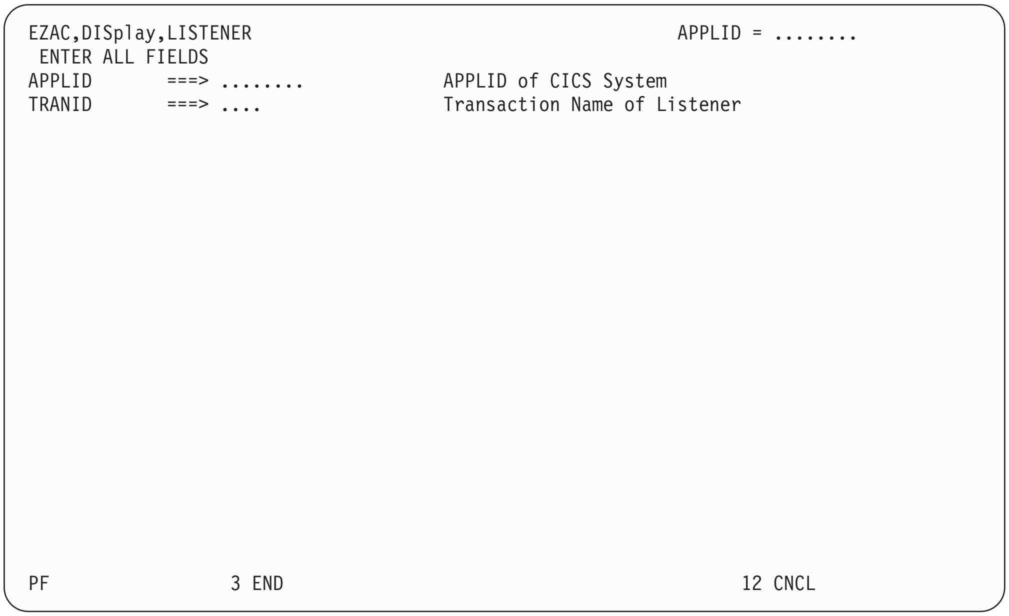
After the APPLID is entered, the following screen is displayed:



```
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












PF                3 END                                          12 CNCL
```

Figure 63. EZAC,DISPLAY,CICS detail screen

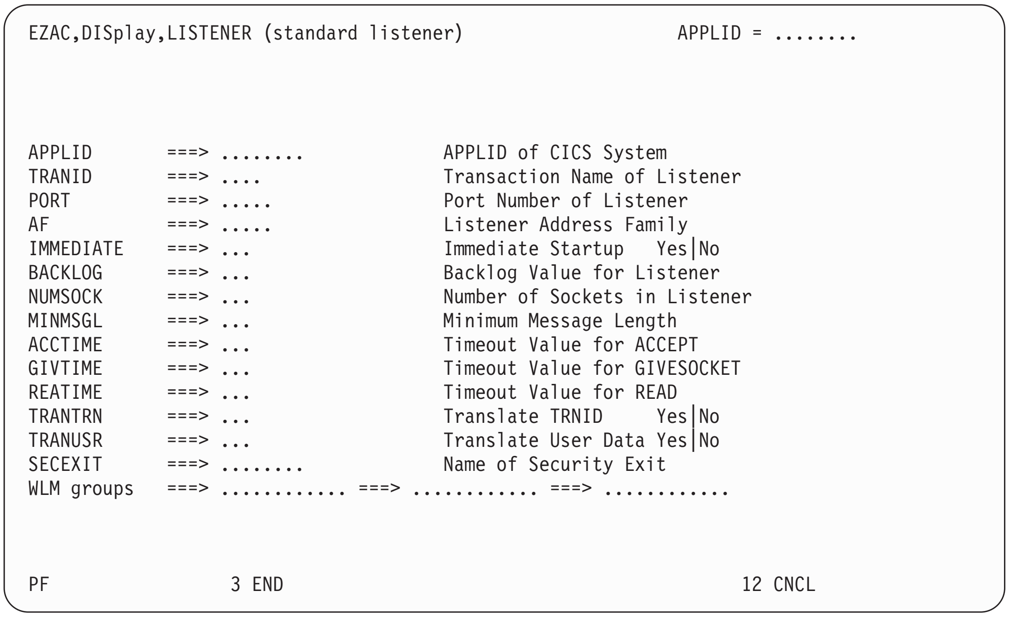
DISPLAY,LISTENER: If you specify DISPLAY,LISTENER, the following screen is displayed:



```
EZAC,DISplay,LISTENER                                   APPLID = ........
 ENTER ALL FIELDS
APPLID      ===> ........             APPLID of CICS System
TRANID      ===> ....                 Transaction Name of Listener















PF               3 END                                        12 CNCL
```

Figure 64. EZAC,DISPLAY,LISTENER screen

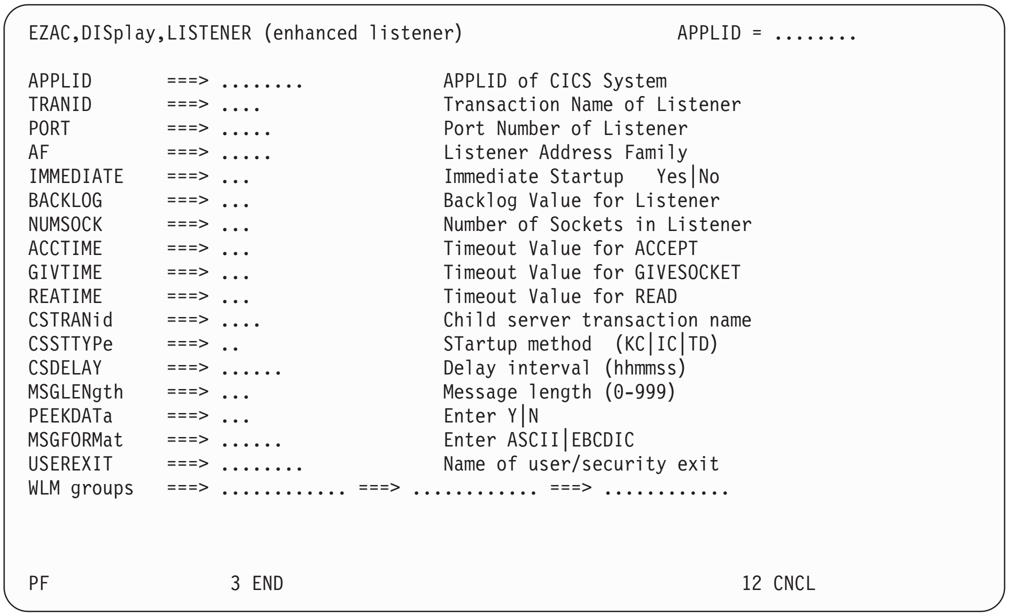
After the APPLID and name are entered, one of the two following screens is displayed. The first screen is displayed for the standard version:



```
EZAC,DISplay,LISTENER (standard listener)               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
MINMSGL     ===> ...                  Minimum Message Length
ACCTIME     ===> ...                  Timeout Value for ACCEPT
GIVTIME     ===> ...                  Timeout Value for GIVESOCKET
REATIME     ===> ...                  Timeout Value for READ
TRANTRN     ===> ...                  Translate TRNID     Yes|No
TRANUSR     ===> ...                  Translate User Data Yes|No
SECEXIT     ===> ........             Name of Security Exit
WLM groups  ===> ............ ===> ............ ===> ............



PF               3 END                                        12 CNCL
```

Figure 65. EZAC,DISPLAY,LISTENER detail screen - Standard version

The following screen is displayed for the enhanced version:

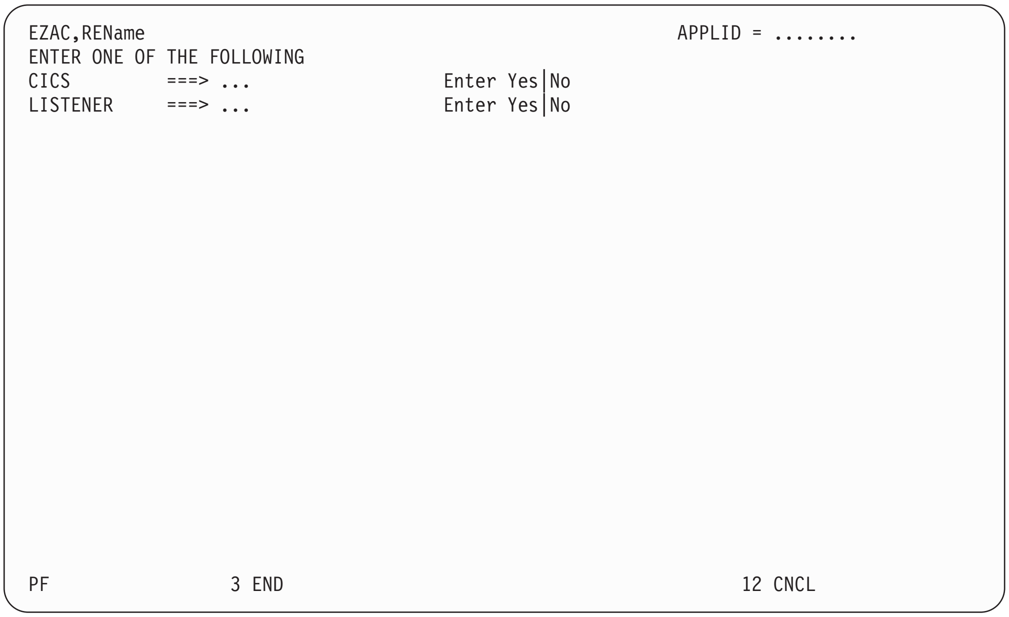


```
EZAC,DISplay,LISTENER (enhanced listener)                  APPLID = ........

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
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
WLM groups   ===> ........... ===> ........... ===> ...........



PF              3 END                                       12 CNCL
```

Figure 66. EZAC,DISPLAY,LISTENER detail screen - Enhanced version

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:

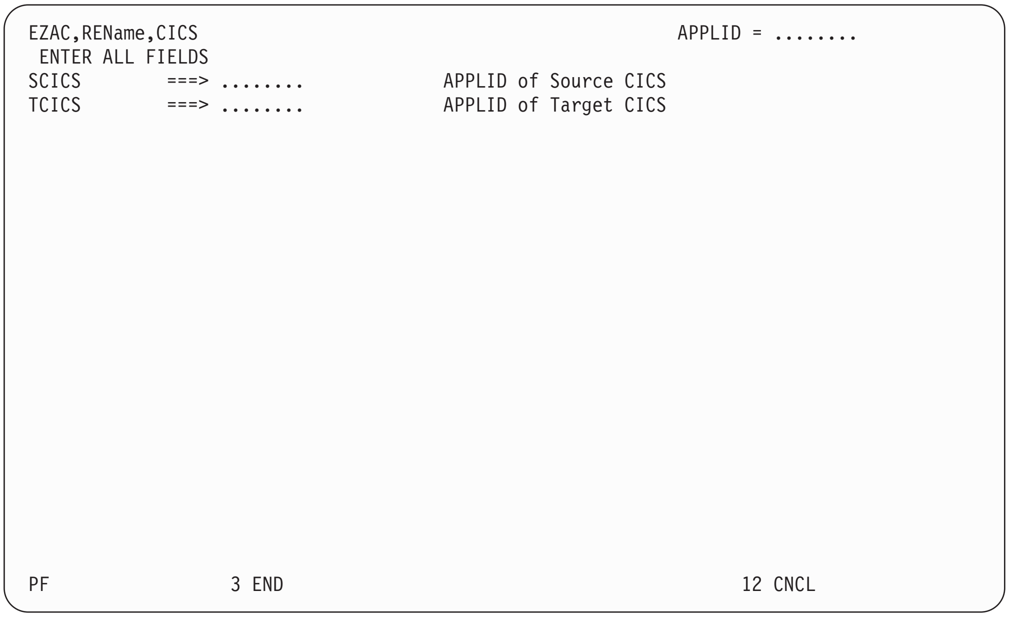


```
EZAC,REName                                                APPLID = ........
ENTER ONE OF THE FOLLOWING
CICS         ===> ...                 Enter Yes|No
LISTENER     ===> ...                 Enter Yes|No


















PF              3 END                                       12 CNCL
```

Figure 67. 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:

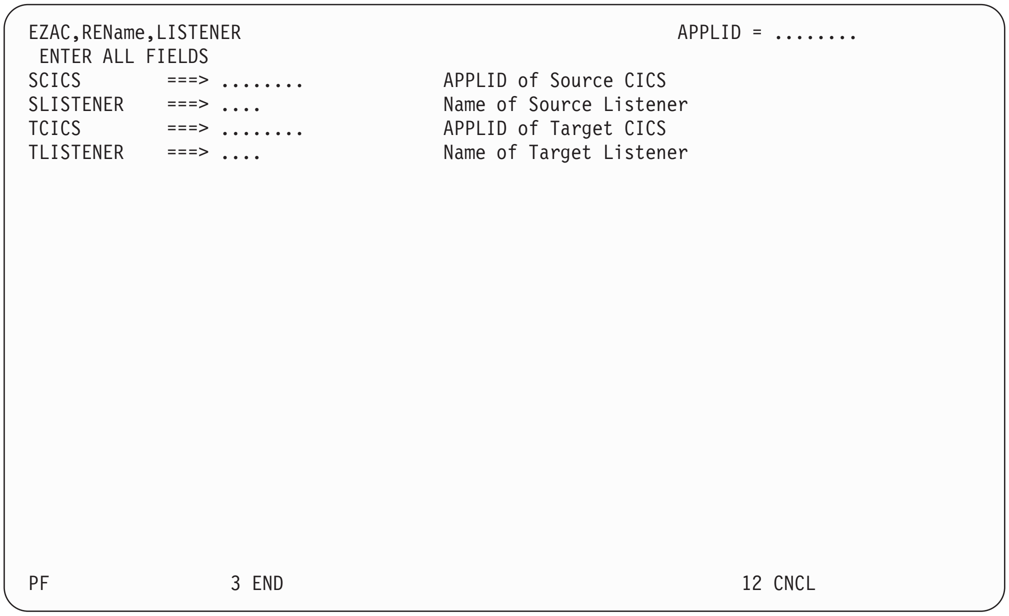


```
EZAC,REName,CICS                                              APPLID = ........
 ENTER ALL FIELDS
SCICS       ===> ........             APPLID of Source CICS
TCICS       ===> ........             APPLID of Target CICS





















PF                3 END                                            12 CNCL
```

Figure 68. 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 69. 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 affects on IP CICS sockets

The UNIX MAXFILEPROC parameter of the BPXPRMxx parmlib member controls the number of sockets that can be opened concurrently in a CICS region. You can use this parameter 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.

For more information on how MAXFILEPROC affects tuning applications, refer to *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.

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 may also be stored and queried. The different possible types of information are defined through input data to the name server in the resource records.

While 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 gets repeated requests for the same set of domain names, using the DNS will improve performance significantly.

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 will be 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 may outweigh the benefits of connection balancing.

Refer to *z/OS Communications Server: IP Configuration Reference* for information on caching issues.

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

- DNS Caching will 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, 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.
- DNS Caching will continue 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 will continue to support CICS Listeners wanting to participate in work load balancing. IPv6 enabled Listeners will still be 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, and BIND 4.9.3 does not support AAAA records.
- The IPv6 client will not be 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. Note that this is not a true IPv6 connection as DNS/WLM will 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 may 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 will build a single record of binary zeros. See “Step 1: Create the initialization module” on page 74.

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 parameters plus the information passed by the invoking application to manage the cache. These parameters are as follows:

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

Minimum refresh time
The minimum time in minutes between refreshes of a cache record. If a cache record is ’younger’ than this time, it will be used. This value is set to 15 (minutes).

Maximum refresh time
The maximum time in minutes between refreshes of a cache record. If a cache record is ’older’ than this time, it will be refreshed. This value is set to 30 (minutes).

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

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 not successful, issue the GETHOSTBYNAME call unless request specifies cache only.
2. If cache retrieval is successful, calculate the ’age’ of the record (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” on page 74.

2. Define the cache files to CICS. See “Step 2: Define the cache file to CICS” on page 77.
3. Use EZACIC25 to replace GETHOSTBYNAME calls in CICS application modules. See “Step 3: Execute EZACIC25” on page 78.

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 wish to preload the file with dummy records for frequently referenced domain names, it would look like this:

```
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 will improve 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 will minimize 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 70 on page 76 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 70. 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 70. Example of defining and initializing a DNS cache file (Part 2 of 2)

Once 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 since 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 will be 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 will be returned regardless of its age.

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

HOSTENT structure

The returned HOSTENT structure is shown in Figure 71.

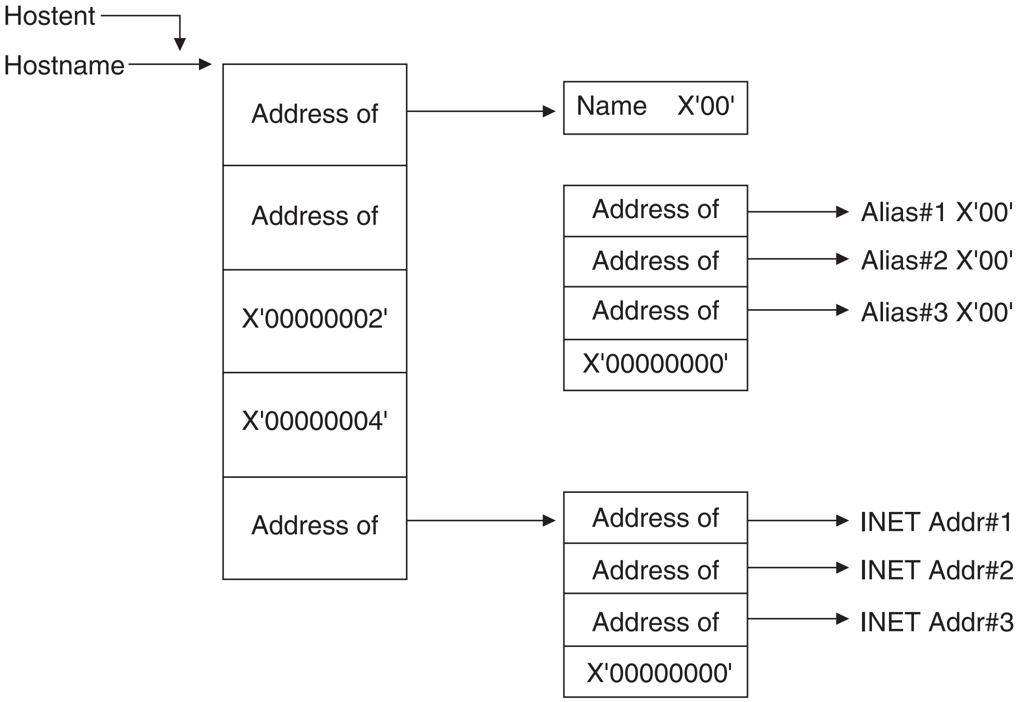


Figure 71. The DNS HOSTENT

Chapter 4. Starting and stopping CICS sockets

This chapter explains how to start and stop (enable and disable) CICS TCP/IP. It describes how:

- You can customize your system so that CICS TCP/IP starts and stops automatically. See “Starting/stopping CICS TCP/IP automatically”.
- An operator can also start and stop CICS TCP/IP manually after CICS has been initialized. See “Starting/stopping CICS TCP/IP manually” on page 82.
- You can also start and stop CICS TCP/IP from a CICS application program. See “Starting/stopping CICS TCP/IP with program link” on page 86.

Starting/stopping CICS TCP/IP automatically

You can start and stop the CICS Sockets Interface automatically by modifying the CICS Program List Table (PLT).

- Startup (PLTPI)

  To start the IP CICS Sockets 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 Sockets 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 Sockets Interface is started in the PLT (started by invoking EZACIC20), then the PLTPIUSR user ID also requires the UPDATE access to the EXITPROGRAM security class. Failure to have at least the UPDATE access to the EXITPROGRAM security class causes the IP CICS Sockets Interface and Listener to not start when starting or not stop when stopping. Message EZY1350E is issued and the IP CICS Sockets Interface does not start.

Starting/stopping CICS TCP/IP manually

You can start CICS TCP/IP manually by using the EZAO transaction. This operational transaction has four functions:

Interface Startup
Starts the interface in a CICS address space and starts all Listeners that are identified for immediate start. Replaces part of the CSKE transaction.

Note: The EZAO transaction *must* be running on the CICS where you want to start the CICS Sockets Interface. You may not start a CICS Sockets Interface from a different CICS.

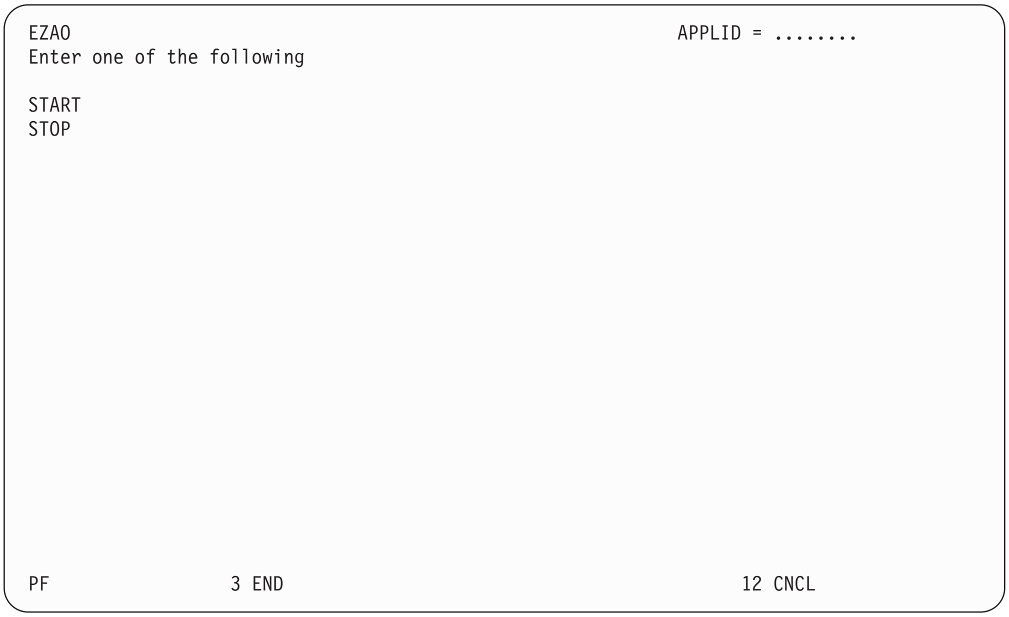
Interface Shutdown
Stops the interface in a CICS address space. Replaces part of the CSKD transaction.

Listener Startup
Starts a Listener in a CICS address space. Replaces part of the CSKE transaction.

Listener Shutdown
Stops a Listener in a CICS address space. Replaces part of the CSKD transaction.

Note: Since the PLT method is now available, the Card Reader Line Printer (CRLP) method of starting the CICS Sockets Interface and Listener is no longer supported. If the EZAO transaction is invoked using CARDIN, it will fail with abend EZAO because the EZAO transaction should be invoked only from a VTAM terminal. The EZAO abend is issued by the EZAO or EZAC transaction program when an EXEC CICS SEND MAP or EXEC CICS RECEIVE MAP command fails in trying to send or receive screens to the VTAM terminal.

When you enter EZAO, the following screen is displayed.

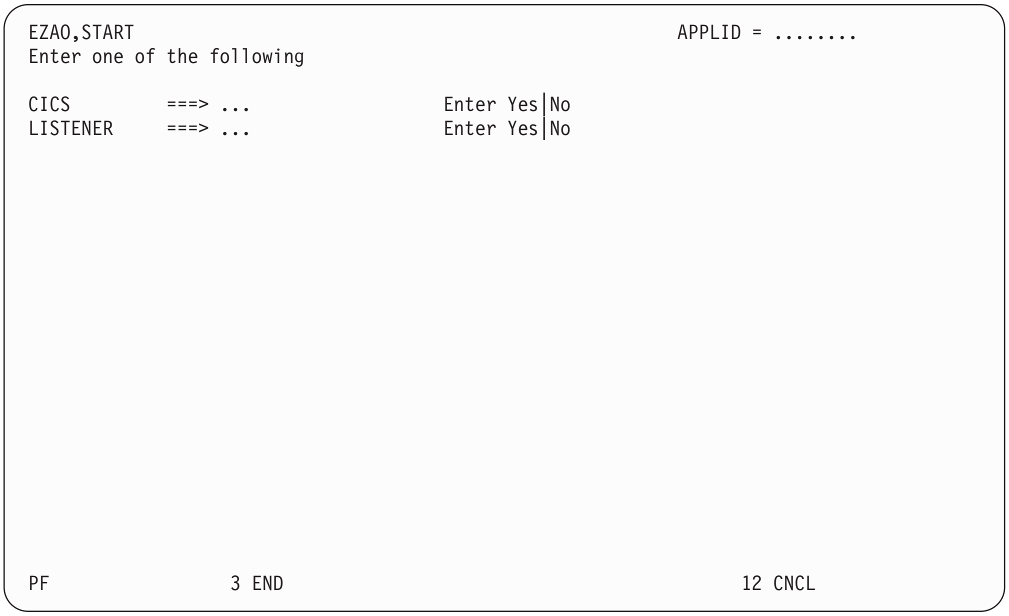


