

66776677

IBM TCP/IP for MVS

CICS TCP/IP Socket Interface Guide and Reference



IBM TCP/IP for MVS



CICS TCP/IP Socket Interface Guide and Reference

Note: Before using this information and the product it supports, be sure to read the general information under "Appendix F. Notices" on page 319.

This book is also available in softcopy and can be viewed with the IBM BookManager READ program.

Fourth Edition (June 1997)

This edition applies to OS/390 (5645-001) and the Version 3 Release 2 of the IBM TCP/IP for MVS licensed program (5655-HAL). See the "Summary of Changes" for a description of the changes made in this edition. Technical changes or additions to text and illustrations are indicated by a vertical line to the left of the change. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

IBM welcomes your comments. A form for readers' comments may be at the back of this publication. If the form has been removed, you may send your comments to the following address:

International Business Machines Corporation
Department CGMD
P.O. Box 12195
Research Triangle Park, North Carolina 27709
USA

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

Fax (USA and Canada):

1-800-227-5088

Internet e-mail:

usib2hpd@vnet.ibm.com

World Wide Web:

<http://www.s390.ibm.com/os390>

IBMLink:

CIBMORCF at RALVM13

IBM Mail Exchange:

USIB2HPD at IBMMAIL

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.

© Copyright International Business Machines Corporation 1994, 1996, 1997. All rights reserved.

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Figures	ix
Tables	xi
About This Book	xiii
Who Should Use This Book	xiii
Where to Find Related Information on the Internet	xiii
How to Contact IBM Service	xiii
Summary of Changes	xv
SC31-7131-03: TCP/IP Version 3 Release 2 for MVS	xv
New Information	xv
Changed Information	xv
Deleted Information	xvi
SC31-7131-02: TCP/IP Version 3 Release 2 for MVS	xvi
New Information	xvi
Changed Information	xvi
Deleted Information	xvii
SC31-7131-01: TCP/IP Version 3 Release 1 for MVS	xvii
SC31-7131-00: TCP/IP Version 3 Release 1 for MVS	xvii
New Information	xvii
Moved Information	xvii
Deleted Information	xvii
Chapter 1. Introduction to CICS TCP/IP.	1
Overview.	1
TCP/IP Internets	2
Telnet	3
Client/Server Processing	3
TCP, UDP, and IP	3
The Socket API	4
Programming with Sockets	5
A Typical Client Server Program Flow Chart	7
The Basic Socket Calls	9
Server TCP/IP calls	10
Client TCP/IP Calls	12
Other Socket Calls	14
What You Must Have to Run CICS TCP/IP	18
CICS TCP/IP Components	18
A Summary of What CICS TCP/IP Provides	19
The Socket Calls.	19
The Listener (3)	20
Conversion routines (4)	20
Chapter 2. Setting Up and Configuring CICS TCP/IP	21
Overview.	21
MVS JCL — Modifying CICS Startup	22
CICS — Defining CICS TCP/IP Resources	23
Transaction Definitions.	23
Program Definitions.	26
File Definitions.	33
Transient Data Definition	37
CICS Monitoring	37

CICS Program List Table (PLT)	42
TCP/IP for MVS — Modifying Data Sets	42
The hlq.PROFILE.TCPIP Data Set	42
The hlq.TCPIP.DATA Data Set	43
Configuring the CICS TCP/IP Environment	44
Building the Configuration Dataset with EZACICD.	44
Customizing the Configuration Dataset.	50
Chapter 3. Configuring the CICS Domain Name Server Cache	69
Overview of the Domain Name Server Cache	69
Function Components	69
How the DNS Handles Requests	70
Using the DNS Cache	71
Step 1: Create the Initialization Module	71
Step 2: Define the Cache File to CICS	74
Step 3: Execute EZACIC25	75
Chapter 4. Starting and Stopping CICS Sockets	79
Overview.	79
Starting/Stopping CICS TCP/IP Automatically	79
Starting/Stopping CICS TCP/IP Manually	79
START Function	80
STOP Function	82
Starting/Stopping CICS TCP/IP with Program Link	85
Chapter 5. Writing Your Own Listener	87
Basic Requirements.	87
Pre-Requisites.	87
Using IBM's Environmental Support	87
Chapter 6. Application Programming Guide	91
Overview.	91
Writing CICS TCP/IP Applications.	92
1. The Client-Listener-Child-Server Application Set	93
2. Writing Your Own Concurrent Server	96
3. The Iterative Server CICS TCP/IP Application	97
4. The Client CICS TCP/IP Application	98
Socket Addresses	99
Address Family (Domain).	99
IP Addresses	99
Ports	99
Address Structures	99
MVS Address Spaces	100
Network Byte Order.	101
GETCLIENTID, GIVESOCKET, and TAKESOCKET	101
The Listener	103
Listener Input Format	103
Listener Output Format	104
Writing Your Own Security Link Module for the Listener	105
Data Conversion Routines	107
Chapter 7. C Language Application Programming	109
Overview.	111
C Socket Library	111
C/370 Compilation	111
Structures Used in Socket Calls	114

The errno Variable	115
C Socket Calls	115
accept()	115
bind()	117
close().	118
connect().	119
fcntl()	120
getclientid()	121
gethostid()	122
gethostname().	122
getpeername().	123
getsockname().	124
getsockopt(), setsockopt()	124
givesocket().	127
initapi()	128
ioctl()	129
listen().	131
read()	132
recv()	133
recvfrom()	134
select()	135
send()	137
sendto()	138
setsockopt()	139
shutdown()	140
socket()	140
takesocket()	141
write()	142
Chapter 8. Sockets Extended Application Programming Interface (API)	145
CALL Instruction Application Programming Interface (API).	146
Call Formats	146
Programming Language Conversions	147
Error Messages and Return Codes	147
CALL Instructions for Assembler, PL/I, and COBOL Programs	148
Data Translation Programs for the Socket Call Interface	203
Appendix A. External Data Structures	211
External Data Structures	211
Configuration Data Set Record Formats	211
Global Work Area	212
Task Interface Element (TIE)	215
Parameter List (COMMAREA) for EZACIC20	220
Listener Control Area (LCA).	221
Appendix B. Original COBOL Application Programming Interface (EZACICAL)	223
Using the EZACICAL or Sockets Extended API.	223
COBOL Compilation	223
The EZACICAL API.	225
COBOL	225
PL/I.	225
Assembler Language	226
COBOL and Assembler Language Socket Calls	226
ACCEPT.	226
BIND	227

CLOSE	228
CONNECT	229
FCNTL	230
GETCLIENTID.	231
GETHOSTID	232
GETHOSTNAME.	232
GETPEERNAME.	233
GETSOCKNAME.	234
GETSOCKOPT	235
GIVESOCKET.	236
INITAPI	237
IOCTL.	238
LISTEN	239
READ	240
RECVFROM	241
SELECT	242
SEND	244
SENDTO.	245
SETSOCKOPT	246
SHUTDOWN	247
SOCKET.	248
TAKESOCKET	249
WRITE	250
Appendix C. Return Codes	253
Error Number from MVS TCP/IP	253
Sockets Extended Return Codes	263
Appendix D. CICS Messages	277
Messages originating from EZACIC01	277
Messages originating from EZACIC02	280
Messages originating from EZACIC03	289
Messages Originating from EZACIC21	289
Messages Originating from EZACIC22	292
Messages Originating from EZACIC25	293
Appendix E. Sample Programs.	295
EZACICSC	295
EZACICSS	302
Appendix F. Notices	319
Trademarks.	320
Bibliography	323
IBM TCP/IP Publications	323
OS/390 TCP/IP OpenEdition Publications.	323
TCP/IP for MVS Publications	323
TCP/IP for VM Publications	324
TCP/IP for OS/2 Publication.	325
TCP/IP for DOS Publications	325
TCP/IP for AIX (RS/6001, PS/2, RT, 370) Publications	325
TCP/IP for AS/400 Publications	325
Other IBM TCP/IP Publications	325
IBM Operating System Publications	325
AIX Publications	325
AS/400 Publications.	326

DOS Publications	326
MVS Publications	326
OS/2 Publications	326
OS/390 Publications	326
VM Publications	326
IBM Software Publications	327
ACF/VTAM Publications	327
DATABASE 2 Publications	327
ISPF Publication	328
JES Publications	328
MVS/DFP Publications.	328
Network Control Program (NCP) Publications	328
TME 10 NetView for OS/390 Publications.	328
OpenEdition MVS Publications.	328
Programming Publications	328
RACF Publications	328
SMP/E Publications.	329
VSAM Publication	329
X.25 NPSI Publications	329
IBM Hardware Publications	329
System/370 and System/390 Publications.	329
3172 Interconnect Controller Publications.	329
3270 Information Display System Publication	329
8232 LAN Channel Station Publications	329
9370 Publications	330
Other TCP/IP-Related Publications	330
OSF/Motif Publications.	330
Sun (RPC) Publications	330
X Window System Publications	330
Network Architecture Publications.	331
Open Systems Interconnection (OSI) Publication	331
Systems Network Architecture (SNA) Publications.	331
Index	333
Readers' Comments — We'd Like to Hear from You.	341

Figures

1.	The Use of CICS Sockets.	2
2.	TCP/IP Protocols Compared to the OSI Model and SNA	3
3.	A Typical Client Server Session	8
4.	An Iterative Server	9
5.	A Concurrent Server.	9
6.	The SELECT Call.	14
7.	How User Applications Access TCP/IP Networks with CICS TCP/IP	19
8.	JCL for CICS Startup with the TCP/IP Socket Interface	22
9.	EZAC, Transaction to Configure the Socket Interface, Definition in RDO	24
10.	EZAO, Transaction to Enable the Socket Interface, Definition in RDO	24
11.	EZAP, Transaction to Disable the Socket Interface	25
12.	CSKL, Listener Task Transaction, Definition in RDO	25
13.	EZACIC00, Connection Manager Program, Definition in RDO	27
14.	EZACIC01, Task Related User Exit Program, Definition in RDO	27
15.	EZACIC02, Listener Program, Definition in RDO	28
16.	EZACIC20, Front-End Module for CICS Sockets, Definition in RDO	28
17.	EZACIC21, Initialization Module for CICS Sockets, Definition in RDO.	29
18.	EZACIC22, Termination Module for CICS Sockets, Definition in RDO.	29
19.	EZACIC23, Primary Module for Transaction EZAC, Definition in RDO	30
20.	EZACIC24, Message Delivery Module for CICS Sockets, Definition in RDO	30
21.	EZACIC25, Domain Name Server Cache Module, Definition in RDO	31
22.	EZACICME, US English Text Delivery Module, Definition in RDO	31
23.	EZACICM, Maps Used by EZAO and EZAP Transactions, Definition in RDO	31
24.	EZACICSS, Sample Iterative Server Program, Definition in RDO	32
25.	EZACICSC, Sample Child Server Program, Definition in RDO	33
26.	EZACONFG, defining to RDO	34
27.	EZACACHE, defining to RDO	36
28.	Addition to the DCT Required by CICS TCP/IP	37
29.	The Monitor Control Table (MCT) for TRUE	39
30.	The Monitor Control Table (MCT) for Listener	41
31.	Definition of the hlq.TCP/IP Profile	43
32.	The TCPIPJOBNAME Parameter in the hlq.TCPIP.DATA Data Set	43
33.	Example of JCL to Define a Configuration File	48
34.	EZAC Initial Screen	51
35.	EZAC ALTER Screen	51
36.	EZAC ALTER CICS screen	52
37.	EZAC ALTER CICS Detail Screen.	52
38.	ALTER LISTENER screen	53
39.	EZAC ALTER LISTENER Detail Screen	53
40.	EZAC COPY Screen	54
41.	EZAC COPY Screen	55
42.	EZAC COPY Screen	56
43.	EZAC DEFINE Screen	57
44.	EZAC DEFINE CICS screen.	57
45.	EZAC DEFINE CICS Detail Screen	58
46.	EZAC DEFINE LISTENER screen.	58
47.	EZAC DEFINE LISTENER Detail Screen	59
48.	EZAC DELETE Screen.	59
49.	EZAC DELETE CICS screen	60
50.	EZAC DELETE LISTENER screen	61
51.	EZAC DISPLAY Screen	62

52.	EZAC DISPLAY CICS screen	62
53.	EZAC DISPLAY CICS Detail Screen	63
54.	EZAC DISPLAY LISTENER screen	63
55.	EZAC DISPLAY LISTENER Detail Screen.	64
56.	EZAC RENAME Screen	65
57.	EZAC RENAME CICS Screen	66
58.	EZAC RENAME LISTENER Screen	67
59.	Example of Defining and Initializing a DNS Cache File	73
60.	The DNS Hostent.	77
61.	EZAO Initial Screen	80
62.	EZAC START Screen	81
63.	EZAC START CICS Response Screen	81
64.	EZAC START LISTENER Screen	82
65.	EZAC START LISTENER Result Screen	82
66.	EZAC STOP Screen.	83
67.	EZAC STOP Screen.	83
68.	EZAO STOP LISTENER Screen	85
69.	The Sequence of Sockets Calls	94
70.	Sequence of Socket Calls with an Iterative Server.	97
71.	Sequence of Socket Calls between a CICS Client and a Remote Iterative Server	98
72.	MVS Address Spaces	100
73.	Transfer of CLIENTID Information.	102
74.	Modified JCL for C/370 Compilation	113
75.	HOSTENT Structure Returned by the GETHOSTBYADDR Call	157
76.	HOSTENT Structure Returned by the GETHOSTBYNAME Call	159
77.	Interface Request Structure (IFREQ) for the IOCTL Call	171
78.	COBOL II Example for SIOCGIFCONF	173
79.	Modified JCL for COBOL Compilation	224

Tables

1.	First Fullword Passed in a Bit String in Select	15
2.	Second Fullword Passed in a Bit String in Select	16
3.	Calls for the Client Application	94
4.	Calls for the Server Application	95
5.	Calls for the Concurrent Server Application	96
6.	CLIENTID Structures	101
7.	Listener Output Format.	105
8.	Security Exit Data	106
9.	C Structures.	114
10.	IOCTL call arguments	171
11.	Configuration File Format	211
12.	Global Work Area Format	212
13.	Task Interface Element Format	215
14.	COMMAREA Format for EZACIC20	220
15.	Listener Control Area (LCA)	221
16.	System Error Return Codes	253
17.	Sockets Extended Return Codes	263

About This Book

This book contains a description of the TCP/IP Version 3 Release 2 for MVS: Socket Interface for CICS (referred to as CICS TCP/IP for short). It comprises an introduction, a guide to initialization, and a guide and reference to writing application programs. Resource Definition On-Line definitions and JCL examples in this book should be used with CICS/ESA Version 3.3 or later.

TCP/IP Version 3 Release 2 for MVS is an integral part of the OS/390 family of products. For an overview and mapping of the documentation available for OS/390, see the *OS/390 Information Roadmap* .

Who Should Use This Book

This book is aimed at 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 book 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 book assumes the reader is also familiar with CICS.

Where to Find Related Information on the Internet

You may find the following information helpful.

For current updates to the TCP/IP Version 3 Release 2 for MVS documentation described in “Bibliography” on page 323, check out the TCP/IP for MVS home page:
<http://www.networking.ibm.com/tcm/tcmprod.html>

To keep in close touch with OS/390, we suggest you look at the OS/390 home page:
<http://www.s390.ibm.com/os390>

To keep abreast of new products and technologies from IBM Networking, take a look at the IBM Networking home page:
<http://www.networking.ibm.com/>

The IBM Networking Software Glossary is now available in HTML format as well as PDF. You can access it directly at the following URL:
<http://www.networking.ibm.com/nsg/nsgmain.htm>

How to Contact IBM Service

For telephone assistance in problem diagnosis and resolution (in the United States or Puerto Rico), call the IBM Software Support Center anytime (1-800-237-5511). 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.

Summary of Changes

SC31-7131-03: TCP/IP Version 3 Release 2 for MVS

The updates contained in this edition are effective only if the latest level of maintenance has been applied. New and changed information is indicated by a revision bar (|).

New Information

This is the fourth edition of this book. This book supports TCP/IP Version 3 Release 2 for MVS and the OS/390 family of products.

The following enhancements are new for this revision:

- Addition of a configuration dataset and macro. This dataset is used to store configuration information for the interface and the listeners it supports. The macro lets users customize the dataset. See “Chapter 2. Setting Up and Configuring CICS TCP/IP” on page 21.
- Creation of startup and shutdown modules. The startup module reads the configuration dataset and automatically starts the CICS/sockets interface and listeners. The shutdown module ends CICS/sockets interface processing. See “Chapter 4. Starting and Stopping CICS Sockets” on page 79.
- Users can now write their own listeners. See “Chapter 5. Writing Your Own Listener” on page 87.
- All messages are developed by a new module, EZACIMx, which translates messages to specific languages. This release provides module EZACICME, which translates messages into U.S. English and accepts message requests from other modules of the interface and creates messages. Other language models will appear in subsequent releases. See Figure 22 on page 31.
- Copybooks used by the interface will be generated by the EZACICA macro.
- A domain name server cache has been added. See “Chapter 3. Configuring the CICS Domain Name Server Cache” on page 69.
- “Appendix A. External Data Structures” on page 211 describes data structures that are available for customer use.
- “Appendix D. CICS Messages” on page 277 includes all the CICS/TCPIP messages.

Changed Information

- An additional 8-byte character field was added to the end of the Security Exit data structure. This field permits a CICS V4+ user to return a USERID valued which is then used in starting the server transaction. This makes CICS V4+ security features available. See Table 8 on page 106.
- CSKE has been replaced by EZAO, and CSKD has been replaced by EZAP. See “Chapter 2. Setting Up and Configuring CICS TCP/IP” on page 21.
- Changes to the operation of the subtask. The subtask will operate in one of two modes:
 - Reusable. A pool of MVS tasks is created at initialization with the size of the pool determined by a value in the Configuration Dataset. One of these

subtasks is associated with a CICS task when the CICS task issues its first socket call and that association will remain in effect until the CICS task terminates.

- Attached. These subtasks function like the subtasks in the current implementation. They are attached when the CICS task issues its first socket call and detached when the CICS task terminates. Attached tasks are used for listeners and when the pool of reusable tasks is exhausted.
- The IBM listener has been rewritten. For each listener instance specified in the Configuration Dataset, there is an instance of the listener started by the Initialization Module. The operating characteristics of each listener instance are controlled by the parameters specified in the configuration file.

Deleted Information

- The section on Terminal Control Table (TCT) entries has been deleted.

SC31-7131-02: TCP/IP Version 3 Release 2 for MVS

This is the third edition of this book. This book supports TCP/IP Version 3 Release 2 for MVS and the OS/390 family of products.

New Information

The following enhancements are new for this revision:

- Improved C Sockets and Sockets Extended application program interfaces (APIs) (also referred to as High-Performance Native Sockets, or HPNS): The Sockets Extended APIs, which are the Macro and Call Instruction interfaces, no longer use the inter-user communication vehicle (IUCV) address space. This change improves performance and reduces CPU cycles in an MVS host transporting data with TCP/IP. Applications written on previous versions of TCP/IP for MVS can run unchanged.
- New calls for the Call Instruction interface:
 - The *READV* call reads data on a socket and stores it in a set of buffers.
 - The *RECVMSG* call receives messages on a socket and stores them in an array of message headers.
 - The *SELECTEX* call monitors a set of sockets, a time value, and an ECB or ECB list. It completes when either one of the sockets has activity, the time value expires, or an ECB is posted.
 - The *SENDMSG* call sends messages on a socket passed from an array of messages.
 - The *WRITEV* call writes data on a socket into a set of buffers.

Changed Information

- Chapter 1. Introduction to CICS TCP/IP has been expanded to include a more complete overview of CICS TCP/IP.
- Minor editorial and technical updates have been applied to this edition.

Deleted Information

“CICS Feature Error Messages” have been removed from the appendix. You can find CICS error messages in *TCP/IP for MVS: Messages and Codes* (SC31-7132-03).

SC31-7131-01: TCP/IP Version 3 Release 1 for MVS

This is the second edition of this book. This book supports TCP/IP V3R1 for MVS and the OS/390 family of products.

The updates contained in this edition are effective only if the latest level of maintenance has been applied. New and changed information is indicated by a revision bar (|).

Various minor editorial and technical updates have been applied to this edition.

SC31-7131-00: TCP/IP Version 3 Release 1 for MVS

This was a new book for IBM TCP/IP Version 3 Release 1 for MVS. This book describes how to install and use TCP/IP for MVS Sockets for CICS/ESA* Version 3.

The first edition incorporated *Sockets Interface for CICS—Using TCP/IP Version 2.2 for MVS: User's Guide*, GC31-7015-00. The following paragraphs describe the changes to the Version 2.2 information.

New Information

The principle added information was the description of the Sockets Extended application programming interface (API), a new simplified interface for COBOL, PL/I, and Assembler. Also added was expanded guidance on configuration and programming.

Moved Information

Version 2.2 book	Version 3.1 for MVS book
Chapter 2, guidance	Chapter 4
Chapter 2, C function calls	Chapter 5
Chapter 2, EZACICAL - original COBOL API	Appendix A
Chapter 2, EZACIC04,-5,-6	Chapter 6
Appendix A	Appendix B
Appendix B	Chapter 2

Deleted Information

“Appendix C. Related Protocol Specifications” has been deleted. This information is not directly relevant to readers of this book.

Chapter 1. Introduction to CICS TCP/IP

Overview	1
TCP/IP Internets	2
Telnet	3
Client/Server Processing	3
TCP, UDP, and IP	3
The Socket API	4
Programming with Sockets	5
Socket types	5
Addressing TCP/IP hosts	5
A Typical Client Server Program Flow Chart	7
Concurrent and Iterative Servers	8
The Basic Socket Calls	9
Server TCP/IP calls	10
Socket	10
Bind	11
Listen	11
Accept	11
GIVESOCKET and TAKESOCKET	12
Read and Write	12
Client TCP/IP Calls	12
The Socket Call	12
The Connect Call	13
Read/Write Calls — the Conversation	13
The Close Call	13
Other Socket Calls	14
The SELECT Call	14
IOCTL and FCNTL Calls	16
GIVESOCKET and TAKESOCKET Calls	16
What You Must Have to Run CICS TCP/IP	18
CICS TCP/IP Components	18
A Summary of What CICS TCP/IP Provides	19
The Socket Calls	19
The Listener (3)	20
Conversion routines (4)	20

Overview

CICS is an on-line 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 is a feature of TCP/IP for MVS that allows remote users to access CICS client/server applications over TCP/IP internets. Figure 1 on page 2 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.

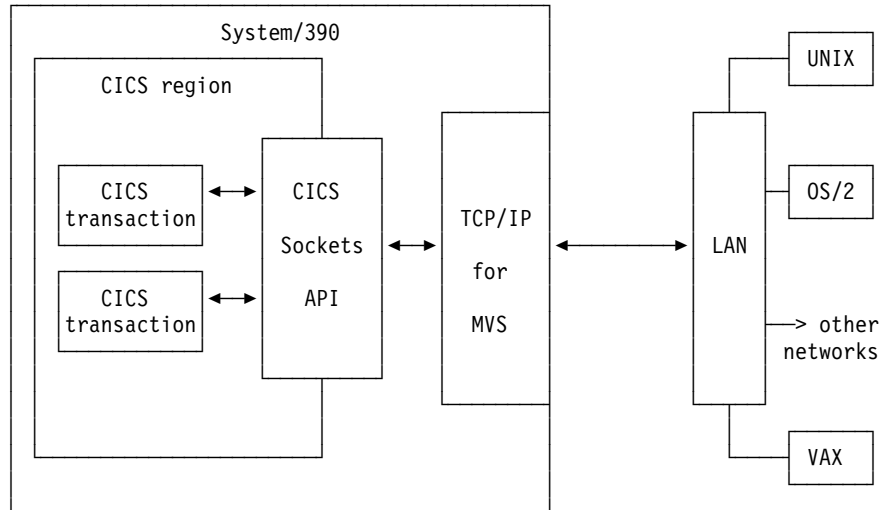


Figure 1. The Use of CICS Sockets

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 books on TCP/IP listed in the Bibliography.

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 for MVS supports traditional 3270 mainframe interactive (MFI) applications with an emulator function called Telnet (TN3270). For these applications, all program logic is housed in the main frame, 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 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.

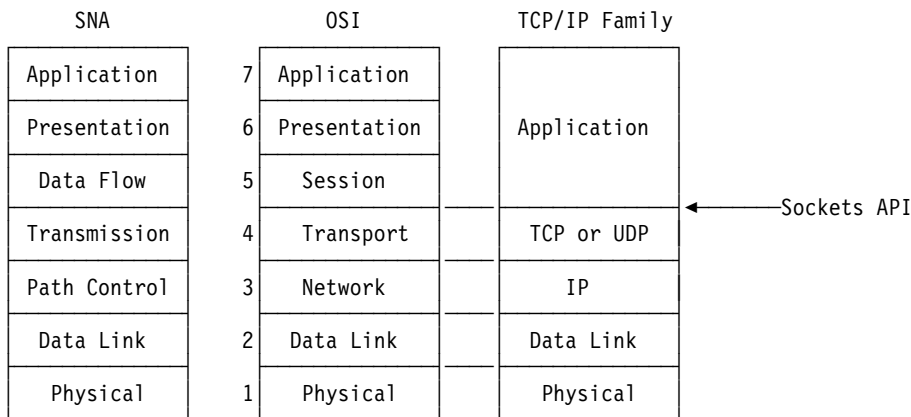


Figure 2. TCP/IP Protocols Compared to the OSI Model and SNA

The protocols implemented by TCP/IP for MVS 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 (that is, data is transmitted link by link; there is no end-to-end connection). The service provides no guarantees: data can be lost or duplicated, and datagrams can arrive out of order.

Internet Protocol (IP)

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

The Socket API

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

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

In addition to these basic functions, the 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 System/370* 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 for MVS 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 109 for the sockets calls provided by TCP/IP for MVS.
- **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 145 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, documented in *IBM Sockets Interface for CICS - Using TCP/IP V2R2 for MVS*. It is similar in use to the Sockets Extended API. The Version 2.2.1 API is available in Version 3.1 and Version 3.2 for those who want to maintain Version 2.2.1 programs. This interface is described in “Appendix B. Original COBOL Application Programming Interface (EZACICAL)” on page 223.

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 newline character boundaries between data. Communicating processes ¹ must agree on a scheme to ensure that both client and server have received all data. One way of doing this is for the sending process to send the *length* of the data, followed by the data itself. The receiving process reads the length and then loops, accepting data until all of it has been transferred.

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

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

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

Addressing TCP/IP hosts

The following section describes how one TCP/IP host addresses another TCP/IP host. ²

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

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

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

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

```
01 NAME
   03 FAMILY      PIC 9(4) BINARY.
   03 PORT        PIC 9(4) BINARY.
   03 IP_ADDRESS  PIC 9(8) BINARY.
   03 RESERVED   PIC X(8).
```

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

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

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

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

As an example, an MVS system with an IP address of 129.9.12.7 might have 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.

Sockets and Ports:

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

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

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 which is different than the listener/server port or any other client ports. Normally, client applications do not share ports, but it can be done using the SO_REUSEADDR option.

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

Network Byte Order: In the open environment of TCP/IP, internet addresses must be defined in terms of the architecture of the machines. Most machine architectures (like IBM PC's and mainframes) define the lowest memory address to be the high-order bit, which is called *big endian*. However, some architectures 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. Since IBM systems are big-endian, the only time you need to be concerned about Network Byte Order is when a little-endian system attempts to make contact with a CICS, system. In such a case, the burden for conversion to Network Byte Order is usually on the little-endian program.

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

A Typical Client Server Program Flow Chart

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



Figure 3. A Typical Client Server Session

Concurrent and Iterative Servers

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

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

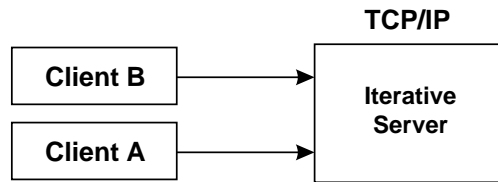


Figure 4. An Iterative Server

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 103.

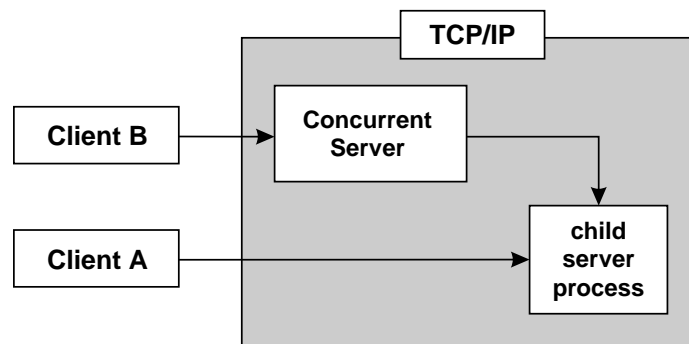


Figure 5. A Concurrent Server

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 145.

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 TCPIP address space, so TCP/IP usually assigns socket numbers in ascending order. In COBOL, the programmer uses the SOCKET call to obtain a new socket.

The socket function specifies the address family (AF_INET), the type of socket (STREAM), and the particular networking protocol (PROTO) to use. (When PROTO is set to 0, the TCPIP 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 198.

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 3 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.

For an example of the BIND call, see “BIND” on page 149.

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 LISTEN³ 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 173.

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

When the ACCEPT call is issued, the server passes its socket descriptor, S, to TCP/IP. When the connection is established, the ACCEPT call returns a new socket descriptor (in RETCODE) that represents the connection with the client. **This is the**

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.

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 148.

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, MVS 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 198 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 [41](#).

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

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

See “CONNECT” on page 152 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 SEND₀ and RECV₀ calls can be used only on sockets that are in the connected state. The SENDTO and RECVFROM, and SENDMSG and RECVMSG calls can be used regardless of whether a connection exists.

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

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

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

- For an example of the READ call, see “READ” on page 174.
- For an example of the WRITE call, see “WRITE” on page 200.

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 151

Other Socket Calls

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

The SELECT Call

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

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

With the Socket SELECT call, you can define which sockets you want to test (the 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 number three, you have to pass a 4-byte bit string, because this is the minimum length. If the highest number is 32, you must pass 8 bytes (2 fullwords).

The number of fullwords in each select mask can be calculated as $\text{INT}(\text{highest socket descriptor} / 32) + 1$

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 left-most bit or byte is relative zero.

If you want to test socket descriptor number 5 for pending read activity, you raise bit 2 in byte 3 of the first fullword (X'00000020'). If you want to test both socket descriptor 4 and 5, you raise both bit 2 and bit 3 in byte 3 of the first fullword (X'00000030').