```
EZAO                                                        APPLID = ........
Enter one of the following

START
STOP




















PF                 3 END                                          12 CNCL
```

Figure 72. EZAO initial screen

START function

The START function starts either the CICS Sockets Interface or a Listener within the interface. When the interface is started, all Listeners marked for immediate start will be started as well. If you enter STA on the previous screen or enter EZAO STA on a blank screen, the following screen is displayed.

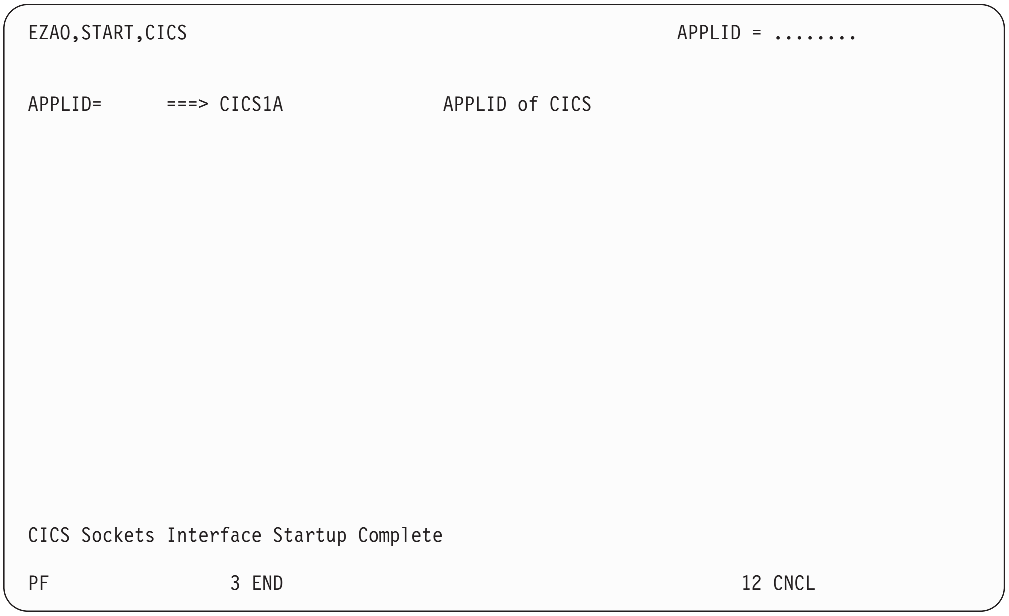


```
EZAO,START                                                 APPLID = ........
Enter one of the following

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




















PF                3 END                                          12 CNCL
```

Figure 73. EZAO START screen

START CICS

If you enter START CICS, the following screen is displayed.

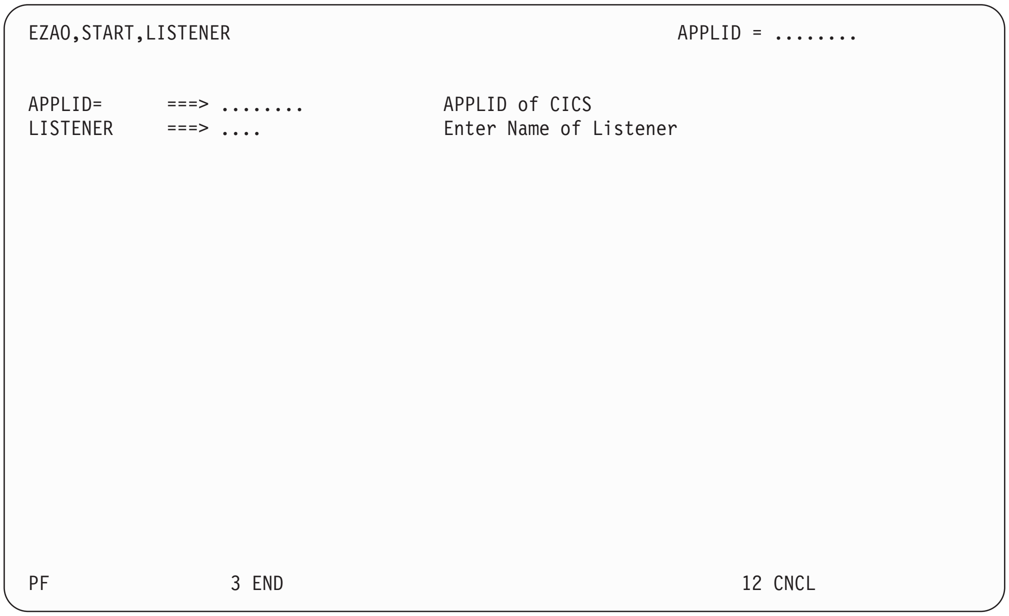


```
EZAO,START,CICS                                            APPLID = ........


APPLID=      ===> CICS1A              APPLID of CICS

















CICS Sockets Interface Startup Complete

PF                3 END                                          12 CNCL
```

Figure 74. EZAO START CICS response screen

START LISTENER

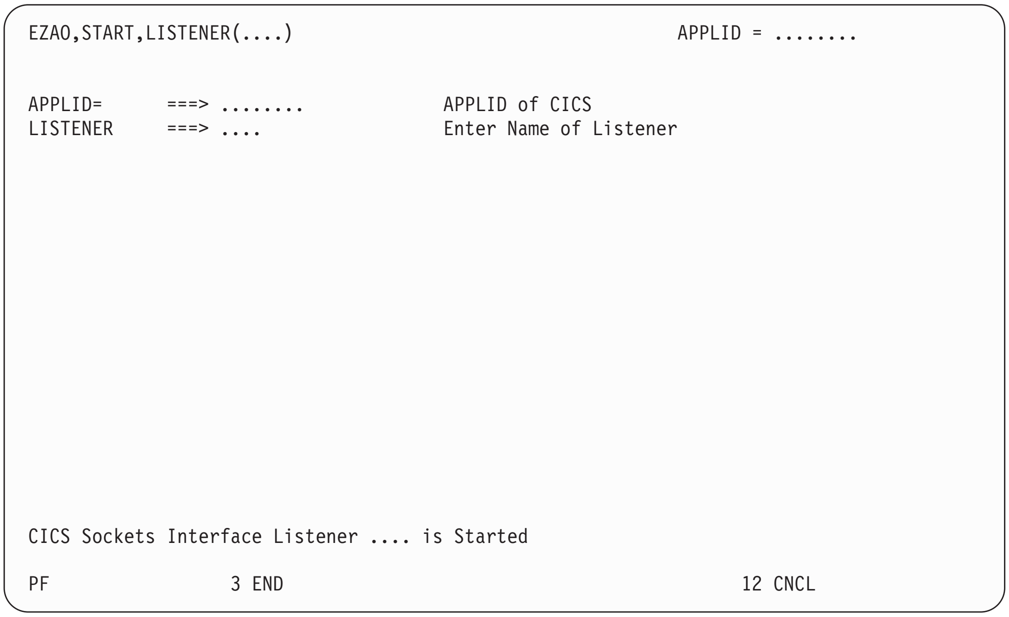
If you enter START LISTENER, the following screen is displayed.



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


APPLID=      ===> ........            APPLID of CICS
LISTENER     ===> ....                Enter Name of Listener
























PF                3 END                                            12 CNCL
```

Figure 75. EZAO START LISTENER screen

After you enter the Listener name, the Listener is started. The following screen is displayed; the results appear in the message area.

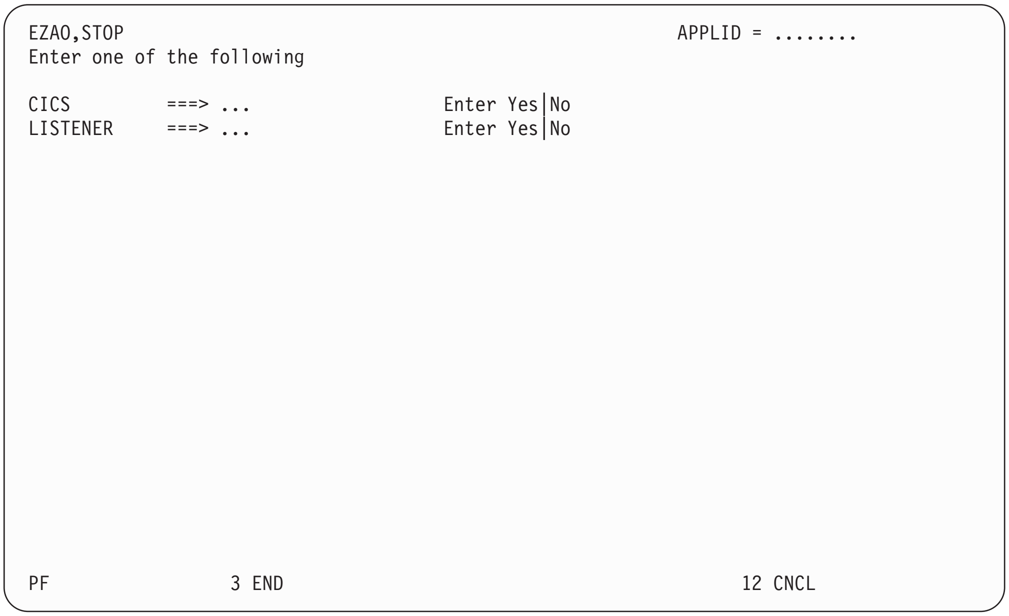


```
EZAO,START,LISTENER(....)                                    APPLID = ........


APPLID=      ===> ........            APPLID of CICS
LISTENER     ===> ....                Enter Name of Listener




















CICS Sockets Interface Listener .... is Started

PF                3 END                                            12 CNCL
```

Figure 76. EZAO START LISTENER result screen

STOP function

The STOP function is used to stop either the CICS Sockets Interface or a Listener within the interface. If the interface is stopped, all Listeners will be stopped before the interface is stopped. If you enter STO on the previous screen or enter EZAO STO on a blank screen, the following screen is displayed.

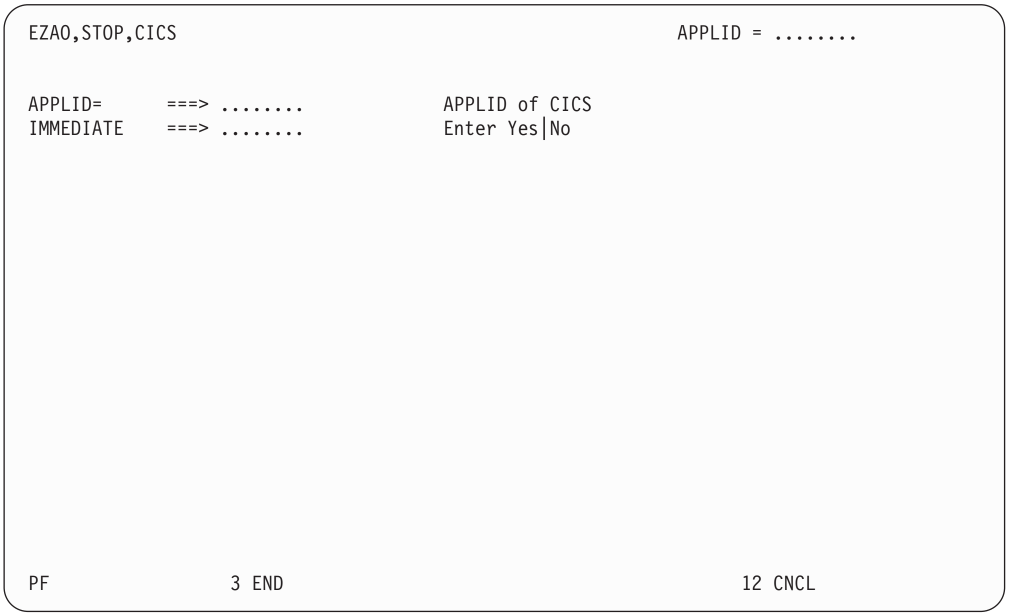


```
EZAO,STOP                                                   APPLID = ........
Enter one of the following

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















PF                3 END                                    12 CNCL
```

Figure 77. 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 78. EZAO STOP CICS screen

Two options are available to stop CICS TCP/IP:

IMMEDIATE=NO
: This should be used in most cases, because it causes the graceful termination of the interface. It has the following effects on applications using this API:

- The Listener transaction (CSKL) quiesces after a maximum wait of 3 minutes provided that no other socket applications are active or suspended.

- If there are active or suspended sockets applications, the Listener will allow them to continue processing. When all of these tasks are completed, 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

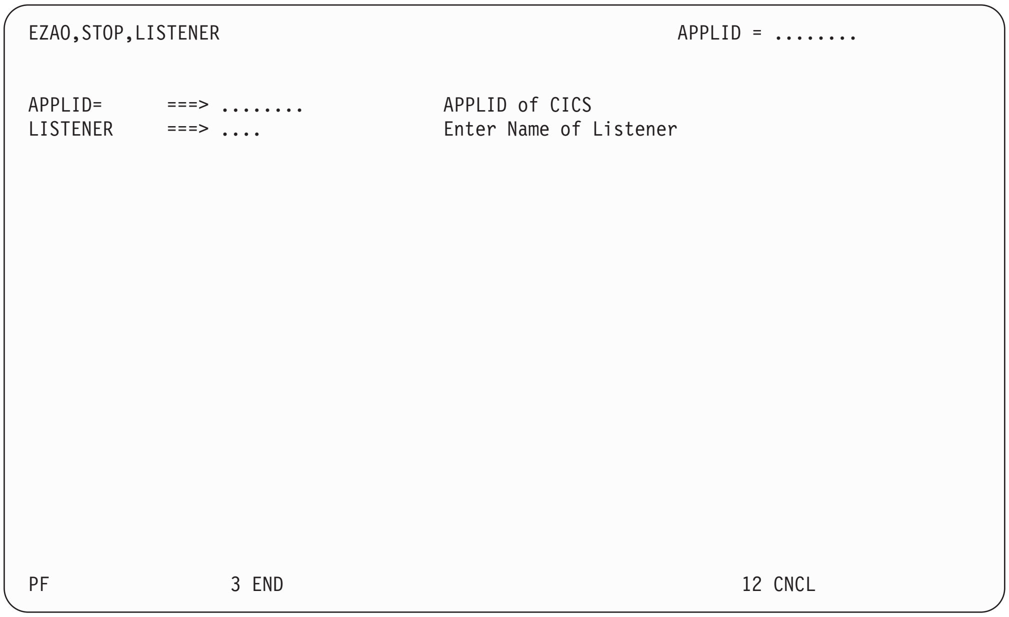
This option is reserved for unusual situations and causes the abrupt termination of the interface. It has the following effect on applications using this API:

- It force purges the master server (Listener) CSKL.
- It denies access to the API for all CICS tasks. Tasks that have successfully called the API previously will 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 will be attempted. The screen redisplays; the results appear in the message line.

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 79. EZAO STOP LISTENER screen

When you enter the Listener named, that Listener will be stopped. The screen redisplays; the results appear in the message line.

Starting/stopping CICS TCP/IP with program link

You can start or stop the CICS Sockets Interface by issuing an EXEC CICS LINK to program EZACIC20. Make sure you include the following steps in the LINKing program:

1. Define the COMMAREA for EZACIC20. This can be done by including the following instruction within 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

   P20OBJ

   C CICS Sockets Interface

   L Listener

   P20LIST
   Name of Listener if this is Listener initialization/termination.

3. Issue the EXEC CICS LINK to program EZACIC20. EZACIC20 *will not* return until the function is complete.
4. Check the P20RET field for the response from EZACIC20.

Note: The following user abend codes may be issued by EZACIC20:

- E20L is issued if the CICS Sockets Interface is not in startup or termination and no COMMAREA was provided.
- E20T is issued if CICS is not active.

Chapter 5. Writing your own Listener

The revised CICS Sockets Interface provides a structure which supports up to 255 Listeners. These Listeners may be multiple copies of the IBM-supplied Listener, user-written Listeners, or a combination of the two. You may choose to run without a Listener as well.

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 don't 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 Listener" on page 106.

Prerequisites

Some installations may 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 Sockets Interface is required.

Using IBM's environmental support

A user-written Listener may 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.
- In the program, define an input area for 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 RACF update access to the EXITPROGRAM. Failure to have at least the UPDATE access to the EXITPROGRAM security class will cause the IP CICS Sockets Interface and Listener to either not start when starting or not stop when stopping.
- 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 configuration macro or the EZAC transaction. The user-written Listener may use this information selectively, but it is highly recommended it uses the port, backlog, and number of sockets data.

  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 will result in file errors in certain shared-file situations.

Requirement: Use of the EXEC CICS ENABLE command requires RACF update access to EXITPROGRAM. Failure to have at least the UPDATE access to the EXITPROGRAM security class will cause the IP CICS Sockets 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 of x’04’ (active) so that the CICS Sockets Interface is aware that the Listener is active. Otherwise the CICS sockets Listener termination logic will bypass 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 an EXEC CICS RETURN 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 then close the passive (listen) socket.
  2. Complete 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 an EXEC CICS RETURN.
- 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 will cause 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 may be checked by looking for a X’40’ in the first byte (post bit).

  This SELECTEX should 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. This is accomplished by issuing the INITAPI with the letter *L* in the last byte of the subtask name. This allows the user-written Listener to implement the termination and detach logic the same way 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
```

An appropriate value to move into LCASTAT would be LCASTATP (B'00000100') when the user-written Listener starts. This will allow the CICS socket logic to correctly post the LCATECB during both deferred and immediate termination.

WLM registration and deregistration for sysplex connection optimization

If you are writing your own Listener(s), an interface to EZACIC12 is available and can be used for registration and deregistration. The registration and deregistration should be done at the same times the IBM Listener does it. It is important to deregister for any termination situation since the Workload Manager will 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. The linking program (Listener) builds a COMMAREA for EZACIC12. The format of this COMMAREA is described below and, for assembler use, issuing the macro EZACICA TYPE={CSECT|DSECT},AREA=P12 will provide 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 Deregistration.

The same value output from Registrations should be input for the associated Deregistration. 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 will pad it to the right with blanks to meet the WLM requirement.

Chapter 6. Application programming guide

This chapter 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 chapter are:

- Four setups for writing CICS TCP/IP applications:
  - 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

Chapter 7, "C language application programming", on page 119 describes the C language calls that can be used with CICS.

Chapter 8, "Sockets extended application programming interface (API)", on page 173 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.

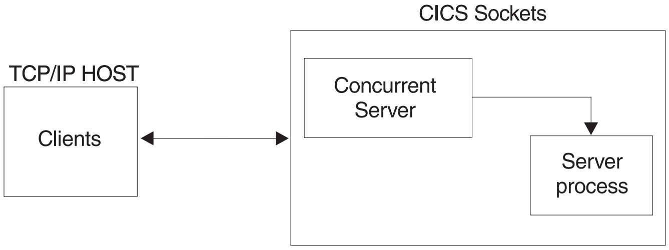
Note: Appendix A, "Original COBOL application programming interface (EZACICAL)", on page 307 provides reference information on the EZACICAL API for COBOL and assembler language. This interface was made available in a prior release of TCP/IP Services and is being retained in the current release for compatibility. For the best results, however, use the Sockets Extended API whenever possible. It is described in Chapter 8, "Sockets extended application programming interface (API)", on page 173.

Writing CICS TCP/IP applications

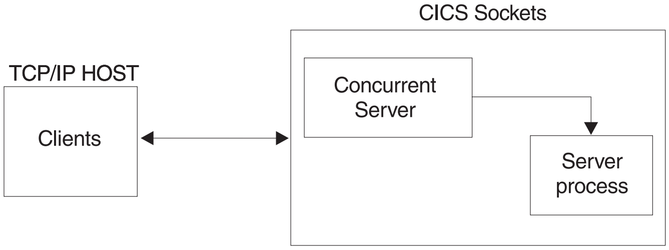
Chapter 1, "Introduction to CICS TCP/IP", on page 1 describes the basics of TCP/IP client/server systems and the two types of server: iterative and concurrent. This chapter considers in detail four TCP/IP setups in which CICS TCP/IP applications are used in various parts of the client/server system.

The setups are:

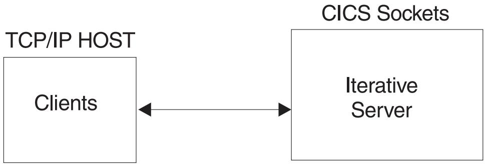
- **The client-Listener-child server application set**. The concurrent server and child server processes run under CICS TCP/IP. The concurrent server is the supplied Listener transaction. The client might be running TCP/IP under one of the various UNIX operating systems such as AIX®.



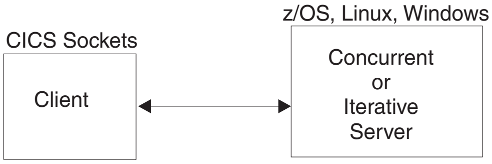
- **Writing your own concurrent server**. This is the same setup as the first except that a user-written concurrent server is being used instead of the IBM Listener.



- **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 119, Chapter 8, “Sockets extended application programming interface (API)”, on page 173, and Appendix A, “Original COBOL application programming interface (EZACICAL)”, on page 307.

1. The client-Listener-child-server application set

Figure 80 on page 97 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.

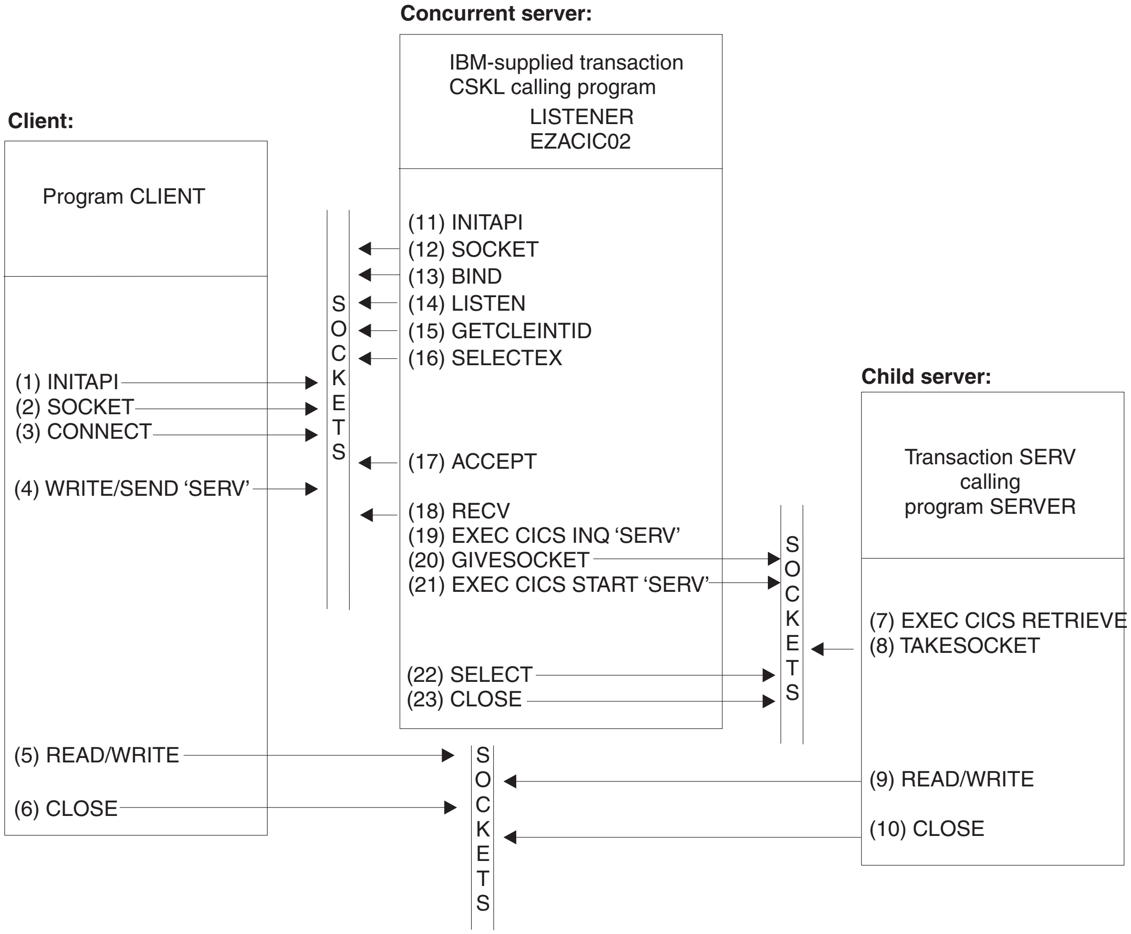


Figure 80. The sequence of sockets calls

Client call sequence

Table 5 explains the functions of each of the calls listed in Figure 80.

Table 5. 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 5. 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 the TCP/IP Internet domain (2 in COBOL, AF_INET in C or 19 in COBOL, AF_INET6 in C). The type can be stream sockets (1 in COBOL, SOCK_STREAM in C), or datagram sockets (2 in COBOL, SOCK_DGRAM in C). 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 (obtained above) 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 99, where the Listener calls in Figure 80 are explained.

Child server call sequence

Table 6 explains the functions of each of the calls listed in Figure 80 on page 97.

Table 6. 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 7 explains the functions of each of the steps listed in Figure 80 on page 97.

Table 7. Calls for the concurrent server application

(11) INITAPI	Connects the application to TCP/IP, as in Table 5.
(12) SOCKET	This obtains a socket, as in Table 5.
(13) BIND	Once 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 will be 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 will return when an incoming CONNECT call is received or when LCATECB was posted because immediate termination was detected, or else will time 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 108, the parameters `LSTN-NAME` and `LSTN-SUBNAME` define the Listener.

Table 7. Calls for the concurrent server application (continued)

(22) SELECTEX [9]	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 81 shows the sequence of socket calls involved in a simple client-iterative server setup.

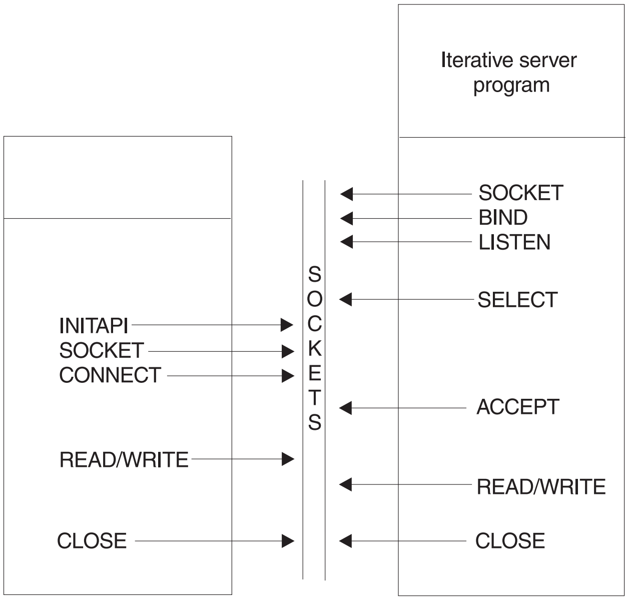


Figure 81. Sequence of socket calls with an iterative server

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.
3. The ACCEPT call obtains a new socket (1). Socket 1 is used to handle the transaction. Once this completed, socket 1 closes.

[9] This SELECTEX is the same as the SELECTEX call in Step 16. They are shown as two calls to clarify the functions being performed.

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 82 shows the sequence of calls in a CICS client-remote server setup. The calls are similar to the previous examples.

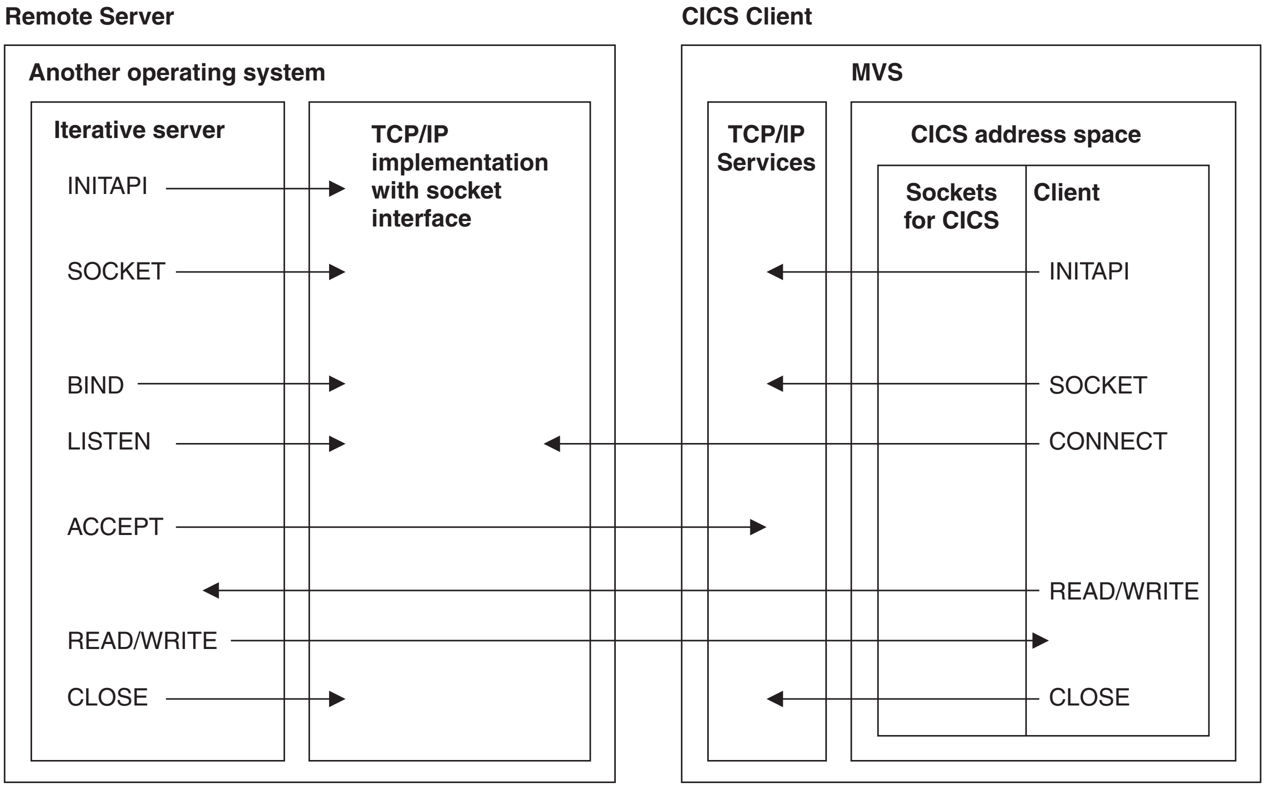


Figure 82. Sequence of socket calls between a CICS client and a remote iterative server

Figure 82 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 80 on page 97.

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 82, 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 82, 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 once. 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 on page 104).

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 on page 104).

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 will fail.

For COBOL, PL/1, and assembler language programs

The address structure of an IPv4 Internet socket address should be defined as follows:

Parameter	Assembler	COBOL	PL/1
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/1
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 13 on page 122.

MVS address spaces

Figure 83 on page 104 shows the relationship between TCP/IP and CICS in terms of MVS address spaces.

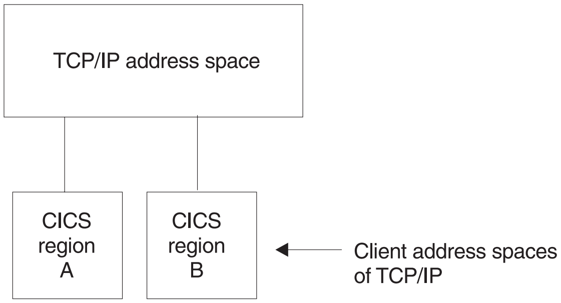


Figure 83. MVS address spaces

Within each CICS region, server and client processes will be 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. Since 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 8. 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 8. CLIENTID structures

C structure	COBOL structure

`struct clientid {`
`    int       domain;`
`    char      name[8];`
`    char      subtaskname[8];`
`    char      reserved[20];`
`};`
	`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).`

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 84.

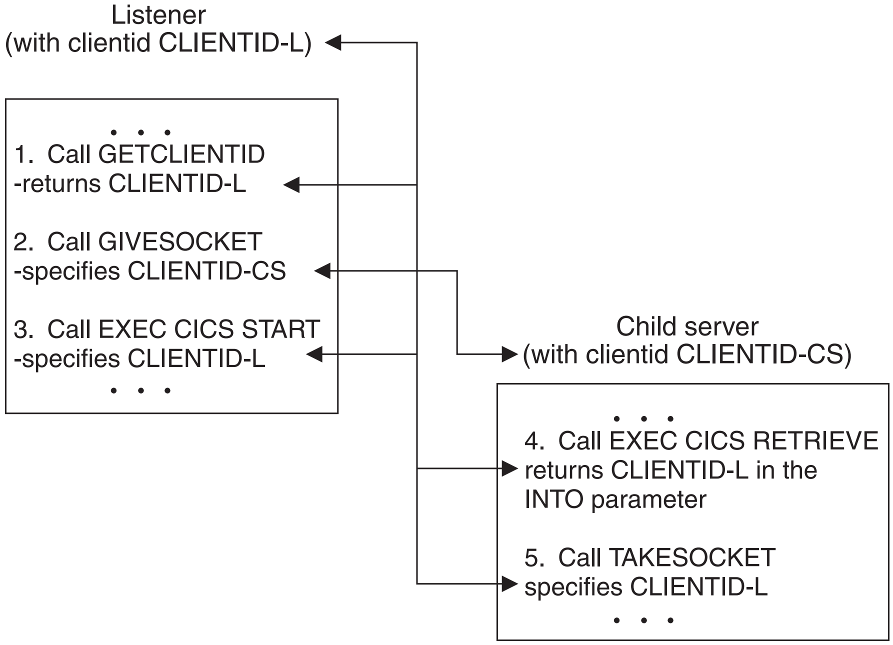


Figure 84. Transfer of CLIENTID information

Figure 84 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 it will get 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 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 may 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:

1. 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.
2. 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.
3. When an incoming connection request arrives, the Listener accepts it and obtains a new socket to pass to the CICS child server application program.
4. For the standard Listener, it 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 107. For the enhanced Listener, it starts the CICS child server transaction based on information in the Configuration file.

5. 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.

```
►►──tran──,──┬────────────────┬──,──┬────┬──,──┬────────┬──────────────────►◄
             └─client-in-data─┘     ├─IC─┤     └─hhmmss─┘
                                    ├─ic─┤
                                    ├─TD─┤
                                    └─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 [10] 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
Optional. Startup type that can be either `IC` for CICS interval control or `TD` for CICS transient data. These can also be entered in lowercase (`ic` or `td`). If this field is left blank, startup is immediate.

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.

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" on page 108. Note that 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.

[10] See "Writing your own security/transaction link module for the Listener" on page 114

Example	Listener response
`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 will be raised if the LENGTH is smaller than the amount of data sent. Coding a HANDLE condition will allow 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 9 on page 109 shows the format of the Listener output data area passed to the child server through a standard Listener.

Table 9. 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)
Filler	+55	1-byte character	Unused byte for fullword alignment
Socket address structure	+56	28 bytes	
Addressing family	+56	Halfword binary	Will be 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  FILLER              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).
```

Example of PL/1 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 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);
```

Example of PL/1 layout of the Listener output format - Standard Listener with an IPv6 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 FILLER_1                   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);
```

Example of assembler layout of the Listener output format - Standard Listener supporting both an IPv4 and 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
         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
```

Example of C structure 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   reserved1[1];
             union {
               struct sockaddr_in sin;
               struct sockaddr_in6 sin6;
             } sockaddr_in_parm;
             char   reserved2[68];
}
```

Table 10 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 10. 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)
Filler	+55	1-byte character	Unused byte for fullword alignment
Socket address structure	+56	28 bytes	
Addressing family	+56	Halfword binary	Will be 2 to indicate AF_INET or 19 to indicate AF_INET6

Table 10. Listener output format - Enhanced Listener (continued)

Description	Offset	Format	Value
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 will be 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  FILLER              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).

where xxx is at least equal to the largest MSGLENgth parameter for the Listeners
that can start this application.
```

Example of PL/1 layout of the Listener output format - Enhanced 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 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),
      2 CLIENT_IN_DATA_LENGTH      FIXED BIN(15),
      2 CLIENT_IN_DATA_2           CHAR(xxx);
```

Example of PL/1 layout of the Listener output format - Enhanced Listener with an IPv6 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 FILLER_1                   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);
```

Example of assembler layout of the Listener output format - Enhanced Listener supporting both an IPv4 and 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
         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
```

Example of C structure 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   reserved1[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];
}
```

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 with this release). You can write this program in COBOL, PL/I, or assembler language and must provide an appropriate entry in the CICS program processing table (PPT).

Specifying in EZAC: Specify the name of the security/transaction module in the SECexit field in Alter or Define. If you do not name the module, CICS will assume you do not have one. See Figure 56 on page 62 for more information.

Just before the 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.

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 11 shows the data area used by the security/transaction module.

Table 11. 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 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 (MSGLENTH 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)
Reserved	+40	4-byte character	Reserved for IBM use.
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. Will contain 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	Switch: **1** Permit the transaction **Not 1** Prohibit the transaction

Table 11. Security/transaction exit data (continued)

Description	Offset	Format	Value
Switch-2	+61	1-byte character	Switch: **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 termid to be associated with the new task.
Socket descriptor	+66	Halfword binary	Current socket descriptor.
User ID	+68	8-byte character	In CICS V1R5 and higher, a userid can be returned so that it is associated with the new task. This is mutually exclusive from termid.
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.
Reserved	+122	40 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.

Table 11. Security/transaction exit data (continued)

Description	Offset	Format	Value

Notes:

1. The security/user exit can change the value of the following fields:
   - CICS transaction identifier
   - Data area
   - Action
   - Interval control time
   - Address family
   - Client's port
   - Client's IPv4 address
   - Switch
   - Terminal identification (output only)
   - User ID (output only)
   - Client's IPv6 address
   - Client's Scope ID
   - Data length
   - Data area -2
2. Though the security exit can alter the contents of Data area, Data length, and Data area -2 when PEEK=YES, the changed values will not be reflected to the child server in the Listener Input data. The child server must read the data itself if the Listener is configured PEEK=YES.

Use the EZACICSX assembler macro contained in the SEZACMAC dataset to format the security/user exit COMMAREA pass by the Listener.

Table 12 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 12. 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
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

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, refer to EZACIC04, EZACIC05, and EZACIC06, EZACIC14, and EZACIC15 in Chapter 8, “Sockets extended application programming interface (API)”, on page 173.

Chapter 7. C language application programming

This chapter describes the C language API provided by CICS TCP/IP.

The chapter is organized under following headings:

- "C socket library" lists the required header files and explains how to make them available to your programs.
- "C socket compilation" on page 120 shows how to compile a C Socket program that contains calls to Sockets for CICS.
- "Structures used in socket calls" on page 122 lists data structures used in C language socket calls.
- "The ERRNO variable" on page 124 describes the use of a global variable used by the socket system to report errors.
- "C socket calls" on page 124 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 chapter, 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
```

The files are in the *hlq*.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 120). 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
```

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 85 on page 121 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 *hlq*.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).
- **4** In the LKED step you must concatenate the *hlq*.SEZATCP and *hlq*.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, refer to *z/OS C/C++ User's Guide* and *z/OS Communications Server: IP Application Programming Interface Guide*.
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 85. 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 85. 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 13 shows the structures used by the calls described in this chapter.

Table 13. C structures

C structure	Format

clientid
 	
Used in many calls
	`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;`
	`};`

Table 13. C structures (continued)

C structure	Format

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;
	};

If_NameIndex
Used in the if_freenameindex(), if_indextoname(), if_nameindex(), and if_nametoindex()
	struct if_nameindex {
	unsigned int if_index;
	char * if_name;
	};

linger
Used in the get/setsockopt() calls only
	struct linger {
	int l_onoff;
	int l_linger;
	};

ip_mreq
Used in the setsockopt() call only
	struct ip_mreq {
	struct in_addr imr_multiaddr;
	struct in_addr imr_interface;
	};

ipv6_mreq
Used in the setsockopt() call only
	struct ipv6_mreq {
	struct in6_addr ipv6mr_multiaddr;
	unsigned int ipv6mr_interface;
	};

sockaddr_in
Used in many calls
	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];
	};

Table 13. C structures (continued)

C structure	Format

sockaddr_in6
 	
Used in many calls
	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;
	};

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;
	};

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 section contains guidance for each C socket call supported by CICS TCP/IP.

For syntax, parameters, and other reference information for each C socket call, refer to *z/OS Communications Server: IP Programmer's 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 will accept 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

```
#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 13 on page 122.

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 see which error has occurred, check the *errno* global variable, which will be set to a return code. Possible codes include:

EBADF
The *s* parameter is not a valid socket descriptor.

EFAULT
Using *name* and *namelen* would result 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 a similar effect to the application specifying an explicit IP address on the bind() function. For more information, refer to *z/OS Communications Server: IP Configuration Reference*.

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 containing 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 13 on page 122.

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 see which error has occurred, check the *errno* global variable, which will be set to a return code. Possible codes include:

EADDRINUSE
The address is already in use. See the SO_REUSEADDR option described under “getsockopt(), setsockopt()” on page 146 for more information.

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* would result 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

```
#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 see which error has occurred, check the *errno* global variable, which will be 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() once only.

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 containing 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 13 on page 122.

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 see which error has occurred, check the *errno* global variable, which will be 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* would result 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

```
#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 see which error has occurred, check the *errno* global variable, which will be 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

```
#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 see which error has occurred, check the errno global variable, which will be 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() will overwrite the buffer containing the text string.

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() will return 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

```
#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.

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 containing 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 containing 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:

Field **Description**

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 IN6ADDR_ANY, for an IPv6 address.

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 will accept IPv4-mapped IPv6 addresses. When the AI_ALL flag is not also specified, if no IPv6 addresses are found, a query is made for IPv4 addresses. If IPv4 addresses are found, they are returned as IPv4-mapped IPv6 addresses. If the *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 will accept 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 will accept 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 will accept any protocol family. The value of a decimal 0 indicates AF_UNSPEC. The value of a decimal 2 indicates AF_INET and the value of a decimal 19 indicates AF_INET6.

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

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 will accept 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 will 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 address information structure contains the following fields:

Field **Description**

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 containing 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 see which error has occurred, check the *errno* global variable, which will be 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

```
#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 see which error has occurred, check the *errno* global variable, which will be set to a return code. Possible codes include:

EFAULT
Using the *clientid* parameter as specified would result 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. Refer to *z/OS Communications Server: IP Configuration Guide* for information on configuring the resolver and using local host tables.

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 containing 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 on the hostent structure, see Figure 96 on page 200. 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. Refer to *z/OS Communications Server: IP Configuration Guide* for information on configuring the resolver and using local host tables.

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 on the hostent structure, see Figure 98 on page 202. 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

```
#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.

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 filled with the host name.

namelen
The length of *name*.

Return values

The value 0 indicates success; the value −1 indicates an error. To see which error has occurred, check the *errno* global variable, which will be set to a return code. Possible codes include:

EFAULT
The *name* parameter specified an address outside of the caller's address space.

getnameinfo()

The getnameinfo() call returns the node name and service location of a socket address that is specified in the call.

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 13 on page 122 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 link scope for an IPv6 address as an interface index. The resolver ignores the *sin6_scope_id* field.

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, which can be a maximum of 256 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 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.

This is an optional field, but if the field is not 0, you must also specify *hostlen*. One or the other or both of the *host* /*hostlen* or *serv*/*servlen* parameters are required. An error occurs if both are omitted.

hostlen A field that contains the length of the *host* storage used to contain the returned resolved host name. *hostlen* must be equal to or greater than the length of the longest host name to be returned. getnameinfo() returns the host name up to the length specified by *hostlen*. If *hostlen* is 0 on input, then the resolved host name is not returned.

This is an optional field, but if the field is not 0, you must also specify *host*. One or the other or both of the *host* /*hostlen* or *serv*/*servlen* parameters are required. 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 field is not 0, you must also specify *servlen*. One or the other or both of the *host* /*hostlen* or *serv*/*servlen* parameters are required. An error occurs if both are omitted.

servlen A field that contains the length of the *serv* 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*. If *servlen* is 0 on input, the service name information is not returned.

This is an optional field, but if the field is not 0, you must also specify *serv*. One or the other or both of the *host* /*hostlen* or *serv*/*servlen* parameters are required. 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
Only return the numeric form of host's address.

NI_NAMEREQD
Return an error if the host's name cannot be located.

NI_NUMERICSERV
Only return 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.

Return values

The value 0 indicates success; the value -1 indicates an error. To see which error has occurred, check the *errno* global variable, which will be 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

```
#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 containing 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 containing 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 see which error has occurred, check the *errno* global variable, which will be 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 would result 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 would cause 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 would cause 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

```
#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 will be 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 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 see which error has occurred, check the *errno* global variable, which will be 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 would result 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
- Setting the multicast interface
- Setting the IP time-to-live of outgoing multicast datagrams
- Looping back multicast datagrams

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 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" on page 148.

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 containing 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 option is recognized at the IPPROTO_IP level:

Option **Description**

IP_ADD_MEMBERSHIP

Use this option to enable 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 *optval* to the *ip_mreq* 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 cannot be specified with getsockopt().

IP_DROP_MEMBERSHIP

Use this option to enable an application to exit a multicast group. This is an IPv4 only socket option.

For setsockopt(), set *optval* to the *ip_mreq* 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 cannot be specified with getsockopt().

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.

For setsockopt(), set *optval* to an IPv4 interface address.

For getsockopt(), *optval* contains an IPv4 interface address.

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.

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
Use this option to control or determine 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.

The following option is recognized at the IPPROTO_IPV6 level:

Option **Description**

IPV6_JOIN_GROUP
Use this option to control 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
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.

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
Use to set or obtain 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 will be 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
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.

For setsockopt(), set *optval* to a value containing an IPv6 interface index.

For getsockopt(), *optval* contains an IPv6 interface index.

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.

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

Use this option to set or obtain 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 will be 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

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.

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 option is recognized at the TCP level:

Option **Description**

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 once 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:

Option **Description**

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
Use this option to set or determine 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 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.

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 IN6ADDR_ANY, refer 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 see which error has occurred, check the *errno* global variable, which will be set to a return code. Possible codes include:

EBADF
The *s* parameter is not a valid socket descriptor.

EFAULT
Using *optval* and *optlen* parameters would result 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.

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

```
#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 see which error has occurred, check the *errno* global variable, which will be 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 would result 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

```
#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 containing 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

```
#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 containing an interface index.

ifname
A buffer used to contain the name of the index value specified in the *ifindex* parameter.

Return values

Possible return 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

```
#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 containing information about each system interface. See 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

```
#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 containing 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 Invalid parameter was 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()

Use the inet_ntop() function to convert numeric IP addresses to their printable form.

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 containing 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()

Use the inet_pton() function to convert IP addresses from presentable text form to numeric form.

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

```
#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 eight-character ID, which should be the 4-byte packed EIBTASKN value in the EIB plus three character 0's and a unique displayable character.

Note: Using L as the last character in the subtaskid parameter causes the tasking mechanism to assume the CICS transaction is a Listener and schedule it using an attached task.

Return values

A positive value indicates success; a value of −1 indicates an error. To see which error has occurred, check the *errno* global variable, which will be 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.

Format

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <bsdtypes.h>
#include <ioctl.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:

Command
Description

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 containing 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.

Return values

The value 0 indicates success; the value −1 indicates an error. To see which error has occurred, check the *errno* global variable, which will be 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

```
#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.

Return values

The value 0 indicates success; the value −1 indicates an error. To see which error has occurred, check the *errno* global variable, which will be 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

```
#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 see which error has occurred, check the *errno* global variable, which will be set to a return code. Possible codes include:

EBADF
s is not a valid socket descriptor.

EFAULT
Using the *buf* and *len* parameters would result 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

```
#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 sees 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 see which error has occurred, check the *errno* global variable, which will be set to a return code. Possible codes include:

EBADF
s is not a valid socket descriptor.

EFAULT
Using the *buf* and *len* parameters would result 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

```
#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 sees 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 containing 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 see which error has occurred, check the *errno* global variable, which will be set to a return code. Possible codes include:

EBADF
s is not a valid socket descriptor.

EFAULT
Using the *buf* and *len* parameters would result 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 would block on a read() call, only one socket could be read at a time. Setting the sockets nonblocking would solve this problem, but would require polling each socket repeatedly until data became available. The select() call allows you to test several sockets and to execute a subsequent I/O call only when one of the tested sockets is ready, thereby ensuring that the I/O call will not block.

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 will not block.
  - A connection has been requested on that socket.
- When a socket is ready to write, TCP/IP can accommodate additional output data. If TCP/IP can accept additional output for a given socket, a write operation on that socket will not block.
- When an exception condition has occurred on a specified socket, it is an indication that a takesocket() has occurred for that socket.

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 will contain 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. Once a socket is selected for write(), the program can determine the amount of TCP/IP buffer space available by issuing the getsockopt() call with the SO_SNDBUF option.

To test whether any of several sockets is ready for writing, set the 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 will return 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 will test 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:

```
#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 [11] with each socket to test its status.

[11] See *z/OS Communications Server: IP Programmer's Reference* for details.

The value −1 indicates an error. To see which error has occurred, check the *errno* global variable, which will be 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

```
#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 containing 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.

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 see which error has occurred, check the *errno* global variable, which will be set to a return code. Possible codes include:

EBADF
s is not a valid socket descriptor.

EFAULT
Using the *buf* and *len* parameters would result 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

```
#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 containing 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 see which error has occurred, check the *errno* global variable, which will be set to a return code. Possible codes include:

EBADF
s is not a valid socket descriptor.

EFAULT
Using the *buf* and *len* parameters would result 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.

setsockopt()

See "getsockopt(), setsockopt()" on page 146.

shutdown()

The shutdown() call shuts down all or part of a duplex connection.