If you want to test socket descriptor number 32, you must pass two fullwords, where the numbering scheme for the second fullword resembles that of the first. Socket descriptor number 32 is bit 7 in byte 3 of the second fullword. If you want to test socket descriptors 5 and 32, you pass two fullwords with the following content: X'0000002000000001'.

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

Table 2. Second Fullword Passed in a Bit String in Select

Socket Descriptor Numbers Represented by Byte	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
Byte 4	63	62	61	60	59	58	57	56
Byte 5	55	54	53	52	51	50	49	48
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 for MVS, will make it easier for you to build and test these bit strings. This routine translates between a character string mask (one byte per socket) and a bit string mask (one bit per socket).

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

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

IOCTL and FCNTL Calls

In addition to SELECT, applications can use the IOCTL or FCNTL calls to help perform asynchronous (nonblocking) socket operations. An example of the use of the IOCTL call is shown in "IOCTL" on page 169.

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

GIVESOCKET and TAKESOCKET Calls

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

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

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

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

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

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

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

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

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

Summary: 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, (as with IMS) any MVS address space requesting a socket can take this socket.
- The server main task then passes the socket descriptor and concurrent server's ID to the subtask using some form of commonly addressable technique such as the CICS START/RETRIEVE commands.
- The concurrent server issues the SELECT call to determine when the GIVESOCKET has successfully completed.

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 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 103. 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 book, you must have the following:

- TCP/IP Version 3 Release 2 requires CICS/ESA Version 3.3 or later.

Note: Use of some security features would require CICS/ESA Version 4.1 or later. Refer to Table 8 on page 106.

- TCP/IP for MVS Version 3 Release 2 or later.
- MVS/ESA Version 4.3 or higher.

TCP/IP for MVS is not described in this book since it is a prerequisite for CICS TCP/IP. However, much material from the TCP/IP library has been repeated in this book in an attempt to make it independent of that library. For more information about TCP/IP for MVS, see the books listed in "TCP/IP for MVS Publications" on page 323 .

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 four main components:

1. The **stub program**. This 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.
2. The **TRUE**. This enables programs to pass calls to the subtask and so to the TCP/IP address space.
3. The **MVS subtask**. It translates commands for accessing TCP/IP into a form acceptable to the TCP/IP resource manager and then pass control to the resource manager. It also handles the MVS waits incurred during socket calls.
4. The **Administration Routine**. It contains the EXEC CICS ENABLE and DISABLE commands that are used to install and withdraw the TRUE program.

A Summary of What CICS TCP/IP Provides

Figure 7 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.

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

Basic calls:	<i>SOCKET, BIND, CONNECT, LISTEN, ACCEPT, SHUTDOWN, CLOSE</i>
Read/write calls:	<i>SEND, SENDTO, RECVFROM, READ, WRITE</i>
Advanced calls:	<i>GETHOSTNAME, GETPEERNAME, GETSOCKNAME, GETSOCKOPT, SETSOCKOPT, SFCNTL, IOCTL, SELECT, GETHOSTBYNAME (Call interface only), GETHOSTBYADDRESS (Call interface only)</i>
IBM-specific calls:	<i>INITAPI, GETCLIENTID, GIVESOCKET, TAKESOCKET</i>

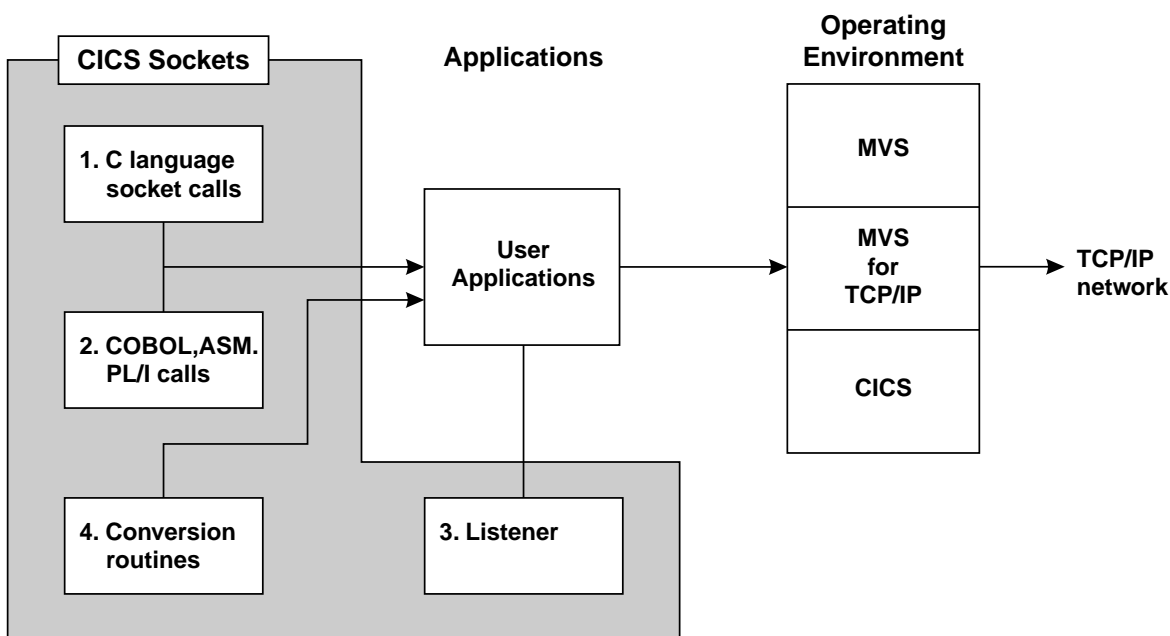


Figure 7. How User Applications Access TCP/IP Networks with CICS TCP/IP

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

The Listener (3)

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

Conversion routines (4)

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

- An EBCDIC-to-ASCII conversion routine, used to convert EBCDIC data within CICS to the ASCII format used in TCP/IP networks and workstations. It is run by calling module EZACIC04.
- A corresponding ASCII-to-EBCDIC conversion routine (EZACIC05).
- 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.

Chapter 2. Setting Up and Configuring CICS TCP/IP

Overview	21
MVS JCL — Modifying CICS Startup	22
CICS — Defining CICS TCP/IP Resources	23
Transaction Definitions.	23
Using Storage Protection	23
Program Definitions	26
Required Programs, CICS Definition Needed	26
Optional Programs, CICS Definition Needed	32
Required Programs, CICS Definition Not Needed	33
File Definitions.	33
EZACONFG	33
EZACACHE	35
Transient Data Definition	37
CICS Monitoring	37
Event Monitoring Points for the TRUE	38
Event Monitoring Points for the Listener	40
CICS Program List Table (PLT)	42
TCP/IP for MVS — Modifying Data Sets	42
The hlq.PROFILE.TCPIP Data Set	42
The hlq.TCPIP.DATA Data Set	43
Configuring the CICS TCP/IP Environment	44
Building the Configuration Dataset with EZACICD.	44
Configuration Macro	44
Customizing the Configuration Dataset.	50
Configuration Transaction (EZAC)	50

Overview

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

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

CICS TCP/IP modules are shipped as a feature of the TCP/IP for MVS install tapes. Before you can start CICS TCP/IP, you need to do the following:

Task	Refer to
Modify the CICS/ESA job stream to enable CICS TCP/IP startup.	“MVS JCL — Modifying CICS Startup” on page 22
Define additional files, programs, maps, and transient data to CICS using RDO.	“CICS — Defining CICS TCP/IP Resources” on page 23
Modify TCP/IP for MVS datasets	“TCP/IP for MVS — Modifying Data Sets” on page 42
Use the configuration macro (EZACICD), to build the TCP Configuration dataset	“Building the Configuration Dataset with EZACICD” on page 44
Use the configuration transaction to customize the Configuration dataset	“Customizing the Configuration Dataset” on page 50

Note: You can modify the dataset while CICS is running by using EZAC. See “Configuration Transaction (EZAC)” on page 50.

MVS JCL — Modifying CICS Startup

Figure 8 illustrates the modifications required to the CICS/ESA Version 3 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=DFHSSIP, REGION=32M, TIME=1440,
//      PARM=SYSIN
//SYSIN  DD *
SIT=6$,
START=AUTO,
DCT=IP,
GRPLIST=TCPLIST,
GMTEXT=' WELCOME TO CICS/ESA V3.3.0 WITH TCP/IP SOCKETS INTERFACE',
APPLID=SCMCICSA
.END
//DFHXRTL DD DISP=SHR, DSN=CICS330.CNTL.CICS.DFHXRCTL
//STEPLIB DD DISP=SHR, DSN=CICS330.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=TCP/IP.SEZALINK
//DFHRPL DD DISP=SHR, DSN=CICS330.SDFHLOAD
//      DD DISP=SHR, DSN=TCP/IP.SEZATCP
//      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=CICS330.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)
. . . .
. . . .
```

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*.SEZALINK to STEPLIB. This data set contains CICS TCP/IP modules EZACIC03 and EZACICMT.

Note: TCP/IP Version 3 Release 2 for MVS data set prefix names might have been modified during installation. When you see the prefix *hlq* in this book, substitute the prefix used in your installation.

2. You must concatenate the data set *hlq*.SEZATCP to DFHRPL. This data set contains all the other CICS TCP/IP modules.

Note: TCP/IP for MVS data set prefix names might have been modified during TCP/IP for MVS installation. When you see the prefix *hlq* in this book, you should substitute the prefix used in your installation.

3. You can add a TCPICCS entry for the output messages from CICS TCP/IP (see “Transient Data Definition” on page 37).

CICS — Defining CICS TCP/IP Resources

The following definitions must be made in CICS:

- Transactions
- Programs (see “Program Definitions” on page 26)
- BMS mapset (EZACICM, shown in Figure 23 on page 31)
- Files (see “File Definitions” on page 33)
- Transient data queues (see “Transient Data Definition” on page 37)

For information on defining transactions, programs, and files to the CICS Resource Definition Online (RDO) facility, refer to *CICS/ESA Resource Definition (Online)* (SC33–0666).

Transaction Definitions

Figures 10, 11, and 12 show the CICS resource definition online (RDO) entries to define the four transactions required to support CICS TCP/IP:

EZAC Configure the socket interface

EZAO Enable the socket interface (replaces CSKE)

EZAP Terminate the socket interface (replaces CSKD)

CSKL Listener task

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

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

Using Storage Protection

When running with CICS 3.3.0 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/ESA 3.3 Release Guide* contains more information on storage protection with task-related user exits (TRUEs).

In Figure 10 on page 24, Figure 11 on page 25, and Figure 12 on page 25 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.

```

CEDA DEFine
  TRansaction   : EZAC
  Group         : TCPIPI
  Description   ==> Configure Sockets Interface
  PROGram      ==> EZACIC23
  TWasize      ==> 00000
  PROFile      ==> DFHCICST
  PArtitionset ==>
  SStatus      ==> Enabled
  PRIMedsize   : 00000
  TASKDATALoc  ==> Any
  TASKDATAKey  ==> USER key
REMOTE ATTRIBUTES
  DYnamic      ==> No
  REMOTESystem ==>
  REMOTEName   ==>
  TRProf       ==>
  Localq       ==>
SCHEDULING
  PRIOrity     ==> 001
  TClass       ==> No
  ALIASES
  Alias        ==>
  TASKReq      ==>
  XTRanid      ==>
  TPName       ==>
  XTPName      ==>
  RECOVERY
  DTimeout     ==> No
  Indoubt      ==> Backout
  REStart      ==> No
  REStart      ==> No
  SPurge       ==> No
  TPurge       ==> No
  DUmP         ==> Yes
  TRACe        ==> Yes
  SECURITY
  RESec        ==> No
  Cmdsec       ==> No
  Extsec       : No
  TRANsec      : 01
  RS1          : 00

```

Figure 9. EZAC, Transaction to Configure the Socket Interface, Definition in RDO

```

CEDA DEFine
  TRansaction   : EZAO
  Group         : TCPIPI
  Description   ==> Enable Sockets Interface
  PROGram      ==> EZACIC00
  TWasize      ==> 00000
  PROFile      ==> DFHCICST
  PArtitionset ==>
  SStatus      ==> Enabled
  PRIMedsize   : 00000
  TASKDATALoc  ==> Any
  TASKDATAKey  ==> CICS key
REMOTE ATTRIBUTES
  DYnamic      ==> No
  REMOTESystem ==>
  REMOTEName   ==>
  TRProf       ==>
  Localq       ==>
SCHEDULING
  PRIOrity     ==> 255
  TClass       ==> No
  ALIASES
  Alias        ==>
  TASKReq      ==>
  XTRanid      ==>
  TPName       ==>
  XTPName      ==>
  RECOVERY
  DTimeout     ==> No
  Indoubt      ==> Backout
  REStart      ==> No
  REStart      ==> No
  SPurge       ==> No
  TPurge       ==> No
  DUmP         ==> Yes
  TRACe        ==> Yes
  SECURITY
  RESec        ==> No
  Cmdsec       ==> No
  Extsec       : No
  TRANsec      : 01
  RS1          : 00

```

Figure 10. EZAO, Transaction to Enable the Socket Interface, Definition in RDO

```

CEDA DEFine
TRAnSACTION   : EZAP
Group        : TCPIPI
DEscription  ==> Disable Sockets Interface
PROGram      ==> EZACIC22
TWAsize      ==> 00000
PROFile      ==> DFHCICST
PARTitionset ==>
STatus       ==> Enabled
PRIMedsize   : 00000
TASKDATAloc  ==> Any
TASKDATAkey  ==> CICS
REMOTE ATTRIBUTES
DYnamic      ==> No
REMOTESystem ==>
REMOTEName   ==>
TRProf       ==>
Localq       ==>
SCHEDULING
PRIOrity     ==> 255
TClass       ==> No

ALIASES
Alias        ==>
TASKReq      ==>
XTRanid     ==>
TPName       ==>
XTPname      ==>

RECOVERY
DTimeout     ==> No
Indoubt      ==> Backout
REStart      ==> No
SPurge       ==> No
TPUrge       ==> No
DUmp         ==> Yes
TRACe        ==> Yes

SECURITY
RESec        ==> No
Cmdsec       ==> No
Extsec       : No
TRANsec      : 01
RS1          : 00

```

Figure 11. EZAP, Transaction to Disable the Socket Interface

```

CEDA DEFine
TRAnSACTION   : CSKL
Group        : TCPIPI
DEscription  ==> Listener task
PROGram      ==> EZACIC02
TWAsize      ==> 00000
PROFile      ==> DFHCICST
PARTitionset ==>
STatus       ==> Enabled
PRIMedsize   : 00000
TASKDATAloc  ==> Any
TASKDATAkey  ==> CICS
REMOTE ATTRIBUTES
DYnamic      ==> No
REMOTESystem ==>
REMOTEName   ==>
TRProf       ==>
Localq       ==>
SCHEDULING
PRIOrity     ==> 255
TClass       ==> No

ALIASES
Alias        ==>
TASKReq      ==>
XTRanid     ==>
TPName       ==>
XTPname      ==>

RECOVERY
DTimeout     ==> No
Indoubt      ==> Backout
REStart      ==> No
SPurge       ==> No
TPUrge       ==> No
DUmp         ==> Yes
TRACe        ==> Yes

SECURITY
RESec        ==> No
Cmdsec       ==> No
Extsec       : No
TRANsec      : 01
RS1          : 00

```

Figure 12. CSKL, Listener Task Transaction, Definition in RDO

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.

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 10 programs and one mapset to run CICS TCP/IP, or to provide supporting functions:

EZACIC00

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

EZACIC01

is the task related user exit (TRUE).

EZACIC02

is 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.

EZACIC20

is the initialization/termination front-end module for CICS sockets.

EZACIC21

is the initialization module for CICS sockets.

EZACIC22

is the termination module for CICS sockets.

EZACIC23

is the primary module for the configuration transaction (EZAC).

EZACIC24

is the message delivery module for transactions EZAC and EZAO.

EZACIC25

is the Domain Name Server (DNS) cache module.

EZACICME

is the US English text delivery module.

EZACICM

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

The following figures show sample RDO definitions of these programs.

Using Storage Protection: When running with CICS 3.3.0 on a storage-protection-enabled machine, all the required CICS TCP/IP programs (EZACIC00/01/02) must have EXECKEY=CICS as part of their CEDA definitions. The *CICS/ESA 3.3 Release 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.

```
CEDA DEFine
  PROGram      : EZACIC00
  Group        : TCPIPI
  DEscription  ==> Primary program for transaction EZA0
  Language     ==> Assembler
  RELoad       ==> No
  RESident     ==> No
  USAge        ==> Transient
  USElpacopy   ==> No
  Status       ==> Enabled
  RS1          : 00
  Cedf         ==> Yes
  DAtalocation ==> Any
  EXECKey      ==> CICS
REMOTE ATTRIBUTES
  REMOTESystem ==>
  REMOTENAME   ==>
  Transid      ==>
  EXECutionset ==> Fullapi
```

Figure 13. EZACIC00, Connection Manager Program, Definition in RDO

```
CEDA DEFine
  Program      : EZACIC01
  Group        : TCPIPI
  DEscription  ==> Task Related User Exit (TRUE)
  Language     ==> Assembler
  RELoad       ==> No
  RESident     ==> Yes
  USAge        ==> Normal
  USElpacopy   ==> No
  Status       ==> Enabled
  RS1          : 00
  Cedf         ==> Yes
  DAtalocation ==> Any
  EXECKey      ==> CICS
REMOTE ATTRIBUTES
  REMOTESystem ==>
  REMOTENAME   ==>
  Transid      ==>
  EXECutionset ==> Fullapi
```

Figure 14. EZACIC01, Task Related User Exit Program, Definition in RDO

```

CEDA DEFine
PROGram      : EZACIC02
Group       : TCPIPI
DEscription ==> IBM Listener
Language    ==> Assembler
RELoad     ==> No
RESident   ==> Yes
USAge      ==> Normal
USElpacopy ==> No
Status     ==> Enabled
RS1        : 00
Cedf       ==> Yes
Datalocation ==> Any
EXECKey    ==> CICS
REMOTE ATTRIBUTES
REMOTESystem ==>
REMOTENAME ==>
Transid    ==>
EXECUTIONset ==> Fullapi

```

Figure 15. EZACIC02, Listener Program, Definition in RDO

```

CEDA DEFine
PROGram      : EZACIC20
Group       : TCPIPI
DEscription ==> Initialization/Termination for CICS Sockets
Language    ==> Assembler
RELoad     ==> No
RESident   ==> No
USAge      ==> Transient
USElpacopy ==> No
Status     ==> Enabled
RS1        : 00
Cedf       ==> Yes
Datalocation ==> Any
EXECKey    ==> CICS
REMOTE ATTRIBUTES
REMOTESystem ==>
REMOTENAME ==>
Transid    ==>
EXECUTIONset ==> Fullapi

```

Figure 16. EZACIC20, Front-End Module for CICS Sockets, Definition in RDO

```

CEDA DEFine
PROGram      : EZACIC21
Group       : TCPIPI
DEscription ==> Initialization Module for CICS Sockets
Language    ==> Assembler
RELoad     ==> No
RESident   ==> No
USAge      ==> Transient
USEIpacopy ==> No
Status     ==> Enabled
RSI        : 00
Cedf       ==> Yes
Datalocation ==> Any
EXECKey    ==> CICS
REMOTE ATTRIBUTES
REMOTESystem ==>
REMOTENAME ==>
Transid    ==>
EXECUTIONset ==> Fullapi

```

Figure 17. EZACIC21, Initialization Module for CICS Sockets, Definition in RDO

```

CEDA DEFine
PROGram      : EZACIC22
Group       : TCPIPI
DEscription ==> Termination Module for CICS Sockets
Language    ==> Assembler
RELoad     ==> No
RESident   ==> No
USAge      ==> Transient
USEIpacopy ==> No
Status     ==> Enabled
RSI        : 00
Cedf       ==> Yes
Datalocation ==> Any
EXECKey    ==> CICS
REMOTE ATTRIBUTES
REMOTESystem ==>
REMOTENAME ==>
Transid    ==>
EXECUTIONset ==> Fullapi

```

Figure 18. EZACIC22, Termination Module for CICS Sockets, Definition in RDO

```

CEDA DEFine
PROGram      : EZACIC23
Group        : TCPIPI
DEscription  ==> Primary Module for Transaction EZAC
Language     ==> Assembler
RELoad      ==> No
RESident     ==> No
USAge       ==> Transient
USEIpacopy   ==> No
Status       ==> Enabled
RSI          : 00
Cedf        ==> Yes
Datalocation ==> Any
EXECKey     ==> User
REMOTE ATTRIBUTES
REMOTESystem ==>
REMOTENAME  ==>
Transid     ==>
EXECUTIONset ==> Fullapi

```

Figure 19. EZACIC23, Primary Module for Transaction EZAC, Definition in RDO

```

CEDA DEFine
PROGram      : EZACIC24
Group        : TCPIPI
DEscription  ==> Message Delivery Module for CICS Sockets
Language     ==> Assembler
RELoad      ==> No
RESident     ==> No
USAge       ==> Transient
USEIpacopy   ==> No
Status       ==> Enabled
RSI          : 00
Cedf        ==> Yes
Datalocation ==> Any
EXECKey     ==> CICS
REMOTE ATTRIBUTES
REMOTESystem ==>
REMOTENAME  ==>
Transid     ==>
EXECUTIONset ==> Fullapi

```

Figure 20. EZACIC24, Message Delivery Module for CICS Sockets, Definition in RDO

```

CEDA DEFine
PROGram      : EZACIC25
Group       : TCPIPI
DEscription ==> Cache Module for the Domain Name Server
Language    ==> Assembler
RELoad     ==> No
RESident   ==> Yes
USAge      ==> Normal
USEIpacopy ==> No
Status     ==> Enabled
RS1        : 00
Cedf       ==> Yes
Datalocation ==> Any
EXECKey    ==> USER
REMOTE ATTRIBUTES
REMOTESystem ==>
REMOTENAME ==>
Transid    ==>
EXECUTIONset ==> Fullapi

```

Figure 21. EZACIC25, Domain Name Server Cache Module, Definition in RDO

```

CEDA DEFine
PROGram      : EZACICME
Group       : TCPIPI
DEscription ==> US English Text Delivery Module
Language    ==> Assembler
RELoad     ==> No
RESident   ==> Yes
USAge      ==> Normal
USEIpacopy ==> No
Status     ==> Enabled
RS1        : 00
Cedf       ==> Yes
Datalocation ==> Any
EXECKey    ==> CICS
REMOTE ATTRIBUTES
REMOTESystem ==>
REMOTENAME ==>
Transid    ==>
EXECUTIONset ==> Fullapi

```

Figure 22. EZACICME, US English Text Delivery Module, Definition in RDO

```

CEDA DEFine
Mapset      : EZACICM
Group       : TCPIPI
Description ==> Mapset for CICS Sockets Interface
RESident   ==> No
USAge      ==> Transient
USEIpacopy ==> No
Status     ==> Enabled
RS1        : 00

```

Figure 23. EZACICM, Maps Used by EZAO and EZAP Transactions, Definition in RDO

Optional Programs, CICS Definition Needed

The following 2 programs are optional. They are the supplied samples. They are also in *hlq.SEZATCP*:

EZACICSS

is a sample iterative server. It establishes the connection between CICS and TCP/IP, and receives client request from workstations. See “EZACICSC” on page 295.

EZACICSC

is sample child server that works with the Listener (EZACIC02). See “EZACICSS” on page 302.

If these sample programs are used, they require RDO definitions as shown in Figures 24 and 25.

```
CEDA DEFine
PROGram      : EZACICSS
Group        : TCPIPI
DEscription  ==> Sample server
Language     ==> Cobol
RELoad      ==> No
RESident    ==> No
USAge       ==> Normal
USElpacopy  ==> No
Status      ==> Enabled
RS1         : 00
Cedf        ==> Yes
DAtalocation ==> Below or above
EXECKey     ==> USER
REMOTE ATTRIBUTES
REMOTESystem ==>
REMOTENAME  ==>
Transid     ==>
EXECutionset ==> Fullapi
```

Figure 24. EZACICSS, Sample Iterative Server Program, Definition in RDO

```

CEDA DEFine
PROGram      : EZACICSC
Group       : TCPIPI
DEscription ==> Sample started server
Language    ==> Cobol
RELoad     ==> No
RESident   ==> No
USAge      ==> Normal
USEIpacopy ==> No
Status     ==> Enabled
RSI        : 00
Cedf      ==> Yes
DAAllocation ==> Below or above
EXECKey    ==> USER
REMOTE ATTRIBUTES
REMOTESystem ==>
REMOTENAME ==>
Transid    ==>
EXECUTIONset ==> Fullapi

```

Figure 25. EZACICSC, Sample Child Server Program, Definition in RDO

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

is 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.SEZALINK*.

EZACIC07

the application stub that handles the C API.

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 information to define EZACONFG to RDO:

```

File          ==> EZACONFG
Group         ==> .....
DEscription  ==> CICS Sockets Configuration file

VSAM PARAMETERS
DSName       ==> U
Password     ==>                                PASSWORD NOT SPECIFIED
Lsrpoolid    ==> 1                               1-8 | None
DSNSharing   ==> Allreqs                         Allreqs | Modifyreqs
STRings      ==> 001                             1 - 255
Nsrgroup     ==> .....

REMOTE ATTRIBUTES
REMOTESystem ==>....
REMOTENAME   ==>.....
RECORDSize   ==>....                             1 - 32767
Keylength    ==>...                              1 - 255

INITIAL STATUS
STATUS       ==>Enabled                          Enabled | Disabled | Unenabled
Opertime     ==>Startup                          Firstref | Startup
Disposition  ==>Share                            Share | Old

BUFFERS
DATabuffers  ==>00002                             2 - 32767
Indexbuffers ==>00001                             1 - 32767

DATATABLE PARAMETERS
Table        ==> No                               No | Cics | User
Maxnumrecs   ==>.....                             16 - 16777215

DATA FORMAT
RECORDFormat ==>V                               V | F

OPERATIONS
Add          ==>No                                No | Yes
BRowse      ==>Yes                                No | Yes
DElete      ==>No                                No | Yes
REAd        ==>Yes                                Yes | No
Update      ==>No                                No | Yes

AUTO JOURNALING
JOurnal     ==>No                                No | 1-99
JNLRead     ==>None                              None | Updatedonly | Readonly | All
JNLSYNCRoad ==>No                                No | Yes
JNLUpdate   ==>No                                No | Yes
JNLAdd      ==>None                              None | Before | After | All
JNLSYNCRoad ==>No                                Yes | No

RECOVERY PARAMETERS
RECOVery    ==>No                                No | Backoutonly | All
Fwdrecovlog ==>No                                No | 1-99
BAckuptype  ==>Static                             STAtic | DYNAmic

SECURITY
RESsecnum   ==>00                                0-24 | Public

```

Figure 26. EZACONFG, defining to RDO

Notes:

1. Choose a DSName to fit installation standards.

EZACACHE

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

| **Note on sharing:** Do not attempt to share a cache file.
| Use the following information to define EZACACHE to RDO:
|

```

File          ==> EZACACHE
Group         ==> .....
DEscription  ==> Domain Name Server Cache Configuration file

VSAM PARAMETERS
DSName       ==> 1
Password     ==>                                PASSWORD NOT SPECIFIED
Lsrpoolid    ==>1                                1-8 | None
DSNSharing   ==> Allreqs                          Allreqs | Modifyreqs
STRings      ==> 2                                1 - 255
Nsrgroup     ==> .....

REMOTE ATTRIBUTES
REMOTESystem ==>....
REMOTENAME   ==>.....
RECORDSize   ==>....                                1 - 32767
Keylength    ==>...                                 1 - 255

INITIAL STATUS
STatus       ==>Enabled                            Enabled | Disabled | Unenabled
Opertime     ==>Startup                            Firstref | Startup
Disposition  ==>Old                                Share | Old

BUFFERS
Databuffers  ==>3                                  2 - 32767
Indexbuffers ==>4                                  1 - 32767

DATATABLE PARAMETERS
5 Table      ==> User                              No | Cics | User
Maxnumrecs   ==>6                                  16 - 16777215

DATA FORMAT
RECORDFormat ==>V                                V | F

OPERATIONS
Add          ==>Yes                                No | Yes
BRowse      ==>Yes                                No | Yes
DElete      ==>Yes                                No | Yes
READ        ==>Yes                                No | Yes
Update      ==>Yes                                No | Yes

AUTO JOURNALING
JOurnal     ==>No                                  No | 1-99
JNLRead     ==>None                                None | Updatedonly | Readonly | All
JNLSYNRead  ==>No                                  No | Yes
JNLUpdate   ==>No                                  No | Yes
JNLAdd      ==>None                                None | Before | After | All
JNLSYNWrite ==>No                                  Yes | No

RECOVERY PARAMETERS
RECOVery    ==>No                                  No | Backoutonly | All
Fwdrecovlog ==>No                                  No | 1-99
BAckuptype  ==>Static                               STAtic | DYNAmic

SECURITY
RESsecnum   ==>00                                  0-24 | Public

```

Figure 27. EZACACHE, defining to RDO

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 2.

4. Indexbuffers equals the number of records in the index set.
5. Although it is optional, we recommend specifying Table=User because it makes the process run faster. For more information on datatables, see *CICS/ESA Resource Definition (Online)* (SC33-0666).
6. Maxnumrecs equals the maximum number of destinations queried.

Transient Data Definition

Figure 28 shows the entries required in the CICS destination control table (DCT) to define the TCPM transient data queue for CICS TCP/IP. For more information on the DCT, refer to *CICS/ESA Resource Definition (Macro)* (SC33-0667).

Note that, in **2** below, the destination TCPM may be changed. If so, it must match the name specified in the ERRORTD parameter of the EZAC DEFINE CICS and/or the EZACICD TYPE=CICS (refer to “Configuration Macro” on page 44).