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 see which error has occurred, check the *errno* global variable, which will be 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. Once a socket is passive, it cannot be used to initiate connection requests.

SOCK_DGRAM supports datagrams (connectionless messages) of a fixed maximum length. Transimission 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

```
#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 see which error has occurred, check the *errno* global variable, which will be 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

```
#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 see which error has occurred, check the *errno* global variable, which will be 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 would result 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()

The write() 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 1000 bytes. Therefore, applications using stream sockets should place this call in a loop, calling this function until all data has been sent.

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 see which error has occurred, check the *errno* global variable, which will be set to a return code. Possible codes include:

EBADF
s is not a valid socket descriptor.

EFAULT
Using the *buf* and *len* parameters would result 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 section 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 *)
```

Macro **Description**

IN6_IS_ADDR_UNSPECIFIED
Returns true if the address is an unspecified IPv6 address. 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 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 will cause 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 application programming interface (API)

This section 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.

For more information about sockets, refer to the *UNIX Programmer's Reference Manual*.

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 141 on page 308.

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 will allow 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/1 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/1 language call format

The following is the ’EZASOKET’ call format for PL/1 language programs.

```
►►──CALL EZASOKET (SOC-FUNCTION──parm1, parm2, ...──ERRNO RETCODE);──────────────────►◄
```

The following is the ’EZACICSO’ call format for the PL/1 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 chapter 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 86 shows examples of storage definition statements for COBOL, PL/1, 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 86. Storage definition statement examples

Error messages and return codes

For information about error messages, refer to *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 on page 337.

Code CALL instructions

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

The following requirements apply to this call:

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
Amode:	31-bit or 24-bit **Note:** See “Addressability mode (Amode) considerations” under “Environmental restrictions and programming requirements” on page 173.
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 87 on page 178 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 87. ACCEPT call instructions example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 337 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, refer to *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 **Note:** See “Addressability mode (Amode) considerations” under “Environmental restrictions and programming requirements” on page 173.
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 88 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 88. BIND call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 337, 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 281 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 will continue to attempt to complete data transmission for a specified period of time after the CLOSE call is issued. See SO-LINGER in the description of "SETSOCKOPT" on page 271.
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 **Note:** See "Addressability mode (Amode) considerations" under "Environmental restrictions and programming requirements" on page 173.

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 89 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 89. CLOSE call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 337, 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 since the socket is nonblocking, the CONNECT call returns normally.

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

The completion cannot be checked by issuing a second CONNECT. For more information, see “SELECT” on page 253.

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 173.
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 90 on page 185 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 90. CONNECT call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 337 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 228 for another way to control a socket's blocking mode.

Values for Command which are supported by the UNIX Systems Services fcntl callable service will also be accepted. Refer to 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 **Note:** See “Addressability mode (Amode) considerations” under “Environmental restrictions and programming requirements” on page 173.
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 91 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 91. FCNTL call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 337 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 92 on page 189 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 **Note:** See “Addressability mode (Amode) considerations” under “Environmental restrictions and programming requirements” on page 173.
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 92 on page 189 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 92. FREEADDRINFO call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 175.

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 337, 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 93 on page 190 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 **Note:** See "Addressability mode (Amode) considerations" under "Environmental restrictions and programming requirements" on page 173.
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 93 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 93. GETADDRINFO call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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.

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 may 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.

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.
- 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 NODE argument is not specified, the IP address portion of the socket address structure pointed to by the returned RES

will be set to INADDR_ANY for an IPv4 address or IN6ADDR_ANY for an IPv6 address.

- 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 NODE argument is not specified, the NAME pointed to by the returned RES will be 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 will accept 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 will accept any protocol family. The value of a decimal 0 indicates AF_UNSPEC. The value of a decimal 2 indicates AF_INET and the value of a decimal 19 indicates AF_INET6.

SOCTYPE

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

The following are the acceptable socket types:

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 will accept 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 address information structure contains the following fields:

Field	**Description**
FLAGS	A fullword binary field that is not used as output.
AF	A fullword binary field. The value returned in this field may be used as the AF argument on the SOCKET call to create a socket suitable for use with the returned address NAME.

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

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

NAMELEN A fullword binary field. The length of the NAME socket address structure. The value returned in this field 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 337, 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/1 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 223 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 **Note:** See “Addressability mode (Amode) considerations” under “Environmental restrictions and programming requirements” on page 173.
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 94 on page 198 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 94. GETCLIENTID call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 337, 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. Refer to *z/OS Communications Server: IP Configuration Guide* for information on 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 **Note:** See “Addressability mode (Amode) considerations” under “Environmental restrictions and programming requirements” on page 173.
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 95 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 95. GETHOSTBYADDR call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 337 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 96.

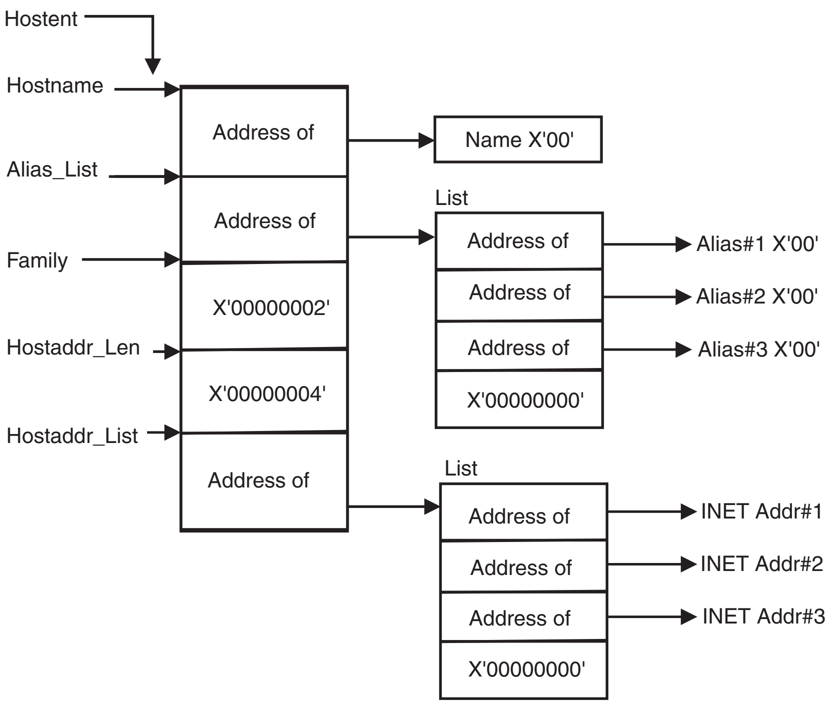


Figure 96. HOSTENT structure returned by the GETHOSTBYADDR call

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/1 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 296. 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. Refer to *z/OS Communications Server: IP Configuration Guide* for information on 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 **Note:** See “Addressability mode (Amode) considerations” under “Environmental restrictions and programming requirements” on page 173.
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 97 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 97. GETHOSTBYNAME call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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

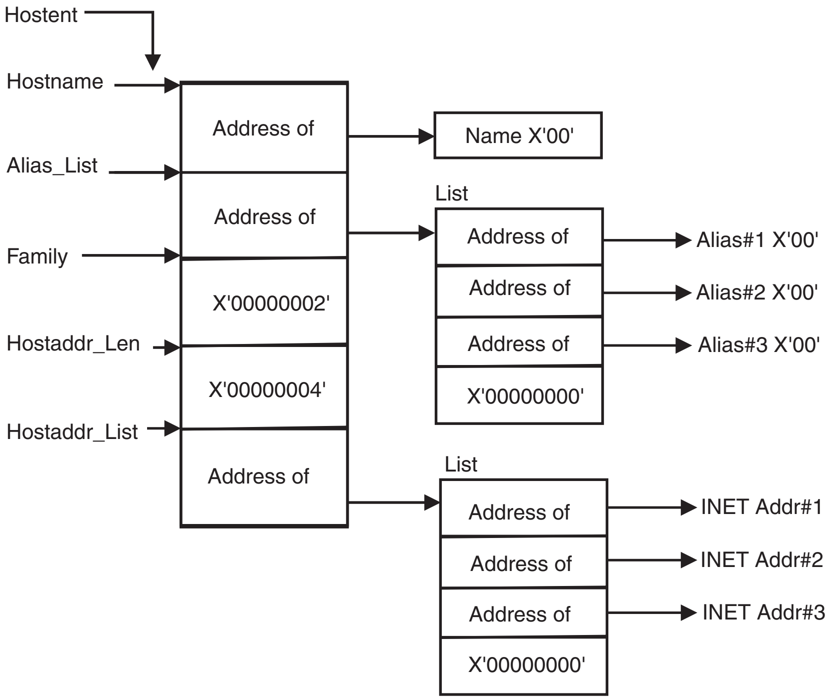


Figure 98. HOSTENT structure returned by the GETHOSTYBYNAME call

GETHOSTBYNAME returns the HOSTENT structure shown in Figure 98. 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/1 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 296. 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 **Note:** See “Addressability mode (Amode) considerations” under “Environmental restrictions and programming requirements” on page 173.
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 99 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 99. GETHOSTID call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 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 173.
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 100 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 100. GETHOSTNAME call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 175.

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.

Parameter values returned to the application

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

ERRNO
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B. Return codes on page 337 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 **Note:** See "Addressability mode (Amode) considerations" under "Environmental restrictions and programming requirements" on page 173.
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 101 on page 206 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.

* 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 101. GETNAMEINFO call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 specifying link scope for an IPv6 address as an interface index. The resolver ignores the SCOPE-ID field.

NAMELEN
A fullword binary field. The length of the socket address structure pointed to by the NAME argument.

HOST
On input, storage capable of holding the returned resolved host name, which can be 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 up to the storage 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 specified, you must also code HOSTLEN. Either the HOST/HOSTLEN parameter, the SERVICE/SERVLEN parameter, or both are required. An error occurs if both are omitted.

HOSTLEN
An output parameter. A fullword binary field that contains the length of the HOST storage used to contain the returned resolved host name. HOSTLEN must be equal to or greater than the length of the longest host name to be returned. GETNAMEINFO returns the host name up to the length specified by HOSTLEN. On output, HOSTLEN contains the length of the returned resolved host name. If HOSTLEN is 0 on input, then the resolved host name is not returned. This is an optional field, but if specified, you must also code HOST. Either the HOST/HOSTLEN parameter, the SERVICE/SERVLEN parameter, or both are required. 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. Either the HOST/HOSTLEN parameter, the SERVICE/SERVLEN parameter, or both are required. 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. Either the HOST/HOSTLEN parameter, the SERVICE/SERVLEN parameter, or both are required. 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	Only return 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	Only return 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.

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 337, 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 **Note:** See “Addressability mode (Amode) considerations” under “Environmental restrictions and programming requirements” on page 173.
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 102 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 102. GETPEERNAME call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 337 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following:

Value	Description
0	Successful call
−1	Check ERRNO for an error code

GETSOCKNAME

The GETSOCKNAME call returns the address currently bound to a specified socket. If the socket is not currently bound to an address, the call returns with the FAMILY field set, and the rest of the structure set to 0.

Since a stream socket is not assigned a name until after a successful call to either BIND, CONNECT, or ACCEPT, the GETSOCKNAME call can be used after an implicit bind to discover which port was assigned to the socket.

The following requirements apply to this call:

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 173.
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 103 on page 212 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 103. GETSOCKNAME call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 will be INADDR_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 will be 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 337 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 **Note:** See “Addressability mode (Amode) considerations” under “Environmental restrictions and programming requirements” on page 173.
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 104 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 104. GETSOCKOPT call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 353 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 337 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 14. 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 *hlq*.SEZAINST(CBLOCK) for the PL/1 example of IP_MREQ.
	The IP_MREQ definition for COBOL:
	`01 IP-MREQ.`
	`  05  IMR-MULTIADDR`
	`       PIC 9(8) BINARY.`
	`  05  IMR-INTERFACE`
	`       PIC 9(8) BINARY.`
		N/A

IP_DROP_MEMBERSHIP
Use this option to enable an application to exit 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 *hlq*.SEZAINST(CBLOCK) for the PL/1 example of IP_MREQ.
	The IP_MREQ definition for COBOL:
	`01 IP-MREQ.`
	`  05  IMR-MULTIADDR`
	`       PIC 9(8) BINARY.`
	`  05  IMR-INTERFACE`
	`       PIC 9(8) BINARY.`
		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.

Table 14. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
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.

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 *hlq*.SEZAINST(CBLOCK) for the PL/1 example of IPV6_MREQ.
	The IPV6_MREQ definition for COBOL:
	`01  IPV6-MREQ.`
	`   05 IPV6MR-MULTIADDR.`
	`     10  FILLER PIC 9(16)`
	`           BINARY.`
	`     10  FILLER PIC 9(16)`
	`           BINARY.`
	`   05 IPV6MR-INTERFACE PIC`
	`        9(8)    BINARY.`
		N/A

Table 14. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)

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 *hlq*.SEZAINST(CBLOCK) for the PL/1 example of IPV6_MREQ.
	The IPV6_MREQ definition for COBOL:
	`01  IPV6-MREQ.`
	`  05 IPV6MR-MULTIADDR.`
	`    10  FILLER PIC 9(16)`
	`          BINARY.`
	`    10  FILLER PIC 9(16)`
	`          BINARY.`
	`  05 IPV6MR-INTERFACE PIC`
	`       9(8)    BINARY.`
		N/A

IPV6_MULTICAST_HOPS Use to set or obtain the hop limit used for outgoing multicast packets. This is an IPv6-only socket option.	Contains a 4-byte binary value specifying the multicast hops. If not specified, then the default is 1 hop. -1 indicates use stack default. 0 - 255 is the valid hop limit range. **Note:** An application must be APF authorized to enable it to set the hop limit value above the system defined hop limit value. CICS applications cannot execute as APF authorized.	Contains a 4-byte binary value in the range from 0 to 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.

Table 14. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
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 from 0 to 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.
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.

Table 14. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
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 14. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	**SETSOCKOPT, OPTVAL (input)**	**GETSOCKOPT, OPTVAL (output)**
SO_KEEPALIVE Use this option to set or determine 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.	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_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.
	Contains an 8-byte field containing two 4-byte binary fields.
	Assembler coding:
	`ONOFF   DS F`
	`LINGER  DS F`
	COBOL coding:
	`ONOFF  PIC 9(8) BINARY.`
	`LINGER PIC 9(8) BINARY.`
	Set ONOFF to a nonzero value to enable and set to 0 to disable this option. Set LINGER to the number of seconds that TCP/IP lingers after the CLOSE is issued.
		Contains an 8-byte field containing two 4-byte binary fields.
		Assembler coding:
		`ONOFF   DS F`
		`LINGER  DS F`
		COBOL coding:
		`ONOFF  PIC 9(8) BINARY.`
		`LINGER PIC 9(8) BINARY.`
		A nonzero value returned in ONOFF indicates enabled, a 0 indicates disabled. LINGER indicates the number of seconds that TCP/IP will try to send data after the CLOSE is issued.

Table 14. 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 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 14. 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 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.

Table 14. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
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.

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, will be closed and will no longer be available for taking. If a select for the socket is outstanding, it will be posted.

The following requirements apply to this call:

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 173.
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 105 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 105. GIVESOCKET call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 will contain the application that is going to take the socket.

- If the socket-taking application is in the *same* address space as the socket-giving application (as in CICS), NAME can be specified. The socket-giving application can determine its own address space name by issuing the GETCLIENTID call.
- If the socket-taking application is in a *different* MVS address space 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 337 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 Sockets Interface is not an INITAPI or INITAPIX, then the CICS Sockets Interface will generate 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 **Note:** See "Addressability mode (Amode) considerations" under "Environmental restrictions and programming requirements" on page 173.
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 106 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 106. INITAPI call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 175.

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 will ever have open at one time. The maximum number is 65535 and the minimum number is 50. This value is used to determine the amount of memory that will be 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 Sockets 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 Sockets 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 Sockets 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.

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 337 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 15 on page 233 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 173.
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 107 on page 229 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 107. IOCTL call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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-VAL pic s9(10) binary value 3222599176.
01  SIOCGHOMEIF6-REDEF REDEFINES SIOCGHOMEIF6-VAL.
     05 FILLER       PIC 9(6) COMP.
     05 SIOCGHOMEIF6 PIC 9(8) COMP.

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 108. 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 15 on page 233.

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 109 on page 232 for the address format.

SIOCGIFBRDADDR

Requests the network interface broadcast address for a given interface name. See the NAME field in Figure 109 on page 232 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 109.

- 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.

```
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 109. 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 109 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-VAL           pic 9(10) binary value 1073804803.
      01 SIOCGIFNAMEINDEX-REDEF REDEFINES SIOCGIFNAMEINDEX-VAL.
          05 FILLER                     PIC 9(6) COMP.
          05 SIOCGIFNAMEINDEX           PIC 9(8) COMP.
      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 110. COBOL language example for SIOCGIFNAMEINDEX

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 15.

Table 15. IOCTL call arguments

COMMAND/CODE	SIZE	REQARG	SIZE	RETARG
FIONBIO X'8004A77E'	4	Set socket mode to: X'00'=blocking; X'01'=nonblocking	0	Not used
FIONREAD X'4004A77F'	0	Not used	4	Number of characters available for read
SIOCATMARK X'4004A707'	0	Not used	4	X'00' = at OOB data X'01' = not at OOB data
SIOCGHOMEIF6 X'C014F608'	20	NetConfHdr		See Figure 108 on page 231.
SIOCGIFADDR X'C020A70D'	32	First 16 bytes—interface name. Last 16 bytes—not used	32	Network interface address (See Figure 109 on page 232 for format.)

Table 15. IOCTL call arguments (continued)

COMMAND/CODE	SIZE	REQARG	SIZE	RETARG
SIOCGIFBRDADDR X'C020A712'	32	First 16 bytes—interface name. Last 16 bytes—not used	32	Network interface address (See Figure 109 on page 232 for format.)
SIOCGIFCONF X'C008A714'	8	Size of RETARG	See note.	
Note: When you call IOCTL with the SIOCGIFCONF command set, REQARG should contain the length in bytes of RETARG. Each interface is assigned a 32-byte array element and REQARG should be set to the number of interfaces times 32. TCP/IP for MVS can return up to 100 array elements.				
SIOCGIFDSTADDR X'C020A70F'	32	First 16 bytes—interface name. Last 16 bytes—not used	32	Destination interface address (See Figure 109 on page 232 for format.)
SIOCGIFNAMEINDEX X'4000F603'	4	First 4 bytes of return buffer		See Figure 110 on page 233.

Parameter values returned to the application

RETARG
Returns an array whose size is based on the value in COMMAND. See Table 15 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 337 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 111 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 111 on page 235 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 111. COBOL II example for SIOCGIFCONF

LISTEN

The LISTEN call:

- Completes the bind, if BIND has not already been called for the socket.
- Creates a connection-request queue of a specified length for incoming connection requests.

Note: The LISTEN call is not supported for datagram sockets or raw sockets.

The LISTEN call is typically used by a server to receive connection requests from clients. When a connection request is received, a new socket is created by a subsequent ACCEPT call, and the original socket continues to listen for additional connection requests. The LISTEN call converts an active socket to a passive socket and conditions it to accept connection requests from clients. Once a socket becomes passive, it cannot initiate connection requests.

The following requirements apply to this call:

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
Amode:	31-bit or 24-bit **Note:** See “Addressability mode (Amode) considerations” under “Environmental restrictions and programming requirements” on page 173.
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 112 on page 236 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 112. LISTEN call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 337 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 **Note:** See “Addressability mode (Amode) considerations” under “Environmental restrictions and programming requirements” on page 173.
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 113 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 113. NTOP call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 337 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 **Note:** See “Addressability mode (Amode) considerations” under “Environmental restrictions and programming requirements” on page 173.
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 114 on page 239 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 114. PTON call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 175.

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 will be 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 337 for information about ERRNO return codes.

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

Value	Description
0	Successful call
–1	Check ERRNO for an error code

READ

The READ call reads the data on socket s. This is the conventional TCP/IP read data operation. If a datagram packet is too long to fit in the supplied buffer, datagram sockets discard extra bytes.

For stream sockets, data is processed as streams of information with no boundaries separating the data. For example, if programs A and B are connected with a stream socket and program A sends 1000 bytes, each call to this function can return any number of bytes up to the entire 1000 bytes. The number of bytes returned will be contained in RETCODE. Therefore, programs using stream sockets should place this call in a loop that repeats until all data has been received.

Note: See "EZACIC05" on page 293 for a subroutine that will translate 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 173.
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 115 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 115. READ call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 337 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 **Note:** See “Addressability mode (Amode) considerations” under “Environmental restrictions and programming requirements” on page 173.
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 116 on page 243 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 116. 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 337 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 will be contained in RETCODE. Therefore, programs using stream sockets should place RECV in a loop that repeats until all data has been received.

If data is not available for the socket, and the socket is in blocking mode, RECV blocks the caller until data arrives. If data is not available and the socket is in nonblocking mode, RECV returns a –1 and sets ERRNO to 35 (EWOULDBLOCK). See “FCNTL” on page 186 or “IOCTL” on page 228 for a description of how to set nonblocking mode.

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

Note: See “EZACIC05” on page 293 for a subroutine that will translate 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 173.
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 117 on page 245 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 117. RECV call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 will read 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 337 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 will be contained in RETCODE. Therefore, programs using stream sockets should place RECVFROM in a loop that repeats until all data has been received.

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

If data is not available for the socket, and the socket is in blocking mode, RECVFROM blocks the caller until data arrives. If data is not available and the socket is in nonblocking mode, RECVFROM returns a −1 and sets ERRNO to 35 (EWOULDBLOCK). See “FCNTL” on page 186 or “IOCTL” on page 228 for a description of how to set nonblocking mode.

Note: See “EZACIC05” on page 293 for a subroutine that will translate 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 173.
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 118 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 118. RECVFROM call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 will read 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 337 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 **Note:** See “Addressability mode (Amode) considerations” under “Environmental restrictions and programming requirements” on page 173.
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 250 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 119. 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 119. RECVMSG call instruction example (Part 2 of 2)

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 will be cleared.

Fullword 3
A pointer to the length of the data buffer referenced in fullword 1.

In COBOL, the IOV structure must be defined separately in the Linkage section, as shown in the example.

IOVCNT
On input, a pointer to a fullword binary field specifying the number of data buffers provided for this call.

ACCRIGHTS
On input, a pointer to the access rights received. This field is ignored.

ACCRLEN
On input, a pointer to the length of the access rights received. This field is ignored.

FLAGS

A fullword binary field with values as follows:

Literal value	Binary value	Description
NO-FLAG	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 will read 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 337 for information about ERRNO return codes.

RETCODE

A fullword binary field with the following values:

Value	Description
<0	Call returned error. See ERRNO field.
0	Connection partner has closed connection.
>0	Number of bytes read.

SELECT

In a process where multiple I/O operations can occur, it is necessary for the program to be able to wait on one or several of the operations to complete.

For example, consider a program that issues a READ to multiple sockets whose blocking mode is set. Because the socket would block on a READ call, only one socket could be read at a time. Setting the sockets nonblocking would solve this problem, but would require polling each socket repeatedly until data became available. The SELECT call allows you to test several sockets and to execute a subsequent I/O call only when one of the tested sockets is ready, thereby ensuring that the I/O call will not block.

To use the SELECT call as a timer in your program, do one of the following:
- 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 **Note:** See “Addressability mode (Amode) considerations” under “Environmental restrictions and programming requirements” on page 173.

ASC mode:	Primary address space control (ASC) mode
Interrupt status:	Enabled for interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space

Defining which sockets to test

The SELECT call monitors for read operations, write operations, and exception operations:

- When a socket is ready to read, one of the following has occurred:
  - A buffer for the specified sockets contains input data. If input data is available for a given socket, a read operation on that socket will not block.
  - A connection has been requested on that socket.
- When a socket is ready to write, TCP/IP stacks can accommodate additional output data. If TCP/IP stacks can accept additional output for a given socket, a write operation on that socket will not block.
- When an exception condition has occurred on a specified socket it is an indication that a TAKESOCKET has occurred for that socket.

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 294.

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 will be 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. Once 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 will return the out-of-band data ahead of program data.

To test whether any of several sockets have an exception condition, set the ESNDMSK bits representing those sockets to 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 returning results. For efficiency, the MAXSOC parameter can be used to specify the largest socket descriptor number that needs to be tested for any event type. The SELECT call tests only bits in the range 0 through the MAXSOC 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.

Figure 120 on page 256 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 120. 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 175.

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 set to the largest socket descriptor number that is to be checked plus 1. (Remember to start counting at 0).

Note: 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. 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 will not check for read events.

WSNDMSK

A bit string sent to request write event status.

- For each socket to be checked for pending write events, the corresponding bit in the string should be set to 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 will not check for write events.

ESNDMSK

A bit string sent to request exception event status.

- For each socket to be checked for pending exception events, the corresponding bit in the string should be set to 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 will not check for exception events.

Parameter values returned to the application

RRETMSK

A bit string returned with the status of read events. The length of the string should be equal to the maximum number of sockets to be checked. For each socket that is ready to read, the corresponding bit in the string will be set to 1; bits that represent sockets that are not ready to read will be set to 0.

WRETMSK

A bit string returned with the status of write events. The length of the string should be equal to the maximum number of sockets to be checked. For each socket that is ready to write, the corresponding bit in the string will be set to 1; bits that represent sockets that are not ready to be written will be set to 0.

ERETMSK

A bit string returned with the status of exception events. The length of the string should be equal to the maximum number of sockets to be checked. For each socket that has an exception status, the corresponding bit will be set to 1; bits that represent sockets that do not have exception status will be set to 0.

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B. Return codes on page 337 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 **Note:** See “Addressability mode (Amode) considerations” under “Environmental restrictions and programming requirements” on page 173.
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 259 shows an example of SELECTEX call instructions.

```
 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.
```

* The bit mask lengths can be determined from the expression:

```
((maximum socket number +32)/32 (drop the remainder))*4
```

Figure 121. SELECTEX call instruction example

Parameter values set by the application

MAXSOC

A fullword binary field specifying the largest socket descriptor number 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 will not check for read interrupts. The length of this bit-mask array is dependent on the value in MAXSOC.

WSNDMSK

The bit-mask array to control checking for write interrupts. If this parameter is not specified or the specified bit-mask is zeros, the SELECT will not check for write interrupts. The length of this bit-mask array is dependent on the value in MAXSOC.

ESNDMSK
The bit-mask array to control checking for exception interrupts. If this parameter is not specified or the specified bit-mask is zeros, the SELECT will not check for exception interrupts. The length of this bit-mask array is dependent on the value in MAXSOC.

SELECB
An ECB which, if posted, causes completion of the SELECTEX.

COBOL users who need more information about ECBs should refer to the section on synchronizing tasks (WAIT, POST, and EVENTS macros) in the *z/OS MVS Programming: Assembler Services Guide*.

Note: The maximum number of ECBs that can be specified in a list is 63.

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 337 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 will be 0) or one of the caller's ECBs has been posted (ECB value will be nonzero and the caller's descriptor sets will be 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 on bits mask conversion.

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 292 for a subroutine that will translate 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 **Note:** See "Addressability mode (Amode) considerations" under "Environmental restrictions and programming requirements" on page 173.
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 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 122. SEND call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 175.

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 337 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 **Note:** See “Addressability mode (Amode) considerations” under “Environmental restrictions and programming requirements” on page 173.
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 on page 264 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 123. 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 123. SENDMSG call instruction example (Part 2 of 2)

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 section, as shown in the example.

IOVCNT
On input, a pointer to a fullword binary field specifying the number of data buffers provided for this call.

ACCRIGHTS
On input, a pointer to the access rights received. This field is ignored.

ACCRLEN
On input, a pointer to the length of the access rights received. This field is ignored.

FLAGS
A fullword 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 337 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 292 for a subroutine that will translate 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 **Note:** See "Addressability mode (Amode) considerations" under "Environmental restrictions and programming requirements" on page 173.
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 on page 269 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 124. SENDTO call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 337 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 **Note:** See “Addressability mode (Amode) considerations” under “Environmental restrictions and programming requirements” on page 173.
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 272 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 125. SETSOCKOPT call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 353 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 337 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 16. 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 *hlq*.SEZAINST(CBLOCK) for the PL/1 example of IP_MREQ.
	The IP_MREQ definition for COBOL:
	`01 IP-MREQ.`
	`  05  IMR-MULTIADDR`
	`       PIC 9(8) BINARY.`
	`  05  IMR-INTERFACE`
	`       PIC 9(8) BINARY.`
		N/A

IP_DROP_MEMBERSHIP
Use this option to enable an application to exit 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 *hlq*.SEZAINST(CBLOCK) for the PL/1 example of IP_MREQ.
	The IP_MREQ definition for COBOL:
	`01 IP-MREQ.`
	`  05  IMR-MULTIADDR`
	`       PIC 9(8) BINARY.`
	`  05  IMR-INTERFACE`
	`       PIC 9(8) BINARY.`
		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.

Table 16. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	**SETSOCKOPT, OPTVAL (input)**	**GETSOCKOPT, OPTVAL (output)**
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.

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 *hlq*.SEZAINST(CBLOCK) for the PL/1 example of IPV6_MREQ.
	The IPV6_MREQ definition for COBOL:
	`01  IPV6-MREQ.`
	`   05 IPV6MR-MULTIADDR.`
	`     10  FILLER PIC 9(16)`
	`           BINARY.`
	`     10  FILLER PIC 9(16)`
	`           BINARY.`
	`   05 IPV6MR-INTERFACE PIC`
	`        9(8)    BINARY.`
		N/A

Table 16. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)

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 *hlq*.SEZAINST(CBLOCK) for the PL/1 example of IPV6_MREQ.
	The IPV6_MREQ definition for COBOL:
	`01  IPV6-MREQ.`
	`  05 IPV6MR-MULTIADDR.`
	`    10  FILLER PIC 9(16)`
	`          BINARY.`
	`    10  FILLER PIC 9(16)`
	`          BINARY.`
	`  05 IPV6MR-INTERFACE PIC`
	`       9(8)    BINARY.`
		N/A

IPV6_MULTICAST_HOPS Use to set or obtain the hop limit used for outgoing multicast packets. This is an IPv6-only socket option.	Contains a 4-byte binary value specifying the multicast hops. If not specified, then the default is 1 hop. -1 indicates use stack default. 0 - 255 is the valid hop limit range. **Note:** An application must be APF authorized to enable it to set the hop limit value above the system defined hop limit value. CICS applications cannot execute as APF authorized.	Contains a 4-byte binary value in the range from 0 to 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.

Table 16. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
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 from 0 to 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.
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.

Table 16. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
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 16. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	SETSOCKOPT, OPTVAL (input)	GETSOCKOPT, OPTVAL (output)
SO_KEEPALIVE Use this option to set or determine 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.	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_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.
	Contains an 8-byte field containing two 4-byte binary fields.
	Assembler coding:
	`ONOFF   DS F`
	`LINGER  DS F`
	COBOL coding:
	`ONOFF  PIC 9(8) BINARY.`
	`LINGER PIC 9(8) BINARY.`
	Set ONOFF to a nonzero value to enable and set to 0 to disable this option. Set LINGER to the number of seconds that TCP/IP lingers after the CLOSE is issued.
		Contains an 8-byte field containing two 4-byte binary fields.
		Assembler coding:
		`ONOFF   DS F`
		`LINGER  DS F`
		COBOL coding:
		`ONOFF  PIC 9(8) BINARY.`
		`LINGER PIC 9(8) BINARY.`
		A nonzero value returned in ONOFF indicates enabled, a 0 indicates disabled. LINGER indicates the number of seconds that TCP/IP will try to send data after the CLOSE is issued.

Table 16. 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 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 16. 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 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.

Table 16. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

OPTNAME options (input)	**SETSOCKOPT, OPTVAL (input)**	**GETSOCKOPT, OPTVAL (output)**
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.

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 will wait before releasing the connection. For example, with a LINGER value of 30 seconds, system resources (including the IMS™ or CICS transaction) will remain in the system for up to 30 seconds after the CLOSE call is issued. In high volume, transaction-based systems like CICS and IMS, this can impact performance severely.

If the SHUTDOWN call is issued, when the CLOSE call is received, the connection can be closed immediately, rather than waiting for the 30-second delay.

If you issue SHUTDOWN for a socket that currently has outstanding socket calls pending, see Table 17 to determine the effects of this operation on the outstanding socket calls.

Table 17. 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 **Note:** See "Addressability mode (Amode) considerations" under "Environmental restrictions and programming requirements" on page 173.
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 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 126. SHUTDOWN call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 337 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 **Note:** See “Addressability mode (Amode) considerations” under “Environmental restrictions and programming requirements” on page 173.
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 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 127. SOCKET call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 337 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 223 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 **Note:** See "Addressability mode (Amode) considerations" under "Environmental restrictions and programming requirements" on page 173.
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 128 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 128. TAKESOCKET call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 175.

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 337 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 will be 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 **Note:** See “Addressability mode (Amode) considerations” under “Environmental restrictions and programming requirements” on page 173.
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 TERMAPI call instructions.

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

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

Figure 129. TERMAPI call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 will be returned in RETCODE. Therefore, programs using stream sockets should place this call in a loop, calling this function until all data has been sent.

See “EZACIC04” on page 292 for a subroutine that will translate 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 173.
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 130 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 130. WRITE call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 337 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 **Note:** See "Addressability mode (Amode) considerations" under "Environmental restrictions and programming requirements" on page 173.
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 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 131. WRITEV call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 175.

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 337 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 which translates EBCDIC data to ASCII data using the translation table listed in “EZACIC14” on page 303.
- EZACIC15—An alternative to EZACIC05 which translates ASCII data to EBCDIC data using the translation table listed in “EZACIC15” on page 305.

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, since 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.

EZACIC04

The EZACIC04 program is used to translate EBCDIC data to ASCII data.

Figure 132 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 132. EZACIC04 call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 175.

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/1, and assembler language programs.

Figure 133 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 133. EZACIC05 call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 134 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 134. EZACIC06 call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 will be checked. The second word of BIT-MASK contains B'00000000000000000000000000000001' to indicate that socket 32 will be checked. These instructions process the bit string shown in the following example.

```
MOVE ZEROS TO CHAR-STRING.
      MOVE '1' TO CHAR-ENTRY(1), CHAR-ENTRY(6), CHAR-ENTRY(33).
      CALL 'EZACIC06' USING TOKEN CTOB BIT-MASK CH-MASK
            CHAR-MASK-LENGTH RETCODE.
      MOVE BIT-MASK TO ....
```

When the select call returns and you want to check the bit-mask string for socket activity, enter the following instructions.

```
MOVE ..... TO BIT-MASK.
      CALL 'EZACIC06' USING TOKEN BTOC BIT-MASK CH-MASK
              CHAR-MASK-LENGTH RETCODE.
      PERFORM TEST-SOCKET THRU TEST-SOCKET-EXIT  VARYING IDX
            FROM 1 BY 1 UNTIL IDX EQUAL CHAR-MASK-LENGTH.
      TEST-SOCKET.
            IF CHAR-ENTRY(IDX) EQUAL '1'
                 THEN PERFORM SOCKET-RESPONSE THRU
                     SOCKET-RESPONSE-EXIT
                 ELSE NEXT SENTENCE.
      TEST-SOCKET-EXIT.
            EXIT.
```

EZACIC08

The GETHOSTBYNAME and GETHOSTBYADDR calls were derived from C socket calls that return a structure known as HOSTENT. A given TCP/IP 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/1 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 135 on page 297 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 135. EZAZIC08 call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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

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/1 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 136 on page 300 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 136. 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 136. EZACIC09 call instruction example (Part 2 of 2)

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

Parameter values set by the application

RES This fullword binary field must contain the address of the ADDRINFO structure (as returned by the GETADDRINFO call). This variable is the same as the RES variable in the GETADDRINFO socket call.

RES-NAME-LEN
A fullword binary field that will contain the length of the socket address structure as returned by the GETADDRINFO call.

Parameter values returned to the application

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 137 shows an example of how EZACIC14 translates a byte of EBCDIC data.

ASCII output by EZACIC14		second hex digit of byte of EBCDIC data															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
first hex digit of byte of EBCDIC data	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 137. EZACIC14 EBCDIC-to-ASCII table

Figure 138 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 138. EZACIC14 call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 139 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 139. EZACIC15 ASCII-to-EBCDIC table

Figure 140 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 140. EZACIC15 call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 175.

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 appendix 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 appendix) 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 141 on page 308. 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 141. 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 on page 311, 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 on page 311, 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 section 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 rest of this chapter 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 on page 176. 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 page 310).

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 will return 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 337.

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 on page 179. 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 page 310).

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 337.

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 on page 182. 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 page 310).

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 337.

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 on page 183. 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 page 310).

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 337.

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 on page 186. 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 page 310).

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 will set the FNDELAY flag; setting ARG to 3 will reset 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 337.

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 on page 197. 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 page 310).

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 337.

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 on page 198. 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 page 310).

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 on page 201. 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 page 310).

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

Parameter values returned to the application

NAMELEN
The length of host name is returned. This cannot exceed 255.

NAME
The host 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 337.

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 on page 209. 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 page 310).

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 337.

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 on page 211. 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 page 310).

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 337.

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 on page 213. 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 page 310).

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 213

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'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 337.

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 on page 223. 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 page 310).

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 337.

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 page 310).

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.

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 on page 228. 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 page 310).

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 228 for values and descriptions.

REQARG
The request argument associated with the command. See “IOCTL” on page 228 for a list and description of possible argument values.

Parameter values returned to the application

RETARG
The return argument. See “IOCTL” on page 228 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 337.

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 on page 235. 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 page 310).

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

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 337.

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 on page 240. 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 page 310).

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 337.

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 293 for a subroutine that will translate ASCII data to EBCDIC.

RECVFROM

This call functions in the same way as the equivalent call described on page 244. 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 page 310).

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 will read 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 337.

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 293 for a subroutine that will translate ASCII data to EBCDIC.

SELECT

This call functions in the same way as the equivalent call described on page 253. 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 page 310).

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 Reference* for more information.

WR-MASK
Set the bit mask array for writes. See *z/OS Communications Server: IP Programmer's Reference* for more information.

EX-MASK
Set the bit mask array for exceptions. See *z/OS Communications Server: IP Programmer's 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 Reference* for more information.

R-W-MASK
Returned bit mask array for writes. See *z/OS Communications Server: IP Programmer's Reference* for more information.

R-E-MASK
Returned bit mask array for exceptions. See *z/OS Communications Server: IP Programmer's 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 337.

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 on page 260. 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 page 310).

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 337.

RETCODE
A value of −1 indicates an error. Other values have no meaning.

See “EZACIC04” on page 292 for a subroutine that will translate EBCDIC data to ASCII.

SENDTO

This call functions in the same way as the equivalent call described on page 267. 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 page 310).

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 337.

RETCODE
A value of −1 indicates an error. Other values have no meaning.

See “EZACIC04” on page 292 for a subroutine that will translate EBCDIC data to ASCII.

SETSOCKOPT

This call functions in the same way as the equivalent call described on page 213. 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 page 310).

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 271 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'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 271.

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 337.

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 on page 281. 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 page 310).

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 337.

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 on page 283. 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 page 310).

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 will select the appropriate protocol for the TYPE specified above.)

SOCKNO
Set to −1. The system will return 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 337.

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 on page 285. 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 page 310).

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 will return 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 337.

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 on page 287. 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 page 310).

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 337.

RETCODE
The number of bytes written is returned. A RETCODE of −1 indicates an error.

See "EZACIC04" on page 292 for a subroutine that will translate EBCDIC data to ASCII.

Appendix B. Return codes

This appendix covers the following return codes and error messages

- Error numbers from MVS TCP/IP
- Error codes from the Sockets Extended interface.

Sockets return codes (ERRNOs)

This section provides the system-wide message numbers and codes set by the system calls. These message numbers and codes are in the TCPERRNO.H include file supplied with TCP/IP Services.

Table 18. 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	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.
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.

Table 18. Sockets ERRNOs (continued)

Error number	Message name	Socket type	Error description	Programmer's response
4	EINTR	All	A system call was interrupted.	Check that the socket connection and TCP/IP are still active.
5	EAI_FAMILY	GETADDRINFO GETNAMEINFO	The AF or the FAMILY is incorrect.	Correct the AF or the FAMILY.
5	EIO	All	An I/O error occurred.	Check status and contents of source database if this occurred during a file access.
6	EAI_MEMORY	GETADDRINFO GETNAMEINFO	The resolver cannot obtain storage to process the host name.	Contact your system administrator.
6	ENXIO	All	The device or driver was not found.	Check status of the device attempting to access.
7	E2BIG	All	The argument list is too long.	Check the number of function parameters.
7	EAI_BADFLAGS	GETADDRINFO GETNAMEINFO	FLAGS has an incorrect value.	Correct the FLAGS.
8	EAI_SERVICE	GETADDRINFO	The SERVICE was not recognized for the specified socket type.	Correct the SERVICE.
8	ENOEXEC	All	An EXEC format error occurred.	Check that the target module on an exec call is a valid executable module.
9	EAI_SOCKTYPE	GETADDRINFO	The SOCTYPE was not recognized.	Correct the SOCTYPE.
9	EBADF	All	An incorrect socket descriptor was specified.	Check socket descriptor value. It might be currently not in use or incorrect.
9	EBADF	Givesocket	The socket has already been given. The socket domain is not AF_INET or AF_INET6.	Check the validity of function parameters.
9	EBADF	Select	One of the specified descriptor sets is an incorrect socket descriptor.	Check the validity of function parameters.
9	EBADF	Takesocket	The socket has already been taken.	Check the validity of function parameters.
9	EAI_SOCKTYPE	GETADDRINFO	The SOCTYPE was not recognized.	Correct the SOCTYPE.
10	ECHILD	All	There are no children.	Check if created subtasks still exist.

Table 18. Sockets ERRNOs (continued)

Error number	Message name	Socket type	Error description	Programmer's response
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 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.
14	EFAULT	All	An incorrect storage address or length was specified.	Check validity of function parameters.
15	ENOTBLK	All	A block device is required.	Check device status and characteristics.
16	EBUSY	All	Listen has already been called for this socket. Device or file to be accessed is busy.	Check if the device or file is in use.
17	EEXIST	All	The data set exists.	Remove or rename existing file.
18	EXDEV	All	This is a cross-device link. A link to a file on another file system was attempted.	Check file permissions.
19	ENODEV	All	The specified device does not exist.	Check file name and if it exists.
20	ENOTDIR	All	The specified directory is not a directory.	Use a valid file that is a directory.
21	EISDIR	All	The specified directory is a directory.	Use a valid file that is not a directory.
22	EINVAL	All types	An incorrect argument was specified.	Check 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.

Table 18. Sockets ERRNOs (continued)

Error number	Message name	Socket type	Error description	Programmer's response
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().
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.
37	EALREADY	Connect	The socket is marked nonblocking and the previous connection has not been completed.	Reissue Connect().
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.

Table 18. Sockets ERRNOs (continued)

Error number	Message name	Socket type	Error description	Programmer's response
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.
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	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.

Table 18. Sockets ERRNOs (continued)

Error number	Message name	Socket type	Error description	Programmer's response
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.
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.	If you want to reuse the same address, use Setsockopt() with SO_REUSEADDR. See Setsockopt(). 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.

Table 18. Sockets ERRNOs (continued)

Error number	Message name	Socket type	Error description	Programmer's response
55	ENOBUFS	Takesocket	Not enough buffer space is available to create the new socket.	Call your system administrator.
56	EISCONN	Connect	The socket is already connected.	Correct the socket descriptor on Connect() or do not issue a Connect() twice for the socket.
57	ENOTCONN	All	The socket is not connected.	Connect the socket before communicating.
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.

Table 18. Sockets ERRNOs (continued)

Error number	Message name	Socket type	Error description	Programmer's response
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.
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 CS 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 not timeout 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.
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 18. 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.
145	E2BIG	All	The argument list is too long.	Eliminate excessive number of arguments.
156	EMVSINITIAL	All	Process initialization error. This indicates an z/OS UNIX process initialization failure. This is usually an indication that a proper OMVS RACF® segment is not defined for the user ID associated with application. The RACF OMVS segment 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.
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.

Table 18. Sockets ERRNOs (continued)

Error number	Message name	Socket type	Error description	Programmer's response
1008	EIBMCONFLICT	All	This request conflicts with a request already queued on the same socket.	Cancel the existing call or wait for its completion before reissuing this call.
1009	EIBMCANCELLED	All	The request was canceled by the CANCEL call.	Informational, no action needed.
1011	EIBMBADTCPNAME	All	A TCP/IP name that is not valid was detected.	Correct the name specified in the IBM_TCPIMAGE structure.
1011	EIBMBADTCPNAME	Setibmopt	A TCP/IP name that is not valid was detected.	Correct the name specified in the IBM_TCPIMAGE structure.
1011	EIBMBADTCPNAME	INITAPI	A TCP/IP name that is not valid was detected.	Correct the name specified on the IDENT option TCPNAME field.
1012	EIBMBADREQUESTCODE	All	A request code that is not valid was detected.	Contact your system administrator.
1013	EIBMBADCONNECTIONSTATE	All	A connection token that is not valid was detected; bad state.	Verify TCP/IP is active.
1014	EIBMUNAUTHORIZEDCALLER	All	An unauthorized caller specified an authorized keyword.	Ensure user ID has authority for the specified operation.
1015	EIBMBADCONNECTIONMATCH	All	A connection token that is not valid was detected. There is no such connection.	Verify TCP/IP is active.
1016	EIBMTCPABEND	All	An abend occurred when TCP/IP was processing this request.	Verify that TCP/IP has restarted.
1023	EIBMTERMERROR	All	Encountered a terminating error while processing.	Call your system administrator.
1026	EIBMINVDELETE	All	Delete requestor did not create the connection.	Delete the request from the process that created it.
1027	EIBMINVSOCKET	All	A connection token that is not valid was detected. No such socket exists.	Call your system programmer.
1028	EIBMINVTCPCONNECTION	All	Connection terminated by TCP/IP. The token was invalidated by TCP/IP.	Reestablish the connection to TCP/IP.
1032	EIBMCALLINPROGRESS	All	Another call was already in progress.	Reissue after previous call has completed.
1036	EIBMNOACTIVETCP	All	TCP/IP is not installed or not active.	Correct TCP/IP name used.
1036	EIBMNOACTIVETCP	Select	EIBMNOACTIVETCP	Ensure TCP/IP is active.

Table 18. Sockets ERRNOs (continued)

Error number	Message name	Socket type	Error description	Programmer's response
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.
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.
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.

Table 18. Sockets ERRNOs (continued)

Error number	Message name	Socket type	Error description	Programmer's response
2017	EIPADDRNOTFOUND	REXX	Address not found.	Call your MVS system programmer.

Sockets extended ERRNOs

Table 19. 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 chapter on general programming.
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.

Table 19. Sockets extended ERRNOs (continued)

Error code	Problem description	System action	Programmer's response
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 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.
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 from 1 to 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 chapter on general programming.
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.

Table 19. Sockets extended ERRNOs (continued)

Error code	Problem description	System action	Programmer's response
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.
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.

Table 19. Sockets extended ERRNOs (continued)

Error code	Problem description	System action	Programmer's response
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.
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 sockets 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 Sockets Interface before executing this call.
20003	The CICS Sockets Interface is not in operation.	End the call.	Contact the CICS Systems programmer. Ensure that the user ID being used is permitted to have at least UPDATE access to the EXITPROGRAM resouce.

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/1 programs.

Table 20. GETSOCKOPT/SETSOCKOPT command values for Macro, Assembler, and PL/1

Command name	Decimal value	Hex value
IP_ADD_MEMBERSHIP	1048581	X'00100005'
IP_DROP_MEMBERSHIP	1048582	X'00100006'
IP_MULTICAST_IF	1048583	X'00100007'
IP_MULTICAST_LOOP	1048580	X'00100004'
IP_MULTICAST_TTL	1048579	X'00100003'
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'
SO_BROADCAST	32	X'00000020'
SO_ERROR	4103	X'00001007'
SO_LINGER	128	X'00000080'
SO_KEEPALIVE	8	X'00000008'
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_NODELAY	2147483649	X'80000001'

Table 21. GETSOCKOPT/SETSOCKOPT optname value for C programs

Option name	Decimal value
IP_ADD_MEMBERSHIP	5
IP_DROP_MEMBERSHIP	6

Table 21. GETSOCKOPT/SETSOCKOPT optname value for C programs (continued)

IP_MULTICAST_IF	7
IP_MULTICAST_LOOP	4
IP_MULTICAST_TTL	3
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_NODELAY	1

Appendix D. CICS sockets messages

This section contains CICS socket interface messages.

EZY1218—EZY1352

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.

User Response: Use CEMT to determine and correct the status of the program.

System Programmer Response: None.

Module: EZACIC02

Destination: LISTENER

EZY1220E *mm/dd/yy hh:mm:ss* **READ FAILURE ON CONFIGURATION FILE PHASE=***xx* **EIBRESP2=***rrrrrr*

Explanation: EZACIC21 was unable to read the configuration file.

System Action: Terminate the transaction.

User Response: Notify the CICS Systems Programmer.

System Programmer Response: Use the EIBRESP2 value to determine the problem and correct the file. See the *CICS User's Handbook* 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 = rrr**

Explanation: The attempt to enable the task related user exit (TRUE) failed.

System Action: Terminate the transaction.

User Response: Notify the CICS Systems 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. See the *CICS User's Handbook* 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 OS/390 failed.

System Action: Terminate the transaction.

User Response: Contact your OS/390 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.

User Response: Contact the CICS Systems 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**

Explanation: The CICS Sockets Interface has completed initialization successfully.

System Action: Continue with execution.

User 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.

User Response: Contact the CICS Systems 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 User's Handbook* 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.

User Response: Contact the CICS Systems 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 User's Handbook* 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.

User 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.

User 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.

User 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.

User 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.

User 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.

User 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.

User 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.

User Response: Contact the CICS Systems 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.