```

DFHDCT TYPE=SDSCI,                                X
        BLKSIZE=136,                               X
        DSCNAME=TCPCICS, 1                       X
        RECFORM=VARUNB,                             X
        RECSIZE=132,                                 X
        TYPEFLE=OUTPUT
        ...
        ...
DFHDCT TYPE=EXTRA,                                X
        DESTID=TCPM, 2                             X
        DSCNAME=TCPCICS
        ...
        ...
DFHDCT TYPE=INTRA,                                X
        DESTID=TRAA,                                 X
        DESTFAC=FILE, 3                             X
        TRIGLEV=1,                                   X
        TRANSID=TRAA
        ...
        ...

```

Figure 28. Addition to the DCT Required by CICS TCP/IP

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 by **1** and **2** in Figure 28.

The CICS startup JCL must include a DD statement for this extrapartition transient data queue (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 103. Entry **3** in Figure 28 shows an entry for an application that is started using the trigger-level mechanism of the DCT.

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'.

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

Figure 29. 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, PERFORM=ADDCNT(1,1)	X
DFHMCT TYPE=EMP, ID=(EZA02.02), CLASS=PERFORM, PERFORM=ADDCNT(2,1)	X
DFHMCT TYPE=EMP, ID=(EZA02.03), CLASS=PERFORM, PERFORM=ADDCNT(3,1)	X
DFHMCT TYPE=EMP, ID=(EZA02.04), CLASS=PERFORM, PERFORM=ADDCNT(4,1)	X
DFHMCT TYPE=EMP, ID=(EZA02.05), CLASS=PERFORM, PERFORM=ADDCNT(5,1)	X
DFHMCT TYPE=EMP, ID=(EZA02.06), CLASS=PERFORM, PERFORM=ADDCNT(6,1)	X
DFHMCT TYPE=EMP, ID=(EZA02.07), CLASS=PERFORM, PERFORM=ADDCNT(7,1)	X
DFHMCT TYPE=EMP, ID=(EZA02.08), CLASS=PERFORM, PERFORM=ADDCNT(8,1)	X
DFHMCT TYPE=EMP, ID=(EZA02.09), CLASS=PERFORM, PERFORM=ADDCNT(9,1)	X
DFHMCT TYPE=EMP, ID=(EZA02.10), CLASS=PERFORM, PERFORM=ADDCNT(10,1)	X
DFHMCT TYPE=EMP, ID=(EZA02.11), CLASS=PERFORM, PERFORM=ADDCNT(11,1)	X
DFHMCT TYPE=EMP, ID=(EZA02.12), CLASS=PERFORM, PERFORM=(MLTCNT(1,11)), COUNT=(1,CONN,STARTED,INVALID,DISTRAN,DISPROG,GIVESOKT,SECEXIT)	X X

Figure 30. 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 CICS Sockets interface automatically, make the following entry in PLTPI *after* the DFHDELIM entry:

```
DFHPLT TYPE=ENTRY,PROGRAM=EZACIC20
```

To shut down CICS Sockets interface automatically, make the following entry in the PLTSD *before* the DFHDELIM entry:

```
DFHPLT TYPE=ENTRY,PROGRAM=EZACIC20
```

TCP/IP for MVS — Modifying Data Sets

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

The *hlq.PROFILE.TCPIP* Data Set

You define the CICS region to TCP/IP on MVS in the *hlq.PROFILE.TCPIP* data set (described in *TCP/IP for MVS: Customization and Administration Guide*). In it, you must provide entries for the CICS region in the PORT statement, as shown in Figure 31 on page 43.

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 31 on page 43 shows 2 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.

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

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


```

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

```

Figure 31. Definition of the hlq.TCPIP Profile

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 for MVS setup. This parameter is the name of the started procedure used to start the TCP/IP MVS address space.

You will need it when you initialize CICS TCP/IP (see “Chapter 4. Starting and Stopping CICS Sockets” on page 79). 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 32. 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 dataset is done in three stages:

1. Create the empty dataset using VSAM IDCAMS (Access Method Services).
2. Initialize the dataset using the program generated by the EZACICD macro. The first two steps are described in “JCL for the Configuration Macro” on page 47.
3. Add to or modify the dataset using the configuration transaction EZAC. This step is described in “Customizing the Configuration Dataset” on page 50.⁷

Building the Configuration Dataset with EZACICD

Configuration Macro

The configuration macro (EZACICD) is used to build the configuration dataset. This dataset can then be incorporated into CICS using RDO and modified using the configuration transactions (see “Configuration Transaction (EZAC)” on page 50). The macro is keyword-driven with the TYPE keyword controlling the specific function request. The dataset 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,      Initialize generation environment
      PRGNAME=EZACICDF,        Name of the generated program
      FILNAME=EZACONFG        DD name of the configuration file
EZACICD TYPE=CICS,            Generate configuration record
      APPLID=CICS01,          APPLID of CICS
      TCPADDR=TCPV3R2,        Address space name for TCP/IP
      NTASKS=20,              Number of reusable subtasks
      DPRTY=10,               Subtask dispatching priority diff.
      CACHMIN=10,             Minimum refresh time for CACHE
      CACHMAX=20,             Maximum refresh time for CACHE
      CACHRES=5,              Maximum number of active resolvers
      ERRORTD=CSKN            Name of TD queue for error messages
EZACICD TYPE=LISTENER,       Create Listener Record
      APPLID=CICS01,          APPLID of CICS
      TRANID=CSKL,            Use standard transaction ID
      PORT=3010,              Use port number 3010
      BACKLOG=40,             Set backlog value to 40
      ACCTIME=30,             Set timeout value to 30 seconds
      GIVTIME=10,            Set givesocket timeout to 10 seconds
      REATIME=300,           Set read timeout to 5 minutes
      NUMSOCK=100,           Support 99 concurrent connections
      MINMSGL=4,             Minimum input message is 4 bytes
      IMMED=YES,             Start listener immediately
      FASTRD=YES             Read after Accept (no select)
EZACICD TYPE=FINAL
```

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

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

per generation and it should be in the first invocation of the macro. For sub-parameters, refer to "TYPE=INITIAL".

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

LISTENER

Identify a Listener object. This will create a listener record. For sub-parameters, refer to "TYPE=LISTENER" on page 46.

FINAL indicates the end of the generation. There are no sub-parameters.

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 non-zero 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.

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.

PORT The port number this listener will use for accepting connections. This parameter is mandatory. The value should be between 2049 and 65535.

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. Setting this value high will minimize CPU consumption on a lightly loaded system but will lengthen shutdown processing. Setting this value low will use more CPU 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.

REETIME

The time in seconds this listener will wait for a response to a READ 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.

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.

MINMSGL

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. IMMED (below) 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.

Blocking note: IMMED=YES acts as a blocking read. IMMED=NO causes the system to ensure that data is present before issuing a read (that is, it does not block).

FASTRD

Specify YES or NO. YES indicates this listener will issue a READ immediately after completion of the ACCEPT, i.e. without issuing an intervening SELECT. NO indicates this listener will issue a SELECT between the ACCEPT and the READ. YES improves performance but relies on the client sending data immediately after its connect request. NO assumes the client may connect without sending data immediately.

TRANTRN

Specify YES or NO. YES indicates the transaction code from the Transaction Initial Message should be translated from ASCII to EBCDIC. NO indicates the first character of the transaction code will be tested to determine if it is Upper Case EBCDIC. If so, it is not translated. The default value is YES.

Note: If the first character is not upper case EBCDIC and NO is specified, translation still occurs.

TRANUSR

Specify YES or NO. YES indicates the user data from the Transaction Initial Message should be translated from ASCII to EBCDIC. The default value is YES.

Note: Previous implementations functioned as if TRANTRN was set to 'NO' and TRANUSR was 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.

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 jobstream to create and initialize the Configuration File. The jobstream 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 which initializes the file.

Figure 33 on page 48 illustrates a jobstream used to define a configuration file.

```

//*****//
//* 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
//*
//DEFINE 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 ) -
DATA ( -
    NAME(CICS.TCP.CONFIG.DATA) -
    KEYS (16 0) ) -
INDEX ( -
    NAME(CICS.TCP.CONFIG.INDEX) )

/*
//

```

Figure 33. Example of JCL to Define a Configuration File (Part 1 of 2)

```

/** THIS STEP ASSEMBLES THE INITIALIZATION PROGRAM
/**

//PRGDEF EXEC PGM=IEV90,PARM='OBJECT,TERM',REGION=1024K
//SYSLIB DD DISP=SHR,DSNAME=SYS1.MACLIB
// DD DISP=SHR,DSNAME=TCPV32.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)
//SYSTEM DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
EZACICD TYPE=INITIAL, Initialize generation environment X
PRGNAME=EZACICDF, Name of the generated program X
FILNAME=EZACONFG DD name of the configuration file
EZACICD TYPE=CICS, Generate configuration record X
APPLID=CICS01, APPLID of CICS X
TCPADDR=TCPV3R2, Address space name for TCP/IP X
NTASKS=20, Number of reusable MVS subtasks X
DPRTY=10, Priority difference (CICS-Subtask) X
CACHMIN=10, Minimum refresh time for CACHE X
CACHMAX=20, Maximum refresh time for CACHE X
CACHRES=5, Maximum number of active resolvers X
ERRORTD=CSKN Name of TD queue for error messages
EZACICD TYPE=LISTENER, Create Listener Record X
APPLID=CICS01, APPLID of CICS X
TRANID=CSKL, Use standard transaction ID X
PORT=3010, Use port number 3010 X
BACKLOG=40, Set backlog value to 40 X
ACCTIME=30, Set timeout value to 30 seconds X
GIVTIME=10, Set givesocket timeout to 10 seconds X
REETIME=300 Set read timeout to 5 minutes X
NUMSOCK=100, Support 99 concurrent connections X
MINMSGL=4 Minimum input message is 4 bytes
EZACICD TYPE=FINAL

/*
/**
/** 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,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 INITIALIZATION PROGRAM
/**
//FILELOAD EXEC PGM=*.LINK.SYSLMOD,COND=(4,LT)
//EZACONFG DD DSNAME=CICS.TCP.CONFIG,DISP=OLD

```

Figure 33. Example of JCL to Define a Configuration File (Part 2 of 2)

Customizing the Configuration Dataset

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 which 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 datasets: You can use EZAC to modify a dataset while CICS is running, as long as the dataset has been run at least once before being loaded.

Command Summary.

<i>Command</i>	<i>Object</i>	<i>Function</i>
ALTER	CICS/Listener	Modifies the attributes of an existing resource definition
COPY	CICS/Listener	<ul style="list-style-type: none">• 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	Create a new resource definition
DELETE	CICS/Listener	<ul style="list-style-type: none">• 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
  ENTER ONE OF THE FOLLOWING
ALter
COpy
DEFine
DELeTe
DISplay
REName

APPLID=DBDCCICS

PF 1 HELP      3 END      6 CRSR      9 MSG      12 CNCL

```

Figure 34. EZAC Initial Screen

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

```

EZAC ALTER
  ENTER ONE OF THE FOLLOWING

CICS      ===>      Enter Yes|No
LISTener  ===>      Enter Yes|No

APPLID=DBDCCICS

PF      3 END      9 MSG      12 CNCL

```

Figure 35. EZAC ALTER Screen

Note: You can short-cut this 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
  ENTER ALL FIELDS

APPLID      ==>

APPLID of CICS System

APPLID=DBDCCICS

PF          3  END

12  CNCL

```

Figure 36. EZAC ALTER CICS screen

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

```

EZAC ALTER CICS
  OVERTYPE TO ENTER

APPLID      ==> .....      APPLID of CICS System
TCPAddr     ==> .....      Name of TCP/IP Address Space
NTasks      ==> ...         Number of Reusable Tasks
DPRty       ==> ...         (CICS-Subtask) dispatch priority
CACHMIN     ==> ...         Minimum Refresh Time for Cache
CACHMAX     ==> ...         Maximum Refresh Time for Cache
CACHRES     ==> ..         Maximum Number of Resolvers
ERRortd     ==> ....        TD queue for Error Messages

APPLID=DBDCCICS

PF 1  HELP      3  END      6  CRSR      9  MSG      12  CNCL

```

Figure 37. 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:
After the names are entered, the following screen is displayed:

```

EZAC ALTER
  ENTER ALL FIELDS

APPLID      ==>
NAME        ==>

                                APPLID of CICS System
                                Transaction Name of Listener

                                                                APPLID=DBDCCICS

PF              3  END
                                                                12  CNCL
  
```

Figure 38. ALTER LISTENER screen

The system will request a confirmation of the values displayed. After the changes

```

EZAC ALTER LISTENER
  OVERTYPE TO MODIFY

APPLID      ==> .....
TRanid      ==> .....
PORt        ==> .....
IMMEDiate   ==> ...
Backlog     ==> ...
NUMsock     ==> ..
MINmsgl     ==> ..
ACCTime     ==> ..
GIVTime     ==> ..
REATime     ==> ..
FASTrd      ==> ...
TRANTrn     ==> ...
TRANUsr     ==> ...
SECexit     ==> .....

                                APPLID of CICS System
                                Transaction Name of Listener
                                Port Number of Listener
                                Immediate Startup      Yes|No
                                Backlog Value for Listener
                                Number of Sockets in Listener
                                Minimum Message Length
                                Timeout Value for Accept
                                Timeout Value for Givesocket
                                Timeout Value for Read
                                Read immediately        Yes|No
                                Translate Trans. Name    Yes|No
                                Translate User Data      Yes|No
                                Security Exit Name

PF 1  HELP      3  END          6  CRSR          9  MSG          12  CNCL
  
```

Figure 39. EZAC ALTER LISTENER Detail Screen

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 CO on a blank screen, the following screen is displayed:

```
EZAC COPY
  ENTER ONE OF THE FOLLOWING

CICS      ==>
LISTener  ==>

Enter Yes|No
Enter Yes|No

APPLID=DBDCCICS

PF          3 END          12 CNCL
```

Figure 40. EZAC COPY Screen

Note: You can short-cut this 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
ENTER ALL FIELDS
SCICS      ==> .....      APPLID of Source CICS
TCICS      ==> .....      APPLID of Target CICS

                                     APPLID=DBDCCICS

PF          3 END          9 MSG          12 CNCL
```

Figure 41. EZAC COPY 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:
After the APPLIDs of the source and target CICS objects and the names of the

```
EZAC COPY
  ENTER ALL FIELDS
SCICS      ==> .....      APPLID of Source CICS
SLISTener  ==> ...         Transaction Name of Source Listener
TCICS      ==> .....      APPLID of Target CICS
TLISTener  ==> ...         Transaction Name of Target Listener

                                     APPLID=DBDCCICS

PF          3  END                    12  CNCL
```

Figure 42. EZAC COPY Screen

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:

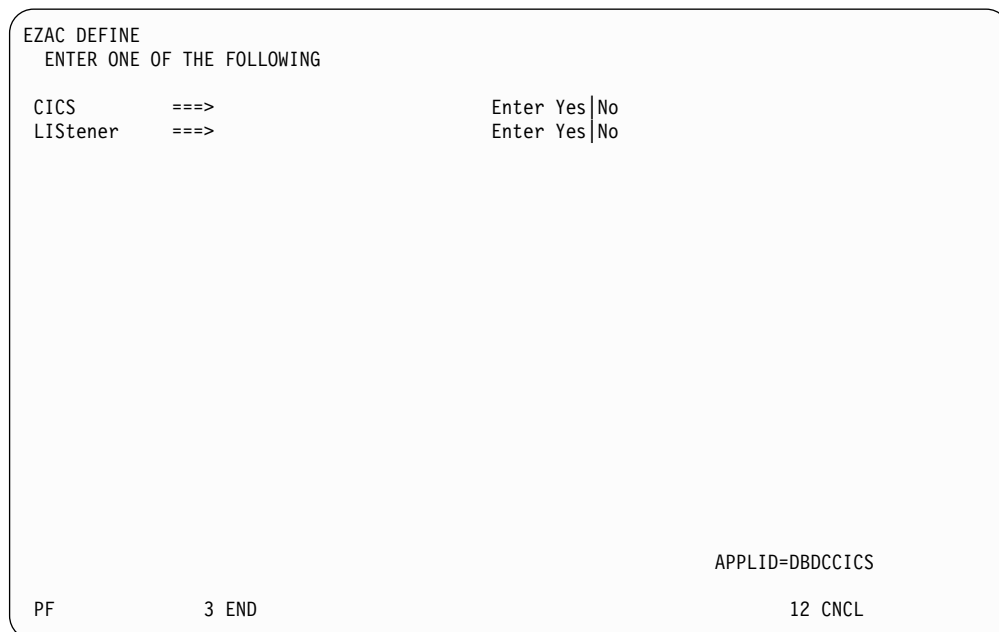


Figure 43. EZAC DEFINE Screen

Note: You can short-cut this by entering either EZAC DEFINE CICS or EZAC DEFINE LISTENER.

DEFINE CICS: For definition of a CICS object, the following screen is displayed: After the APPLID is entered, the following screen is displayed.

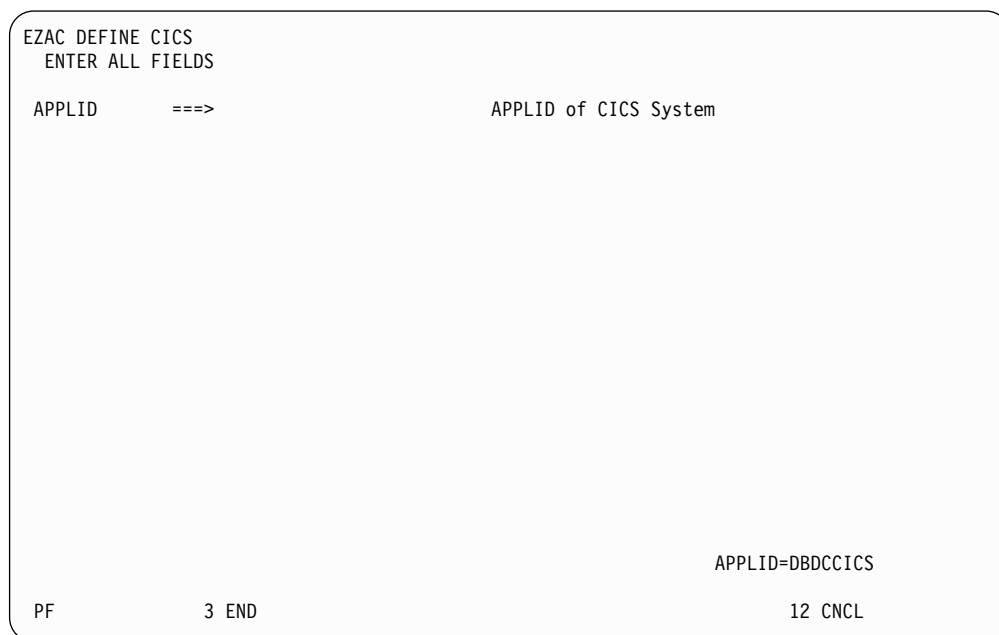


Figure 44. EZAC DEFINE CICS screen

```

EZAC DEFINE CICS
OVERTYPE TO ENTER

APPLID      ==> .....      APPLID of CICS System
TCPAddr     ==> .....      Name of TCP/IP Address Space
NTAsks      ==> ...        Number of Reusable Tasks
DPRty       ==> ...        (CICS-Subtask) dispatch priority
CACHMIN     ==> ...        Minimum Refresh Time for Cache
CACHMAX     ==> ...        Maximum Refresh Time for Cache
CACHRES     ==> ..        Maximum Number of Resolvers
ERRortd     ==> ....      TD queue for Error Messages

APPLID=DBDCCICS

PF          3 END          9 MSG          12 CNCL

```

Figure 45. 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: After the names are entered, the following screen is displayed:

```

EZAC DEFINE LISTENER
ENTER ALL FIELDS

APPLID      ==> .....      APPLID of CICS System
NAME        ==> .....      Transaction Name of Listener

APPLID=DBDCCICS

PF          3 END          12 CNCL

```

Figure 46. EZAC DEFINE LISTENER screen


```

EZAC DEFINE LISTENER
OVERTYPE TO MODIFY

APPLID      ==> .....          APPLID of CICS System
TRanname    ==> .....          Transaction Name of Listener
POrt        ==> .....          Port Number of Listener
IMMediate   ==> Yes           Immediate Startup          Yes|No
BAcklog     ==> 020           Backlog Value for Listener
NUMsock     ==> 50           Number of Sockets in Listener
MINmsgl     ==> 04           Minimum Message Length
ACCTime     ==> 60           Timeout Value for Accept
GIVTime     ==> 60           Timeout Value for Givesocket
REATime     ==> 10           Timeout Value for Read
FASTread    ==> Yes           Read immediately          Yes|No
TRANTrn     ==> Yes           Translate Trans. Name     Yes|No
TRANUsr     ==> Yes           Translate User Data       Yes|No
SECexit     ==> .....          Security Exit Name

PF          3 END          12 CNCL

```

Figure 47. EZAC DEFINE LISTENER Detail Screen

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
ENTER ONE OF THE FOLLOWING

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

APPLID=DBDCCICS

PF          3 END          12 CNCL

```

Figure 48. EZAC DELETE Screen

DELETE CICS: If you specify DELETE CICS, the following screen is displayed:
After the APPLID is entered, confirmation is requested. When the confirmation is

```
EZAC DELETE CICS
  ENTER ALL FIELDS

APPLID      ==>>>                APPLID of CICS System

                                           APPLID=DBDCCICS

PF          3 END                    12 CNCL
```

Figure 49. EZAC DELETE CICS screen

entered, the CICS object is deleted.

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

```

EZAC DELETE LISTENER
ENTER ALL FIELDS

APPLID      ==>
NAME        ==>

APPLID of CICS System
Transaction Name of Listener

APPLID=DBDCCICS

PF          3 END          12 CNCL

```

Figure 50. 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:

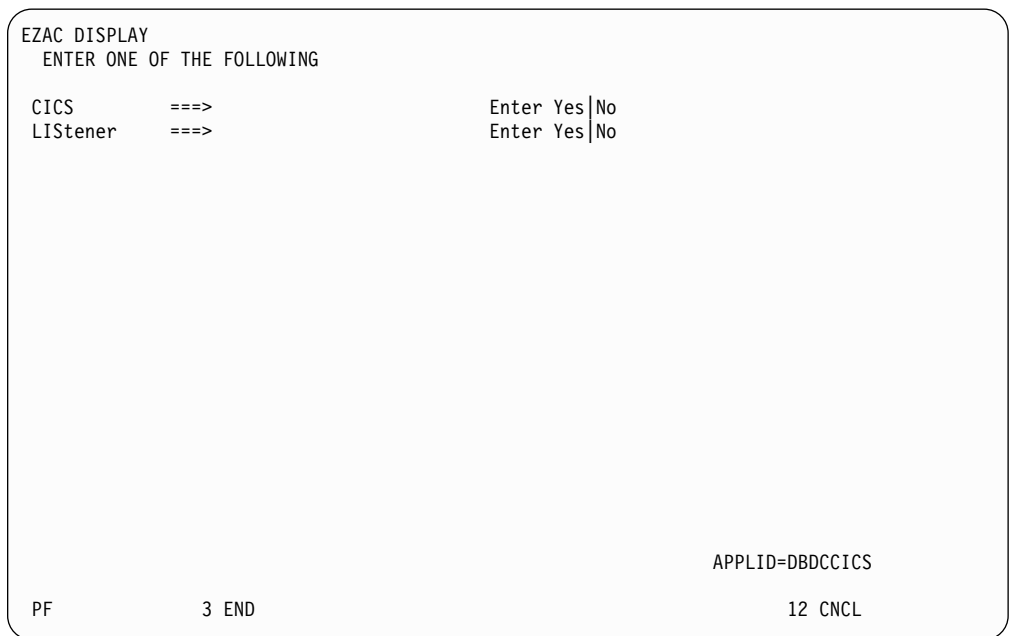


Figure 51. EZAC DISPLAY Screen

Note: You can short-cut this by entering either EZAC DISPLAY CICS or EZAC DISPLAY LISTENER.

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

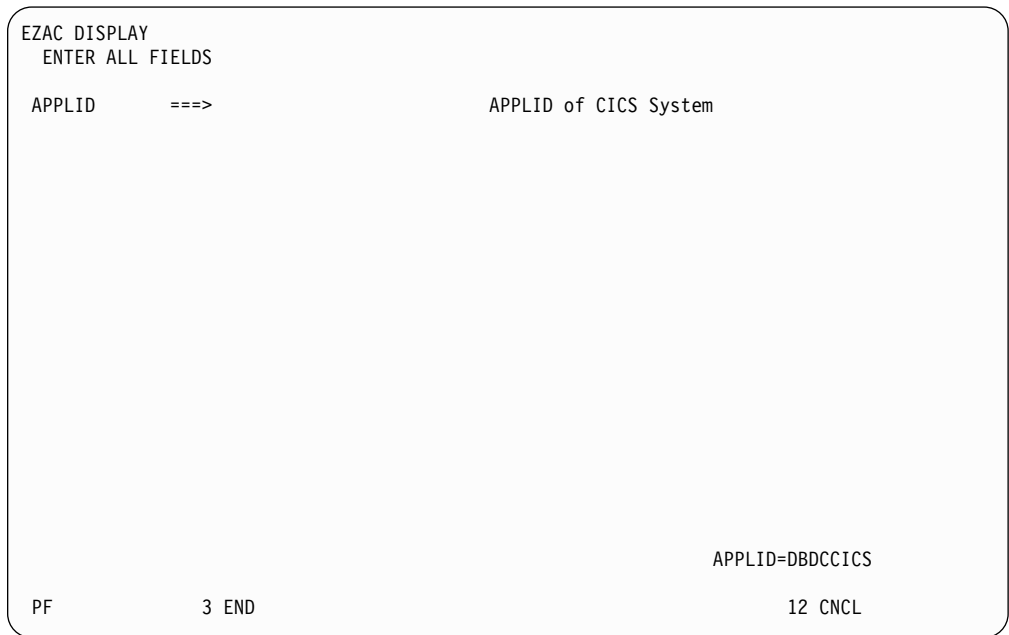


Figure 52. EZAC DISPLAY CICS screen

```

EZAC DISPLAY CICS

APPLID      ==> .....      APPLID of CICS System
TCPAddr     ==> .....      Name of TCP/IP Address Space
NTasks      ==> ...        Number of Reusable Tasks
DPRty       ==> ...        (CICS-Subtask) dispatch priority
CACHMIN     ==> ...        Minimum Refresh Time for Cache
CACHMAX     ==> ...        Maximum Refresh Time for Cache
CACHRES     ==> ..        Maximum Number of Resolvers
ERRortd     ==> ....      TD queue for Error Messages

APPLID=DBDCCICS

PF          3 END          9 MSG          12 CNCL

```

Figure 53. EZAC DISPLAY CICS Detail Screen

DISPLAY LISTENER: If you specify DISPLAY LISTENER, the following screen is displayed:
 After the APPLID and name are entered, the following screen is displayed

```

EZAC DISPLAY
ENTER ALL FIELDS

APPLID      ==> .....      APPLID of CICS System
NAME        ==> .....      Transaction Name of Listener

APPLID=DBDCCICS

PF          3 END          12 CNCL

```

Figure 54. EZAC DISPLAY LISTENER screen

```

EZAC DISPLAY LISTENER

APPLID      ==> .....      APPLID of CICS System
TRaname     ==> .....      Transaction Name of Listener
POrt        ==> .....      Port Number of Listener
IMMediate   ==> Yes        Immediate Startup          Yes|No
BAcklog     ==> 020        Backlog Value for Listener
NUMsock     ==> 50         Number of Sockets in Listener
MINmsgl     ==> 04         Minimum Message Length
ACCTime     ==> 60         Timeout Value for Accept
GIVTime     ==> 60         Timeout Value for Givesocket
REATime     ==> 10         Timeout Value for Read
FASTread    ==> Yes        Read immediately          Yes|No
TRANtrn     ==> Yes        Translate Trans. Name     Yes|No
TRANusr     ==> Yes        Translate User Data       Yes|No
SECexit     ==> .....      Security Exit Name

PF              3 END              12 CNCL

```

Figure 55. EZAC DISPLAY LISTENER Detail Screen

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  
ENTER ONE OF THE FOLLOWING  
  
CICS      ==>  
LIStener  ==>  
  
Enter Yes|No  
Enter Yes|No  
  
APPLID=DBDCCICS  
  
PF          3 END          12 CNCL
```

Figure 56. EZAC RENAME Screen

Note: You can short-cut this 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:
After the APPLIDs of the source CICS object and the target CICS object are

```
EZAC RENAME
ENTER ALL FIELDS

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

                                           APPLID=DBDCCICS

PF          3 END          9 MSG          12 CNCL
```

Figure 57. EZAC RENAME CICS Screen

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
ENTER ALL FIELDS

SCICS      ==> .....      APPLID of Source CICS
SLISTener  ==> ....       Transaction Name of Source Listener
TCICS      ==> .....      APPLID of Target CICS
TLISTener  ==> ....       Transaction Name of Target Listener

APPLID=DBDCCICS

PF          3 END          12 CNCL
```

Figure 58. 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.

Chapter 3. Configuring the CICS Domain Name Server Cache

Overview of the Domain Name Server Cache	69
Function Components	69
VSAM Cache File	69
EZACICR macro	70
EZACIC25 Module	70
How the DNS Handles Requests	70
Using the DNS Cache	71
Step 1: Create the Initialization Module	71
Step 2: Define the Cache File to CICS	74
Step 3: Execute EZACIC25	75
HOSTENT Structure	76

Overview of the Domain Name Server Cache

The Domain Name Server (DNS) cache 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 via input data to the name server in the resource records.

While the CICS Domain Name Server Cache function is optional, it is useful in a highly active CICS 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.

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 domain 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 hosts 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 71.

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 as follows:

Error Destination

The Transient Data destination to which error messages are sent. It is set to 'CSMT'.

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 Handles Requests".

How the DNS 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 gethostbyname 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, go issue the gethostbyname call and refresh the cache record with the results.
 - If the age is between the minimum and maximum cache refresh values, do the following:

- a. Calculate the difference between the maximum and minimum cache refresh times and divide it by the maximum number of concurrent resolver requests. The result is called the time increment.
 - b. Multiply the time increment by the number of currently active resolver requests. Add this time to the minimum refresh time giving the adjusted refresh time.
 - c. If the age of the record is less than the adjusted refresh time, use the cache record.
 - d. If the age of the record is greater than the adjusted refresh time, issue the gethostbyname call and refresh the cache record with the results.
- If the gethostbyname is issued and is successful, the cache is updated and the update time for the entry is changed to the current time.

Using the DNS Cache

There are three steps to using the DNS cache.

1. Create the initialization module, which in turn defines and initializes the file and the EZACIC25 module. See “Step 1: Create the Initialization Module”.
2. Define the cache files to CICS. See “Step 2: Define the Cache File to CICS” on page 74 .
3. Use EZACIC25 to replace gethostbyname calls in CICS application modules. See “Step 3: Execute EZACIC25” on page 75.