User 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.

User Response: Contact the CICS Systems Programmer.

System Programmer Response: Use the RESP2 value to determine the error and correct the cache file. See the *CICS User's Handbook* 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.

User Response: Use the RESP2 value to determine the error. Contact the IBM Software Support Center. See the *CICS User's Handbook* 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 Sockets Interface was inactive.

System Action: Return error and terminate transaction EZAO.

User Response: Use transaction EZAO to start the CICS Sockets 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.

User 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 *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.

User 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 *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.

User Response: Contact the CICS Systems 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=** *xxxxxxxx* **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 which cannot be accommodated within the pool of reusable tasks.

System Action: Processing continues.

User Response: If this message happens frequently, increase the size of the reusable task pool for this CICS.

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.

User Response: Use EZAO to stop (immediate) and start the CICS Sockets 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.

User Response: Use EZAO to stop (immediate) and start the CICS Sockets 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 *z/OS UNIX System Services Messages and Codes*.

System Action: The TRUE is disabled and the task abends with an AEY9.

User Response: Use EZAO to stop (immediate) and start the CICS Sockets 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 *z/OS UNIX System Services Messages and Codes*.

System Action: The TRUE is disabled and the task abends with an AEY9.

User Response: Contact the CICS Systems Programmer.

System Programmer Response: The CICS Sockets 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 *z/OS UNIX System Services Messages and Codes*.

System Action: The TRUE is disabled and the task abends with an AEY9.

User Response: If this happens repeatedly, use EZAO to STOP (immediate) the CICS Sockets 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 *z/OS UNIX System Services Messages and Codes*.

System Action: The TRUE is disabled and the task abends with an AEY9.

User 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 *z/OS UNIX System Services Messages and Codes*.

System Action: The TRUE is disabled and the task abends with an AEY9.

User Response: Check the application program calls to the CICS Sockets 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 *z/OS UNIX System Services Messages and Codes*.

System Action: The TRUE is disabled and the task abends with an AEY9.

User Response: Check the application program calls to the CICS Sockets 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 *z/OS UNIX System Services Messages and Codes*.

System Action: The TRUE is disabled and the task abends with an AEY9.

User 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 *z/OS UNIX System Services Messages and Codes*.

System Action: The TRUE is disabled and the task abends with an AEY9.

User 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 in invalid function.

errno is the UNIX System Services Return Code. These return codes are listed and described in the *z/OS UNIX System Services Messages and Codes*.

System Action: The TRUE is disabled and the task abends with an AEY9.

User 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 *z/OS UNIX System Services Messages and Codes*.

System Action: The TRUE is disabled and the task abends with an AEY9.

User 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 *z/OS UNIX System Services Messages and Codes*.

System Action: The TRUE is disabled and the task abends with an AEY9.

User 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 *z/OS UNIX System Services Messages and Codes*.

System Action: The TRUE is disabled and the task abends with an AEY9.

User 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 *z/OS UNIX System Services Messages and Codes*.

System Action: The TRUE detaches the MVS subtask.

User 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.

User 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.

User 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.

User 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.

User Response: Contact the IBM Software Support Center. See the *CICS User's Handbook* 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.

User Response: Contact CICS Systems 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.

User Response: Contact CICS Systems 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 User's Handbook* 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.

User Response: Contact CICS Systems Programmer.

System Programmer Response: Use the CICS APR to interpret the value of EIBRESP2. Correct your CICS configuration as indicated.

See the *CICS User's Handbook* 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 Sockets Interface.

System Action: Listener terminates.

User Response: See the *CICS User's Handbook* 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.

User 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.

User Response: None.

System Programmer Response: None.

Module: EZACIC02

Destination: LISTENER

EZY1291I *mm/dd/yy hh:mm:ss* **LISTENER TRANSACTION** *tran* **TASKID=** *cicstask* **ACCEPTING REQUESTS VIA PORT** *pppppp*

Explanation: Listener transaction *tran* is now available to receive connection requests on port *pppppp*.

System Action: Listener *tran* continues

User Response: None.

System Programmer Response: None.

Module: EZACIC02

Destination: LISTENER

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 Sockets Interface did not complete successfully and this Listener cannot continue.

System Action: Listener transaction *tran* terminates.

User Response: If EZAO is being used to start the Listener, ensure that the CICS Sockets 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 Sockets Interface failed.

See the *CICS User's Handbook* 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 *z/OS UNIX System Services Messages and Codes*.

System Programmer Response: None.

System Action: Listener transaction *tran* terminates.

User 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 *z/OS UNIX System Services Messages and Codes*.

System Programmer Response: None.

System Action: Listener transaction *tran* terminates.

User 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 *z/OS UNIX System Services Messages and Codes*.

System Action: Listener transaction *tran* terminates.

User 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 *z/OS UNIX System Services Messages and Codes*.

System Action: Listener transaction *tran* terminates.

User 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 *z/OS UNIX System Services Messages and Codes*.

System Action: Listener transaction *tran* terminates.

User 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 *z/OS UNIX System Services Messages and Codes*.

System Action: Listener transaction *tran* continues.

User 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 *z/OS UNIX System Services Messages and Codes*.

System Action: Listener transaction *tran* terminates.

User 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 *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.

User 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.

User 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.

User 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.

User 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.

User 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.

User 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.

User 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 *z/OS UNIX System Services Messages and Codes*.

System Action: The ACCEPT call is delayed until a socket is available.

User 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 *z/OS UNIX System Services Messages and Codes*.

System Action: Listener transaction *tran* terminates.

User 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 *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.

User 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.

User 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.

User 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.

User 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.

User 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.

User 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.

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.

User 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: None.

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.

User 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.

User 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= qqqq**

Explanation: The Listener transaction started a server transaction through transient data queue qqqq.

System Action: Listener transaction continues and the server transaction is ready to start.

User 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.

User 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.

User Response: Contact the CICS Systems 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.

User Response: Contact the CICS Systems 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.

User 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.

User Response: Contact the CICS Systems 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.

User Response: Contact the CICS Systems 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.

User 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.

User Response: Contact the CICS Systems 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.

User Response: Contact the CICS Systems 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.

User 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.

User Response: Contact the CICS Systems 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.

User 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.

User Response: Contact the CICS Systems 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.

User 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.

User Response: Contact the CICS Systems 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.

User 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 *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.

User 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 *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.

User 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.

User 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.

User 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 Sockets Interface was requested but the interface is not enabled.

System Action: The termination request is ignored.

User 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 Sockets interface has been made but one is already is progress.

System Action: Ignore the second request.

User 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 Sockets interface has been made but one is already is progress.

System Action: Ignore the second request.

User 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=***term* **TRAN=***tranxxx*

Explanation: A quiesce is in progress and is waiting for the completion of all outstanding CICS tasks using the CICS sockets interface.

System Action: Continue with the quiesce.

User 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 Sockets Interface has been successfully completed.

System Action: Terminate the CICS Sockets Interface.

User 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 Sockets Interface has been successfully completed.

System Action: Terminate the CICS Sockets Interface.

User 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.

User 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.

User 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.

User 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 *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.

User Response: Correct the invalid socket function and retry.

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.

User 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 Sockets 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 Sockets 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 Sockets interface is being enabled.
- DISABLE means the IP CICS Sockets interface is being disabled.
- STARTUP means the IP CICS Sockets 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 Sockets 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 Sockets Interface is shutting down, then the disable action is ignored and the interface remains active.
- If the IP CICS Sockets interface is being started, then the startup action is ignored and the interface remains inactive.

User 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 Sockets 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 Sockets 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 Sockets interface is being enabled.
- DISABLE means the IP CICS Sockets 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 Sockets Interface is not initialized.

User 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 Sockets 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.

User Response: Contact the CICS Systems 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)

Appendix E. Sample programs

This appendix contains the following samples:

- 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

EZACICSC

The following COBOL socket program is in the *hlq*.SEZAINST data set.

```
* $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. *
*                                                              *
*  COPYRIGHT = LICENSED MATERIALS - PROPERTY OF IBM            *
*              5694-A01 (C) COPYRIGHT IBM CORP. 1993, 2003     *
*              This module is restricted materials of IBM      *
*              REFER TO IBM COPYRIGHT INSTRUCTIONS.            *
*                                                              *
*   Status :  CSV1R5                                           *
*                                                              *
*                                                              *
*--------------------------------------------------------------*
*
 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       '.
 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 '.
```

Figure 142. EZACICSC IPv4 child server sample (Part 1 of 8)

```
 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'.
 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            '.
     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  BITMASK-TOKEN           PIC X(16) VALUE 'TCPIPBITMASKCOBL'.
 77  TOEBCDIC-TOKEN          PIC X(16) VALUE 'TCPIPTOEBCDICXLT'.
```

Figure 142. EZACICSC IPv4 child server sample (Part 2 of 8)

```
 77  TOASCII-TOKEN           PIC X(16) VALUE 'TCPIPTOASCIIXLAT'.
 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.
     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 FILLER                        PIC X(1).
     05 SOCKADDR-IN.
```

Figure 142. EZACICSC IPv4 child server sample (Part 3 of 8)

```
       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.
     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 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('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)
```

Figure 142. EZACICSC IPv4 child server sample (Part 4 of 8)

```
                         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
        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.
     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 TOASCII-TOKEN TCP-BUF TCPLENG.
     CALL 'EZASOKET' USING SOKET-WRITE SOCKID TCPLENG
           TCP-BUF ERRNO RETCODE.
     IF RETCODE <  0 THEN
        MOVE 'Y' TO WRITE-SW
        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
```

Figure 142. EZACICSC IPv4 child server sample (Part 5 of 8)

```
     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
        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 TOEBCDIC-TOKEN 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.
```

Figure 142. EZACICSC IPv4 child server sample (Part 6 of 8)

```
     MOVE RETCODE TO TCPLENG.
*
*    REMOVE FOLLOWING STATEMENT IF USING EBCDIC CLIENT
*
     CALL 'EZACIC04' USING TOASCII-TOKEN TCP-BUF TCPLENG.
     CALL 'EZASOKET' USING SOKET-WRITE SOCKID TCPLENG
          TCP-BUF ERRNO RETCODE.
     IF RETCODE <  0 THEN
        MOVE 'Y' TO WRITE-SW
        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.
     IF WRITE-SW = 'Y' THEN
         MOVE 78 TO CLENG
         MOVE EIBTASKN TO TASK-NUMBER
         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 TOASCII-TOKEN TCP-BUF TCPLENG.
        CALL 'EZASOKET' USING SOKET-WRITE SOCKID TCPLENG
             TCP-BUF ERRNO RETCODE.
        IF RETCODE <  0 THEN
           MOVE 'Y' TO WRITE-SW
           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.
```

Figure 142. EZACICSC IPv4 child server sample (Part 7 of 8)

```
    MOVE 'Y' TO WRITE-SW
    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
    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
    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
    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
    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
    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
    MOVE ENDDATA-ERR TO CICS-MSG-AREA.
    PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
    GO TO PGM-EXIT.
```

Figure 142. EZACICSC IPv4 child server sample (Part 8 of 8)

EZACICSS

The following COBOL socket program is in the *hlq*.SEZAINST data set.

```
***************************************************************
*                                                             *
* Communications Server for OS/390, Version 1, Release 5      *
*                                                             *
*                                                             *
* Copyright:    Licensed Materials - Property of IBM          *
*                                                             *
*               "Restricted Materials of IBM"                 *
*                                                             *
*               5647-A01                                      *
*                                                             *
*               (C) Copyright IBM Corp. 1977, 2003            *
*                                                             *
*               US Government Users Restricted Rights -       *
*               Use, duplication or disclosure restricted by  *
*               GSA ADP Schedule Contract with IBM Corp.      *
*                                                             *
* Status:       CSV1R5                                        *
*                                                             *
* $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.     *
*                  It processes request received from          *
*                  clients for updates to a DB2 database.      *
*                  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.                                     *
*                                                              *
*                                                              *
*--------------------------------------------------------------*
```

Figure 143. EZACICSS IPv4 iterative server sample (Part 1 of 20)

```
*                                                              *
*   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)
      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)
```

Figure 143. EZACICSS IPv4 iterative server sample (Part 2 of 20)

```
            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)
                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.
```

Figure 143. EZACICSS IPv4 iterative server sample (Part 3 of 20)

```
        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)
         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.
```

Figure 143. EZACICSS IPv4 iterative server sample (Part 4 of 20)

```
     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  TCP-TOKEN                PIC X(16) VALUE 'TCPIPIUCVSTREAMS'.
 77  BITMASK-TOKEN            PIC X(16) VALUE 'TCPIPBITMASKCOBL'.
 77  TOEBCDIC-TOKEN           PIC X(16) VALUE 'TCPIPTOEBCDICXLT'.
 77  TOASCII-TOKEN            PIC X(16) VALUE 'TCPIPTOASCIIXLAT'.
 77  CONTRACE                 PIC X(8)  VALUE 'CONTRACE'.
 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.
 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 '????'.
```

Figure 143. EZACICSS IPv4 iterative server sample (Part 5 of 20)

```
     88  DEFAULT-SPECIFIED              VALUE '1950'.
 01  COMMAND.
     05  INITAPI-CMD          PIC 9(4)  COMP VALUE 0.
     05  ACCEPT-CMD           PIC 9(4)  COMP VALUE 1.
     05  BIND-CMD             PIC 9(4)  COMP VALUE 2.
     05  CLOSE-CMD            PIC 9(4)  COMP VALUE 3.
     05  CONNECT-CMD          PIC 9(4)  COMP VALUE 4.
     05  FCNTL-CMD            PIC 9(4)  COMP VALUE 5.
     05  GETHOSTID-CMD        PIC 9(4)  COMP VALUE 7.
     05  GETHOSTNAME-CMD      PIC 9(4)  COMP VALUE 8.
     05  GETPEERNAME-CMD      PIC 9(4)  COMP VALUE 9.
     05  GETSOCKNAME-CMD      PIC 9(4)  COMP VALUE 10.
     05  GETSOCKOPT-CMD       PIC 9(4)  COMP VALUE 11.
     05  IOCTL-CMD            PIC 9(4)  COMP VALUE 12.
     05  LISTEN-CMD           PIC 9(4)  COMP VALUE 13.
     05  READ-CMD             PIC 9(4)  COMP VALUE 14.
     05  RECVFROM-CMD         PIC 9(4)  COMP VALUE 16.
     05  SELECT-CMD           PIC 9(4)  COMP VALUE 19.
     05  SELECTX-CMD          PIC 9(4)  COMP VALUE 19.
     05  SEND-CMD             PIC 9(4)  COMP VALUE 20.
     05  SENDTO-CMD           PIC 9(4)  COMP VALUE 22.
     05  SETSOCKOPT-CMD       PIC 9(4)  COMP VALUE 23.
     05  SHUTDOWN-CMD         PIC 9(4)  COMP VALUE 24.
     05  SOCKET-CMD           PIC 9(4)  COMP VALUE 25.
     05  WRITE-CMD            PIC 9(4)  COMP VALUE 26.
     05  GETCLIENTID-CMD      PIC 9(4)  COMP VALUE 30.
     05  GIVESOCKET-CMD       PIC 9(4)  COMP VALUE 31.
     05  TAKESOCKET-CMD       PIC 9(4)  COMP VALUE 32.
*----------------------------------------------------------------*
*    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  L-DESC                   PIC 9(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).
```

Figure 143. EZACICSS IPv4 iterative server sample (Part 6 of 20)

```
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  INITAPI-SOCKET.
    05  INIT-API2            PIC X(8)  VALUE 'IUCVAPI '.
    05  INIT-API3            PIC 9(4)  COMP VALUE 50.
    05  INIT-API4            PIC 9(4)  COMP VALUE 2.
    05  INIT-SUBTASKID.
        10  SUBTASKNO        PIC X(7)  VALUE LOW-VALUES.
        10  SUBT-CHAR        PIC A(1)  VALUE 'L'.
    05  INIT-API6            PIC 9(8)  COMP VALUE 0.
    05  NFDS                 PIC 9(8)  COMP.
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  SIN-FAMILY           PIC 9(4)  COMP VALUE 0.
    05  SIN-PORT             PIC 9(4)  COMP VALUE 0.
    05  SIN-ADDR             PIC 9(8)  COMP VALUE 0.
    05  SIN-ZERO             PIC X(8)  VALUE LOW-VALUES.
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 '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  SOCKADDR-IN-PARM.
        10  SIN-FAMILY-PARM  PIC 9(4).
        10  SIN-PORT-PARM    PIC 9(4).
```

Figure 143. EZACICSS IPv4 iterative server sample (Part 7 of 20)

```
         10  SIN-ADDR-PARM    PIC 9(8)  COMP.
         10  SIN-ZERO-PARM    PIC X(8).
 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),
       MGRNO                         CHAR(6),
       ADMRDEPT                      CHAR(3)
     ) END-EXEC.
*******************************************************************
* COBOL DECLARATION FOR TABLE TCPCICS.DEPT                        *
*******************************************************************
 01  DCLDEPT.
     10 DEPTNO               PIC X(3).
```

Figure 143. EZACICSS IPv4 iterative server sample (Part 8 of 20)

```
     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
          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 143. EZACICSS IPv4 iterative server sample (Part 9 of 20)

```
*---------------------------------------------------------------*
*                                                               *
*  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.
     MOVE 2                           TO CLI-SOCKID
```

Figure 143. EZACICSS IPv4 iterative server sample (Part 10 of 20)

```
                                         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 BITMASK-TOKEN 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.
*----------------------------------------------------------------*
*                                                                *
*  LISTENER STARTED TASK                                         *
*                                                                *
*----------------------------------------------------------------*
 LISTENER-STARTED-TASK.
     MOVE CLIENTID-PARM               TO CID-LSTN-INFO.
```

Figure 143. EZACICSS IPv4 iterative server sample (Part 11 of 20)

```
     MOVE -1 TO L-DESC.
     CALL 'EZACICAL' USING TCP-TOKEN  TAKESOCKET-CMD
                           ZERO-HWRD  CLIENTID-LSTN
                           GIVE-TAKE-SOCKET L-DESC
                           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 TOASCII-TOKEN TCP-BUF TCPLENG
          CALL 'EZACICAL' USING TCP-TOKEN     WRITE-CMD
                                SRV-SOCKID    TCPLENG
                                ZERO-FWRD     ZERO-PARM
                                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 'EZACICAL' USING TCP-TOKEN   CLOSE-CMD
                                     SRV-SOCKID  ZERO-8
                                     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 'EZACICAL' USING TCP-TOKEN  INITAPI-CMD  INIT-API2
                           INIT-API3  INIT-API4    INIT-SUBTASKID
```

Figure 143. EZACICSS IPv4 iterative server sample (Part 12 of 20)

```
                           INIT-API6  ERRNO        RETCODE.
*---------------------------------------------------------------*
*                          CONTRACE.                            *
*  NOTE:  The CONTRACE parameter places trace output for this   *
*         SERVER in your system log for debugging purposes.     *
*         The parameter should be removed from the INITAPI-CMD  *
*         Once you are comfortable that your server is working. *
*                                                               *
*---------------------------------------------------------------*
     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.
*--------------------------------------------------------------*
*                                                              *
*  PERFORM TCP SOCKET FUNCTIONS BY PASSING SOCKET COMMAND TO   *
*  EZACICAL ROUTINE.  SOCKET COMMAND ARE TRANSLATED TO PRE-    *
*  DEFINE INTEGER.                                             *
*                                                              *
*--------------------------------------------------------------*
 SCKET-BIND-LSTN.
     MOVE  -1                    TO SRV-SOCKID-FWD.
*--------------------------------------------------------------*
*                                                              *
*   CREATING A SOCKET (SOCKET CALL, INTEGER 17) TO ALLOCATE    *
*   AN OPEN SOCKET FOR INCOMING CONNECTIONS                    *
*                                                              *
*--------------------------------------------------------------*
     CALL 'EZACICAL' USING TCP-TOKEN       SOCKET-CMD  ZERO-HWRD
                           AF-INET         SOCK-TYPE   PROTOCOL
                           SRV-SOCKID-FWD  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 (BIND CALL, INTEGER 02) TO THE SERVICE PORT *
*  TO ESTABLISH A LOCAL ADDRESS FOR PROCESSING INCOMING        *
*  CONNECTIONS.                                                *
*                                                              *
*--------------------------------------------------------------*
```

Figure 143. EZACICSS IPv4 iterative server sample (Part 13 of 20)

```
     MOVE AF-INET                TO SIN-FAMILY.
     MOVE 0                      TO SIN-ADDR.
     MOVE PORT                   TO SIN-PORT.
     CALL 'EZACICAL' USING TCP-TOKEN    BIND-CMD  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 'LISTEN' COMMAND (INTEGER 09) TO ALLOWS SERVERS TO      *
*  PREPARE A SOCKET FOR INCOMING CONNECTIONS AND SET MAXIMUM    *
*  CONNECTIONS.                                                 *
*                                                               *
*---------------------------------------------------------------*
     CALL 'EZACICAL' USING TCP-TOKEN  LISTEN-CMD  SRV-SOCKID
                           ZERO-FWRD  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.
*---------------------------------------------------------------*
*                                                               *
*  SOCKET HAS BEEN SET UP, THEN CALL 'ACCEPT' (INTEGER 1) TO    *
*  ACCEPT A REQUEST WHEN A CONNECTION ARRIVES.                  *
*                                                               *
*  THIS SAMPLE PROGRAM WILL ONLY USE 5 SOCKETS.                 *
*                                                               *
*---------------------------------------------------------------*
 ACCEPT-CLIENT-REQ.
     CALL 'EZACICAL' USING  TCP-TOKEN     SELECT-CMD
                            LOM           NFDS
                            NONZERO-FWRD  NONZERO-FWRD
                            ZERO-FWRD     ZERO-FWRD
                            TIMEVAL       READMASK
                            DUMYMASK      DUMYMASK
                            ZERO-8        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.
*---------------------------------------------------------------*
*                                                               *
```

Figure 143. EZACICSS IPv4 iterative server sample (Part 14 of 20)

```
*  ACCEPT REQUEST                                              *
*                                                              *
*--------------------------------------------------------------*
     MOVE -1 TO CLI-SOCKID-FWD.
     CALL 'EZACICAL' USING TCP-TOKEN       ACCEPT-CMD
                           SRV-SOCKID      ZERO-FWRD
                           CLI-SOCKID-FWD  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.
     CALL 'EZACICAL' USING TCP-TOKEN  CLOSE-CMD  CLI-SOCKID
                           ZERO-8     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 'EZACICAL' USING TCP-TOKEN    RECVFROM-CMD  CLI-SOCKID
                           ZERO-FWRD    TCP-FLAG      TCPLENG
                           SOCKADDR-IN  TCP-BUF       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 TOEBCDIC-TOKEN
                                     TCP-BUF
```

Figure 143. EZACICSS IPv4 iterative server sample (Part 15 of 20)

```
                                     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
                       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
```

Figure 143. EZACICSS IPv4 iterative server sample (Part 16 of 20)

```
                    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
                  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 TOASCII-TOKEN TCP-BUF TCPLENG
          CALL 'EZACICAL' USING TCP-TOKEN  WRITE-CMD  CLI-SOCKID
                                TCPLENG    ZERO-FWRD  ZERO-PARM
                                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 143. EZACICSS IPv4 iterative server sample (Part 17 of 20)

```
          END-IF
     END-IF.
 TALK-CLIENT-EXIT.
     EXIT.
*----------------------------------------------------------------*
*                                                                *
*   CLOSE ORIGINAL SOCKET DESCRIPTOR                             *
*                                                                *
*----------------------------------------------------------------*
 CLOSE-SOCKET.
     CALL 'EZACICAL' USING TCP-TOKEN  CLOSE-CMD  SRV-SOCKID
                           ZERO-8     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
                  ELSE
                    IF RESPONSE = DFHRESP(IOERR)
                       THEN MOVE TS-IOERR-ERR TO MSG-AREA
                       ELSE MOVE WRITETS-ERR  TO MSG-AREA
```

Figure 143. EZACICSS IPv4 iterative server sample (Part 18 of 20)

```
                    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 TOASCII-TOKEN TCP-BUF TCPLENG
        MOVE ' '                    TO TCP-INDICATOR
        CALL 'EZACICAL' USING TCP-TOKEN  WRITE-CMD  CLI-SOCKID
                              TCPLENG    ZERO-FWRD  ZERO-PARM
                              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)
     END-EXEC.
     MOVE LOW-VALUES      TO TCP-BUF.
     MOVE TCPCICS-MSG-2   TO TCP-BUF.
     CALL 'EZACIC04' USING TOASCII-TOKEN TCP-BUF TCPLENG.
     CALL 'EZACICAL' USING TCP-TOKEN  WRITE-CMD  CLI-SOCKID
                           TCPLENG    ZERO-FWRD  ZERO-PARM
                           TCP-BUF    ERRNO      RETCODE.
     IF RETCODE < 0 THEN
        MOVE ERRNO        TO WRITE-ERRNO
        MOVE WRITE-ERR    TO MSG-AREA
```

Figure 143. EZACICSS IPv4 iterative server sample (Part 19 of 20)

```
        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 143. EZACICSS IPv4 iterative server sample (Part 20 of 20)

EZACIC6C

The following COBOL socket program is in the *hlq*.SEZAINST data set.

```
* $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. *
*                                                              *
*  COPYRIGHT = LICENSED MATERIALS - PROPERTY OF IBM            *
*              5694-A01 (C) COPYRIGHT IBM CORP. 2003           *
*              This module is restricted materials of IBM      *
*              REFER TO IBM COPYRIGHT INSTRUCTIONS.            *
*                                                              *
*   Status :  CSV1R5                                           *
*                                                              *
*                                                              *
*--------------------------------------------------------------*
*
 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      '.
 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)
```

Figure 144. EZACIC6C IPv6 child server sample (Part 1 of 13)

```
            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'.
       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-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 '.
```

Figure 144. EZACIC6C IPv6 child server sample (Part 2 of 13)

```
*---------------------------------------------------------------*
*    program's variables                                        *
*---------------------------------------------------------------*

 77  SUBTRACE                PIC X(8)  VALUE 'CONTRACE'.
 77  BITMASK-TOKEN           PIC X(16) VALUE 'TCPIPBITMASKCOBL'.
 77  TOEBCDIC-TOKEN          PIC X(16) VALUE 'TCPIPTOEBCDICXLT'.
 77  TOASCII-TOKEN           PIC X(16) VALUE 'TCPIPTOASCIIXLAT'.
 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 DUMMYMASK               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-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.
```

Figure 144. EZACIC6C IPv6 child server sample (Part 3 of 13)

```
           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.
             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.
```

Figure 144. EZACIC6C IPv6 child server sample (Part 4 of 13)

```
      *
      * 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 FILLER                        PIC X(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.
           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'.
```

Figure 144. EZACIC6C IPv6 child server sample (Part 5 of 13)

```
 CLOSE-SOCK.
*---------------------------------------------------------------*
*                                                               *
*   CLOSE 'accept descriptor'                                   *
*                                                               *
*---------------------------------------------------------------*

     CALL 'EZASOKET' USING SOKET-CLOSE SOCKID
           ERRNO RETCODE.

     IF RETCODE <  0 THEN
        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.
```

Figure 144. EZACIC6C IPv6 child server sample (Part 6 of 13)

```
           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.
           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.                                              *
      *                                                               *
      *---------------------------------------------------------------*
```

Figure 144. EZACIC6C IPv6 child server sample (Part 7 of 13)

```
 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
        MOVE 'Y' TO WRITE-SW
        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
```

Figure 144. EZACIC6C IPv6 child server sample (Part 8 of 13)

```
         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 TOASCII-TOKEN TCP-BUF TCPLENG.

     CALL 'EZASOKET' USING SOKET-WRITE SOCKID TCPLENG
           TCP-BUF ERRNO RETCODE.

     IF RETCODE <  0 THEN
        MOVE 'Y' TO WRITE-SW
        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
        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.
```

Figure 144. EZACIC6C IPv6 child server sample (Part 9 of 13)

```
        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
               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
            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
               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
```

Figure 144. EZACIC6C IPv6 child server sample (Part 10 of 13)

```
            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 TOEBCDIC-TOKEN 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 TOASCII-TOKEN TCP-BUF TCPLENG.
            CALL 'EZASOKET' USING SOKET-WRITE SOCKID TCPLENG
                 TCP-BUF ERRNO RETCODE.


            IF RETCODE <  0 THEN
               MOVE 'Y' TO WRITE-SW
               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.
            IF WRITE-SW = 'Y' THEN
                MOVE 78 TO MSGLENG
```

Figure 144. EZACIC6C IPv6 child server sample (Part 11 of 13)

```
         MOVE EIBTASKN TO TASK-NUMBER
         EXEC CICS WRITEQ TD QUEUE('CSMT') FROM(TD-MSG)
              LENGTH(MSGLENG) 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 TOASCII-TOKEN TCP-BUF TCPLENG.
        CALL 'EZASOKET' USING SOKET-WRITE SOCKID TCPLENG
             TCP-BUF ERRNO RETCODE.

        IF RETCODE <  0 THEN
           MOVE 'Y' TO WRITE-SW
           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
     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
     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
     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
     MOVE NOSPACE-ERR TO CICS-MSG-AREA.
```

Figure 144. EZACIC6C IPv6 child server sample (Part 12 of 13)

```
            PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
            GO TO PGM-EXIT.
        QIDERR-SEC.
            MOVE 'Y' TO WRITE-SW
            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
            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
            MOVE ENDDATA-ERR TO CICS-MSG-AREA.
            PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
            GO TO PGM-EXIT.
```

Figure 144. EZACIC6C IPv6 child server sample (Part 13 of 13)

EZACIC6S

The following COBOL socket program is in the *hlq*.SEZAINST data set.

```
****************************************************************
*                                                              *
* Communications Server for zOS/390, Version 1, Release 5      *
*                                                              *
*                                                              *
* Copyright:    Licensed Materials - Property of IBM           *
*                                                              *
*               "Restricted Materials of IBM"                  *
*                                                              *
*               5694-A01                                       *
*                                                              *
*               (C) Copyright IBM Corp. 2003                   *
*                                                              *
*               US Government Users Restricted Rights -        *
*               Use, duplication or disclosure restricted by   *
*               GSA ADP Schedule Contract with IBM Corp.       *
*                                                              *
* Status:       CSV1R5                                         *
*                                                              *
* $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.        *
*                                                               *
*                  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.                                      *
*                                                               *
*                                                               *
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 1 of 28)

```
*--------------------------------------------------------------*
*                                                              *
*   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)
      VALUE IS 'INPUT KEYWORD ERROR           '.
 77  LENGERR-ERR                     PIC X(30)
      VALUE IS 'LENGERR ERROR                 '.
 77  NOSPACE-ERR                     PIC X(30)
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 2 of 28)

```
            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)
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 3 of 28)

```
                VALUE IS 'CLIENTS HOST NAME: '.
           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'.
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 4 of 28)

```
               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)
                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)
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 5 of 28)

```
                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  BITMASK-TOKEN            PIC X(16) VALUE 'TCPIPBITMASKCOBL'.
       77  TOEBCDIC-TOKEN           PIC X(16) VALUE 'TCPIPTOEBCDICXLT'.
       77  TOASCII-TOKEN            PIC X(16) VALUE 'TCPIPTOASCIIXLAT'.
       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.
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 6 of 28)

```
 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     '.
     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.
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 7 of 28)

```
       77  L-DESC                   PIC 9(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.
           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.
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 8 of 28)

```
      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.
    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.
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 9 of 28)

```
           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.
                    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  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).
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 10 of 28)

```
           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),
      ***    MGRNO                          CHAR(6),
      ***    ADMRDEPT                       CHAR(3)
      ***  ) END-EXEC.
      ******************************************************************
      * COBOL DECLARATION FOR TABLE TCPCICS.DEPT                       *
      ******************************************************************
       01  DCLDEPT.
           10 DEPTNO               PIC X(3).
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 11 of 28)

```
     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 *
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 12 of 28)

```
      *  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.
      *
      *
      *---------------------------------------------------------------*
      *                                                               *
      *  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
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 13 of 28)

```
     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.

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 BITMASK-TOKEN
                      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
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 14 of 28)

```
           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.


      *----------------------------------------------------------------*
      *                                                                *
      *  LISTENER STARTED TASK                                         *
      *                                                                *
      *----------------------------------------------------------------*

       LISTENER-STARTED-TASK.
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 15 of 28)

```
MOVE CLIENTID-PARM               TO CID-LSTN-INFO.
MOVE -1 TO L-DESC.

CALL 'EZASOKET' USING SOKET-TAKESOCKET
                      GIVE-TAKE-SOCKET
                      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 TOASCII-TOKEN 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
          END-IF
      END-IF
END-IF.
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 16 of 28)

```
     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
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 17 of 28)

```
                           SOCK-TYPE
                           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.
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 18 of 28)

```
     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.


*---------------------------------------------------------------*
*                                                               *
*  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
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 19 of 28)

```
                           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.

     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.
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 20 of 28)

```
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
   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
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 21 of 28)

```
        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 TOEBCDIC-TOKEN
                                     TCP-BUF
                                     TCPLENG
               IF TCP-BUF-H = LOW-VALUES OR SPACES
                  THEN
                    MOVE NULL-DATA           TO MSG-AREA
                    PERFORM HANDLE-TCPCICS   THRU
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 22 of 28)

```
                            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
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 23 of 28)

```
             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
                  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 TOASCII-TOKEN TCP-BUF TCPLENG

          CALL 'EZASOKET' USING SOKET-WRITE
                                CLI-SOCKID
                                TCPLENG
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 24 of 28)

```
                                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
          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.
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 25 of 28)

```
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
             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 TOASCII-TOKEN TCP-BUF TCPLENG

   MOVE ' '                    TO TCP-INDICATOR

   CALL 'EZASOKET' USING SOKET-WRITE
                         CLI-SOCKID
                         TCPLENG
                         TCP-BUF
                         ERRNO
                         RETCODE
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 26 of 28)

```
              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)
           END-EXEC.

           MOVE LOW-VALUES      TO TCP-BUF.
           MOVE TCPCICS-MSG-2   TO TCP-BUF.

           CALL 'EZACIC04' USING TOASCII-TOKEN TCP-BUF TCPLENG.

           CALL 'EZASOKET' USING SOKET-WRITE
                                 CLI-SOCKID
                                 TCPLENG
                                 TCP-BUF
                                 ERRNO
                                 RETCODE.
```

Figure 145. EZACIC6S IPv6 iterative server sample (Part 27 of 28)

```
     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 145. EZACIC6S IPv6 iterative server sample (Part 28 of 28)

EZACICAC

The following COBOL socket program is in the *hlq*.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                                              *
*                                                                     *
*               (C) Copyright IBM Corp. 2003                          *
*                                                                     *
*               US Government Users Restricted Rights -               *
*               Use, duplication or disclosure restricted by          *
*               GSA ADP Schedule Contract with IBM Corp.              *
*                                                                     *
* Status:       CSV1R5                                                *
*                                                                     *
*                                                                     *
*   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
SOCSTG   DS    0F                  PROGRAM STORAGE
*
* Storage to format messages
*
TDMSG    DS    0F                  WRITEQ TD Message area
```

Figure 146. EZACICAC assembler child server sample (Part 1 of 10)

```
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
*
* 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
```

Figure 146. EZACICAC assembler child server sample (Part 2 of 10)

```
SOKLTID  DS    CL8                Listener task identifier
SOKDATA1 DS    CL35               Client input data
         DS    CL1                Unused
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)
*
* Issue the 'TAKESOCKET' call to acquire the socket which was
* given by the listener program.
*
         XC    CLIENTID,CLIENTID  Clear the clientid structure
```

Figure 146. EZACICAC assembler child server sample (Part 3 of 10)

```
         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,(TOASCII_TOKEN,TCP_BUF,TCPLENG),               X
*              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?
         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
```

Figure 146. EZACICAC assembler child server sample (Part 4 of 10)

```
*
* 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
*
AGAIN1   DS    0H
*
         XC    TCP_BUF,TCP_BUF    Clear the buffer storage
*
         CALL  EZASOKET,(SOCRECV,SOCDESC,RECV_FLAG,TCPLENG,TCP_BUF,    X
```

Figure 146. EZACICAC assembler child server sample (Part 5 of 10)

```
               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 EZACIC04 if using an EBCDIC client.
*
*       CALL  EZACIC04,(TOASCII_TOKEN,TCP_BUF,TCPLENG),               X
*             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
*
* 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
*
* 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,(TOASCII_TOKEN,TCP_BUF,TCPLENG),               X
*             VL,MF=(E,PARMLIST)
*
* Notify the client that the connection will end.
```

Figure 146. EZACICAC assembler child server sample (Part 6 of 10)

```
*
         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
         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
*
         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
         EXEC CICS ASKTIME ABSTIME(UTIME)
         EXEC CICS FORMATTIME ABSTIME(UTIME)                            X
              DATESEP('/') DDMMYY(TDDATE)                              X
```

Figure 146. EZACICAC assembler child server sample (Part 7 of 10)

```
              TIME(TDTIME) TIMESEP
         LA    R6,TDMSGL
         STH   R6,TDLEN
         EXEC CICS WRITEQ TD QUEUE('CSMT')                             X
               FROM(TDMSG)                                             X
               LENGTH(TDLEN)
         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       '
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         '
```

Figure 146. EZACICAC assembler child server sample (Part 8 of 10)

```
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'
*
* EZACIC06 parms
*
TOEBCDIC_TOKEN DC CL16'TCPIPTOEBCDICXLT'
TOASCII_TOKEN DC CL16'TCPIPTOASCIIXLAT'
*
* 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
         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
*
```

Figure 146. EZACICAC assembler child server sample (Part 9 of 10)

```
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          '
         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 146. EZACICAC assembler child server sample (Part 10 of 10)

EZACICAS

The following COBOL socket program is in the *hlq*.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                                              *
*                                                                     *
*               (C) Copyright IBM Corp. 2003                          *
*                                                                     *
*               US Government Users Restricted Rights -               *
*               Use, duplication or disclosure restricted by          *
*               GSA ADP Schedule Contract with IBM Corp.              *
*                                                                     *
* Status:       CSV1R5                                                *
*                                                                     *
*                                                                     *
*   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):                              *
*                                                                     *
*                                                                     *
***********************************************************************
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 ...
```

Figure 147. EZACICAS assembler iterative server sample (Part 1 of 20)

```
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)
*
*  THE LEADING SPACES ARE SIGNIFICANT.
*
         XC    TCP_INPUT_DATA,TCP_INPUT_DATA Clear input data area
```

Figure 147. EZACICAS assembler iterative server sample (Part 2 of 20)

```
         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
*
         CALL  EZASOKET,(SOCINIT,MAXSOC,IDENT,INIT_SUBTASKID,MAXSNO,   X
               ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
```

Figure 147. EZACICAS assembler iterative server sample (Part 3 of 20)

```
         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
*
         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
```

Figure 147. EZACICAS assembler iterative server sample (Part 4 of 20)

```
*
* 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
         BNE   SET_SELECT_BIT_MASK No, continue with SELECT
         B     CLOSEDOWN
*
* Create the read bitmask
```

Figure 147. EZACICAS assembler iterative server sample (Part 5 of 20)

```
*
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
         MVC   MSGRESULT(L'SUCC),SUCC Move SUCCESSFUL msg to TD area
         BAL   R7,HANDLE_TCPCICS  Write to TD Queue
*
* Get our peers' socket address
```

Figure 147. EZACICAS assembler iterative server sample (Part 6 of 20)

```
*
         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
*
         MVC   TDSERV(L'TDSERV),PEER_SERVICENAME
         MVC   MSGAREA(L'TDSERVMSG),TDSERVMSG Move message to TD area
         BAL   R7,HANDLE_TCPCICS  Write to TD Queue
```

Figure 147. EZACICAS assembler iterative server sample (Part 7 of 20)

```
*
* 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,(TOEBCDIC_TOKEN,TCP_BUF,TCPLENG),             X
*              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.
*
* Inform the cleint that the server has process the message
*
         XC    MSGAREA,MSGAREA
         MVC   MSGAREA(L'SERVER_PROC_MSG),SERVER_PROC_MSG
```

Figure 147. EZACICAS assembler iterative server sample (Part 8 of 20)

```
*
         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,(TOASCII_TOKEN,TCP_BUF,TCPLENG),               X
*              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
         MVI   TASK_FLAG,C'1'
         B     CLOSE_CLIENT
SET_TERM DS 0H
         MVI   TASK_FLAG,C'2'
         B     CLOSE_CLIENT
*
```

Figure 147. EZACICAS assembler iterative server sample (Part 9 of 20)

```
*   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
         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.
```

Figure 147. EZACICAS assembler iterative server sample (Part 10 of 20)

```
*
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,(TOASCII_TOKEN,TCP_BUF,TCPLENG),               X
*              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
*
* 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
```

Figure 147. EZACICAS assembler iterative server sample (Part 11 of 20)

```
         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
         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
```

Figure 147. EZACICAS assembler iterative server sample (Part 12 of 20)

```
*
         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,(TOASCII_TOKEN,TCP_BUF,TCPLENG),               X
*              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?
         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
```

Figure 147. EZACICAS assembler iterative server sample (Part 13 of 20)

```
         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     '
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            '
```

Figure 147. EZACICAS assembler iterative server sample (Part 14 of 20)

```
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          '
*
* EZACIC04/EZACIC05 parms
*
TOEBCDIC_TOKEN DC CL16'TCPIPTOEBCDICXLT'
TOASCII_TOKEN DC CL16'TCPIPTOASCIIXLAT'
*
* 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' '
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' '
```

Figure 147. EZACICAS assembler iterative server sample (Part 15 of 20)

```
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
*
TDTEXT5  DS    0CL40
         DC    CL10'RETCODE = '
TDRETC   DC    CL7' '             Printable RETCODE
         DC    CL3' '
         DC    CL9'ERRNO = '
```

Figure 147. EZACICAS assembler iterative server sample (Part 16 of 20)

```
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
         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
```

Figure 147. EZACICAS assembler iterative server sample (Part 17 of 20)

```
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
RECVFROM_RETCODE DS F
*
* Client ID from Listener to be used by the TAKESOKET command
*
CLIENTID_LSTN DS 0CL40
```

Figure 147. EZACICAS assembler iterative server sample (Part 18 of 20)

```
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
         DS    20F
STIM_SIN#LEN EQU *-STIM_SIN
          ORG  STIM_DATA
STIM_SIN6 DS 0C
STIM_SIN6_PORT DS H
```

Figure 147. EZACICAS assembler iterative server sample (Part 19 of 20)

```
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 147. EZACICAS assembler iterative server sample (Part 20 of 20)

Appendix F. Related protocol specifications (RFCs)

This appendix lists the related protocol specifications 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.

These documents can be obtained from:

Government Systems, Inc.
Attn: Network Information Center
14200 Park Meadow Drive
Suite 200
Chantilly, VA 22021

You can see Internet drafts at http://www.ietf.org/ID.html. See "Internet Drafts" on page 495 for draft RFCs implemented in this and previous Communications Server releases.

You can also request RFCs through electronic mail, from the automated 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`.

Hard copies of all RFCs are available from the NIC, either individually or by subscription. Many RFCs are available online. Online copies are available using FTP from the NIC at the following Web address: http://www.rfc-editor.org/rfc.html.

Use FTP to download the files, using the following format:

```
RFC:RFC-INDEX.TXT
RFC:RFCnnnn.TXT
RFC:RFCnnnn.PS
```

where:

nnnn	Is the RFC number.
TXT	Is the text format.
PS	Is the PostScript format.

Many features of TCP/IP Services are based on the following RFCs:

RFC	Title and Author
768	*User Datagram Protocol* J.B. Postel
791	*Internet Protocol* J.B. Postel
792	*Internet Control Message Protocol* J.B. Postel

793 *Transmission Control Protocol* J.B. Postel

821 *Simple Mail Transfer Protocol* J.B. Postel

822 *Standard for the Format of ARPA Internet Text Messages* D. Crocker

823 *DARPA Internet Gateway* R.M. Hinden, A. Sheltzer

826 *Ethernet Address Resolution Protocol or Converting Network Protocol Addresses to 48.Bit Ethernet Address for Transmission on Ethernet Hardware* D.C. Plummer

854 *Telnet Protocol Specification* J.B. Postel, J.K. Reynolds

855 *Telnet Option Specification* J.B. Postel, J.K. Reynolds

856 *Telnet Binary Transmission* J.B. Postel, J.K. Reynolds

857 *Telnet Echo Option* J.B. Postel, J.K. Reynolds

858 *Telnet Suppress Go Ahead Option* J.B. Postel, J.K. Reynolds

859 *Telnet Status Option* J.B. Postel, J.K. Reynolds

860 *Telnet Timing Mark Option* J.B. Postel, J.K. Reynolds

861 *Telnet Extended Options—List Option* J.B. Postel, J.K. Reynolds

862 *Echo Protocol* J.B. Postel

863 *Discard Protocol* J.B. Postel

864 *Character Generator Protocol* J.B. Postel

877 *Standard for the Transmission of IP Datagrams over Public Data Networks* J.T. Korb

885 *Telnet End of Record Option* J.B. Postel

896 *Congestion Control in IP/TCP Internetworks* J. Nagle

903 *Reverse Address Resolution Protocol* R. Finlayson, T. Mann, J.C. Mogul, M. Theimer

904 *Exterior Gateway Protocol Formal Specification* D.L. Mills

919 *Broadcasting Internet Datagrams* J.C. Mogul

922 *Broadcasting Internet Datagrams in the Presence of Subnets* J.C. Mogul

950 *Internet Standard Subnetting Procedure* J.C. Mogul, J.B. Postel

952 *DoD Internet Host Table Specification* K. Harrenstien, M.K. Stahl, E.J. Feinler

959 *File Transfer Protocol* J.B. Postel, J.K. Reynolds

974 *Mail Routing and the Domain Name System* C. Partridge

1006 *ISO Transport Service on top of the TCP Version 3* M.T.Rose, D.E. Cass

1009 *Requirements for Internet Gateways* R.T. Braden, J.B. Postel

1011 *Official Internet Protocols* J. Reynolds, J. Postel

1013 *X Window System Protocol, Version 11: Alpha Update* R.W. Scheifler

1014 *XDR: External Data Representation Standard* Sun Microsystems Incorporated

1027 *Using ARP to Implement Transparent Subnet Gateways* S. Carl-Mitchell, J.S. Quarterman

1032 *Domain Administrators Guide* M.K. Stahl

1033 *Domain Administrators Operations Guide* M. Lottor

1034 *Domain Names—Concepts and Facilities* P.V. Mockapetris

1035 *Domain Names—Implementation and Specification* P.V. Mockapetris

1042 *Standard for the Transmission of IP Datagrams over IEEE 802 Networks* J.B. Postel, J.K. Reynolds

1044 *Internet Protocol on Network System's HYPERchannel: Protocol Specification* K. Hardwick, J. Lekashman

1055 *Nonstandard for Transmission of IP Datagrams over Serial Lines: SLIP* J.L. Romkey

1057 *RPC: Remote Procedure Call Protocol Version 2 Specification* Sun Microsystems Incorporated

1058 *Routing Information Protocol* C.L. Hedrick

1060 *Assigned Numbers* J. Reynolds, J. Postel

1073 *Telnet Window Size Option* D. Waitzman

1079 *Telnet Terminal Speed Option* C.L. Hedrick

1091 *Telnet Terminal-Type Option* J. VanBokkelen

1094 *NFS: Network File System Protocol Specification* Sun Microsystems Incorporated

1096 *Telnet X Display Location Option* G. Marcy

1101 *DNS encoding of network names and other types* P.V. Mockapetris

1112 *Host Extensions for IP Multicasting* S. Deering

1118 *Hitchhikers Guide to the Internet* E. Krol

1122 *Requirements for Internet Hosts—Communication Layers* R.T. Braden

1123 *Requirements for Internet Hosts—Application and Support* R.T. Braden

1155 *Structure and Identification of Management Information for TCP/IP-Based Internets* M.T. Rose, K. McCloghrie

1156 *Management Information Base for Network Management of TCP/IP-Based Internets* K. McCloghrie, M.T. Rose

1157 *Simple Network Management Protocol (SNMP)* J.D. Case, M. Fedor, M.L. Schoffstall, C. Davin

1158 *Management Information Base for Network Management of TCP/IP-based internets: MIB-II* M.T. Rose

1179 *Line Printer Daemon Protocol* The Wollongong Group, L. McLaughlin III

1180 *TCP/IP Tutorial* T.J. Socolofsky, C.J. Kale

1183 *New DNS RR Definitions* C.F. Everhart, L.A. Mamakos, R. Ullmann, P.V. Mockapetris, (Updates RFC 1034, RFC 1035)

1184 *Telnet Linemode Option* D. Borman

1187 *Bulk Table Retrieval with the SNMP* M.T. Rose, K. McCloghrie, J.R. Davin

1188 *Proposed Standard for the Transmission of IP Datagrams over FDDI Networks* D. Katz

1191 *Path MTU Discovery* J. Mogul, S. Deering

1198 *FYI on the X Window System* R.W. Scheifler

1207 *FYI on Questions and Answers: Answers to Commonly Asked “Experienced Internet User” Questions* G.S. Malkin, A.N. Marine, J.K. Reynolds

1208 *Glossary of Networking Terms* O.J. Jacobsen, D.C. Lynch

1213 *Management Information Base for Network Management of TCP/IP-Based Internets: MIB-II* K. McCloghrie, M.T. Rose

1215 *Convention for Defining Traps for Use with the SNMP* M.T. Rose

1228 *SNMP-DPI Simple Network Management Protocol Distributed Program Interface* G.C. Carpenter, B. Wijnen

1229 *Extensions to the Generic-Interface MIB* K. McCloghrie

1230 *IEEE 802.4 Token Bus MIB* K. McCloghrie, R. Fox

1231 *IEEE 802.5 Token Ring MIB* K. McCloghrie, R. Fox, E. Decker

1236 *IP to X.121 Address Mapping for DDN* L. Morales, P. Hasse

1267 *A Border Gateway Protocol 3 (BGP-3)* K. Lougheed, Y. Rekhter

1268 *Application of the Border Gateway Protocol in the Internet* Y. Rekhter, P. Gross

1269 *Definitions of Managed Objects for the Border Gateway Protocol (Version 3)* S. Willis, J. Burruss

1270 *SNMP Communications Services* F. Kastenholz, ed.