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 be the same as the first sub-parameter 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 59 on page 73 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
//*
//DEFINE EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DEFINE CLUSTER (NAME(CICS.USER.CACHE) VOLUMES(CICSVOL) -
    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=IEV90,PARM='OBJECT,TERM',REGION=1024K
//SYSLIB DD DISP=SHR,DSNAME=SYS1.MACLIB
// DD DISP=SHR,DSNAME=TCPV32.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)
//SYSTEM DD SYSOUT=*
//SYSPRINT DD SYSOUT=*

```

Figure 59. 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 DSNNAME=&&LOADSET(GO),DISP=(MOD,PASS),UNIT=SYSDA,
//          SPACE=(TRK,(1,1,1)),
//          DCB=(DSORG=P0,RECFM=U,BLKSIZE=32760)
//SYSLIN   DD DSNNAME=&&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 59. 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 of the file.

Record Type

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

Last Refresh Time

A 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 name(s) returned from the domain 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 e.g.
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 60 on page 77

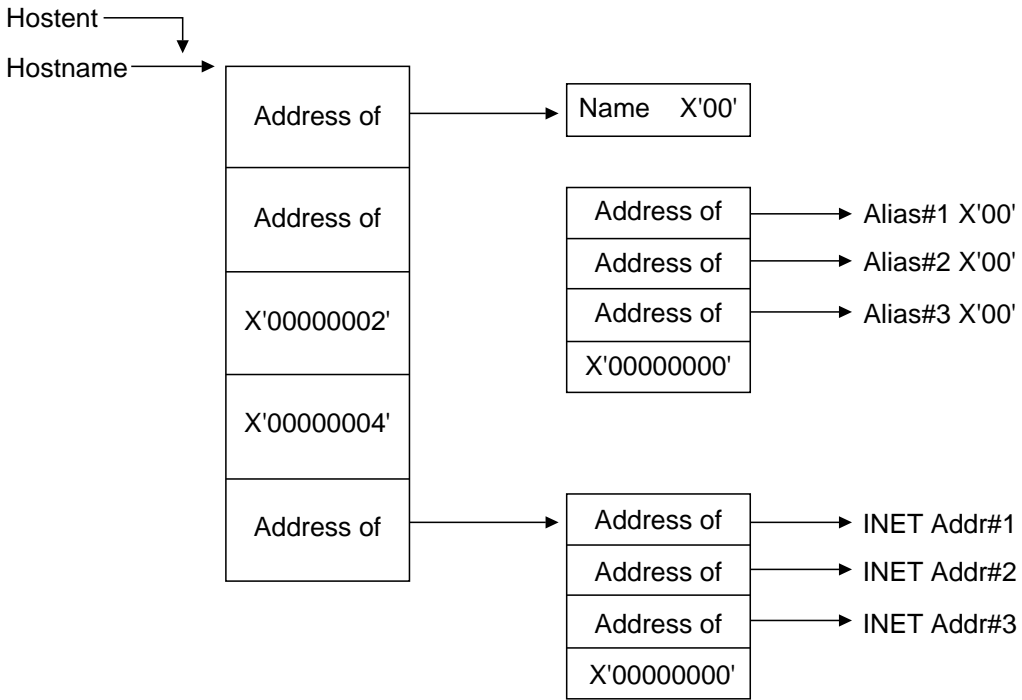


Figure 60. The DNS Hostent

|

Chapter 4. Starting and Stopping CICS Sockets

Overview	79
Starting/Stopping CICS TCP/IP Automatically	79
Starting/Stopping CICS TCP/IP Manually	79
START Function	80
START CICS	81
START LISTENER	81
STOP Function	82
STOP CICS.	83
STOP LISTENER	84
Starting/Stopping CICS TCP/IP with Program Link	85

Overview

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

- 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”.
- 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 85.

This section describes all three methods.

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 CICS Sockets interface automatically, make the following entry in PLTPI *after* the DFHDELIM entry:

```
DFHPLT TYPE=ENTRY,PROGRAM=EZACIC20
```

- Shutdown (PLTSD)

To shut down CICS Sockets interface automatically, make the following entry in the PLTSD *before* the DFHDELIM entry:

```
DFHPLT TYPE=ENTRY,PROGRAM=EZACIC20
```

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 which 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.

When you enter EZAO, the following screen displays.

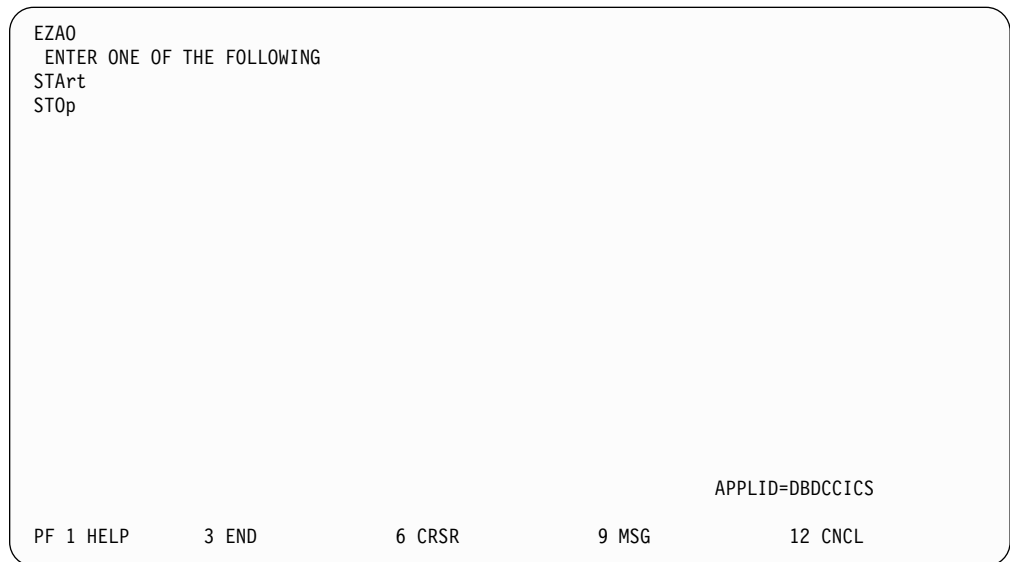


Figure 61. EZAO Initial Screen

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 displays.

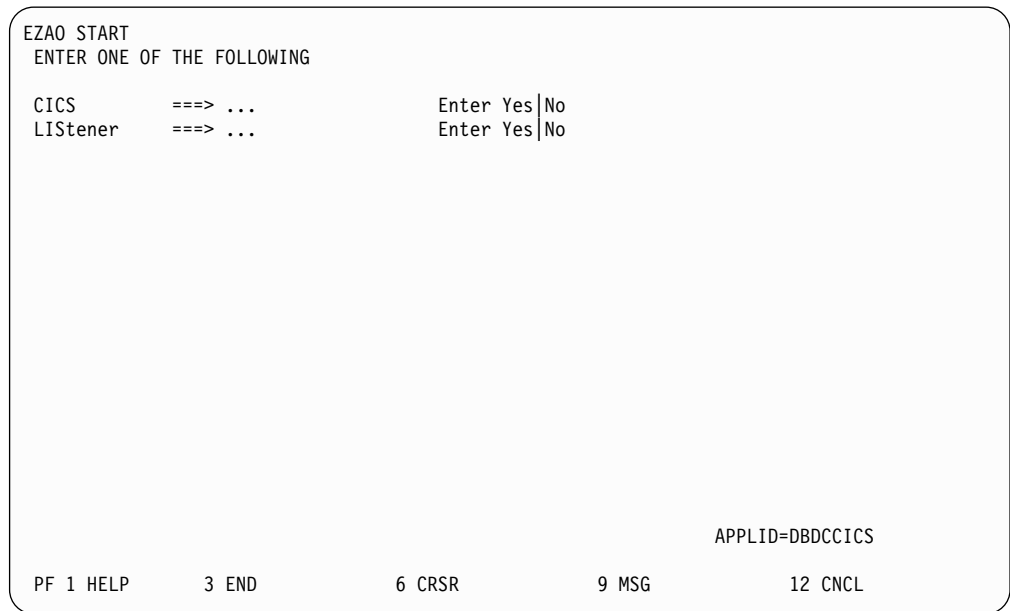


Figure 62. EZAC START Screen

START CICS

If you enter START CICS, the following screen displays.

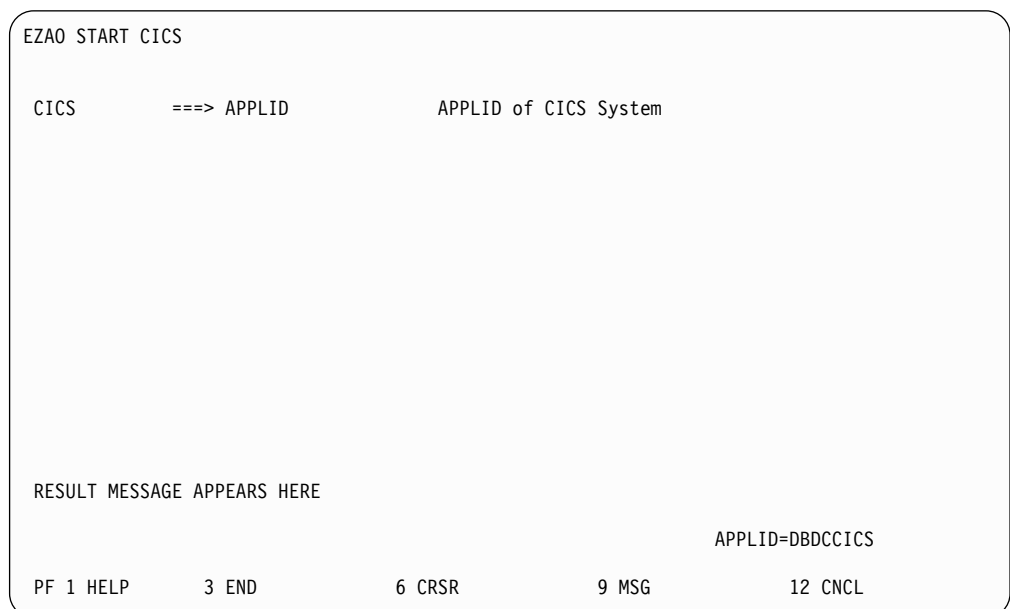


Figure 63. EZAC START CICS Response Screen

START LISTENER

If you enter START LISTENER, the following screen displays.

```

EZA0 START LISTENER
ENTER LISTENER NAME

CICS      ==> APPLID      APPLID of CICS System
NAME      ==>           Transaction Name of Listener

                                           APPLID=DBDCCICS

PF          3 END          9 MSG          12 CNCL

```

Figure 64. EZAC START LISTENER Screen

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

```

EZA0 START LISTENER

CICS      ==> APPLID      APPLID of CICS system
NAME      ==> XXXX      Transaction Name of Listener

RESULT MESSAGE APPEARS HERE

                                           APPLID=DBDCCICS

PF          3 END          9 MSG          12 CNCL

```

Figure 65. EZAC 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 will be displayed:

```
EZAO STOP
ENTER ONE OF THE FOLLOWING

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

APPLID=DBCDCICS

PF 1 HELP      3 END      6 CRSR      9 MSG      12 CNCL
```

Figure 66. EZAC STOP Screen

STOP CICS

If you specify STOP CICS, the following screen is displayed

```
EZAO STOP CICS
SPECIFY IMMEDIATE STOP

CICS      ==> ...      APPLID of CICS
IMMEDIATE ==> ...      Enter Yes|No

APPLID=DBCDCICS

PF 1 HELP      3 END      6 CRSR      9 MSG      12 CNCL
```

Figure 67. EZAC STOP 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, then 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 re-displays; the results appear in the message line.

STOP LISTENER

If you specify STOP LISTENER, the following screen displays.

```

EZAO STOP
ENTER LISTENER NAME
CICS      ==> DBDCCICS      APPLID of this CICS
LISTener  ==> .....      Transaction Name of Listener

APPLID=DBDCCICS

PF 1 HELP      3 END      6 CRSR      9 MSG      12 CNCL

```

Figure 68. EZAO STOP LISTENER Screen

When you enter the listener named, that listener will be stopped. The screen re-displays; 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.

Chapter 5. Writing Your Own Listener

Basic Requirements.	87
Pre-Requisites.	87
Using IBM's Environmental Support	87

Basic Requirements

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 must have a unique port assigned to it. Assignment of the same port to multiple listeners will cause an error on the bind call for the IBM-supplied listener. If you attempt to circumvent this in a user-written listener using the SOREUSADDR option, the results will be unpredictable (and probably undesirable).
- Each listener should have an entry in the CICS Sockets Configuration Dataset. 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 103.

Pre-Requisites

Some installations may require a customized, user-written listener. Writing your own listener has the following prerequisites:

1. Determine what capability is required which 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 as 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 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 Global Work Area (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*.

- 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 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 via 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.

- 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 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, i.e. 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 which 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.

Chapter 6. Application Programming Guide

Overview	91
Writing CICS TCP/IP Applications.	92
1. The Client-Listener-Child-Server Application Set	93
Client Call Sequence	94
Listener Call Sequence	95
Child Server Call Sequence	95
2. Writing Your Own Concurrent Server	96
Concurrent Server Call Sequence	96
Passing Sockets	97
3. The Iterative Server CICS TCP/IP Application	97
Iterative Server Use of Sockets	98
4. The Client CICS TCP/IP Application	98
Socket Addresses	99
Address Family (Domain).	99
IP Addresses	99
Ports	99
Address Structures	99
For COBOL and Assembler Language Programs	100
For C Programs	100
MVS Address Spaces	100
Network Byte Order.	101
GETCLIENTID, GIVESOCKET, and TAKESOCKET	101
The Listener	103
Listener Input Format	103
Examples	104
Listener Output Format	104
Writing Your Own Security Link Module for the Listener	105
Data Conversion Routines	107

Overview

This chapter describes how to write applications that use the 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:
 1. Concurrent server (the supplied Listener transaction) and child server processes run under CICS TCP/IP
 2. The same as 1 but with a user-written concurrent server
 3. An iterative server running under CICS TCP/IP
 4. 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 109 describes the C language calls that can be used with CICS.

“Chapter 8. Sockets Extended Application Programming Interface (API)” on page 145 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.***

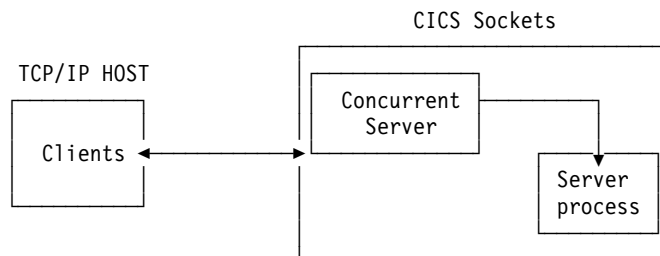
Note: “Appendix B. Original COBOL Application Programming Interface (EZACICAL)” on page 223 provides reference information on the EZACICAL API for COBOL and assembler language. This interface was made available with Release 2.2.1 of TCP/IP for MVS and is being retained in the current release for compatibility. ***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 145.

Writing CICS TCP/IP Applications

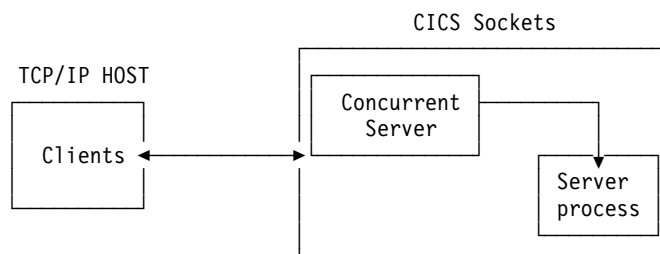
“Chapter 1. Introduction to CICS TCP/IP” on page 1 describes the basics of TCP/IP client/server systems and the two types of server: iterative and concurrent. This chapter considers in detail 4 TCP/IP setups in which CICS TCP/IP applications are used in various parts of the client/server system.

The setups are:

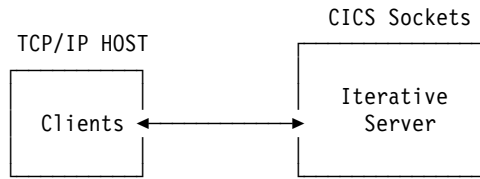
1. **The Client-Listener-Child Server Application Set.** The concurrent server and child server processes run under CICS TCP/IP. The concurrent server is the supplied **Listener** transaction. The client might be running TCP/IP under the OS/2 operating system or one of the various UNIX operating systems such as AIX*.



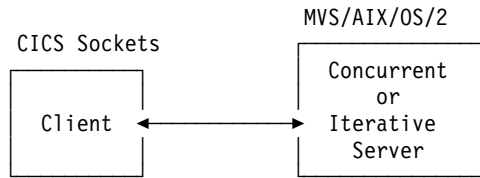
2. **Writing Your Own Concurrent Server.** This is the same setup as the first except that a user-written concurrent server is being used instead of the IBM Listener.



3. **The Iterative Server CICS TCP/IP Application.** This setup is designed to process one socket at a time.



4. **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., Chapter 8., and Appendix 2.

1. The Client-Listener-Child-Server Application Set

Figure 69 on page 94 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.

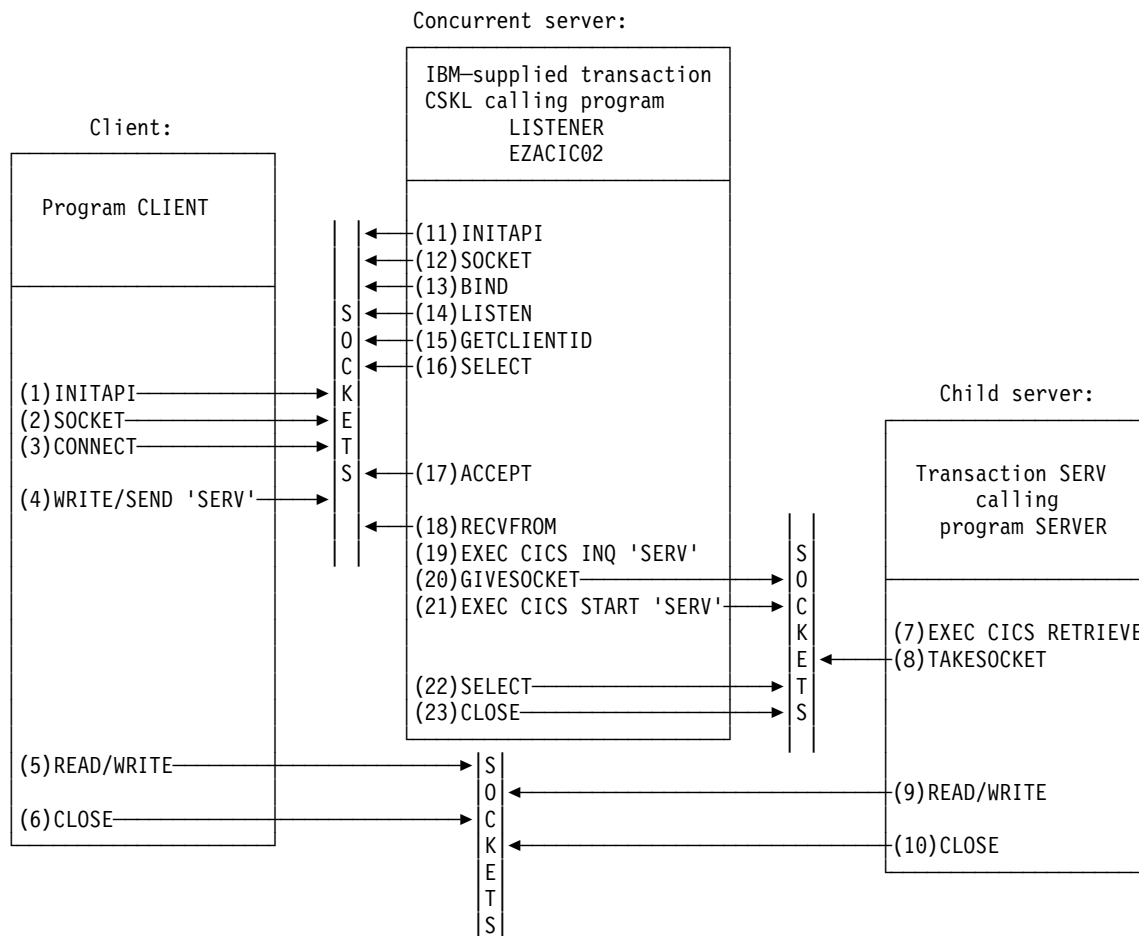


Figure 69. The Sequence of Sockets Calls

Client Call Sequence

Table 3 explains the functions of each of the calls listed in Figure 69

Table 3. 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 MAX-SOCK parameter to specify the maximum number of sockets to be used by the application.
-------------	--

Table 3. Calls for the Client Application (continued)

(2) SOCKET	<p>This obtains a socket. You define a socket with 3 parameters:</p> <ul style="list-style-type: none">• The domain, or addressing family• The type of socket• The protocol <p>For CICS TCP/IP, the domain can only be the TCP/IP internet domain (2 in COBOL, AF_INET 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.</p> <p>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.</p>
(3) CONNECT	<p>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).</p>
(4) WRITE	<p>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.</p>
(5) READ/WRITE	<p>These calls continue the conversation with the server until it is complete.</p>
(6) CLOSE	<p>This closes a specified socket and so ends the connection. The socket resources are released for other applications.</p>

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 96, where the Listener calls in Figure 69 are explained.

Child Server Call Sequence

Table 4 explains the functions of each of the calls listed in Figure 69 on page 94

Table 4. Calls for the Server Application

(7) EXEC CICS RETRIEVE	<p>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.</p>
(8) TAKESOCKET	<p>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.</p>
(9) READ/WRITE	<p>The conversation with the client continues until complete.</p>

Table 4. Calls for the Server Application (continued)

(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 5 explains the functions of each of the steps listed in Figure 69 on page 94

Table 5. Calls for the Concurrent Server Application

(11) INITAPI	Connects the application to TCP/IP, as in Table 3.
(12) SOCKET	This obtains a socket, as in Table 3.
(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 <i>bound</i> 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) SELECT	The SELECT 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 else will time out after an interval specified by one of the SELECT 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) RECVFROM	This reads one message from the client to determine what service is required. This message contains, at a minimum, the CICS transaction ID of the server.
(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.

Table 5. Calls for the Concurrent Server Application (continued)

(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 104, the parameters LSTN-NAME and LSTN-SUBNAME define the Listener.
(22) SELECT ⁸	Again, the SELECT call is used to monitor TCP/IP activity. This time, SELECT 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 70 shows the sequence of socket calls involved in a simple client-iterative server setup.

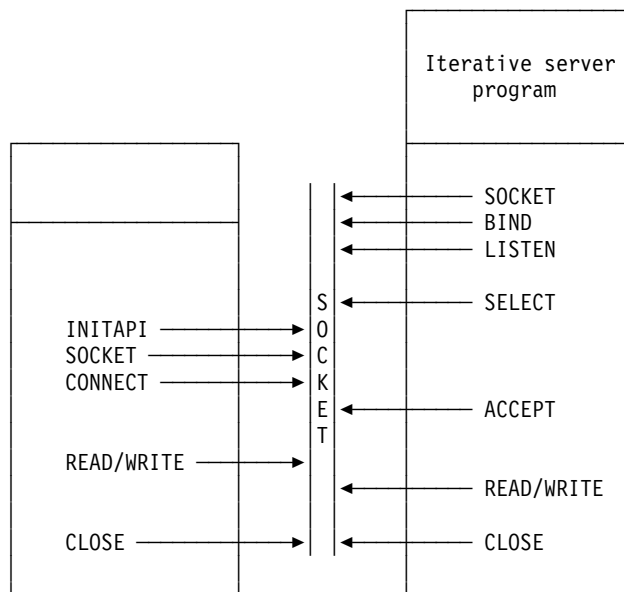


Figure 70. Sequence of Socket Calls with an Iterative Server

The setup with an iterative server is much simpler than the previous cases with concurrent servers.

8. This SELECT is the same as the SELECT call in Step 16. They are shown as two calls to clarify the functions being performed.

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

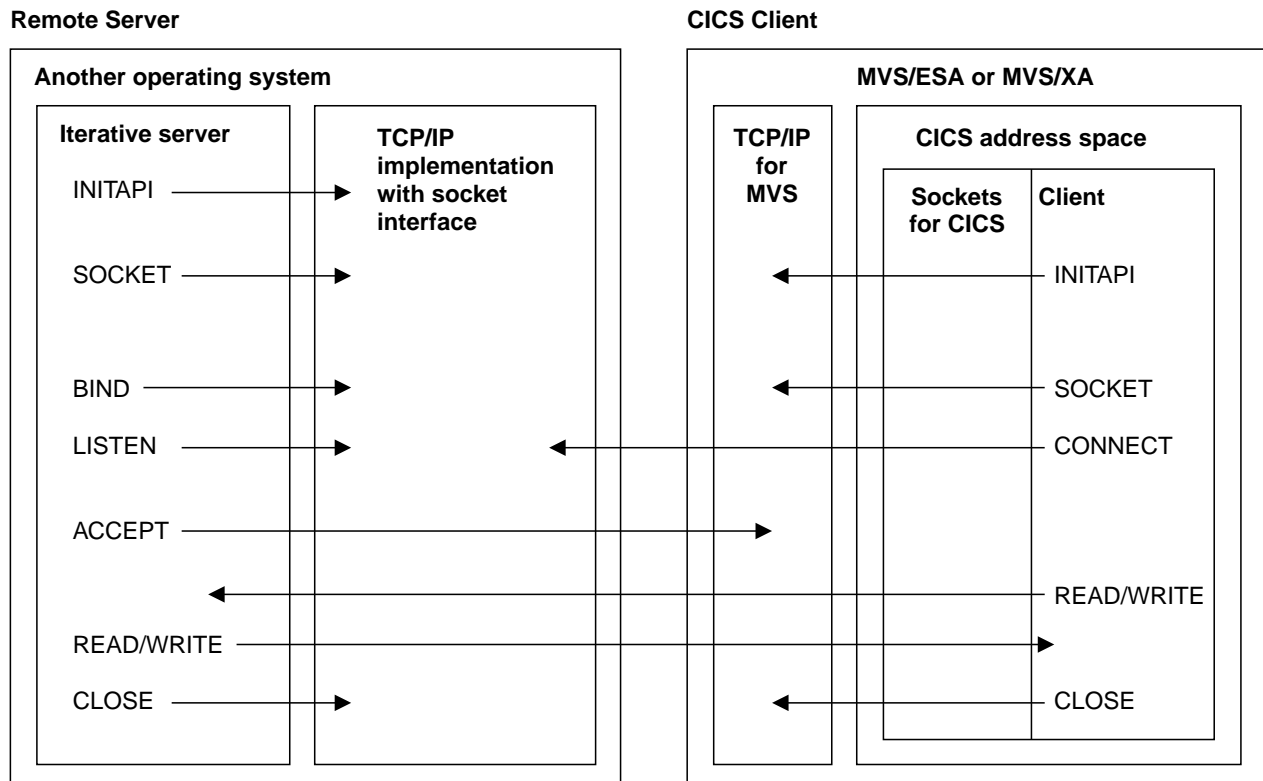


Figure 71. Sequence of Socket Calls between a CICS Client and a Remote Iterative Server

Figure 71 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 69 on page 94.

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 71, 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 71, 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 only one TCP/IP addressing family (or domain, as it is called in the UNIX system). This is the internet domain, denoted by AF_INET 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 for MVS address on a TCP/IP internet. Each address is a unique 32-bit 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 multi-homed 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

A socket address in an IP addressing family comprises 4 fields: the address family, an IP address, a port, and a character array (zeros), set as follows:

- The family field is set to AF_INET in C, or to 2 in other languages.
- The port field is the port used by the application, in network byte order (which is explained on page 101).

- The address field is the IP address of the network interface used by the application. It is also in network byte order.
- The character array field should always be set to all zeros.

For COBOL and Assembler Language Programs

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

Parameter	Assembler COBOL	
NAME		
STRUCTURE:		
FAMILY	H	PIC 9(4) BINARY
PORT	H	PIC 9(4) BINARY
ADDRESS	F	PIC 9(8) BINARY
ZEROS	XL8	PIC X(8)

For C Programs

The structure of an internet socket address is defined by the *sockaddr_in* structure, which is found in the IN.H header file. The format of this structure is shown in Table 9 on page 114.

MVS Address Spaces

Figure 72 shows the relationship between TCP/IP and CICS in terms of MVS address spaces.

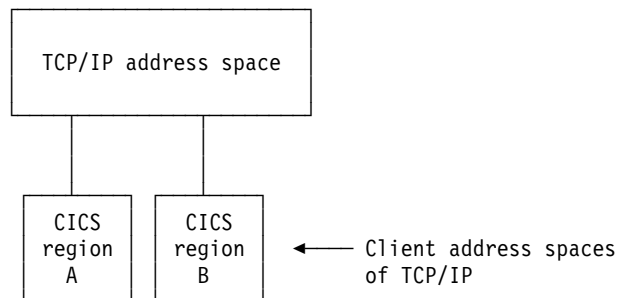


Figure 72. MVS Address Spaces

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!

The TCP/IP address space is capable of supporting a maximum of 5000 sockets; however, a single task can support 2000 sockets. Therefore, using multiple tasks, a single CICS region can support up to 5000 sockets.

The structure of CLIENTID is shown in Table 6. 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 6. CLIENTID Structures

C structure	COBOL structure
<pre>struct clientid { int domain; char name[8]; char subtaskname[8]; char reserved[20]; };</pre>	<pre>CLIENTID STRUCTURE: Domain PIC 9(8) BINARY Name PIC X(8) Task PIC X(8) Reserved PIC X(20)</pre>

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 IBM's 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 73 on page 102.

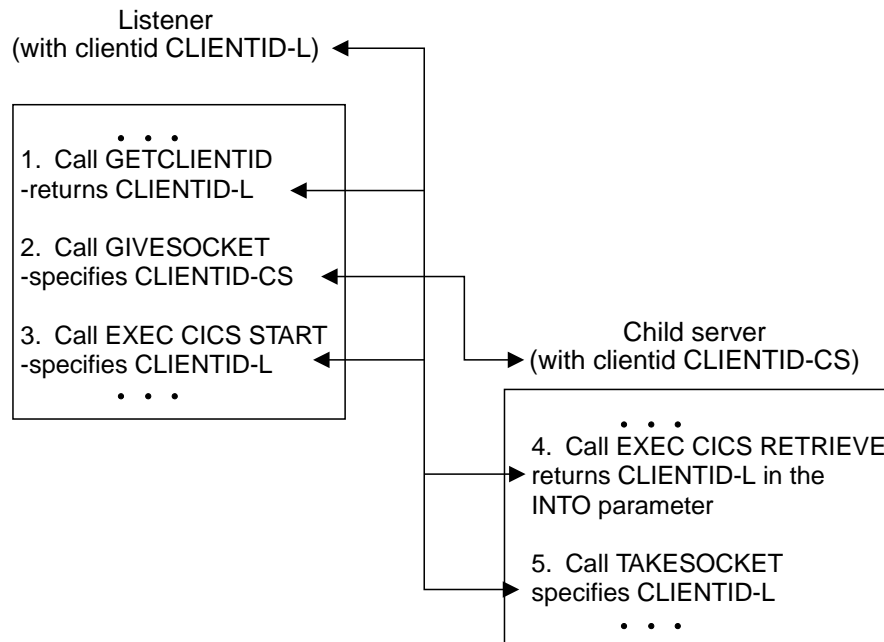


Figure 73. Transfer of CLIENTID Information

Figure 73 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 enters its own MVS address space name and sets the subtask identifier to blanks. This makes the socket available to a TAKESOCKET command from any task in the Listener's address space, but only the child server receives the socket descriptor number, so the exposure is minimal. For total integrity, the child server's subtask identifier 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 Listener is CSKL.

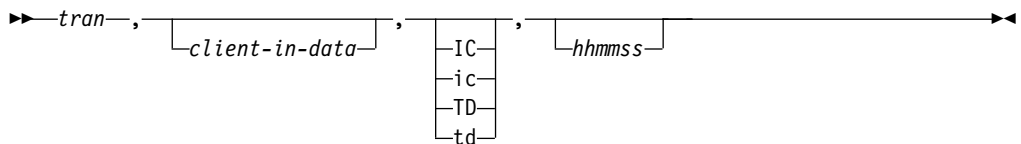
The Listener performs the following functions:

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*.
2. When an incoming connection request arrives, the Listener accepts it and obtains a new socket to pass to the CICS child server application program.
3. 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”.
4. 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 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

The CICS transaction ID (in uppercase) that the Listener is going to start. This field can be 1 to 4 characters.

client-in-data

Optional. Application data, used by the optional security exit ⁹ or the server transaction. The maximum length of this field is 35 characters.

9. (See “Writing Your Own Security Link Module for the Listener” on page 105)

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 6 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". 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.
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	<p>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.</p> <p>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.</p>
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

Table 7 on page 105 shows the format of the Listener output data area passed to the child server. The Listener program uses the following COBOL definition:

```
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.
```

15 SIN-FAMILY	PIC 9(4) COMP.
15 SIN-PORT	PIC 9(4) COMP.
15 SIN-ADDRESS	PIC 9(8) COMP.
15 SIN-ZERO	PIC X(8).

Table 7. Listener Output Format

Description	Format	Value
Socket descriptor	Fullword binary	The socket descriptor to be used by the child server in the TAKESOCKET command
MVS address space identifier	8-byte character	Name of the Listener's address space
TCP/IP task identifier	8-byte character	Listener's task identifier
Data area	35-byte character plus 1-byte filler	Client-in-data from Listener input received from the client
Socket address	Structure containing remaining 4 fields	See each field
TCP/IP addressing family	Halfword binary	2, indicating AF-INET
Port descriptor	Halfword binary	Descriptor of the port bound to the socket (Listener's port number from CSYS).
32-bit IP address	Fullword binary	IP address of the socket's host machine in network byte order
Unused	Doubleword	Binary zeros

Writing Your Own Security 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 a 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 module, you can name it anything you want, as long as you define it in the configuration dataset. (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 module in the SECexit field in Alter or Define. If you don't name the module, CICS will assume you don't have one. See Figure 47 on page 59 for more information.

Just before the task creation process, the Listener invokes the security 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 module should perform a security check and set the switch accordingly.

When the security 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.

Table 8 shows the data area used by the security module.

Table 8. Security Exit Data

Description	Format	Value
CICS transaction identifier	4-byte character	CICS transaction requested by the client
Data area	40-byte character	User data received from the client
Action	2-byte character	Method of starting the task: IC Interval control KC Task control TD Transient data
Interval control time	6-byte character	Interval requested for IC start Has the form <i>hhmmss</i>
Address family	Halfword binary	Network address family. A value of 2 must be set.
Port	Halfword binary	The port number of the requester's port.
Address	Fullword binary	The IP address of the requester's host.
Switch	1-byte character	Switch: 1 Permit the transaction Not 1 Prohibit the transaction
Switch-2	1-byte character	Switch: 1 Listener sends message to Client. Not 1 Security Exit program sends message to client.
Terminal identification	4-byte character	LOW-VALUES and binary zeros if a CICS terminal is not associated with the new task. CICS terminal identifier if a CICS terminal is associated with the new task.
Socket descriptor	Halfword binary	Current socket descriptor
User ID	8-byte character	Users who have CICS/ESA V4 and later can return a USERID value which is used in starting the server transaction. This enables the CICS V4+ security features.

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 for MVS for:

- Converting data from EBCDIC to ASCII and back, when sending and receiving data to and from the TCP/IP network, with the SEND, RECEIVE, READ, and WRITE calls.
- Converting between bit arrays and character strings when using the SELECT call.

For details of these routines, refer to EZACIC04, EZACIC05, and EZACIC06 in “Chapter 8. Sockets Extended Application Programming Interface (API)” on page 145.

Chapter 7. C Language Application Programming

Overview	111
C Socket Library	111
C/370 Compilation	111
Structures Used in Socket Calls	114
The errno Variable	115
C Socket Calls	115
accept()	115
Format	116
Parameters	116
Return Values	116
bind()	117
Format	117
Parameters	117
Return Values	118
close().	118
Format	118
Parameter	118
Return Values	118
connect().	119
Format	119
Parameters	119
Return Values	119
fcntl()	120
Format	120
Parameters	121
Return Values	121
getclientid()	121
Format	121
Parameters	121
Return Values	122
gethostid()	122
Format	122
Parameters	122
Return Values	122
gethostname().	122
Format	122
Parameters	122
Return Values	123
getpeername().	123
Format	123
Parameters	123
Return Values	123
getsockname().	124
Format	124
Parameters	124
Return Values	124
getsockopt(), setsockopt()	124
Format	125
Parameters	125
Possible Entries for optname	126
Return Values	127
givesocket().	127
Format	128

Parameters	128
Return Values	128
initapi()	128
Format	129
Parameters	129
Return Values	129
ioctl()	129
Format	129
Parameters	130
Return Values	131
listen().	131
Format	132
Parameters	132
Return Values	132
read()	132
Format	132
Parameters	132
Return Values	132
recv()	133
Format	133
Parameters	133
Return Values	133
recvfrom()	134
Format	134
Parameters	134
Return Values	134
select()	135
Defining which sockets to test	135
send()	137
Format	137
Parameters	138
Return Values	138
sendto()	138
Format	139
Parameters	139
Return Values	139
setsockopt()	139
shutdown()	140
Format	140
Parameters	140
Return Values	140
socket()	140
Format	141
Parameters	141
Return Values	141
takesocket()	141
Format	141
Parameters	142
Return Values	142
write()	142
Format	142
Parameters	142

Overview

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/370 Compilation” shows how to compile a C/370* program that contains calls to Sockets for CICS.
- “Structures Used in Socket Calls” on page 114 lists data structures used in C language socket calls.
- “The errno Variable” on page 115 describes the use of a global variable used by the socket system to report errors.
- “C Socket Calls” on page 115 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 have these header files:

```

cics.h          manifest.h
fcntl.h        netdb.h
if.h           rtroute.h
in.h          saiucv.h
inet.h        socket.h
ioctl.h       uio.h
              bsdtypes.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 3 of “C/370 Compilation”). These files carry a .h extension in this text to distinguish them as header files.

In the IBM implementation, you must include *manifest.h* to remap the socket function long names to 8-character names. To reference *manifest.h*, you should also include the following statement as the first `#include` at the beginning of each program:

```
#include <manifest.h>
```

C/370 Compilation

To compile a C/370 program that contains calls to CICS TCP/IP, you must change the standard procedure for C/370 compilation provided with CICS. Figure 74 on page 113 shows a sample job for the compilation of a C/370 program that contains calls to CICS TCP/IP. It includes the following modifications:

- ¶ The prototyping statement is required for CICS.

- If you compile with TCP/IP V3R2 headers and you plan to linkedit with TCP/IP V3R1 for MVS C socket libraries, you **must** specify this during a C compile to get the older macro versions of the functions HTONL, HTONS, NTOHL and NTOHS. Failure to do so will result in unresolved references at linkedit time.
- **2** in the C step (running the C/370 compiler) you must concatenate the *hlq.SEZACMAC* data set to the SYSLIB DD.
- **3** in the LKED step you must concatenate the following data sets to the SYSLIB DD:
 - *hlq.SEZATCP*
 - *hlq.SEZACMTX*
- **4** also in the LKED step you must add an INCLUDE for the module EZACIC07 to the SYSIN DD *.

Notes:

1. Further to Step 4 above, Sockets for CICS application programs must include EZACIC07 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. For more information about compiling and linking, see *OS/390 C/C++ User's Guide* and *TCP/IP for MVS: Application Programming Interface Reference* .

```

//CICRS1C JOB (999,POK),'CICRS1',NOTIFY=CICRS1,
// CLASS=A,MSGCLASS=T,TIME=1439,
// REGION=5000K,MSGLEVEL=(1,1)
//DFHEITDL PROC SUFFIX=1$,
// INDEX='CICS330',
// INDEX2='CICS330',
//CPARM='DEFINE(MVS,_TCP31_PROTOS) ',
//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
// DSN=&COMHD..&COMVER..SIBMLINK,DISP=SHR
// DSN=&VSCCHD..&CVER..SEDCCOMP,DISP=SHR
//SYMSGS DD DSN=&VSCCHD..&CVER..SEDCMSG(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=h1q.SEZACMAC,DISP=SHR
//SYSLIN DD DSN=&&LOAD,DISP=(,PASS),
// UNIT=&WORK,SPACE=&WRKSPC,DCB=&DCB80
//SYSPRINT DD SYSOUT=&OUTC
//SYSCPRT DD SYSOUT=&OUTC
//SYSTEM 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)),
// REGION=&REG,PARM='&PPARM'
//*
//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=h1q.SEZATCP,DISP=SHR
// DD DSN=h1q.SEZACMTX,DISP=SHR
//SYSLIN DD DSN=*.PLKED.SYSMOD,DISP=(OLD,DELETE)
// DD DSN=*.COPYLINK.SYSUT2,DISP=(OLD,DELETE)
// DD DDNAME=SYSIN
//SYSLMOD DD DSN=CICRS2.CICS330.PGMLIB,DISP=SHR
//*RESLIB DD DSN=&IMSIND..RESLIB,DISP=SHR
//SYSUT1 DD DSN=&&SYSUT1L,DISP=(,PASS),
// UNIT=&WORK,SPACE=&WRKSPC,DCB=&DCB80

```

Figure 74. Modified JCL for C/370 Compilation (Part 1 of 2)

```

//SYSPRINT DD SYSOUT=&OUTC
// PEND
//APPLPROG EXEC DFHEITDL
//TRN.SYSIN DD DISP=SHR,DSN=CICRSR1.JCL.DATA(SICUCCLD)
//LKED.SYSIN DD *
  INCLUDE SYSLIB(EZACIC07)
  NAME SICUCCLD(R)
/*

```

Figure 74. Modified JCL for C/370 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 9 shows the structures used by the calls described in this chapter.

Table 9. C Structures

C Structure	Format
clientid Used in many calls	<pre> struct clientid { int domain; char name[8]; char subtaskname[8]; char reserved[20]; }; </pre>
ifconf Used in the ioctl() call only	<pre> struct ifconf { int ifc_len; union { caddr_t ifcu_buf; struct ifreq *ifcu_req; } ifc_ifcu; }; </pre>
ifreq Used in the ioctl() call only	<pre> 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; }; </pre>
linger Used in the get/setsockopt() calls only	<pre> struct linger { int l_onoff; int l_linger; }; </pre>

Table 9. C Structures (continued)

C Structure	Format
sockaddr_in Used in many calls	<pre> struct in_addr { unsigned long s_addr; }; struct sockaddr_in { short sin_family; ushort sin_port; struct in_addr sin_addr; char sin_zero[8]; }; </pre>
timeval Used in the select() call only	<pre> struct timeval { long tv_sec; long tv_usec; }; </pre>

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 the include statement:

```
#include <ezacichd.h>
```

Note: Do not include `<errno.h>` in the module and do not use `tcperror()`.

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 *TCP/IP for MVS: 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.

accept()

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 pre-determine 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>
#include <bsdtypes.h>
#include <in.h>
#include <socket.h>

int accept(int s, struct sockaddr_in *name, [cont]
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_in` 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`, as defined in the header file `in.h`. The format of the structure is shown in Table 9 on page 114 .

namelen

The size, in bytes, of the buffer pointed to by `name`.

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:

errno Value

Description

EBADF

The `s` parameter is not a valid socket descriptor.

EFAULT

Using `addr` and `addrlen` 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.

Format

```
#include <manifest.h>
#include <bsdtypes.h>
#include <socket.h>
#include <in.h>
```

```
int bind(int s, struct sockaddr_in *name, [cont]
int namelen)
```

Parameters

s The socket descriptor returned by a previous socket() call.

name The pointer to a *sockaddr_in* structure containing the name that is to be bound to *s*. The format of the name buffer is expected to be *sockaddr_in*, as defined in the header file *in.h*. The format of the structure is shown in Table 9 on page 114.

The *sin_family* field must be set to AF_INET.

The *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.

The *in_addr.s_addr* field is set to the 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.

The *sin_zero* field is not used and must be set to all zeros.

bind()

namelen

The size, in bytes, of the buffer pointed to by *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:

errno Value

Description

EADDRINUSE

The address is already in use. See the SO_REUSEADDR option described under “getsockopt(), setsockopt()” on page 124 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).

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>
#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:

errno Value

Description

close()

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>
#include <bsdtypes.h>
#include <socket.h>
#include <in.h>
```

```
int connect(int s, struct sockaddr_in *name, [cont]
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*, as defined in the header file *in.h*. The format of the structure is shown in Table 9 on page 114

The *sin_family* field must be set to `AF_INET`. The *sin_port* field is set to the port to which the server is bound. It must be specified in network byte order. The *sin_zero* field is not used and must be set to all zeros.

namelen

The size of the *socket address* pointed to by *name* in bytes.

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:

connect()

errno Value	Description
EADDRNOTAVAIL	The calling host cannot reach the specified destination.
EAFNOSUPPORT	The address family is not supported.
EALREADY	The socket <i>s</i> is marked nonblocking, and a previous connection attempt has not completed.
EBADF	The <i>s</i> parameter is not a valid socket descriptor.
ECONNREFUSED	The connection request was rejected by the destination host.
EFAULT	Using <i>name</i> and <i>namelen</i> 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 <i>s</i> is marked nonblocking, and the connection cannot be completed immediately. The EINPROGRESS value does not indicate an error condition.
EINVAL	The <i>namelen</i> parameter is not a valid length.
EISCONN	The socket <i>s</i> 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>
#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:

errno Value

Description

EBADF

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

EINVAL

The *arg* parameter is not a valid flag.

getclientid()

A `getclientid()` call returns the identifier by which the calling application is known to the TCPIP 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>
#include <bsdtypes.h>
#include <socket.h>
```

```
int getclientid(int domain, struct clientid)
```

Parameters

domain

The *domain* must be `AF_INET`.

getclientid()

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:

errno Value

Description

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

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>
#include <socket.h>

unsigned long gethostid()
```

Parameters

The `gethostid()` call has no parameters.

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>
#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:

errno Value

Description

EFAULT

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

getpeername()

The `getpeername()` call returns the name of the peer connected to a specified socket.

Format

```
#include <manifest.h>
#include <socket.h>
#include <bsdtypes.h>
```

```
int getpeername(int s, struct sockaddr *name, [cont]
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.

namelen

A pointer to the structure containing the size of the address structure pointed to by *name* in bytes.

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:

errno Value

Description

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 socket would cause the name to point to a `sockaddr` structure with the `sin_family` field set to `AF_INET` 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>
#include <socket.h>
#include <bsdtypes.h>
#include <in.h>

int getsockname(int s, [cont]
struct sockaddr_in *name, int *namelen)
```

Parameters

`s` The socket descriptor.

`name` The address of the buffer into which `getsockname()` copies the name of `s`.

`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`.

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:

errno Value

Description

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 socket level:

- The ability to broadcast messages (UDP socket only).
- The ability to toggle the TCP keep-alive mechanism for a stream socket.
- Linger on close if data is present.
- Reception of out-of-band data.
- Local address reuse.

As well as checking current options, `getsockopt()` can return pending errors and the type of socket.

Format

```
#include <manifest.h>
#include <socket.h>
#include <bsdtypes.h>

int getsockopt(int s, int level, [cont]
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:

- `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*. To manipulate options at any other level, such as the TCP or IP level, supply the appropriate protocol number for the protocol controlling the option. Currently, only the `SOL_SOCKET` level is 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 126.

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*.

getsockopt(),setsockopt()

This structure is defined in the following example:

```
#include <manifest.h>
struct linger
{
    int    l_onoff;           /* option on/off */
    int    l_linger;        /* linger time */
};
```

The *l_onoff* field is set to zero if the SO_LINGER option is being disabled. A nonzero value enables the option. The *l_linger* field specifies the amount of time to linger on close. The units of *l_linger* are seconds.

Possible Entries for optname

The following options are recognized at the 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_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 TCPIP address space waits to try to send the data. Although the data transfer is usually successful, it cannot be guaranteed, because the TCPIP address space waits a finite amount of time trying to send the data. The close() call returns without blocking the caller.

SO_OOINLINE

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.

SO_SNDBUF

(Applies to getsockopt() only.) Returns the size of the data portion of the

getsockopt(),setsockopt()

TCP/IP send buffer in *optval*. The size of the data portion of the send buffer is protocol-specific, based on the DATABUFFERPOOLSIZ 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:

errno Value

Description

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.

ENOPROTOPT

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().

givesocket()

Format

```
#include <manifest.h>
#include <bsdtypes.h>
#include <socket.h>
int givesocket(int s, struct 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

AF_INET (2).

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:

errno Value

Description

EBADF

The *s* parameter is not a valid socket descriptor or the socket has already been given or the socket domain is not AF_INET.

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.

initapi()

The *initapi()* call connects your application to the TCP/IP interface.

Format

```
#include <manifest.h>
int initapi(int max_sock, char *subtaskid)
```

Parameters

max_sock

The maximum number of sockets requested.

subtaskid

A unique 8-character ID, which should be the 4-byte packed EIBTASKN value in the EIB plus three character 0's and a unique displayable character.

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.
- Add a routing table entry.
- Query whether the current location in the data input is pointing to out-of-band data.
- Delete a routing table entry.
- Get the network interface address.
- Get the network interface broadcast address.
- Get the network interface configuration.
- Get the network interface destination address.
- Get the network interface flags.
- Get the network interface routing metric.
- Get the network interface network mask.
- Set the network interface routing metric.

Format

```
#include <manifest.h>
#include <bsdtypes.h>
#include <ioctl.h>
#include <rtrouteh.h>
#include <if.h>

int ioctl(int s, unsigned long cmd, [cont]
char *arg)
```

ioctl()

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 non-blocking 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.

SIOCADDRT

Adds a routing table entry. *arg* is a pointer to a *rtenry* structure, as defined in *rtroute.h*. The routing table entry, passed as an argument, is added to the routing tables.

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.

SIOCDELRT

Deletes a routing table entry. The *arg* parameter is a pointer to a *rtenry* structure, as defined in *rtroute.h*. If it exists, the routing table entry passed as an argument is deleted from the routing tables.

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.

SIOCGIFFLAGS

Gets the network interface flags. *arg* is a pointer to an *ifreq* structure, as defined in *if.h*. The interface flags are returned in the argument.

SIOCGIFMETRIC

Gets the network interface routing metric. The *arg* parameter is a pointer to an *ifreq* structure, as defined in *if.h*. The interface routing metric is returned in the argument.

SIOCGIFNETMASK

Gets the network interface network mask. The *arg* parameter is a pointer to an *ifreq* structure, as defined in *if.h*. The interface network mask is returned in the argument.

SIOCSIFDSTADDR

Sets the network interface destination address.

SIOCSIFFLAGS

Sets the network interface flags.

SIOCSIFMETRIC

Sets the network interface routing metric. The *arg* parameter is a pointer to an *ifreq* structure, as defined in *if.h*. Set the interface routing metric to the value passed 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:

errno Value**Description****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 4 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.

listen()

Format

```
#include <manifest.h>
#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:

errno Value

Description

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 1 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>

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:

errno Value

Description

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>
#include <bsdtypes.h>
#include <socket.h>
```

```
int recvfrom(int s, char *buf, [cont]
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:

errno Value**Description****EBADF**

s is not a valid socket descriptor.

recv()

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>
#include <bsdtypes.h>
#include <socket.h>

int recvfrom(int s, char *buf, [cont]
int len, int flags,
struct sockaddr_in *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.

namelen

A pointer to an integer containing the size of *name* in bytes.

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:

errno Value**Description****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 left-most bit represents socket 31, etc. 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.

select()

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, `RETCODE` is set to 0.

Format:

```
#include <manifest.h>
#include <socket.h>
#include <bsdtypes.h>
#include <bsdtime.h>

int select(int nfd, fd_set *readfds, [cont]
fd_set *writefds, fd_set *exceptfds,
struct timeval *timeout)
```

Parameters:

nfd The number of socket descriptors to check.

readfds

The pointer to a bit mask of descriptors to check for reading.

writelfds

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 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 zero indicates an expired time limit. The three bit masks indicate status (with one bit for each socket). A 1-bit indicates that the respective socket is ready; a 0-bit indicates that the respective socket is not ready. You can use the macro `FD_ISSET`¹⁰ with each socket to test its status.

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

errno Value**Description****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>
#include <bsdtypes.h>
#include <socket.h>

int send(int s, char *msg, [cont]
int len, int flags)
```

¹⁰ See *TCP/IP for MVS: Programmer's Reference* for details.

send()

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:

errno Value

Description

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>
#include <bsdtypes.h>
#include <socket.h>

int sendto(int s, char *msg, [cont]
int len, int flags,
struct sockaddr_in *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.
- tolen* The size of the structure pointed to by *to*.

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:

errno Value

Description

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 124.

shutdown()

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

Format

```
#include <manifest.h>
#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:

errno Value

Description

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. Transmission reliability is not guaranteed. Datagrams can be corrupted, received out of order, lost, or delivered multiple times.

Sockets are deallocated with the close() call.

Format

```
#include <manifest.h>
#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 family supported by CICS TCP/IP is AF_INET, which is the internet domain. The AF_INET 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:

errno Value

Description

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>
#include <bsdtypes.h>
#include <socket.h>
```

```
int takesocket(struct clientid *client_id, [cont]
int hisdesc)
```

takesocket()

Parameters

clientid

A pointer to the *clientid* of the application from which you are taking a socket.

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:

errno Value

Description

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 TCPIP address space.

EPFNOSUPPORT

The domain field of the *clientid* parameter is not AF_INET.

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>
#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:

errno Value

Description

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.

write()

Chapter 8. Sockets Extended Application Programming Interface (API)

CALL Instruction Application Programming Interface (API)	146
Call Formats	146
COBOL language call format	146
Assembler language call format	146
PL/I language call format	147
Programming Language Conversions	147
Error Messages and Return Codes	147
CALL Instructions for Assembler, PL/I, and COBOL Programs	148
ACCEPT	148
BIND	149
CLOSE	151
CONNECT	152
FCNTL	154
GETCLIENTID	155
GETHOSTBYADDR	156
GETHOSTBYNAME	158
GETHOSTID	160
GETHOSTNAME	160
GETPEERNAME	161
GETSOCKNAME	162
GETSOCKOPT	163
GIVESOCKET	165
INITAPI	167
IOCTL	169
LISTEN	173
READ	174
READV	175
RECV	176
RECVFROM	178
RECVMSG	180
SELECT	183
SELECTEX	187
SEND	189
SENDMSG	190
SENDTO	193
SETSOCKOPT	195
SHUTDOWN	197
SOCKET	198
TAKESOCKET	199
TERMAPI	200
WRITE	200
WRITEV	201
Data Translation Programs for the Socket Call Interface	203
Data Translation	203

CALL Instruction Application Programming Interface (API)

This chapter describes the CALL instruction API for TCP/IP for MVS 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, see *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. A C run-time library is required when you use the GETHOSTBYADDR or GETHOSTBYNAME call.

Call Formats

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

COBOL language call format

►►—CALL 'EZASOKET' USING SOC-FUNCTION—*parm1*, *parm2*, ...—ERRNO RETCODE.—◄◄

SOC-FUNCTION

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

parm*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.

Assembler language call format

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

PARMLIST

CALL EZASOKET,(SOC-FUNCTION,—*parm1*, *parm2*, ...—ERRNO RETCODE),VL,MF=(E, 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.

PL/I language call format

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

SOC-FUNCTION

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

parm*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 `tcpcerror()` 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.

Programming Language Conversions

The parameter descriptions in this chapter are written using the VS COBOL II PIC language syntax and conventions. Use the syntax and conventions that are appropriate for the language you want to use. The following are examples of storage definition statements for COBOL, PL/I, and assembler language programs.

VS COBOL II PIC

PIC S9(4) BINARY	HALFWORD BINARY VALUE
PIC S9(8) BINARY	FULLWORD BINARY VALUE
PIC X(n)	CHARACTER FIELD OF N BYTES

COBOL PIC

PIC S9(4) COMP	HALFWORD BINARY VALUE
PIC S9(8) COMP	FULLWORD BINARY VALUE
PIC X(n)	CHARACTER FIELD OF N BYTES

PL/I DECLARE STATEMENT

DCL HALF	FIXED BIN(15),	HALFWORD BINARY VALUE
DCL FULL	FIXED BIN(31),	FULLWORD BINARY VALUE
DCL CHARACTER	CHAR(n)	CHARACTER FIELD OF n BYTES

ASSEMBLER DECLARATION

DS H	HALFWORD BINARY VALUE
DS F	FULLWORD BINARY VALUE
DS CLn	CHARACTER FIELD OF n BYTES

Error Messages and Return Codes

For information about error messages, see *TCP/IP for MVS: Messages and Codes*.

For information about error codes that are returned by TCP/IP, see “Sockets Extended Return Codes” on page 263.

CALL Instructions for Assembler, PL/I, and COBOL Programs

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 socket (s) remains 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.

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

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

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

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 A socket address structure that contains the client's socket address.

FAMILY

A halfword binary field specifying the addressing family. The call returns the value 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 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.

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See "Sockets Extended Return Codes" on page 263, 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.

In the AF_INET 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 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 ADDRESS field to a fullword of zeros.

BIND

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

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

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

Parameter Values Set by the Application:

SOC-FUNCTION

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

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

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

FAMILY

A halfword binary field specifying the addressing family. The value is always set to 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 zero when the call is issued, the system assigns the port number for the socket. The application can call the GETSOCKNAME macro after the BIND macro 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 8-byte character field that is required but not used.

Parameter Values Returned to the Application:

ERRNO

A fullword binary field. If RETCODE is negative, this field contains an error number. See “Sockets Extended Return Codes” on page 263, 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 197 for more information.

Notes:

1. If a stream socket is closed while input or output data is queued, the TCP connection is reset and data transmission may be incomplete. The 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 195.
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.

```
WORKING STORAGE
  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.
```

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

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

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

CLOSE

number. See “Sockets Extended Return Codes” on page 263, 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:

- The *server* issues BIND and LISTEN to create a passive open socket.
- The *client* issues CONNECT to request the connection.
- 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 zero RETCODE indicates that the connection was completed.
 - A nonzero RETCODE with an ERRNO of 36 (EINPROGRESS) indicates that the connection is not completed but since the socket is nonblocking, the CONNECT call returns normally.

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

The completion cannot be checked by issuing a second CONNECT. For more information, see “SELECT” on page 183.

```

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

```

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

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

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 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 2 for AF_INET.

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

IP-ADDRESS

A fullword binary field that is set to the 32-bit 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 hex.

RESERVED

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

Parameter Values Returned to the Application:

ERRNO

A fullword binary field. If RETCODE is negative, this field contains an error number. See “Sockets Extended Return Codes” on page 263, 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 169 for another way to control a socket’s blocking mode.

```
WORKING STORAGE
  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
CALL 'EZASOKET' USING SOC-FUNCTION S COMMAND REQARG
ERRNO RETCODE.
```

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

Parameter Values Set by the Application:

SOC-FUNCTION

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

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

COMMAND

A fullword binary number with the following values.

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 “Sockets Extended Return Codes” on page 263, 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).

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 165 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.

```

WORKING STORAGE
  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
  CALL 'EZASOKET' USING SOC-FUNCTION CLIENT ERRNO RETCODE.

```

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

Parameter Values Set by the Application:**SOC-FUNCTION**

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

Parameter Values Returned to the Application:**CLIENT**

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

DOMAIN

A fullword binary number specifying the caller's domain. For TCP/IP the value is set to 2 for AF_INET.

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.

GETCLIENTID

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See “Sockets Extended Return Codes” on page 263, 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.

Note: The C runtime libraries are required when GETHOSTBYADDR is issued by your program. For CICS, the C runtime library must be included in the link list.

```
WORKING STORAGE
  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
  CALL 'EZASOKET' USING SOC-FUNCTION HOSTADDR HOSTENT RETCODE.
```

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

Parameter Values Set by the Application:

SOC-FUNCTION

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

HOSTADDR

A fullword binary field set to the internet address (specified in network byte order) of the host whose name is being sought.

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

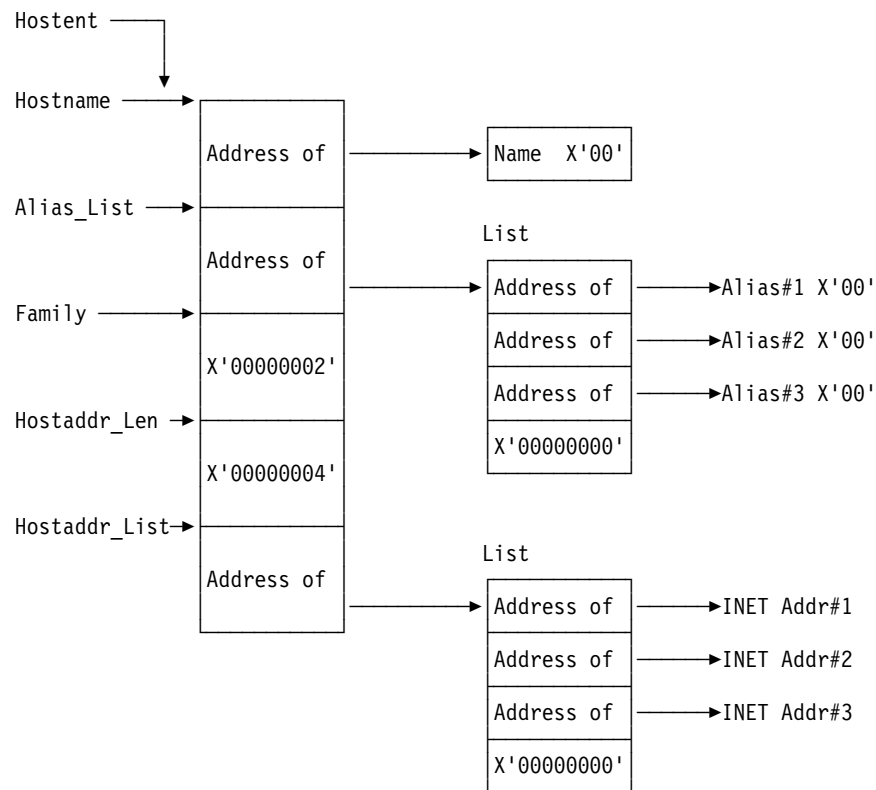


Figure 75. HOSTENT Structure Returned by the GETHOSTBYADDR Call

GETHOSTBYADDR returns the HOSTENT structure shown in Figure 75. This structure contains:

- The address of the host name that is returned by the call. The name length is variable and is ended by X'00'.
- The address of a list of addresses that point to the alias names returned by the call. This list is ended by the pointer X'00000000'. Each alias name is a variable length field ended by X'00'.
- The value returned in the FAMILY field is always 2 for AF_INET.
- The length of the host internet address returned in the HOSTADDR_LEN field is always 4 for AF_INET.
- The address of a list of addresses that point to the host internet addresses returned by the call. The list is ended by the pointer X'00000000'. If the call cannot be resolved, the HOSTENT structure contains the ERRNO 10214.

The HOSTENT structure uses indirect addressing to return a variable number of alias names and internet addresses. If you are coding in PL/I or assembler language, this structure can be processed in a relatively straight-forward manner. If you are coding in COBOL, this structure may be difficult to interpret. You can use the subroutine EZACIC08 to simplify interpretation of the information returned by the GETHOSTBYADDR and GETHOSTBYNAME calls. For more information about EZACIC08, see "EZACIC08" on page 208.

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.

TCP/IP tries to resolve the host name through a name server, if one is present. If a name server is not present, the system searches the HOSTS.SITEINFO data set until a matching host name is found or until an EOF marker is reached.

Notes:

1. HOSTS.LOCAL, HOSTS.ADDRINFO, and HOSTS.SITEINFO are described in *TCP/IP for MVS: Customization and Administration Guide*.
2. The C runtime libraries are required when GETHOSTBYNAME is issued by your program. For CICS, the C runtime library must be included in the link list.

```
WORKING STORAGE
  01 SOC-FUNCTION    PIC X(16) VALUE IS 'GETHOSTBYNAME'.
  01 NAMELEN        PIC 9(8)  BINARY.
  01 NAME           PIC X(24).
  01 HOSTENT        PIC 9(8)  BINARY.
  01 RETCODE        PIC S9(8) BINARY.
```

```
PROCEDURE
  CALL 'EZASOKET' USING SOC-FUNCTION NAMELEN NAME
                      HOSTENT RETCODE.
```

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

Parameter Values Set by the Application:

SOC-FUNCTION

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

NAMELEN

A value set to the length of the host name.

NAME A character string, up to 24 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
-----------	-------------------

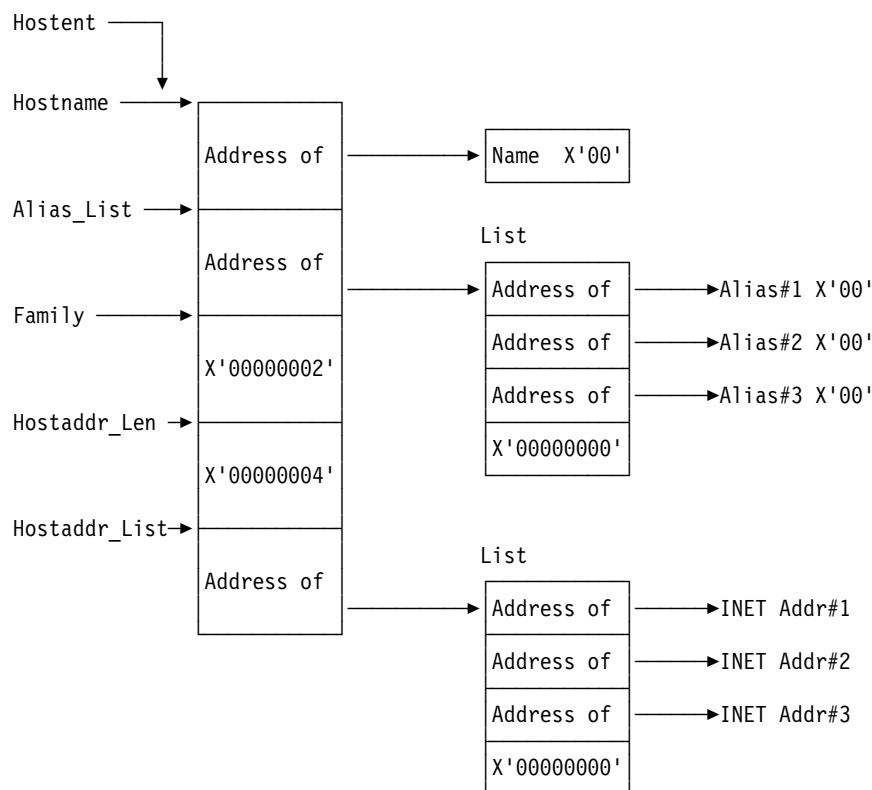


Figure 76. HOSTENT Structure Returned by the GETHOSTBYNAME Call

GETHOSTBYNAME returns the HOSTENT structure shown in Figure 76. This structure contains:

- The address of the host name that is returned by the call. The name length is variable and is ended by X'00'.
- The address of a list of addresses that point to the alias names returned by the call. This list is ended by the pointer X'00000000'. Each alias name is a variable length field ended by X'00'.
- The value returned in the FAMILY field is always 2 for AF_INET.
- The length of the host internet address returned in the HOSTADDR_LEN field is always 4 for AF_INET.
- The address of a list of addresses that point to the host internet addresses returned by the call. The list is ended by the pointer X'00000000'. If the call cannot be resolved, the HOSTENT structure contains the ERRNO 10214.

The HOSTENT structure uses indirect addressing to return a variable number of alias names and internet addresses. If you are coding in PL/I or assembler language, this structure can be processed in a relatively straight-forward manner. If you are coding in COBOL, this structure may be difficult to interpret. You can use the subroutine EZACIC08 to simplify interpretation of the information returned by the GETHOSTBYADDR and GETHOSTBYNAME calls. For more information about EZACIC08, see "EZACIC08" on page 208.

GETHOSTID

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

```
WORKING STORAGE
  01 SOC-FUNCTION    PIC X(16) VALUE IS 'GETHOSTID'.
  01 RETCODE        PIC S9(8) BINARY.

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

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

SOC-FUNCTION

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

RETCODE

Returns a fullword binary field containing the 32-bit internet address of the host. There is no ERRNO parameter for this call.

GETHOSTNAME

The GETHOSTNAME call returns the domain name of the local host.

```
WORKING STORAGE
  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
  CALL 'EZASOKET' USING SOC-FUNCTION NAMELEN NAME
                    ERRNO RETCODE.
```

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

Parameter Values Set by the Application:

SOC-FUNCTION

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

NAMELEN

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

Parameter Values Returned to the Application:

NAMELEN

A fullword binary field set to the length of the host name.

NAME Indicates the receiving field for the host name. TCP/IP for MVS 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

GETHOSTNAME

number. See “Sockets Extended Return Codes” on page 263, 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.

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

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

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

Parameter Values Set by the Application:

SOC-FUNCTION

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

S A halfword binary number set to the socket descriptor of the local socket connected to the remote peer whose address is required.

Parameter Values Returned to the Application:

NAME A 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 addressing family. The call always returns the 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 internet address of the connection peer's host machine.

RESERVED

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

GETPEERNAME

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See “Sockets Extended Return Codes” on page 263, 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 zero.

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.

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

PROCEDURE

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

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

Parameter Values Set by the Application:

SOC-FUNCTION

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

S A halfword binary number set to the descriptor of local socket whose address is required.

Parameter Values Returned to the Application:

NAME Specifies the socket address structure returned by the call.

FAMILY

A halfword binary field containing the addressing family. The call always returns the value 2, indicating AF_INET.

PORT A halfword binary field set to the port number bound to this socket. If the socket is not bound, 0 is returned.

IP-ADDRESS

A fullword binary field set to the 32-bit internet address of the local host machine.

RESERVED

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

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See “Sockets Extended Return Codes” on page 263, 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.

```

WORKING STORAGE
01 SOC-FUNCTION    PIC X(16) VALUE IS 'GETSOCKOPT'.
01 S              PIC 9(4) BINARY.
01 OPTNAME        PIC 9(8) BINARY.
   88 SO-REUSEADDR VALUE 4.
   88 SO-KEEPALIVE VALUE 8.
   88 SO-BROADCAST VALUE 32.
   88 SO-LINGER    VALUE 128.
   88 SO-OOBINLINE VALUE 256.
   88 SO-SNDBUF   VALUE 4097.
   88 SO-ERROR    VALUE 4103.
   88 SO-TYPE     VALUE 4104.
01 OPTVAL        PIC X(16) BINARY.
01 OPTLEN        PIC 9(8) BINARY.
01 ERRNO         PIC 9(8) BINARY.
01 RETCODE       PIC S9(8) BINARY.

```

```

PROCEDURE
CALL 'EZASOKET' USING SOC-FUNCTION S OPTNAME
OPTVAL OPTLEN ERRNO RETCODE.

```

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

Parameter Values Set by the Application:**SOC-FUNCTION**

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

S A halfword binary number specifying the socket descriptor for the socket requiring options.

GETSOCKOPT

OPTNAME

Set OPTNAME to the required option before you issue GETSOCKOPT. The option are as follows:

SO-REUSEADDR

Returns the status of local address reuse. When enabled, this option allows local addresses that are already in use to be bound. Instead of checking at BIND time (the normal algorithm) the system checks at CONNECT time to ensure that the local address and port do not have the same remote address and port. If the association already exists, Error 48 (EADDRINUSE) is returned when the CONNECT is issued.

SO-BROADCAST

Requests the status of the broadcast option, which is the ability to send broadcast messages. This option has no meaning for stream sockets.

SO-KEEPALIVE

Requests the status of the TCP keep-alive mechanism for a stream socket. When activated, the keep-alive mechanism periodically sends a packet on an otherwise idle connection. If the remote TCP does not respond to the packet or to retransmissions of the packet, the connection is terminated with the error ETIMEDOUT.

SO-LINGER

Requests the status of LINGER.

- When the LINGER option has been enabled, and data transmission has not been completed, a CLOSE call blocks the calling program until the data is transmitted or until the connection has timed out.
- If LINGER is not enabled, a CLOSE call returns without blocking the caller. TCP/IP attempts to send the data; although the data transfer is usually successful, it cannot be guaranteed, because TCP/IP only attempts to send the data for a specified amount of time.

SO-OOBINLINE

Requests the status of how out-of-band data is to be received. This option has meaning only for stream sockets.

- When this option is enabled, out-of-band data is placed in the normal data input queue as it is received, making it available to RECV, and RECVFROM without having to specify the MSG_OOB flag in those calls.
- When this option is disabled, out-of-band data is placed in the priority data input queue as it is received, making it available to RECV and RECVFROM only when the MSG_OOB flag is set.

SO-SNDBUF

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 DATABUFFERPOOLSIZ statement in the PROFILE.TCPIP data set. This value is adjusted to allow for protocol header information.

SO-ERROR

Requests 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-TYPE

Returns socket type: stream, datagram, or raw.

Parameter Values Returned to the Application:**OPTVAL**

- For all values of OPTNAME other than SO-LINGER, OPTVAL is a 32-bit fullword, containing the status of the specified option.
 - If the requested option is enabled, the fullword contains a positive value; if the requested option is disabled, the fullword contains zero.
 - If OPTNAME is set to SO-ERROR, OPTVAL contains the most recent ERRNO for the socket. This error variable is then cleared.
 - If OPTNAME is set to SO-TYPE, OPTVAL returns X'1' for SOCK-STREAM, to X'2' for SOCK-DGRAM, or to X'3' for SOCK-RAW.

- If SO-LINGER is specified in OPTNAME, the following structure is returned:

```

      ONOFF      PIC X(8)
      LINGER     PIC 9(8)
  
```

- A nonzero value returned in ONOFF indicates that the option is enabled; a zero value indicates that it is disabled.
- The LINGER value indicates the amount of time (in seconds) TCP/IP will continue to attempt to send the data after the CLOSE call is issued. To *set* the Linger time, see “SETSOCKOPT” on page 195.

OPTLEN

A fullword binary field containing the length of the data returned in OPTVAL.

- For all values of OPTNAME except SO-LINGER, OPTLEN will be set to 4 (one fullword).
- For OPTNAME of SO-LINGER, OPTVAL contains two fullwords, so OPTLEN will be set to 8 (two fullwords).

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See “Sockets Extended Return Codes” on page 263, 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 |

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.

GIVESOCKET

TCP/IP normally uses GETCLIENTID, GIVESOCKET, and TAKESOCKET calls in the following sequence:

1. A process issues a GETCLIENTID call to get the jobname 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 which 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.

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

For equivalent PL/I and assembler language declarations, see "Programming Language Conversions" on page 147.

Parameter Values Set by the Application:

SOC-FUNCTION

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

S A halfword binary number set to the socket descriptor of the socket to be given.

CLIENT

A structure containing the identifier of the application to which the socket should be given.

DOMAIN

A fullword binary number that must be set to 2, indicating AF_INET.

NAME Specifies an 8-character field, left-justified, padded to the right with blanks, that can be set to the name of the MVS address space that will contain the application that is going to take the socket.

GIVESOCKET

- 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 8-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.

- As used by IMS and CICS, the field should be set to blanks.
- If TASK identifier is non-blank, the socket-receiving task should already be in execution when the GIVESOCKET is issued.

RESERVED

A 20-byte reserved field. This field is required, but not used.

Parameter Values Returned to the Application:

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See "Sockets Extended Return Codes" on page 263, 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

The INITAPI call connects an application to the TCP/IP interface. Almost all sockets programs that are written in COBOL, PL/I, or assembler language must issue the INITAPI macro before they issue other sockets macros.

The exceptions to this rule are the following calls, which, when issued first, will generate a default INITAPI call.

- GETCLIENTID
- GETHOSTID
- GETHOSTNAME
- SELECT
- SELECTEX
- SOCKET
- TAKESOCKET.

INITAPI

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

PROCEDURE
  CALL 'EZASOKET' USING SOC-FUNCTION MAXSOC IDENT SUBTASK
  MAXSNO ERRNO RETCODE.
```

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

Parameter Values Set by the Application:

SOC-FUNCTION

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

MAXSOC

A halfword binary field set to the maximum number of sockets this application will ever have open at one time. The maximum number is 2000 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 structure containing the identities of the TCP/IP address space and the calling program's address space. Specify IDENT on the INITAPI call from an address space.

TCPNAME

An 8-byte character field which should be set to the MVS jobname of the TCP/IP address space with which you are connecting.

ADSNAME

An 8-byte character field set to the identity of the calling program's address space. It is the name of the CICS startup job.

SUBTASK

Indicates an 8-byte field, containing a unique subtask identifier which is used to distinguish between multiple subtasks within a single address space. For your subtask name, we suggest you 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 0. If you have 50 sockets, they are numbered from 0 to 49. If MAXSNO is not specified, the value for MAXSNO is 49.

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error

number. See “Sockets Extended Return Codes” on page 263, 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 macro, 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 10 on page 171, for information about REQARG and RETARG.

```

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,
   3  NAME              PIC X(16).
   3  FAMILY            PIC 9(4)  BINARY.
   3  PORT              PIC 9(4)  BINARY.
   3  ADDRESS           PIC 9(8)  BINARY.
   3  RESERVED         PIC X(8).

01  IFREQOUT,
   3  NAME              PIC X(16).
   3  FAMILY            PIC 9(4)  BINARY.
   3  PORT              PIC 9(4)  BINARY.
   3  ADDRESS           PIC 9(8)  BINARY.
   3  RESERVED         PIC X(8).

01  GRP_IOCTL_TABLE(100)
02  IOCTL_ENTRY,
   3  NAME              PIC X(16).
   3  FAMILY            PIC 9(4)  BINARY.
   3  PORT              PIC 9(4)  BINARY.
   3  ADDRESS           PIC 9(8)  BINARY.
   3  NULLS            PIC X(8).

01  IOCTL_REQARG       POINTER ;
01  IOCTL_RETARG       POINTER ;
01  ERRNO              PIC 9(8)  BINARY.
01  RETCODE            PIC 9(8)  BINARY.

```

PROCEDURE

```

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

```

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

IOCTL

Parameter Values Set by the Application:

SOC-FUNCTION

A 16-byte character field containing 'IOCTL'. The field is left justified and padded to the right with blanks.

S A halfword binary number set to the descriptor of the socket to be controlled.

COMMAND

To control an operating characteristic, set this field to one of the following symbolic names. A value in a bit mask is associated with each symbolic name. By specifying one of these names, you are turning on a bit in a mask which communicates the requested operating characteristic to TCP/IP.

'FIONBIO'

Sets or clears blocking status.

'FIONREAD'

Returns the number of immediately readable bytes for the socket.

'SIOCADDRT'

Adds a specified routing table entry.

'SIOCATMARK'

Determines whether the current location in the data input is pointing to out-of-band data.

'SIOCDELRT'

Deletes a specified routing table entry.

'SIOCGIFADDR'

Requests the network interface address for a given interface name. See the NAME field in Figure 77 on page 171 for the address format.

'SIOCGIFBRDADDR'

Requests the network interface broadcast address for a given interface name. See the NAME field in Figure 77 on page 171 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 77.

- 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 77. 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 77 for format.)

'SIOCGIFFLAGS'

Requests the network interface flags.

'SIOCGIFMETRIC'

Requests the network interface routing metric.

'SIOCGIFNETMASK'

Requests the network interface network mask.

'SIOCSIFMETRIC'

Sets the network interface routing metric.

'SIOCSIFDSTADDR'

Sets the network interface destination address.

'SIOCSIFFLAGS'

Sets the network interface flags.

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 10.

Table 10. 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
SIOCADDRT X'8030A70A'	48	For IBM use only	0	For IBM use only
SIOCATMARK X'4004A707'	0	Not used	4	X'00'= at OOB data X'01'= not at OOB data
SIOCDELRT X'8030A70B'	48	For IBM use only	0	For IBM use only
SIOCGIFADDR X'C020A70D'	32	First 16 bytes—interface name. Last 16 bytes—not used	32	Network interface address (See Figure 77 for format.)

Table 10. 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 77 on page 171 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 77 on page 171 for format.)
SIOCGIFFLAGS 'X'C020A711'	32	For IBM use only	32	For IBM use only
SIOCGIFMETRIC 'X'C020A717'	32	For IBM use only	32	For IBM use only
SIOCGIFNETMASK 'X'C020A715'	32	For IBM use only	32	For IBM use only
SIOCSIFMETRIC 'X'8020A718'	32	For IBM use only	0	For IBM use only
SIOCSIFDSTADDR 'X'8020A70E'	32	For IBM use only	0	For IBM use only
SIOCSIFFLAGS 'X'8020A710'	32	For IBM use only	0	For IBM use only

Parameter Values Returned to the Application:**RETARG**

Returns an array whose size is based on the value in COMMAND. See Table 10 for information about REQARG and RETARG.

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See “Sockets Extended Return Codes” on page 263, 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 78 contains an example of a COBOL II routine which can be used to work with such a structure.

Note: This call can only be programmed in languages which support address pointers.

```

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

```

Figure 78. 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.

```

WORKING STORAGE
  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
  CALL 'EZASOKET' USING SOC-FUNCTION S BACKLOG ERRNO RETCODE.

```

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

Parameter Values Set by the Application:

SOC-FUNCTION

A 16-byte character field containing 'LISTEN'. The field is left-justified and padded to the right with blanks.

S A halfword binary number set to the socket descriptor.

LISTEN

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 C. Return Codes" on page 253, 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 205 for a subroutine that will translate ASCII input data to EBCDIC.

```
WORKING STORAGE
 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
  CALL 'EZASOKET' USING SOC-FUNCTION S NBYTE BUF
  ERRNO RETCODE.
```

For equivalent PL/I and assembler language declarations, see "Programming Language Conversions" on page 147.

Parameter Values Set by the Application:

SOC-FUNCTION

A 16-byte character field containing 'READ'. The field is left justified and padded to the right with blanks.

S A halfword binary number set to the socket descriptor of the socket that is going to read the data.

NBYTE

A fullword binary number set to the size of BUF. READ does not return more than the number of bytes of data in NBYTE even if more data is available.

Parameter Values Returned to the Application:

BUF On input, a buffer to be filled by completion of the call. The length of BUF must be at least as long as the value of NBYTE.

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See "Sockets Extended Return Codes" on page 263, for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following:

Value Description

- 0** A zero 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.

```

WORKING-STORAGE SECTION.
01 SOKET-FUNCTION      PIC X(16) VALUE 'READV      '.
01 S                   PIC 9(4)  BINARY.
01 IOVAMT              PIC 9(4)  BINARY.

01 MSG-HDR.
03 MSG_NAME            POINTER.
03 MSG_NAME_LEN        POINTER.
03 IOVPTR              POINTER.
03 IOVCNT              POINTER.
03 MSG_ACCRIGHTS       PIC X(4).
03 MSG_ACCRIGHTS_LEN  PIC 9(4)  BINARY.

01 IOV.
03 BUFFER-ENTRY OCCURS N TIMES.
05 BUFFER_ADDR         POINTER.
05 RESERVED            PIC X(4).
05 BUFFER_LENGTH       PIC 9(4).

01 ERRNO               PIC 9(8)  BINARY.
01 RETCODE              PIC 9(8)  BINARY.

```

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:

READV

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 1.

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.

RETCODE

A fullword binary field that returns one of the following:

Value	Description
-------	-------------

0	A zero 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 154 or "IOCTL" on page 169 for a description of how to set nonblocking mode.

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

Note: See “EZACIC05” on page 205 for a subroutine that will translate ASCII input data to EBCDIC.

```

WORKING STORAGE
01 SOC-FUNCTION PIC X(16) VALUE IS 'RECV'.
01 S PIC 9(4) BINARY.
01 FLAGS PIC 9(8) BINARY.
    88 NO-FLAG VALUE IS 0.
    88 OOB VALUE IS 1.
    88 PEEK VALUE IS 2.
01 NBYTE PIC 9(8) BINARY.
01 BUF PIC X(length of buffer).
01 ERRNO PIC 9(8) BINARY.
01 RETCODE PIC S9(8) BINARY.
    
```

```

PROCEDURE
CALL 'EZASOKET' USING SOC-FUNCTION S FLAGS NBYTE BUF
ERRNO RETCODE.
    
```

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

Parameter Values Set by the Application:

SOC-FUNCTION

A 16-byte character field containing 'RECV'. The field is left justified and padded to the right with blanks.

S A halfword binary number set to the socket descriptor of the socket to receive the data.

FLAGS

A fullword binary field with values as follows:

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 “Sockets Extended Return Codes” on page 263, for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following :

Value Description

RECV

- 0 The socket is closed
- >0 A positive return code indicates the number of bytes copied into the buffer.
- 1 Check ERRNO for an error code

RECVFROM

The RECVFROM call receives data on a socket with descriptor S and stores it in a buffer. The RECVFROM call applies to both connected and unconnected sockets. The socket address is returned in the NAME structure. If a datagram packet is too long to fit in the supplied buffers, datagram sockets discard extra bytes.

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 154 or "IOCTL" on page 169 for a description of how to set nonblocking mode.

Note: See "EZACIC05" on page 205 for a subroutine that will translate ASCII input data to EBCDIC.

```
WORKING STORAGE
01 SOC-FUNCTION PIC X(16) VALUE IS 'RECVFROM'.
01 S PIC 9(4) BINARY.
01 FLAGS PIC 9(8) BINARY.
   88 NO-FLAG VALUE IS 0.
   88 OOB VALUE IS 1.
   88 PEEK VALUE IS 2.
01 NBYTE PIC 9(8) BINARY.
01 BUF PIC X(length of buffer).
01 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).
01 ERRNO PIC 9(8) BINARY.
01 RETCODE PIC S9(8) BINARY.
```

```
PROCEDURE
CALL 'EZASOKET' USING SOC-FUNCTION S FLAGS
NBYTE BUF NAME ERRNO RETCODE.
```

For equivalent PL/I and assembler language declarations, see "Programming Language Conversions" on page 147.

Parameter Values Set by the Application:

SOC-FUNCTION

A 16-byte character field containing 'RECVFROM'. The field is left justified and padded to the right with blanks.

S A halfword binary number set to the socket descriptor of the socket to receive the data.

FLAGS

A fullword binary field containing flag values as follows:

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 A 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 always 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 internet address of the sending socket.

RESERVED

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

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See "Sockets Extended Return Codes" on page 263, 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.

```

WORKING STORAGE
01 SOC-FUNCTION      PIC X(16)  VALUE IS 'RECVMSG      '
01 S                 PIC 9(4)   BINARY.
01 MSG-HDR.
03 MSG-NAME          USAGE IS POINTER.
03 MSG-NAME-LEN      USAGE IS POINTER.
03 IOV               USAGE IS POINTER.
03 IOVCNT            USAGE IS POINTER.
03 MSG-ACCRIGHTS     USAGE IS POINTER.
03 MSG-ACCRIGHTS-LEN USAGE IS POINTER.

01 FLAGS             PIC 9(8)   BINARY.
88 NO-FLAG           VALUE IS 0.
88 OOB               VALUE IS 1.
88 PEEK              VALUE IS 2.

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

```

LINKAGE SECTION.

```

01 RECVMSG-IOVECTOR.
03 IOV1A             USAGE IS POINTER.
05 IOV1AL            PIC 9(8) COMP.
05 IOV1L             PIC 9(8) COMP.
03 IOV2A             USAGE IS POINTER.
05 IOV2AL            PIC 9(8) COMP.
05 IOV2L             PIC 9(8) COMP.
03 IOV3A             USAGE IS POINTER.
05 IOV3AL            PIC 9(8) COMP.
05 IOV3L             PIC 9(8) COMP.

01 RECVMSG-BUFFER1   PIC X(16).
01 RECVMSG-BUFFER2   PIC X(16).
01 RECVMSG-BUFFER3   PIC X(16).
01 RECVMSG-BUFNO     PIC 9(8) COMP.

```

PROCEDURE

```

SET MSG-NAME TO NULLS.
SET MSG-NAME-LEN TO NULLS.
SET IOV TO ADDRESS OF RECVMSG-IOVECTOR.
MOVE 3 TO RECVMSG-BUFNO.
SET MSG-IOVCNT TO ADDRESS OF RECVMSG-BUFNO.
SET IOV1A TO ADDRESS OF RECVMSG-BUFFER1.
MOVE 0 TO MSG-IOV1AL.
MOVE LENGTH OF RECVMSG-BUFFER1 TO MSG-IOV1L.
SET IOV2A TO ADDRESS OF RECVMSG-BUFFER2.
MOVE 0 TO IOV2AL.
MOVE LENGTH OF RECVMSG-BUFFER2 TO IOV2L.
SET IOV3A TO ADDRESS OF RECVMSG-BUFFER3.
MOVE 0 TO IOV3AL.
MOVE LENGTH OF RECVMSG-BUFFER3 TO IOV3L.
SET MSG-ACCRIGHTS TO NULLS.
SET MSG-ACCRIGHTS-LEN TO NULLS.
MOVE X'00000000' TO FLAGS.
MOVE SPACES TO RECVMSG-BUFFER1.
MOVE SPACES TO RECVMSG-BUFFER2.
MOVE SPACES TO RECVMSG-BUFFER3.

```

CALL 'EZASOKET' USING SOC-FUNCTION S MSGHDR FLAGS ERRNO RETCODE.

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

Parameter Values Set by the Application:

RECVMSG

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's address is stored upon completion of the call.

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 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

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 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.

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 either of the following:

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

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 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 right-most bit represents socket descriptor 0; the left-most bit represents socket descriptor 31, and so on. 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 full words. You define the sockets that you want to test by turning on bits in the string.

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 206.

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.

SELECT

To test whether any of several sockets is ready for reading, set the appropriate bits in RSNDSK to '1' 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 can accept additional outgoing data.
- The socket is marked nonblocking and a previous CONNECT did not complete immediately. In this case, CONNECT returned an ERRNO with a value of 36 (EINPROGRESS). This socket will be selected for write when the CONNECT completes.

A call to WRITE, SEND, or SENDTO blocks when the amount of data to be sent exceeds the amount of data TCP/IP can accept. To avoid this, you can precede the write operation with a SELECT call to ensure that the socket is ready for writing. Once a socket is selected for WRITE, the program can determine the amount of TCP/IP buffer space available by issuing the GETSOCKOPT call with the SO-SNDBUF option.

To test whether any of several sockets is ready for writing, set the WSNDSK bits representing those sockets to '1' 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 ESNDSK bits representing those sockets to '1'. When the SELECT call returns, the corresponding bits in the ERETMSK indicate sockets with exception conditions.

MAXSOC Parameter: The SELECT call must test each bit in each string before returning results. For efficiency, the MAXSOC parameter can be used to specify the largest socket descriptor number that needs to be tested for any event type. The SELECT call tests only bits 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, RETCODE is set to 0.