1321 *The MD5 Message-Digest Algorithm* R. Rivest

1323 *TCP Extensions for High Performance* V. Jacobson, R. Braden, D. Borman

1325 *FYI on Questions and Answers: Answers to Commonly Asked "New Internet User" Questions* G.S. Malkin, A.N. Marine

1340 *Assigned Numbers* J.K. Reynolds, J.B. Postel

1348 *DNS NSAP RRs* B. Manning

1349 *Type of Service in the Internet Protocol Suite* P. Almquist

1350 *TFTP Protocol* K.R. Sollins

1351 *SNMP Administrative Model* J. Davin, J. Galvin, K. McCloghrie

1352 *SNMP Security Protocols* J. Galvin, K. McCloghrie, J. Davin

1353 *Definitions of Managed Objects for Administration of SNMP Parties* K. McCloghrie, J. Davin, J. Galvin

1354 *IP Forwarding Table MIB* F. Baker

1356 *Multiprotocol Interconnect on X.25 and ISDN in the Packet Mode* A. Malis, D. Robinson, R. Ullmann

1363 *A Proposed Flow Specification* C. Partridge

1372 *Telnet Remote Flow Control Option* D. Borman, C. L. Hedrick

1374 *IP and ARP on HIPPI* J. Renwick, A. Nicholson

1381 *SNMP MIB Extension for X.25 LAPB* D. Throop, F. Baker

1382 *SNMP MIB Extension for the X.25 Packet Layer* D. Throop

1387 *RIP Version 2 Protocol Analysis* G. Malkin

1388 *RIP Version 2—Carrying Additional Information* G. Malkin

1389 *RIP Version 2 MIB Extension* G. Malkin

1390 *Transmission of IP and ARP over FDDI Networks* D. Katz

1393 *Traceroute Using an IP Option* G. Malkin

1397 *Default Route Advertisement In BGP2 And BGP3 Versions of the Border Gateway Protocol* D. Haskin

1398 *Definitions of Managed Objects for the Ethernet-Like Interface Types* F. Kastenholz

1416 *Telnet Authentication Option* D. Borman, ed.

1464 *Using the Domain Name System to Store Arbitrary String Attributes* R. Rosenbaum

1469 *IP Multicast over Token-Ring Local Area Networks* T. Pusateri

1535 *A Security Problem and Proposed Correction With Widely Deployed DNS Software* E. Gavron

1536 *Common DNS Implementation Errors and Suggested Fixes* A. Kumar, J. Postel, C. Neuman, P. Danzig, S.Miller

1537 *Common DNS Data File Configuration Errors* P. Beertema

1540 *IAB Official Protocol Standards* J.B. Postel

1571 *Telnet Environment Option Interoperability Issues* D. Borman

1572 *Telnet Environment Option* S. Alexander

1577 *Classical IP and ARP over ATM* M. Laubach

1583 *OSPF Version 2* J. Moy

1591 *Domain Name System Structure and Delegation* J. Postel

1592 *Simple Network Management Protocol Distributed Protocol Interface Version 2.0* B. Wijnen, G. Carpenter, K. Curran, A. Sehgal, G. Waters

1594 *FYI on Questions and Answers: Answers to Commonly Asked "New Internet User" Questions* A.N. Marine, J. Reynolds, G.S. Malkin

1695 *Definitions of Managed Objects for ATM Management Version 8.0 Using SMIv2* M. Ahmed, K. Tesink

1706 *DNS NSAP Resource Records* B. Manning, R. Colella

1713 *Tools for DNS debugging* A. Romao

1723 *RIP Version 2—Carrying Additional Information* G. Malkin

1766 *Tags for the Identification of Languages* H. Alvestrand

1794 *DNS Support for Load Balancing* T. Brisco

1832 *XDR: External Data Representation Standard* R. Srinivasan

1850 *OSPF Version 2 Management Information Base* F. Baker, R. Coltun

1876 *A Means for Expressing Location Information in the Domain Name System* C. Davis, P. Vixie, T. Goodwin, I. Dickinson

1886 *DNS Extensions to support IP version 6* S. Thomson, C. Huitema

1901 *Introduction to Community-Based SNMPv2* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

1902 *Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

1903 *Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

1904 *Conformance Statements for Version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

1905 *Protocols Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

1906 *Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

1907 *Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

1908 *Coexistence between Version 1 and Version 2 of the Internet-Standard Network Management Framework* J. Case, K. McCloghrie, M. Rose, S. Waldbusser

1912 *Common DNS Operational and Configuration Errors* D. Barr

1918 *Address Allocation for Private Internets* Y. Rekhter, B. Moskowitz, D. Karrenberg, G.J. de Groot, E. Lear

1928 *SOCKS Protocol Version 5* M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, L. Jones

1939 *Post Office Protocol-Version 3* J. Myers, M. Rose

1981 *Path MTU Discovery for IP version 6* J. McCann, S. Deering, J. Mogul

1982 *Serial Number Arithmetic* R. Elz, R. Bush

1995 *Incremental Zone Transfer in DNS* M. Ohta

1996 *A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)* P. Vixie

2010 *Operational Criteria for Root Name Servers* B. Manning, P. Vixie

2011 *SNMPv2 Management Information Base for the Internet Protocol Using SMIv2* K. McCloghrie

2012 *SNMPv2 Management Information Base for the Transmission Control Protocol Using SMIv2* K. McCloghrie

2013 *SNMPv2 Management Information Base for the User Datagram Protocol Using SMIv2* K. McCloghrie

2030 *Simple Network Time Protocol* D. Mills

2052 *A DNS RR for specifying the location of services (DNS SRV)* A. Gulbrandsen, P. Vixie

2065 *Domain Name System Security Extensions* D. Eastlake, C. Kaufman

2080 *RIPng for IPv6* G. Malkin, R. Minnear

2096 *IP Forwarding Table MIB* F. Baker

2104 *HMAC: Keyed-Hashing for Message Authentication* H. Krawczyk, M. Bellare, R. Canetti

2132 *DHCP Options and BOOTP Vendor Extensions* S. Alexander, R. Droms

2133 *Basic Socket Interface Extensions for IPv6* R. Gilligan, S. Thomson, J. Bound, W. Stevens

2137 *Secure Domain Name System Dynamic Update* D. Eastlake

2163 *Using the Internet DNS to Distribute MIXER Conformant Global Address Mapping (MCGAM)* C. Allocchio

2168 *Resolution of Uniform Resource Identifiers using the Domain Name System* R. Daniel, M. Mealling

2178 *OSPF Version 2* J. Moy

2181 *Clarifications to the DNS Specification* R. Elz, R. Bush

2205 *Resource ReSerVation Protocol (RSVP) Version 1* R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin

2210 *The Use of RSVP with IETF Integrated Services* J. Wroclawski

2211 *Specification of the Controlled-Load Network Element Service* J. Wroclawski

2212 *Specification of Guaranteed Quality of Service* S. Shenker, C. Partridge, R. Guerin

2215 *General Characterization Parameters for Integrated Service Network Elements* S. Shenker, J. Wroclawski

2219 *Use of DNS Aliases for Network Services* M. Hamilton, R. Wright

2228 *FTP Security Extensions* M. Horowitz, S. Lunt

2230 *Key Exchange Delegation Record for the DNS* R. Atkinson

2233 *The Interfaces Group MIB Using SMIv2* K. McCloghrie, F. Kastenholz

2240 *A Legal Basis for Domain Name Allocation* O. Vaughn

2246 *The TLS Protocol Version 1.0* T. Dierks, C. Allen

2308 *Negative Caching of DNS Queries (DNS NCACHE)* M. Andrews

2317 *Classless IN-ADDR.ARPA delegation* H. Eidnes, G. de Groot, P. Vixie

2320 *Definitions of Managed Objects for Classical IP and ARP over ATM Using SMIv2* M. Greene, J. Luciani, K. White, T. Kuo

2328 *OSPF Version 2* J. Moy

2345 *Domain Names and Company Name Retrieval* J. Klensin, T. Wolf, G. Oglesby

2352 *A Convention for Using Legal Names as Domain Names* O. Vaughn

2355 *TN3270 Enhancements* B. Kelly

2373 *IP Version 6 Addressing Architecture* R. Hinden, M. O'Dell, S. Deering

2374 *An IPv6 Aggregatable Global Unicast Address Format* R. Hinden, M. O'Dell, S. Deering

2375 *IPv6 Multicast Address Assignments* R. Hinden, S. Deering

2389 *Feature negotiation mechanism for the File Transfer Protocol* P. Hethmon, R. Elz

2428 *FTP Extensions for IPv6 and NATs* M. Allman, S. Ostermann, C. Metz

2460 *Internet Protocol, Version 6 (IPv6) S pecification* S. Deering, R. Hinden

2461 *Neighbor Discovery for IP Version 6 (IPv6)* T. Narten, E. Nordmark, W. Simpson

2462 *IPv6 Stateless Address Autoconfiguration* S. Thomson, T. Narten

2464 *Transmission of IPv6 Packets over Ethernet Networks* M. Crawford

2474 *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers* K. Nichols, S. Blake, F. Baker, D. Black

2535 *Domain Name System Security Extensions* D. Eastlake

2539 *Storage of Diffie-Hellman Keys in the Domain Name System (DNS)* D. Eastlake

2571 *An Architecture for Describing SNMP Management Frameworks* D. Harrington, R. Presuhn, B. Wijnen

2572 *Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)* J. Case, D. Harrington, R. Presuhn, B. Wijnen

2573 *SNMP Applications* D. Levi, P. Meyer, B. Stewart

2574 *User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)* U. Blumenthal, B. Wijnen

2575 *View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)* B. Wijnen, R. Presuhn, K. McCloghrie

2578 *Structure of Management Information Version 2 (SMIv2)* K. McCloghrie, D. Perkins, J. Schoenwaelder

2640 *Internationalization of the File Transfer Protocol* B. Curtin

2665 *Definitions of Managed Objects for the Ethernet-like Interface Types* J. Flick, J. Johnson

2672 *Non-Terminal DNS Name Redirection* M. Crawford

2710 *Multicast Listener Discovery (MLD) for IPv6*S. Deering, W. Fenner, B. Haberman

2711 *IPv6 Router Alert Option*C. Partridge, A. Jackson

2758 *Definitions of Managed Objects for Service Level Agreements Performance Monitoring* K. White

2845 *Secret Key Transaction Authentication for DNS (TSIG)* P. Vixie, O. Gudmundsson, D. Eastlake, B. Wellington

2874 *DNS Extensions to Support IPv6 Address Aggregation and Renumbering* M. Crawford, C. Huitema

2941 *Telnet Authentication Option* T. Ts'o, ed., J. Altman

2942 *Telnet Authentication: Kerberos Version 5* T. Ts'o

2946 *Telnet Data Encryption Option* T. Ts'o

2952 *Telnet Encryption: DES 64 bit Cipher Feedback* T. Ts'o

2953 *Telnet Encryption: DES 64 bit Output Feedback* T. Ts'o, ed.

3060 *Policy Core Information Model—Version 1 Specification* B. Moore, E. Ellesson, J. Strassner, A. Westerinen

3484 *Default Address Selection for Internet Protocol version 6 (IPv6)* R. Draves

3493 *Basic Socket Interface Extensions for IPv6* R.E. Gilligan, S. Thomson, J. Bound, J. McCann, W. R. Stevens

3513 *Internet Protocol Version 6 (IPv6) Addressing Architecture*R. Hinden, S. Deering

3542 *Advanced Sockets API for IPv6*W. Richard Stevens, Matt Thomas, Erik Nordmark, Tatuya Jinmei

Internet Drafts

Several areas of IPv6 implementation include elements of the following Internet drafts and are subject to change during the RFC review process.

Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification
A. Conta, S. Deering

Appendix G. Information APARs

This appendix lists information APARs for IP and SNA documents.

Notes:

1. Information APARs contain updates to previous editions of the manuals listed below. Documents updated for V1R5 are complete except for the updates contained in the information APARs that may be issued after V1R5 documents went to press.
2. Information APARs are predefined for z/OS V1R5 Communications Server and may not contain updates.
3. Information APARs for OS/390 documents are in the document called *OS/390 DOC APAR and PTF ++HOLD Documentation*, which can be found at http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/IDDOCMST/CCONTENTS.
4. 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:80/cgi-bin/bookmgr_OS390/BOOKS/ZIDOCMST/CCONTENTS.

Information APARs for IP documents

Table 22 lists information APARs for IP documents.

Table 22. IP information APARs

Title	z/OS CS V1R5	z/OS CS V1R4	z/OS CS V1R2	CS for OS/390 2.10 and z/OS CS V1R1
IP API Guide	II13577	ii13255	ii12861	ii12371
IP CICS Sockets Guide	II13578	ii13257	ii12862	
IP Configuration Guide	II13568	ii13244 ii13541	ii12498 ii13087 ii13364	ii12362 ii12493 ii13006
IP Configuration Reference	II13569	ii13245 ii13521	ii12499 ii13394	ii12363 ii12494 ii12712 ii13325 ii13406
IP Diagnosis	II13571	ii13249 ii13493	ii12503 ii13473	ii12366 ii12495 ii13472
IP Messages Volume 1	II13572	ii13250	ii12857 ii13229 ii13405	ii12367
IP Messages Volume 2	II13573	ii13251	ii12858	ii12368
IP Messages Volume 3	II13574	ii13252	ii12859	ii12369 ii12990
IP Messages Volume 4	II13575	ii13253	ii12860	
IP Migration	II13566	ii13242	ii12497	ii12361

Table 22. IP information APARs (continued)

Title	z/OS CS V1R5	z/OS CS V1R4	z/OS CS V1R2	CS for OS/390 2.10 and z/OS CS V1R1
IP Network and Application Design Guide	II13567	ii13243		
IP Network Print Facility			ii12864	
IP Programmer's Reference	II13581	ii13256	ii12505	
IP and SNA Codes	II13576	ii13254	ii12504	ii12370
IP User's Guide				ii12365 ii13060
IP User's Guide and Commands	II13570	ii13247	ii12501 ii13404	ii12365 ii13060 ii13403
IP System Admin Guide	II13580	ii13248	ii12502	
Quick Reference		ii13246	ii12500	ii12364

Information APARs for SNA documents

Table 23 lists information APARs for SNA documents.

Table 23. SNA information APARs

Title	z/OS CS V1R5	z/OS CS V1R4	z/OS CS V1R2	CS for OS/390 2.10 and z/OS CS V1R1
IP and SNA Codes	II13576	ii13254	ii12504	ii12370
SNA Customization	II13560	ii13240	ii12872	ii12388
SNA Diagnosis	II13558	ii13236	ii12490 ii13034	ii12389
SNA Messages	II13559	ii13238	ii12491	ii12382 ii12383
SNA Network Implementation Guide	II13555	ii13234	ii12487	ii12381 ii12383
SNA Operation	II13557	ii13237	ii12489	ii12384
SNA Migration	II13554	ii13233	ii12486	ii12386
SNA Programming		ii13241	ii13033	ii12385
Quick Reference		ii13246	ii12500	ii12364
SNA Resource Definition Reference	II13556	ii13235	ii12488	ii12380 ii12567
SNA Data Areas		ii13239	ii12492	ii12387

Other information APARs

Table 24 on page 499 lists information APARs not related to documents.

Table 24. Non-document information APARs

Content	Number
index of recommended maintenance for VTAM	ii11220
index of Communication Server IP information APARs	ii12028
AHHC, MPC, and CTC	ii01501
Collecting TCPIP CTRACEs	II12014
CSM for VTAM	ii12657
CSM for TCP/IP	ii12658
DLUR/DLUS for z/OS V1R2	ii12986
DOCUMENTATION REQUIRED FOR OSA/2, OSA EXPRESS AND OSA QDIO	II13016
DYNAMIC VIPA (BIND)	II13215
DNS — common problems and solutions	II13453
Enterprise Extender	ii12223
FTPing doc to z/OS Support	II12030
FTP problems	II12079
Generic resources	ii10986
HPR	ii10953
iQDIO	ii11220
LPR problems	II12022
MNPS	ii10370
NCPROUTE problems	II12025
OMPROUTE	ii12026
OROUTED problems	ii12024
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
Telnet	ii11574 ii13135
TN3270 TELNET 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 it 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 Volume 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.

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.

Copyright © 1983 The Regents of the University of California. All rights reserved.

Redistribution and use in source and binary forms are permitted provided that the above copyright notice and this paragraph are duplicated in all such forms and that any documentation, advertising materials, and other materials related to such distribution and use acknowledge that the software was developed by the University of California, Berkeley. The name of the University may not be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED ``AS IS’’ AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Copyright © 1991, 1993 The Regents of the University of California. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement:

   `This product includes software developed by the University of California, Berkeley and its contributors.`
4. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS’’ AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT 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 if you modify this software you must label your software as modified software and not distribute it in such a fashion that it might be confused with the original M.I.T. software. M.I.T. makes no representations about the suitability of this software for any purpose. It is provided ″as is″ without express or implied warranty.

Copyright © 1998 by the FundsXpress, INC. All rights reserved.

Export of this software from the United States of America may require a specific license from the United States Government. It is the responsibility of any person or organization contemplating export to obtain such a license before exporting.

WITHIN THAT CONSTRAINT, permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of FundsXpress not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. FundsXpress makes no representations about the suitability of this software for any purpose. It is provided ″as is″ without express or implied warranty.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Copyright © 1999, 2000 Internet Software Consortium.

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED ″AS IS″ AND INTERNET SOFTWARE CONSORTIUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL INTERNET SOFTWARE CONSORTIUM BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Copyright © 1995-1998 Eric Young (eay@cryptsoft.com) All rights reserved.

This package is an SSL implementation written by Eric Young (eay@cryptsoft.com). The implementation was written so as to conform with Netscape's SSL.

This library is free for commercial and non-commercial use as long as the following conditions are adhered to. The following conditions apply to all code found in this distribution, be it the RC4, RSA, lhash, DES, etc., code; not just the SSL code. The SSL documentation included with this distribution is covered by the same copyright terms except that the holder is Tim Hudson (tjh@cryptsoft.com).

Copyright remains Eric Young's, and as such any Copyright notices in the code are not to be removed. If this package is used in a product, Eric Young should be given attribution as the author of the parts of the library used. This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement: ″This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)″. The word ’cryptographic’ can be left out if the routines from the library being used are not cryptographic related.
4. If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include acknowledgement:

   ″This product includes software written by Tim Hudson (tjh@cryptsoft.com)″

THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS’’ AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The license and distribution terms for any publicly available version or derivative of this code cannot be changed. i.e. this code cannot simply be copied and put under another distribution license [including the GNU Public License.]

This product includes cryptographic software written by Eric Young.

Copyright © 1999, 2000 Internet Software Consortium.

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED ″AS IS″ AND INTERNET SOFTWARE CONSORTIUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL INTERNET SOFTWARE CONSORTIUM BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

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 of unlicensed books and the z/OS Licensed Product Library (LK3T-4307), which contains BookManager and PDF formats of licensed books.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

Advanced Peer-to-Peer Networking
AFP
AD/Cycle
AIX
AIX/ESA
AnyNet
APL2
AS/400
AT
BookManager
BookMaster
C/370
CICS
CICS/ESA
C/MVS
Common User Access
C Set ++
CT
CUA
DB2
DFSMSdfp
DFSMShsm
DFSMS/MVS
DPI
Domino
DRDA
Enterprise Systems Architecture/370
ESCON
eServer
ES/3090
ES/9000
ES/9370
EtherStreamer
Extended Services
FFST
FFST/2
First Failure Support Technology
GDDM
IBM
IBMLink
IMS
IMS/ESA
Language Environment
LANStreamer
Library Reader
LPDA
Micro Channel
MVS
MVS/DFP
MVS/ESA
MVS/SP
MVS/XA
NetView
Network Station
Nways
Notes
OfficeVision/MVS
OfficeVision/VM
Open Class
OpenEdition
OS/2
OS/390
OS/400
Parallel Sysplex
PR/SM
PROFS
PS/2
RACF
Resource Link
RETAIN
RISC System/6000
RMF
RS/6000
S/370
S/390
SAA
SecureWay
SP
SP2
SQL/DS
System/360
System/370
System/390
SystemView
Tivoli
TURBOWAYS
UNIX System Services
VM/ESA
VSE/ESA
VTAM
WebSphere
XT
z/Architecture
z/OS
z/OS.e
zSeries
400
3090
3890

Lotus, Freelance, and Word Pro are trademarks of Lotus Development Corporation in the United States, or other countries, or both.

Tivoli and NetView are trademarks of Tivoli Systems Inc. in the United States, or other countries, or both.

DB2 and NetView are registered trademarks of International Business Machines Corporation or Tivoli Systems Inc. in the U.S., other countries, or both.

The following terms are trademarks of other companies:

ATM is a trademark of Adobe Systems, Incorporated.

BSC is a trademark of BusiSoft Corporation.

CSA is a trademark of Canadian Standards Association.

DCE is a trademark of The Open Software Foundation.

HYPERchannel is a trademark of Network Systems Corporation.

UNIX is a registered trademark in the United States, other countries, or both and is licensed exclusively through X/Open Company Limited.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

ActionMedia, LANDesk, MMX, Pentium, and ProShare are trademarks of Intel Corporation in the United States, other countries, or both. For a complete list of Intel trademarks, see http://www.intel.com/sites/corporate/tradmarx.htm .

Other company, product, and 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 xviii.

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”, on page 497 for a list of the documents and the info APARs associated with them.

- Info APARs for OS/390 documents are in the document called *OS/390 DOC APAR and PTF ++HOLD Documentation* which can be found at http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/ BOOKS/IDDOCMST/CCONTENTS.
- 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:80/cgi-bin/bookmgr_OS390/ BOOKS/ZIDOCMST/CCONTENTS.

Planning and migration

Title	Number	Description
z/OS Communications Server: SNA Migration and Exploitation	GC31-8774	This document is intended to help you plan for SNA, whether you are migrating from a previous version or installing SNA for the first time. This document also identifies the optional and required modifications needed to enable you to use the enhanced functions provided with SNA.
z/OS Communications Server: IP Migration and Exploitation	GC31-8773	This document is intended to help you plan for TCP/IP Services, whether you are migrating from a previous version or installing IP for the first time. This document also identifies the optional and required modifications needed to enable you to use the enhanced functions provided with TCP/IP Services.
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: AnyNet SNA over TCP/IP	SC31-8832	This guide provides information to help you install, configure, use, and diagnose SNA over TCP/IP.
z/OS Communications Server: AnyNet Sockets over SNA	SC31-8831	This guide provides information to help you install, configure, use, and diagnose sockets over SNA. It also provides information to help you prepare application programs to use sockets over SNA.
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	LY43-0092	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 Application Programming Interface Guide	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 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*	LY43-0088 LY43-0089	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*	LY43-0090 LY43-0091	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, ISU, 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)	SC31-8784	This volume contains TCP/IP messages beginning with EZB.
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.

APPC Application Suite

Title	Number	Description
z/OS Communications Server: APPC Application Suite User's Guide	SC31-8809	This documents the end-user interface (concepts, commands, and messages) for the AFTP, ANAME, and APING facilities of the APPC application suite. Although its primary audience is the end user, administrators and application programmers may also find it useful.

Title	Number	Description
z/OS Communications Server: APPC Application Suite Administration	SC31-8835	This document contains the information that administrators need to configure the APPC application suite and to manage the APING, ANAME, AFTP, and A3270 servers.
z/OS Communications Server: APPC Application Suite Programming	SC31-8834	This document provides the information application programmers need to add the functions of the AFTP and ANAME APIs to their application programs.

Index

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.

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this document.
- If you prefer to send comments by FAX, use this number: 1-800-254-0206
- If you prefer to send comments electronically, use this network ID: usib2hpd@vnet.ibm.com

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.

Readers' Comments — We'd Like to Hear from You

z/OS Communications Server
IP CICS Sockets Guide
Version 1 Release 5

Publication No. SC31-8807-02

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	□	□	□	□	□

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	□	□	□	□	□
Complete	□	□	□	□	□
Easy to find	□	□	□	□	□
Easy to understand	□	□	□	□	□
Well organized	□	□	□	□	□
Applicable to your tasks	□	□	□	□	□

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? □ Yes □ No

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.

Name

Company or Organization

Phone No.

Address

Readers' Comments — We'd Like to Hear from You
SC31-8807-02

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
G7IA 7G Globalization Services
Department QSJA/500/2
P.O. Box 12195, 3039 Cornwallis Road
Research Triangle Park, NC 27709-2195

Fold and Tape **Please do not staple** Fold and Tape

Cut or Fold
Along Line

Program Number: 5694–A01 and 5655–G52

Printed in U.S.A.

SC31-8807-02

Spine information:

z/OS Communications Server

z/OS V1R5.0 CS: IP CICS Sockets Guide

Version 1
Release 5