```

WORKING STORAGE
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 RSNDSK        PIC X(*).
01 WSNDSK        PIC X(*).
01 ESNDSK        PIC X(*).
01 RRETSK        PIC X(*).
01 WRETSK        PIC X(*).
01 ERETSK        PIC X(*).
01 ERRNO         PIC 9(8) BINARY.
01 RETCODE       PIC S9(8) BINARY.

```

```

PROCEDURE
    CALL 'EZASOKET' USING SOC-FUNCTION MAXSOC TIMEOUT
                        RSNDSK WSNDSK ESNDSK
                        RRETSK WRETSK ERETSK
                        ERRNO RETCODE.

```

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

$((\text{maximum socket number} + 32) / 32 \text{ (drop the remainder)}) * 4$

Bit masks are 32-bit fullwords with 1 bit for each socket. Up to 32 sockets fit into 1 32-bit mask (PIC X(4)). If you have 33 sockets, you must allocate 2 32-bit masks (PIC X(8)).

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

Parameter Values Set by the Application:

SOC-FUNCTION

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

MAXSOC

A fullword binary field set to the largest socket descriptor number that is to be checked plus 1. (Remember to start counting at zero).

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 zero.

TIMEOUT is specified in the 2-word TIMEOUT as follows:

- TIMEOUT-SECONDS, word 1 of the TIMEOUT field, is the seconds component of the time-out value.
- TIMEOUT-MICROSEC, word 2 of the TIMEOUT field, is the microseconds component of the time-out value (0 through 999999).

For example, if you want SELECT to timeout after 3.5 seconds, set TIMEOUT-SECONDS to 3 and TIMEOUT-MICROSEC to 500000.

RSNDSK

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.

SELECT

- For sockets to be ignored, the value of the corresponding bit should be set to 0.

If this parameter is set to all zeros, the SELECT will not check for read events.

WSNDMSK

A bit string sent to request write event status.

- For each socket to be checked for pending write events, the corresponding bit in the string should be set to 1.
- For sockets to be ignored, the value of the corresponding bit should be set to 0.

If this parameter is set to all zeros, the SELECT will not check for write events.

ESNDMSK

A bit string sent to request exception event status.

- For each socket to be checked for pending exception events, the corresponding bit in the string should be set to 1.
- For each socket to be ignored, the corresponding bit should be set to 0.

If this parameter is set to all zeros, the SELECT will not check for exception events.

Parameter Values Returned to the Application:

RRETMSK

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

WRETMSK

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

ERETMSK

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

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See "Sockets Extended Return Codes" on page 263, 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.

```

WORKING STORAGE
  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 RSNDSK        PIC X(*).
  01 WSNDSK        PIC X(*).
  01 ESNDSK        PIC X(*).
  01 RRETSK        PIC X(*).
  01 WRETSK        PIC X(*).
  01 ERETSK        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
  CALL 'EZASOKET' USING SOC-FUNCTION MAXSOC TIMEOUT
                    RSNDSK WSNDSK ESNDSK
                    RRETSK WRETSK ERETSK
                    SELECB ERRNO RETCODE.

```

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

$((\text{maximum socket number} + 32) / 32 \text{ (drop the remainder)}) * 4$

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 2-word TIMEOUT as follows:

- TIMEOUT-SECONDS, word 1 of the TIMEOUT field, is the seconds component of the time-out value.
- TIMEOUT-MICROSEC, word 2 of the TIMEOUT field, is the microseconds component of the time-out value (0 through 999999).

SELECTEX

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 should see the *Assembler Macro Guide* for their operating system.

Parameter Values Returned by the Application:

ERRNO

A fullword binary field. If RETCODE is negative, this contains an error number.

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 non-zero and the caller's descriptor sets will be set to 0). The caller must initialize the ECB values to 0 before issuing the SELECTEX macro. |
| -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 address family support out-of-band data.
- Suppress use of local routing tables. This implies that the caller takes control of routing, 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 204 for a subroutine that will translate EBCDIC input data to ASCII.

```
WORKING STORAGE
 01 SOC-FUNCTION  PIC X(16)  VALUE IS 'SEND'.
 01 S            PIC 9(4)  BINARY.
 01 FLAGS       PIC 9(8)  BINARY.
      88 NO-FLAG          VALUE IS 0.
      88 OOB             VALUE IS 1.
      88 DONT-ROUTE      VALUE IS 4.
 01 NBYTE       PIC 9(8)  BINARY.
 01 BUF         PIC X(length of buffer).
 01 ERRNO       PIC 9(8)  BINARY.
 01 RETCODE     PIC S9(8) BINARY.

PROCEDURE
  CALL 'EZASOKET' USING SOC-FUNCTION S FLAGS NBYTE
                          BUF ERRNO RETCODE.
```

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

Parameter Values Set by the Application:

SOC-FUNCTION

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

S A halfword binary number specifying the socket descriptor of the socket that is sending data.

FLAGS

A fullword binary field with values as follows:

SEND

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 "Sockets Extended Return Codes" on page 263, for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following:

Value	Description
-------	-------------

0 or >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.

```

WORKING STORAGE
01 SOC-FUNCTION      PIC X(16)  VALUE IS 'SENDMSG      '.
01 S                 PIC 9(4)   BINARY.
01 MSG-HDR.
03 MSG-NAME          USAGE IS POINTER.
03 MSG-NAME-LEN      USAGE IS POINTER.
03 IOV               USAGE IS POINTER.
03 IOVCNT            USAGE IS POINTER.
03 MSG-ACCRIGHTS     USAGE IS POINTER.
03 MSG-ACCRIGHTS-LEN USAGE IS POINTER.

01 FLAGS             PIC 9(8)   BINARY.
88 NO-FLAG           VALUE IS 0.
88 OOB               VALUE IS 1.
88 DONTRROUTE        VALUE IS 4.
01 ERRNO             PIC 9(8)   BINARY.
01 RETCODE           PIC S9(8)  BINARY.

```

LINKAGE SECTION.

```

01 SENDMSG-IOVECTOR.
03 IOV1A             USAGE IS POINTER.
05 IOV1AL            PIC 9(8) COMP.
05 IOV1L             PIC 9(8) COMP.
03 IOV2A             USAGE IS POINTER.
05 IOV2AL            PIC 9(8) COMP.
05 IOV2L             PIC 9(8) COMP.
03 IOV3A             USAGE IS POINTER.
05 IOV3AL            PIC 9(8) COMP.
05 IOV3L             PIC 9(8) COMP.

01 SENDMSG-BUFFER1  PIC X(16).
01 SENDMSG-BUFFER2  PIC X(16).
01 SENDMSG-BUFFER3  PIC X(16).
01 SENDMSG-BUFNO    PIC 9(8) COMP.

```

PROCEDURE

```

SET MSG-NAME TO NULLS.
SET MSG-NAME-LEN TO NULLS.
SET IOV TO ADDRESS OF SENDMSG-IOVECTOR.
MOVE 3 TO SENDMSG-BUFNO.
SET MSG-IOVCNT TO ADDRESS OF SENDMSG-BUFNO.
SET IOV1A TO ADDRESS OF SENDMSG-BUFFER1.
MOVE 0 TO MSG-IOV1AL.
MOVE LENGTH OF SENDMSG-BUFFER1 TO MSG-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 MSG-ACCRIGHTS TO NULLS.
SET MSG-ACCRIGHTS-LEN TO NULLS.
MOVE X'00000000' TO FLAGS.
MOVE SPACES TO SENDMSG-BUFFER1.
MOVE SPACES TO SENDMSG-BUFFER2.
MOVE SPACES TO SENDMSG-BUFFER3.

```

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

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

Parameter Values Set by the Application:

SENDMSG

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.

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.

RETCODE

A fullword binary field that returns one of the following:

Value Description**0 or >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 or not 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 204 for a subroutine that will translate EBCDIC input data to ASCII.

```

WORKING STORAGE
01 SOC-FUNCTION PIC X(16) VALUE IS 'SENDTO'.
01 S PIC 9(4) BINARY.
01 FLAGS. PIC 9(8) BINARY.
   88 NO-FLAG VALUE IS 0.
   88 OOB VALUE IS 1.
   88 DONT-ROUTE VALUE IS 4.
01 NBYTE PIC 9(8) BINARY.
01 BUF PIC X(length of buffer).
01 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).
01 ERRNO PIC 9(8) BINARY.
01 RETCODE PIC S9(8) BINARY.

```

PROCEDURE

```

CALL 'EZASOKET' USING SOC-FUNCTION S FLAGS NBYTE
BUF NAME ERRNO RETCODE.

```

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

Parameter Values Set by the Application:

SENDTO

SOC-FUNCTION

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

S A halfword binary number set to the socket descriptor of the socket sending the data.

FLAGS

A fullword field that returns one of the following:

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 socket name structure as follows:

FAMILY

A halfword binary field containing the addressing family. For TCP/IP the value must be 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 internet address.

RESERVED

Specifies 8-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 "Sockets Extended Return Codes" on page 263, for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following:

Value Description

0 or >0

A successful call. The value is set to the number of bytes transmitted.

-1 Check ERRNO for an error code

SETSOCKOPT

The SETSOCKOPT call sets the options associated with a socket. SETSOCKOPT can be called only for sockets in the AF_INET domain.

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 0, if data is not needed by the command. The OPTLEN parameter must be set to the size of the data pointed to by OPTVAL.

```
WORKING STORAGE
  01 SOC-FUNCTION    PIC X(16) VALUE IS 'SETSOCKOPT'.
  01 S              PIC 9(4) BINARY.
  01 OPTNAME        PIC 9(8) BINARY.
      88 SO-REUSEADDR VALUE 4.
      88 SO-KEEPALIVE VALUE 8.
      88 SO-BROADCAST VALUE 32.
      88 SO-LINGER   VALUE 128.
      88 SO-OOBINLINE VALUE 256.
  01 OPTVAL         PIC 9(16) BINARY.
  01 OPTLEN         PIC 9(8) BINARY.
  01 ERRNO          PIC 9(8) BINARY.
  01 RETCODE        PIC S9(8) BINARY.

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

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

Parameter Values Set by the Application:

SOC-FUNCTION

A 16-byte character field containing 'SETSOCKOPT'. The field is left justified and padded to the right with blanks.

S A halfword binary number set to the socket whose options are to be set.

OPTNAME

Specify one of the following values.

SO-REUSEADDR

Toggles local address re-useability. This option allows local addresses that are already in use to be bound. Instead of checking at BIND time (the normal algorithm) the system checks at CONNECT time to ensure that the local address and port do not have the same remote address and port. If the association already exists, Error 48 (EADDRINUSE) is returned when the CONNECT is issued.

The default is DISABLED.

SO-BROADCAST

Toggles the ability to broadcast messages. This option has no meaning for stream sockets.

If SO-BROADCAST is enabled, the program can send broadcast messages over the socket to destinations which support the receipt of packets.

The default is DISABLED.

SETSOCKOPT

SO-KEEPALIVE

Toggles the TCP keep-alive mechanism for a stream socket. The default is disabled. When activated, the keep-alive mechanism periodically sends a packet on an otherwise idle connection. If the remote TCP does not respond to the packet or to retransmissions of the packet, the connection is terminated with the error ETIMEDOUT.

SO-LINGER

Controls how TCP/IP deals with data that it has not been able to transmit when the socket is closed. This option has meaning only for stream sockets.

- When LINGER is enabled and CLOSE is called, the calling program is blocked until the data is successfully transmitted or the connection has timed out.
- When LINGER is disabled, the CLOSE call returns without blocking the caller, and TCP/IP continues to attempt to send the data for a specified period of time. Although this usually provides sufficient time to complete the data transfer, use of the LINGER option does not guarantee successful completion because TCP/IP only waits the amount of time specified in OPTVAL LINGER.

The default is DISABLED.

SO-OOBINLINE

Toggles the ability to receive out-of-band data. This option has meaning only for stream sockets.

- When this option is enabled, out-of-band data is placed in the normal data input queue as it is received, and is available to a RECVFROM or a RECV call whether or not the OOB flag is set in the call.
- 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 call only when the OOB flag is set.

The default is DISABLED.

OPTVAL

Contains data which further defines the option specified in OPTNAME.

- For OPTNAME of SO-BROADCAST, SO-OOBINLINE, and SO-REUSEADDR, OPTVAL is a one-word binary integer. Set OPTVAL to a nonzero positive value to enable the option; set OPTVAL to zero to disable the option.
- For SO-LINGER, OPTVAL assumes the following structure:

```
ONOFF      PIC X(4).  
LINGER     PIC 9(8) BINARY.
```

Set ONOFF to a nonzero value to enable the option; set it to zero to disable the option. Set the LINGER value to the amount of time (in seconds) TCP/IP will linger after the CLOSE call.

OPTLEN

A fullword binary number specifying the length of the data returned in OPTVAL.

Parameter Values Returned to the Application:

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See “Sockets Extended Return Codes” on page 263, 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

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.

```

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

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

```

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

Parameter Values Set by the Application:**SOC-FUNCTION**

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

S A halfword binary number set to the socket descriptor of the socket to be shutdown.

HOW A fullword binary field. Set to specify whether all or part of a connection is to be shut down. The following values can be set:

Value	Description
-------	-------------

SHUTDOWN

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 "Sockets Extended Return Codes" on page 263, 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.

```
WORKING STORAGE
01 SOC-FUNCTION    PIC X(16) VALUE IS 'SOCKET'.
01 AF             PIC 9(8) COMP VALUE 2.
01 SOCTYPE       PIC 9(8) BINARY.
    88 STREAM          VALUE 1.
    88 DATAGRAM      VALUE 2.

01 PROTO         PIC 9(8) BINARY.
01 ERRNO        PIC 9(8) BINARY.
01 RETCODE      PIC S9(8) BINARY.
```

PROCEDURE

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

For equivalent PL/I and assembler language declarations, see "Programming Language Conversions" on page 147.

Parameter Values Set by the Application:

SOC-FUNCTION

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

AF A fullword binary field set to the addressing family. For TCP/IP the value is set to 2 for AF_INET.

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 zero, 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 “Sockets Extended Return Codes” on page 263, 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 which it obtained from the concurrent server. See “GIVESOCKET” on page 165 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.

```

WORKING STORAGE
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
CALL 'EZASOKET' USING SOC-FUNCTION SOCRECV CLIENT
ERRNO RETCODE.
    
```

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

TAKESOCKET

Parameter Values Set by the Application:

SOC-FUNCTION

A 16-byte character field containing 'TAKESOCKET'. The field is left justified and padded to the right with blanks.

SOCRCV

A halfword binary field set to the descriptor of the socket to be taken. The socket to be taken is passed by the concurrent server.

CLIENT

Specifies the client ID of the program that is giving the socket. In CICS and IMS, these parameters are passed by the Listener program to the program that issues the TAKESOCKET call.

- In CICS, the information is obtained using EXEC CICS RETRIEVE.
- In IMS, the information is obtained by issuing GU TIM.

DOMAIN

A fullword binary field set to domain of the program giving the socket. It is always 2, indicating AF_INET.

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 "Sockets Extended Return Codes" on page 263 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.

In the CICS environment, the use of TERMAPI is not recommended. CICS transaction termination automatically performs the functions of TERMAPI.

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.

WRITE

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 204 for a subroutine that will translate EBCDIC output data to ASCII.

```
WORKING STORAGE
  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
  CALL 'EZASOKET' USING SOC-FUNCTION S NBYTE BUF
                      ERRNO RETCODE.
```

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

Parameter Values Set by the Application:

SOC-FUNCTION

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

S A halfword binary field set to the socket descriptor.

NBYTE

A fullword binary field set to the number of bytes of data to be transmitted.

BUF Specifies the buffer containing the data to be transmitted.

Parameter Values Returned to the Application:

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See “Sockets Extended Return Codes” on page 263, for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following:

Value Description

0 or >0

A successful call. A return code greater than 0 indicates the number of bytes of data written.

-1 Check ERRNO for an error code

WRITEV

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

WRITEV

```
WORKING-STORAGE SECTION.
01 SOKET-FUNCTION      PIC X(16) VALUE 'WRITEV      '.
01 S                   PIC 9(4)  BINARY.
01 IOVAMT              PIC 9(4)  BINARY.

01 MSG-HDR.
   03 MSG_NAME         POINTER.
   03 MSG_NAME_LEN    POINTER.
   03 IOVPTR           POINTER.
   03 IOVCNT           POINTER.
   03 MSG_ACCRIGHTS   PIC X(4).
   03 MSG_ACCRIGHTS_LEN PIC 9(4)  BINARY.

01 IOV.
   03 BUFFER-ENTRY OCCURS N TIMES.
       05 BUFFER_ADDR  POINTER.
       05 RESERVED     PIC X(4).
       05 BUFFER_LENGTH PIC 9(4).

01 ERRNO               PIC 9(8)  BINARY.
01 RETCODE             PIC 9(8)  BINARY.

PROCEDURE

   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.
```

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

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.

RETCODE

A fullword binary field.

Value Meaning

<0	Error. Check ERRNO.
0	Connection partner has closed connection.
>0	Number of bytes sent.

Data Translation Programs for the 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
- EZACIC05—Translates ASCII data to EBCDIC data

Bit String Processing

In C-language, bit strings are often used to convey flags, switch settings, etc; TCP/IP 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.

EZACIC04:

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

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

```
PROCEDURE
  CALL 'EZACIC04' USING OUT-BUFFER LENGTH.
```

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

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.

Examples: None.

EZACIC05:

: The EZACIC05 program is used to translate ASCII data to EBCDIC data. EBCDIC data is required by COBOL, PL/I, and assembler language programs.

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

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

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

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.

Examples: None.

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, you might want to use the assembler language program EZACIC06 to translate them to character strings to be used with the SELECT call.

```
WORKING STORAGE
 01 TOKEN                                PIC X(16) VALUE 'TCPIPBITMASKCOBL'.
 01 CH-MASK.
    05 CHAR-STRING                        PIC X(nn).
 01 CHAR-ARRAY REDEFINES CH-MASK.
    05 CHAR-ENTRY-TABLE OCCURS nn TIMES.
      10 CHAR-ENTRY                        PIC X(1).
 01 BIT-MASK.
    05 BIT-ARRAY-FWDS                      PIC X(*) BINARY.
 01 COMMAND.
    05 CTOB                                PIC X(4) VALUE 'CTOB'.
    05 BTOC                                PIC X(4) VALUE 'BTOC'.
 01 BIT-LNGTH                            PIC 9(8) BINARY VALUE '8'.
 01 RETCODE                              PIC 9(8) BINARY.

PROCEDURE
  CALL 'EZACIC06' USING TOKEN BTOC BIT-MASK CH-MASK
  BIT-LENGTH RETCODE.
```

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

TOKEN

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

CH-MASK

Specifies the character array where *nn* is the maximum number of sockets in the array.

BIT-MASK

Specifies the bit string to be translated for the SELECT call. The bits are ordered right-to-left with the right-most bit representing socket 0. The socket positions in the character array are indexed starting with 1 making socket 0 index number 1 in the character array. You should keep this in mind when turning character positions on and off.

COMMAND

BTOC—Specifies bit string to character array translation.

CTOB—Specifies character array to bit string translation.

BIT-LNGTH

Specifies the length of the bit-mask.

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 9, and you are using a character array to represent the sockets, you must set the appropriate characters in the character array to 1. In this example, index positions 1, 6 and 10

in the character array are set to 1. Then you can call EZACIC06 with the COMMAND parameter set to CTOB. When EZACIC06 returns, BIT-MASK contains a fullword with bits 0, 5, and 9 (numbered from the right) turned on as required by the SELECT call. These instructions process the bit string shown in the following example.

```
MOVE ZEROS TO CHAR-ENTRY-TABLE.  
MOVE '1' TO CHAR-ENTRY(1), CHAR-ENTRY(6), CHAR-ENTRY(10).  
CALL 'EZACIC06' USING TOKEN CTOB BIT-MASK CH-MASK  
      BIT-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  
      BIT-LENGTH RETCODE.  
PERFORM TEST-SOCKET THRU TEST-SOCKET-EXIT VARYING IDX  
      FROM 1 BY 1 UNTIL IDX EQUAL 10.  
  
TEST-SOCKET.  
  IF CHAR-ENTRY(10) EQUAL '1'  
    THEN PERFORM SOCKET-RESPONSE THRU SOCKET-RESPONSE-EXIT  
    ELSE NEXT SENTENCE.  
TEST-SOCKET-EXIT.  
  EXIT.
```

EZACIC08:

: The GETHOSTBYNAME and GETHOSTBYADDR calls were derived from C socket calls that return a structure known as HOSTENT. A given TCP/IP host can have multiple alias names and host internet addresses.

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

If you are coding in PL/I or assembler language, the HOSTENT structure can be processed in a relatively straight-forward manner. However, if you are coding in COBOL, HOSTENT can be more difficult to process and you should use the EZACIC08 subroutine to process it for you.

Here is how it works:

- 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 (3 and 9 above), the application program should repeat the call to EZACIC08 until all alias names and host internet addresses have been retrieved.

WORKING STORAGE

```

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

```

CALL 'EZASOKET' USING 'GETHOSTBYxxxx'
                    HOSTENT-ADDR
                    RETCODE.

```

Where xxxx is ADDR or NAME.

```

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

```

For equivalent PL/I and assembler language declarations, see “Programming Language Conversions” on page 147.

Parameter Values set by the Application:

HOSTENT-ADDR

This fullword binary field must contain the address of the HOSTENT structure (as returned by the GETHOSTBYxxxx call). This variable is the same as the variable HOSTENT in the GETHOSTBYADDR and GETHOSTBYNAME socket calls.

HOSTALIAS-SEQ

This halfword field is used by EZACIC08 to index the list of alias names. When EZACIC08 is called, it adds 1 to the current value of HOSTALIAS-SEQ and uses the resulting value to index into the table of alias names. Therefore, for a given instance of GETHOSTBYxxxx, this field should be set to zero for the initial call to EZACIC08. For all subsequent calls to EZACIC08, this field should contain the HOSTALIAS-SEQ number returned by the previous invocation.

HOSTADDR-SEQ

This halfword field is used by EZACIC08 to index the list of IP addresses. When EZACIC08 is called, it adds 1 to the current value of HOSTADDR-SEQ and uses the resulting value to index into the table of IP addresses. Therefore, for a given instance of GETHOSTBYxxxx, this field should be set to zero 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

Appendix A. External Data Structures

External Data Structures

The data structures available for customer use are as follows:

Configuration Data Set Record Formats

DSECT/Structure Name

CFG0000

Length of Structure

CFGLEN

Macro Expansion

EZACICA AREA=CFG,TYPE=DSECT

EZACICA AREA=CFG,TYPE=CSECT

Table 11. Configuration File Format

Field Name	Field Type	Description	Default Value
CFHAPPL	8 byte char	APPLID of CICS Object to which this record refers.	
CFHRTYPE	1 byte char	Record type <ul style="list-style-type: none"> • C = CICS Object Record • L = Listener Object Record 	
(Reserved)	3 byte hex	Reserved for IBM Use	00
<i>CICS Record Format</i>			
CFLTRAN	4 byte char	Transaction name for Configuration Record	<<<<
CFCTCPIP	8-byte char	Address space name of TCP/IP	
CFCNOTSK	Halfword bin	Number of reusable tasks	20
CFCSTIME	Halfword bin	Resolver Cache minimum refresh time	15
CFCLTIME	Halfword bin	Resolver Cache maximum refresh time	30
CFCNORES	Halfword bin	Resolver Cache number of concurrent resolvers	10
CFCDPRTY	Halfword bin	Limit Priority of Subtask (LPMOD value in ATTACH macro)	0
CFCENAME	4-byte character	Name of Transient Data Message Queue	CSMT
<i>Listener Record Format</i>			
CFLTRAN	4 byte char	Transaction name for this listener	CSKL
CFLPORT	Halfword bin	Port number for this listener	
CFLBKLOG	Halfword bin	Backlog value for listen call	10
CFLNSOCK	Halfword bin	Number of sockets used by listener	50
CFLMMIN	Halfword bin	Minimum length of input message	4
CFLLTIM	Halfword bin	Timeout value (seconds) for accept	60

Table 11. Configuration File Format (continued)

Field Name	Field Type	Description	Default Value
CFLRTIM	Halfword bin	Timeout value (seconds) for read	0
CFLGTIM	Halfword bin	Timeout value (seconds) for givesocket	30
CFLOPT	1 byte hex	Listener Options Value Description B'00000001' Immediate Startup B'00000110' Translate entire message B'00000010' Translate trans code only B'00000100' Translate user data only B'00001000' Issue READ immediately after ACCEPT	B'00001111'
CFLSECXT	8-byte char	Name of Security Exit	EZACICSE
CFLUSRID	8-byte char	Surrogate User ID for this Listener (CICS V4 only)	

Global Work Area

DSECT/Structure Name

GWA0000

Length of Structure

GWALENTH (Length of Fixed Area)

Macro Expansion

EZACICA AREA=GWA,TYPE=DSECT

EZACICA AREA=GWA,TYPE=CSECT

Table 12. Global Work Area Format

Field Name	Field Type	Description	Default Value
<i>Beginning of Global Work Area Eyecatcher</i>			
GWACMDSC	8 byte char	Identifier	GWATCPIP
<i>Beginning of Startup Module Heritage</i>			
GWACMNAM	8 byte hex	Startup Module Name	EZACIC21
GWACMVER	2 byte char	Startup Module Version Number	
GWACMREL	2 byte char	Startup Module Release Number	
GWACMMOD	2 byte char	Startup Module Mod Number	
GWACMDAT	8 byte char	Startup Module Assembled Date	
GWACMTIM	8 byte char	Startup Module Assembled Time	
<i>End of Startup Module Heritage</i>			
<i>Beginning of Task-Related User Exit Heritage</i>			

Table 12. Global Work Area Format (continued)

Field Name	Field Type	Description	Default Value
GWATRNAM	8 byte hex	Task Related User Exit Module Name	EZACIC01
GWATRVER	2 byte char	Task Related User Exit Version Number	
GWATRREL	2 byte char	Task Related User Exit Release Number	
GWATRMOD	2 byte char	Task Related User Exit Mod Number	
GWATRDAT	8 byte char	Task Related User Exit Assembled Date	
GWATRTIM	8 byte char	Task Related User Exit Assembled Time	
<i>End of Task-Related User Exit Heritage</i>			
<i>Beginning of IBM Listener Heritage</i>			
GWAMSNAM	8 byte hex	IBM Listener Module Name	EZACIC02
GWAMSVR	2 byte char	IBM Listener Version Number	
GWAMSREL	2 byte char	IBM Listener Release Number	
GWAMSMOD	2 byte char	IBM Listener Mod Number	
GWAMSDAT	8 byte char	IBM Listener Assembled Date	
GWAMSTIM	8 byte char	IBM Listener Assembled Time	
<i>End of IBM Listener Heritage</i>			
<i>Beginning of Attached Subtask Heritage</i>			
GWASTNAM	8 byte hex	Attached Subtask Module Name	EZACIC03
GWASTVER	2 byte char	Attached Subtask Version Number	
GWASTREL	2 byte char	Attached Subtask Release Number	
GWASTMOD	2 byte char	Attached Subtask Mod Number	
GWASTDAT	8 byte char	Attached Subtask Assembled Date	
GWASTTIM	8 byte char	Attached Subtask Assembled Time	
<i>End of Attached Subtask Heritage</i>			
GWACMIBM	154 byte char	Copyright Statement	
	42 byte char	Reserved Area	
<i>End of Global Work Area Eyecatcher</i>			
GWAUSCNT	Fullword bin	Use count for this GWA	
GWABKWRD	Fullword bin	Attached (non-pool) task chain anchor backward address	
GWAFOWRD	Fullword bin	Attached (non-pool) task chain anchor forward address	
GWACAPPL	8 byte char	VTAM APPLID of the CICS System	
GWATRUEN	8 byte char	Name of Task Related User Exit Load Module	
GWASTSKN	8 byte char	Name of Attached Subtask Load Module	
GWATCPID	8 byte char	TCPIP Address Space Name	
GWALCAAD	Fullword bin	Address of First Listener Control Area	
GWA03PSA	Fullword bin	Address of EZASOH03 Load Module	
GWANTASK	Halfword bin	Number of Reusable Tasks	
GWANLIST	Halfword bin	Number of Listeners	

Table 12. Global Work Area Format (continued)

Field Name	Field Type	Description	Default Value
GWATSTAT	1 byte char	Task Related User Exit Status Value Meaning E TRUE is enabled I Immediate Shutdown Requested/Processing Q Quiescent Shutdown Requested/Processing	
GWARSHUT	1 byte char	EZAO Shutdown Request Indicator Value Meaning I Immediate Shutdown Requested/Processing Q Quiescent Shutdown Requested/Processing	
GWACSTAT	1 byte bin	CICS Execution Status Value Meaning B'00000001' 1st Stage of Initialization B'00000010' 2nd Stage of Initialization B'00000100' 3rd Stage of Initialization B'00001000' Orderly of Shutdown B'00010000' Immediate of Shutdown B'00100000' Abending, TCB's Dispatchable(RETRY) B'01000000' Abending, TCB's Dispatchable(NORTY) B'10000000' Operator Cancelled, TCB's Not Dispatchable	
GWAVOSYS	1 byte bin	MVS Version Value Meaning B'00000001' B'00000010' B'00000100' B'00001000' B'00010000' B'00100000' B'01000000' B'10000000'	
GWAOPREL	2 byte bin	MVS Release	

Table 12. Global Work Area Format (continued)

Field Name	Field Type	Description	Default Value
GWACIVER	2 byte char	CICS Version	
GWACIREL	1 byte char	CICS Release	
GWACIMOD	1 byte char	CICS Modification	
GWATOKEN	8 byte char	Token for OS/390 Registration/Deregistration	
GWAMSGMD	8 byte char	Name of Message Module	
<i>End of Global Work Area Eyecatcher</i>			
GWATDMSG	4 byte char	Name of TD Queue for Message Delivery	
<i>End of Fixed Part of GWA</i>			

Task Interface Element (TIE)

DSECT/Structure Name

TIE0000

Length of Structure

TIELENTH

Macro Expansion

EZACICA AREA=TIE,TYPE=DSECT

EZACICA AREA=TIE,TYPE=CSECT

Table 13. Task Interface Element Format

Field Name	Field Type	Description	Default Value
Beginning of Task Interface Element Eyecatcher			
TIETRDC	8 byte char	Identifier	EZACICTI
Beginning of Task-Related User Exit Heritage			
TIETRNAM	8 byte char	Task Related User Exit Module Name	EZACIC01
TIETRDAT	8 byte char	Task Related User Exit Assembled Date	
TIETRSRV	8 byte char	Task Related User Exit Service Level	
TIETRRSV	6 byte char	Task Related User Exit Reserved Field	
End of Task-Related User Exit Heritage			
Beginning of Startup Module Heritage			
TIECMNAM	8 byte hex	Startup Module Name	EZACIC21
TIECMDAT	8 byte char	Startup Module Assembled Date	
TIECMSRV	8 byte char	Startup Module Service Level	
TIECMRSV	6 byte char	Startup Module Reserved Field	
End of Startup Module Heritage			
<i>Beginning of IBM Listener Heritage</i>			
TIEMSNAM	8 byte hex	IBM Listener Module Name	EZACIC02
TIEMSDAT	8 byte char	IBM Listener Assembled Date	
TIEMSSRV	2 byte char	IBM Listener Service Level	

Table 13. Task Interface Element Format (continued)

Field Name	Field Type	Description	Default Value
TIEMRSRV	6 byte char	IBM Listener Reserved Field	
<i>End of IBM Listener Heritage</i>			
<i>Beginning of Attached Subtask Heritage</i>			
TIESTNAM	8 byte char	Attached Subtask Module Name	EZACIC03
TIESTDTE	8 byte char	Attached Subtask Assembled Date	
TIESDSRV	8 byte char	Attached Subtask Service Level	
TIESTRSV	6 byte char	Attached Subtask Reserved Field	
<i>End of Attached Subtask Heritage</i>			
TIETRIBM	154 byte char	Copyright Statement	
	46 byte char	Reserved Area	
<i>End of Task Interface Element Eyecatcher</i>			
TIEBKWRD	Fullword bin	Task Chain Anchor Backward Address (Attached Tasks Only)	
TIEFOWRD	Fullword bin	Task Chain Anchor Forward Address (Attached Tasks Only)	
TIESTTCB	Fullword bin	Address of Subtask's TCB	
TIESTPSA	Fullword bin	Address of Subtask's Program Storage	
TIESTPSL	Fullword bin	Length of Subtask's Program Storage	
TIESTCSA	Fullword bin	Address of Subtask's Common Storage	
TIESTCSL	Fullword bin	Length of Subtask's Common Storage	
TIECTECB	Fullword bin	TRUE Event Complete ECB posted by Subtask	
TIECKECB	Fullword bin	TRUE Immediate Termination ECB posted by termination management	
TIESTECCB	Fullword bin	Subtask ECB Posted by TRUE to initiate function	
TIESDECB	Fullword bin	Subtask ECB Posted by TRUE to signal task end	
<i>Wait List for TRUE</i>			
TIECTADS	Fullword bin	Address of TIECTECB (Event Complete ECB)	
TIECKADS	Fullword bin	Address of TIECKECB (Termination ECB)	
<i>Wait List for Subtask</i>			
TIESTADS	Fullword bin	Address of TIESTECCB (Start event ECB)	
TIESDADS	Fullword bin	Address of TIESDADS (Task Termination ECB)	
<i>Beginning of Audit Trail Data</i>			
TIECCALL	16 byte char	Eyecatcher	CURRENT CALL
TIECALLT	16 byte char	Current Call Type	
TIESOKCL	Halfword bin	HPNS Function	
TIESOKDS	Halfword bin	Socket Descriptor	
TIEAPARM	Fullword bin	Address of Application Parameter List	
TIEERRNO	Fullword bin	ERRNO returned from call	

Table 13. Task Interface Element Format (continued)

Field Name	Field Type	Description	Default Value
TIEUENAD	Fullword bin	Address of user's ERRNO field	
TIERCODE	Fullword bin	Return code from call	
TIEURCAD	Fullword bin	Address of User's Return Code Field	
TIEMACRC	Fullword bin	MVS Macro Return Code Returned in R15	
TIEPCALL	16 byte char	Eyecatcher	PREVIOUS CALL
TIESCALL	16 byte char	Previous Call Type	
TISSOKD	Halfword bin	Socket Descriptor	
TIESAPAR	Fullword bin	Address of Application Parameter List	
TIESERRN	Fullword bin	ERRNO returned from call	
TIESUENA	Fullword bin	Address of user's ERRNO field	
TIESRCOD	Fullword bin	Return code from call	
TIESURCA	Fullword bin	Address of User's Return Code Field	
TIESMACR	Fullword bin	MVS Macro Return Code Returned in R15	
TIEFSTAT	1 byte bin	<p>TRUE Function Status Flags</p> <p>Value Meaning</p> <p>B'00000000' TRUE uninitialized (Never Called)</p> <p>B'00000001' TRUE Called by Application</p> <p>B'00000010' TRUE Called by Diagnostic Facility</p> <p>B'00000100' Unused</p> <p>B'00001000' TRUE Called by Syncpoint Manager</p> <p>B'00010000' TRUE Called by Task Termination</p> <p>B'00100000' TRUE Called by System (CICS) Termination</p> <p>B'10000000' TRUE Called by Unknown Source</p>	

Table 13. Task Interface Element Format (continued)

Field Name	Field Type	Description	Default Value
TIEASTAT	1 byte bin	<p>Application Status Flags (Calling Application)</p> <p>Value Meaning</p> <p>B'00000000' Application not initialized (Has Never Called)</p> <p>B'00000001' Application's Initial Call to TRUE</p> <p>B'00000010' Application is Active (Not Waiting on TRUE)</p> <p>B'00000100' Unused</p> <p>B'00001000' Application is Waiting for Subtask Service</p> <p>B'01000000' Application is at Task Termination</p> <p>B'10000000' Application is at System Termination</p>	
TIESSTAT	1 byte bin	<p>Subtask Status Flags</p> <p>Value Meaning</p> <p>B'00000000' Subtask not initialized</p> <p>B'00000001' Using Pool Subtask</p> <p>B'00000010' Using Attached Subtask</p> <p>B'00000100' Subtask Initialized</p> <p>B'00001000' Subtask Waiting on TRUE</p> <p>B'00010000' Subtask has HPNS Connection</p> <p>B'00100000' Subtask has Detected Errors</p> <p>B'01000000' Subtask has Abnormally Terminated</p> <p>B'10000000' Subtask has Detached</p>	

Table 13. Task Interface Element Format (continued)

Field Name	Field Type	Description	Default Value
TIESSTAT	1 byte bin	Task Related Exit (TRUE) Status Flags Value Meaning B'00000000' TRUE not initialized B'00000001' TRUE's Initial Entry B'00000010' TRUE is Active B'00000100' TRUE is Enabled for Application Calls B'00001000' TRUE Waiting on Subtask B'00010000' TRUE Enabled for Termination Manager Calls B'00100000' TRUE is Disabled for Application Calls B'01000000' TRUE is Disabled for Termination Manager Calls	
TIESUBID	8 byte bin	TCP/IP Subtask ID	
TIEAPLID	8 byte char	VTAM APPLID for this CICS	
TIEOPID	3 byte char	CICS Operator Identification from Sign-On Table	
TIETRMID	4 byte char	CICS Terminal Identification from EIBTRMID	
TIETRNID	4 byte char	CICS Transaction Identification from EIBTRNID	
TIETASKN	4 byte packed	CICS Task Number from EIBTASKN	
TIEDOWEK	3 byte char	Abbreviated Day of Week	
TIEDATFM	8 byte char	Date Format as Specified in SID DATEFORM= Option	
TIECLKTM	8 byte char	24-hour clock time in the form HH:MM:SS	
E\$APARMS	30 times Fullword bin	Parameter List for EZASOKET Calls	
E\$ASMTOK	16 byte char	Token Field	
E\$ASMCMD	2 byte bin	Command Field	
E\$ASMDUM	32 byte bin	Dummy field Used by EZASMI Macro	
E\$ASMLOM	Fullword bin	Length of Mask field for Select	
E\$ASMDUM	32 byte bin	Dummy field Used by EZASMI Macro	
E\$USECB	Fullword bin	Address of User's ECB Area	
E\$AUPARM	Fullword bin	Address of User's Parameter List	
E\$USOCK	4 byte bin	Socket Field for Accept, Socket and Takesocket	
E\$TSOCK	Fullword bin	Socket Field for Takesocket	

Parameter List (COMMAREA) for EZACIC20

DSECT/Structure Name
P20PARMS

Length of Structure
P20PARML

Macro Expansion
EZACICA AREA=P20,TYPE=DSECT
EZACICA AREA=P20,TYPE=CSECT

Table 14. COMMAREA Format for EZACIC20

Field Name	Field Type	Description	Default Value
P20TYPE	1 byte char	Type of Function Value Meaning I Initialization T Immediate Termination D Deferred Termination	
P20OBJ	1 byte char	Type of Function Value Meaning C CICS Sockets Interface L Listener	
P20LIST	4 byte char	Transaction Name of Listener	
P20RET	1 byte bin	Return Code Value Meaning B'00000000' No Errors Encountered B'00000001' Errors in CICS Sockets Interface Initialization B'00000010' Errors in Listener Initialization B'00000100' Errors in CICS Sockets Interface Termination B'00001000' Errors in Listener Termination B'00010000' Errors in COMMAREA Contents. B'00100000' Errors in CICS/MVS.	

Listener Control Area (LCA)

DSECT/Structure Name

LCA0000

Length of Structure

LCALEN

Macro Expansion

EZACICA AREA=LCA,TYPE=DSECT

EZACICA AREA=LCA,TYPE=CSECT

Table 15. Listener Control Area (LCA)

Field Name	Field Type	Description	Default Value
LCATECB	Fullword bin	ECB Posted by Termination Manager	
LCATRAN	4 byte char	Transaction Name for this Listener	
LCASTAT	1 byte bin	Status of this Listener Value Meaning B'00000000' Listener Not in Operation B'00000001' Listener Not in Operation B'00000001' Listener in Initialization B'00000010' Listener in SELECT B'00000100' Listener Processing B'00001000' Listener Had Initialization Error B'00010000' Immediate Termination in Progress B'00100000' Deferred Termination in Progress	
LCAPHASE	1 byte char	Execution Phase for IBM Listener	

Appendix B. Original COBOL Application Programming Interface (EZACICAL)

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 79 on page 224. 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

```

//CICRS2C JOB (999,POK),'CICRS2',NOTIFY=CICRS2,
// CLASS=A,MSGCLASS=T,TIME=1439,
// REGION=5000K,MSGLEVEL=(1,1)
//DFHEITVL PROC SUFFIX=1$,
// INDEX='CICS330',
// INDEX2='CICS330',
// OUTC=*,
// REG=2048K,
// LNKPARAM='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=CICRS2.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))
//*
//LKD EXEC PGM=IEWL,REGION=&REG,
// PARM='&LNKPARAM',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=h1q.SEZATCP,DISP=SHR
//SYSLMOD DD DSN=CICRS2.CICS330.PGMLIB,DISP=SHR
//SYSUT1 DD UNIT=&WORK,DCB=BLKSIZE=1024,
// SPACE=(1024,(200,20))
//SYSPRINT DD SYSOUT=&OUTC
//
//SYSLIN DD DSN=&&LOADSET,DISP=(OLD,DELETE)
// DD DDNAME=SYSIN
// PEND
//APPLPROG EXEC DFHEITVL
//TRN.SYSIN DD DISP=SHR,DSN=CICRS2.JCL.DATA(SISSRR1C)
//LKED.SYSIN DD *
INCLUDE SYSLIB(EZACICAL)
NAME SISSRR1C(R)
/*

```

X

X

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 'TCPPIUCVSTREAMS'

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 227 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 'TCPPIUCVSTREAMS'

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 227 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
CLn	Character format, length <i>n</i> bytes
XLn	Hexadecimal format, length <i>n</i> bytes

ACCEPT

This call functions in the same way as the equivalent call described on page 148. 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).

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 9(8) BINARY
NAME STRUCTURE:		
<i>Internet Family</i>	H	PIC 9(4) BINARY
<i>Port</i>	H	PIC 9(4) BINARY
<i>Internet Address</i>	F	PIC 9(8) BINARY
<i>Zeros</i>	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 'TCP/IPUCVSTREAMS'

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 C. Return Codes" on page 253.

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 149. 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 226).

BIND

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:		
<i>Internet Family</i>	H	PIC 9(4) BINARY
<i>Port</i>	H	PIC 9(4) BINARY
<i>Internet Address</i>	F	PIC 9(8) BINARY
<i>Zeros</i>	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 'TCPIUCVSTREAMS'

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 zero, the system returns an available port.

ERRNO

If RETCODE is negative, this contains an error number. Error numbers are described in "Appendix C. Return Codes" on page 253.

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 151. 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 226).

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 'TCPIIUCVSTREAMS'

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 C. Return Codes" on page 253.

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 152. 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 on page 226).

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:		
<i>Internet Family</i>	H	PIC 9(4) BINARY
<i>Port</i>	H	PIC 9(4) BINARY
<i>Internet Address</i>	F	PIC 9(8) BINARY
<i>Zeros</i>	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 'TCPIIUCVSTREAMS'

COMMAND

Must be set to 4 for the CONNECT command

S The descriptor of the local socket to be used to establish a connection

CONNECT

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 C. Return Codes" on page 253.

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 154. 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 226).

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 'TCPIUCVSTREAMS'

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 C. Return Codes” on page 253.

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 155. 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 on page 226).

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:		
<i>Domain</i>	F	PIC 9(8) BINARY
<i>Name</i>	CL8	PIC X(8)
<i>Task</i>	CL8	PIC X(8)
<i>Reserved</i>	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 'TCP/IPUCVSTREAMS'

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

GETCLIENTID

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 C. Return Codes" on page 253.

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 156. 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 226).

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 'TCPIIUCVSTREAMS'

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 158. The format of the COBOL call for the GETHOSTNAME function is:

GETHOSTNAME

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 226).

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	NAMELEN or larger 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 'TCPIIUCVSTREAMS'

COMMAND

Must be set to 8 for the GETHOSTNAME command

HZERO

Set to zero

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 C. Return Codes" on page 253.

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 161. 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 226).

Parameter Lengths in Assembler Language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY

GETPEERNAME

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 'TCPIIUCVSTREAMS'

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 C. Return Codes" on page 253.

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 162. 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 226).

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:		
<i>Internet Family</i>	H	PIC 9(4) BINARY
<i>Port</i>	H	PIC 9(4) BINARY
<i>Internet Address</i>	F	PIC 9(8) BINARY
<i>Zeros</i>	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 'TCPIIUCVSTREAMS'

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 C. Return Codes" on page 253.

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 163. 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 on page 226).

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 'TCPIIUCVSTREAMS'

COMMAND

Must be set to 11 for the GETSOCKOPT command

GETSOCKOPT

S The descriptor of the socket whose option settings are required

LEVEL

This must be set to X'0000FFFF'. It is only possible to get options for the socket level.

OPTNAME

Set this field to specify the option to be queried, as shown below. For a description of these options, see "GETSOCKOPT" on page 163

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

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 zero 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 zero, 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 C. Return Codes" on page 253.

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 165. 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 226).

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:		
<i>Domain</i>	F	PIC 9(8) BINARY
<i>Name</i>	CL8	PIC X(8)
<i>Task</i>	CL8	PIC X(8)
<i>Reserved</i>	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 'TCPIIUCVSTREAMS'

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 C. Return Codes" on page 253.

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 IDENT MAX-SOCK API SUBTASK FZERO [cont]
ERRNO RETCODE.
```

In assembler language, issue the macro call CALL EZACICAL, using standard assembler call syntax (for the call format, see page 226).

INITAPI

Parameter Lengths in Assembler Language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY
IDENT	CL8	PIC X(8)
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 'TCPIIUCVSTREAMS'

COMMAND

Must be set to 0 for the INITAPI command

IDENT Must be set to 'IUCVAPI'.

MAX-SOCK

The maximum number of sockets to be supported in this application. For performance reasons, this should be one greater than the actual maximum. This value cannot exceed 2000. 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 169. 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 on page 226).

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 'TCPIUCVSTREAMS'

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 169 for values and descriptions.

REQARG

The request argument associated with the command. See "IOCTL" on page 169 for a list and description of possible argument values.

Parameter Values Returned to the Application

RETARG

The return argument. See "IOCTL" on page 169 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 C. Return Codes" on page 253.

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 173. 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 226).

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

LISTEN

Parameter Values to Be Set by the Application

TOKEN

Must be set to 'TCPIIUCVSTREAMS'

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 C. Return Codes" on page 253.

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 174. 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 on page 226).

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 'TCPIIUCVSTREAMS'

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 C. Return Codes” on page 253.

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 205 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 176. 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 on page 226).

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 'TCPIIUCVSTREAMS'

COMMAND

Must be set to 16 for the RECVFROM command

S The descriptor of the socket receiving data

RECVFROM

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 C. Return Codes" on page 253.

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 205 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 183. 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 on page 226).

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:		
<i>Seconds</i>	F	PIC 9(8) BINARY
<i>Milliseconds</i>	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 $NUM-FDS \leq 32$, $LOM = 4$ bytes.
3. For $33 \leq NUM-FDS \leq 64$, $LOM = 8$ bytes, and so on.

Parameter Values to Be Set by the Application

TOKEN

Must be set to 'TCPIUCVSTREAMS'

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 time-out 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 time-out value if no activity is detected. Setting this structure to (0,0) indicates that SELECT should act as a polling function; that is, as non-blocking.

Seconds

Set to the seconds component of the time-out value.

Milliseconds

Set to the milliseconds component of the time-out value (in the range 0 through 999).

RD-MASK

Set the bit mask array for reads. See *TCP/IP for MVS: Programmer's Reference* for more information.

SELECT

WR-MASK

Set the bit mask array for writes. See *TCP/IP for MVS: Programmer's Reference* for more information.

EX-MASK

Set the bit mask array for exceptions. See *TCP/IP for MVS: 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 *TCP/IP for MVS: Programmer's Reference* for more information.

R-W-MASK

Returned bit mask array for writes. See *TCP/IP for MVS: Programmer's Reference* for more information.

R-E-MASK

Returned bit mask array for exceptions. See *TCP/IP for MVS: Programmer's Reference* for more information.

ERRNO

If RETCODE is negative, this contains an error number. Error numbers are described in "Appendix C. Return Codes" on page 253.

RETCODE

A positive value indicates the total number of ready sockets in all bit masks. A value of zero 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 189. 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 226).

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 'TCPIUCVSTREAMS'

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 C. Return Codes" on page 253.

RETCODE

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

See "EZACIC04" on page 204 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 193. 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 on page 226).

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:		
<i>in-family</i>	H	PIC 9(4) BINARY
<i>in-port</i>	H	PIC 9(4) BINARY
<i>in-address</i>	F	PIC 9(8) BINARY
<i>dzero</i>	D	PIC X(8)
BUF	LEN or larger	LEN or larger
ERRNO	F	PIC 9(8) BINARY

SENDTO

RETCODE F PIC S9(8) BINARY

Parameter Values to Be Set by the Application

TOKEN

Must be set to 'TCPIPUVCVSTREAMS'

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 C. Return Codes" on page 253.

RETCODE

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

See "EZACIC04" on page 204 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 163. 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 on page 226).

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 'TCPIIUCVSTREAMS'

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'. It is only possible to set options for the socket level.

OPTNAME

Set this field to specify the option to be set, as shown below. See "SETSOCKOPT" on page 195 for a description of these settings.

Value Meaning

X'00000020'
SO-BROADCAST

X'00000080'
SO-LINGER

X'00000100'
SO-OOBINLINE

X'00000004'
SO-REUSEADDR

OPTVAL

For SO-BROADCAST, SO-OOBINLINE, and SO-REUSEADDR, set to a nonzero integer to enable the option specified in OPTNAME, and set to zero to disable the option. For SO-LINGER, see the equivalent OPTVAL parameter in "SETSOCKOPT" on page 195.

Parameter Values Returned to the Application

ERRNO

If RETCODE is negative, this contains an error number. Error numbers are described in "Appendix C. Return Codes" on page 253.

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 197. The format of the COBOL call for the SHUTDOWN function is:

```
CALL 'EZACICAL' USING TOKEN COMMAND S FZERO HOW ERRNO RETCODE.
```

SHUTDOWN

In assembler language, issue the macro call CALL EZACICAL, using standard assembler call syntax (for the call format, see page 226).

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 'TCPIUCVSTREAMS'

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 C. Return Codes" on page 253.

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 198. 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 226).

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 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 'TCPIUCVSTREAMS'

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 C. Return Codes" on page 253.

RETCODE

The socket number for 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 199. 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 on page 226).

Parameter Lengths in Assembler Language and COBOL

TOKEN	CL16	PIC X(16)
COMMAND	H	PIC 9(4) BINARY
HZERO	H	PIC 9(4) BINARY

TAKESOCKET

CLIENTID STRUCTURE:

<i>Domain</i>	F	PIC 9(8) BINARY
<i>Name</i>	CL8	PIC X(8)
<i>Task</i>	CL8	PIC X(8)
<i>Reserved</i>	CL20	PIC X(20)
L-DESC	F	PIC 9(8) BINARY
SOCKNO	F	PIC 9(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 'TCPIIUCVSTREAMS'

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 C. Return Codes" on page 253.

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 200. 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 226).

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 'TCPIIUCVSTREAMS'

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 C. Return Codes" on page 253.

RETCODE

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

See "EZACIC04" on page 204 for a subroutine that will translate EBCDIC data to ASCII.

Appendix C. 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.

Error Number from MVS TCP/IP

The error numbers in this section are returned by MVS TCP/IP.

Table 16. System Error Return Codes

Error Number	Message Name	Socket Type	Error Description
1	EPERM	All	Permission is denied. No owner exists.
1	EPERM	AD/Cycle C compiler	No owner exists.
1	EDOM	AD/Cycle C compiler	Argument too large.
2	ENOENT	All	The data set or directory was not found.
2	ENOENT	AD/Cycle C compiler	The data set or directory was not found.
2	ERANGE	AD/Cycle C compiler	The result is too large.
3	ESRCH	All	The process was not found.
4	EINTR	All	A system call was interrupted.
5	EIO	All	An I/O error occurred.
6	ENXIO	All	The device or driver was not found.
7	E2BIG	All	The argument list is too long.
8	ENOEXEC	All	An EXEC format error occurred.
9	EBADF	All	An incorrect socket descriptor was specified.
9	EBADF	Givesocket	The socket has already been given. The socket domain is not AF_INET.
9	EBADF	Select	One of the specified descriptor sets is an incorrect socket descriptor.
9	EBADF	Takesocket	The socket has already been taken.

Table 16. System Error Return Codes (continued)

Error Number	Message Name	Socket Type	Error Description
10	ECHILD	All	There are no children.
11	EAGAIN	All	There are no more processes.
12	ENOMEM	All	There is not enough storage.
13	EACCES	Takesocket	The other application (listener) did not give the socket to your application.
13	EACCES	Socket	Access denied. The client's ID is not in the OBEY file.
14	EFAULT	All	An incorrect storage address or length was specified.
15	ENOTBLK	All	A block device is required.
16	EBUSY	Givesocket	Listen has already been called for this socket.
17	EEXIST	All	The data set exists.
18	EXDEV	All	This is a cross-device link.
19	ENODEV	All	The specified device does not exist.
20	ENOTDIR	All	The specified directory is not a directory.
21	EISDIR	All	The specified directory is a directory.
22	EINVAL	All	An incorrect argument was specified.
22	EINVAL	Accept	Listen was not called for this socket.
22	EINVAL	Bind	The socket is already bound to an address.
22	EINVAL	Connect	The specified name length is incorrect.
22	EINVAL	Fcntl	Incorrect flags were specified.
22	EINVAL	Givesocket	An incorrect client ID was entered.
22	EINVAL	loctl	The request is incorrect or not supported.
22	EINVAL	Select	One of the fields in the time-out structure is incorrect.
22	EINVAL	Sendto	The target address length is incorrect for the specified address family.
22	EINVAL	Shutdown	The shutdown condition is not 0, 1, or 2.
22	EINVAL	Takesocket	The specified client ID is incorrect.

Table 16. System Error Return Codes (continued)

Error Number	Message Name	Socket Type	Error Description
23	ENFILE	All	Data set table overflow occurred.
24	EMFILE	Takesocke	The socket descriptor table is full.
25	ENOTTY	All	An incorrect device call was specified.
26	ETXTBSY	All	A text data set is busy.
27	EFBIG	All	The specified data set is too large.
28	ENOSPC	All	There is no space left on the device.
29	ESPIPE	All	An incorrect seek was attempted.
30	EROFS	All	The data set system is Read only.
31	EMLINK	All	There are too many links.
32	EPIPE	All	The connection is broken.
33	EDOM	All	The specified argument is too large.
34	ERANGE	All	The result is too large.
35	EWOULDBLOCK	Accept	The socket is in nonblocking mode and connections are not queued. This is not an error condition.
35	EWOULDBLOCK	Read Recvfrom	The socket is in nonblocking mode and read data is not available. This is not an error condition.
35	EWOULDBLOCK	Send Sendto Write	The socket is in nonblocking mode and buffers are not available.
36	EINPROGRESS	Connect	The socket is marked nonblocking and the connection cannot be completed immediately. This is not an error condition.
37	EALREADY	Connect	The socket is marked nonblocking and the previous connection has not been completed.
38	ENOTSOCK	All	A socket operation was requested on a nonsocket connection.
39	EDESTADDRREQ	All	A destination address is required.
40	EMSGSIZE	Sendto	The message is too long. The default is 8192 and the maximum is 32,767. The LARGEENVELOPEPOOLSIZE statement in PROFILE.TCPIP may restrict this value.
41	EPROTOTYPE	All	The specified protocol type is incorrect for this socket.

Table 16. System Error Return Codes (continued)

Error Number	Message Name	Socket Type	Error Description
42	ENOPROTOOPT	Getsockopt, Setsockopt	The socket option specified is incorrect or the level is not SOL_SOCKET.
43	EPROTONOSUPPORT	Socket	The specified protocol is not supported.
44	ESOCKTNOSUPPORT	All	The specified socket type is not supported.
45	EOPNOTSUPP	Accept, Givesocket	The selected socket is not a stream socket.
45	EOPNOTSUPP	Listen	The socket does not support the Listen call.
45	EOPNOTSUPP	Getibmopt, Setibmopt	The socket does not support this function call. This command is not supported for this function.
45	EOPNOTSUPP	Connect	Either a previous Connect failed or Connect was issued after Listen. This command is not supported for this function.
46	EPFNOSUPPORT	All, Getclientid, Takesocket	The specified protocol family is not especially supported or the specified domain for the client identifier is not AF_INET=2.
47	EAFNOSUPPORT	Bind, Connect, Socket	The specified address family is not supported by this protocol family.
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.
49	EADDRNOTAVAIL	bind	The specified address is incorrect for this host.
49	EADDRNOTAVAIL	connect	The calling host cannot reach the specified destination.
50	ENETDOWN	All	The network is down.
51	ENETUNREACH	Connect	The network cannot be reached.
52	ENETRESET	All	The network dropped a connection on a reset.
53	ECONNABORTED	All	The software caused a connection abend.
54	ECONNRESET	All	The connection to the destination host is not available.
55	ENOBUFS	All	No buffer space is available.

Table 16. System Error Return Codes (continued)

Error Number	Message Name	Socket Type	Error Description
55	ENOBUFS	Accept	Not enough buffer space is available to create the new socket.
55	ENOBUFS	Send Sendto Write	Not enough buffer space is available to send the new message.
55	ENOBUFS	Takesocket	There is a socket control block (SCB) or socket interface control block (SKCB) shortage in the TCPIP address space.
56	EISCONN	Connect	The socket is already connected.
57	ENOTCONN	All	The socket is not connected.
58	ESHUTDOWN	All	A Send cannot be processed after socket shutdown.
59	ETOOMANYREFS	All	There are too many references. A splice cannot be completed.
60	ETIMEDOUT	Connect	The connection timed out before it was completed.
61	ECONNREFUSED	Connect	The requested connection was refused.
62	ELOOP	All	There are too many symbolic loop levels.
63	ENAMETOOLONG	All	The file name is too long.
64	EHOSTDOWN	All	The host is down.
65	EHOSTUNREACH	All	There is no route to the host.
66	ENOTEMPTY	All	The directory is not empty.
67	EPROCLIM	All	There are too many processes in the system.
68	EUSERS	All	There are too many users on the system.
69	EDQUOT	All	The disk quota has been exceeded.
70	ESTALE	All	An old NFS** data set handle was found.
71	EREMOTE	All	There are too many levels of remote in the path.
72	ENOSTR	All	The device is not a stream device.
73	ETIME	All	The timer has expired.
74	ENOSR	All	There are no more stream resources.
75	ENOMSG	All	There is no message of the desired type.

Table 16. System Error Return Codes (continued)

Error Number	Message Name	Socket Type	Error Description
76	EBADMSG	All	The system cannot read the message.
77	EIDRM	All	The identifier has been removed.
78	EDEADLK	All	A deadlock condition has occurred.
79	ENOLCK	All	No record locks are available.
80	ENONET	All	The requested machine is not on the network.
81	ERREMOTE	All	The object is remote.
82	ENOLINK	All	The link has been severed.
83	EADV	All	An ADVERTISE error has occurred.
84	ESRMNT	All	An SRMOUNT error has occurred.
85	ECOMM	All	A communication error has occurred on a Send call.
86	EPROTO	All	A protocol error has occurred.
87	EMULTIHOP	All	A multihop address link was attempted.
88	EDOTDOT	All	A cross-mount point was detected. This is not an error.
89	EREMCHG	All	The remote address has changed.
90	ECONNCLOSED	All	The connection was closed by a peer.
113	EBADF	All	An incorrect socket descriptor was specified.
113	EBADF	Givesocket	The socket has already been given. The socket domain is not AF_INET.
113	EBADF	Select	One of the specified descriptor sets is an incorrect socket descriptor.
113	EBADF	Takesocket	The socket has already been taken.
121	EINVAL	All	An incorrect argument was specified.
121	EINVAL	Accept	Listen was not called for this socket.
121	EINVAL	Bind	The socket is already bound to an address.
121	EINVAL	Connect	The connect call contained a parameter that is not valid.
121	EINVAL	Fcntl	Incorrect flags were specified.

Table 16. System Error Return Codes (continued)

Error Number	Message Name	Socket Type	Error Description
121	EINVAL	Givesocke	An incorrect client ID was entered.
121	EINVAL	loctl	The request is incorrect or not supported.
121	EINVAL	Select	One of the fields in the time-out structure is incorrect.
121	EINVAL	Sendto	The target address length is incorrect for the specified address family.
121	EINVAL	Shutdown	The shutdown condition is not 0, 1, or 2.
121	EINVAL	Takesocke	The specified client ID is incorrect.
145	E2BIG	All	The argument list is too long.
1000	EIBMBADCALL	All	An incorrect socket-call constant was found in the IUCV header.
1001	EIBMBADPARM	All	An IUCV header error, type OTHER, has occurred.
1002	EIBMSOCKOUTOFRANGE	Socket	A socket number assigned by the client interface code is out of range.
1003	EIBMSOCKINUSE	Socket	A socket number assigned by the client interface code is already in use.
1004	EIBMIUCVERR	All	The request failed because of an IUCV error. This error is generated by the client stub code.
1005		All	The Offload host encountered an error.
1006		All	The Offload host was restarted.
1007		All	The Offload host went down.
1008	EIBMCONFLICT	All	This request conflicts with a request already queued on the same socket.
1009	EIBMCANCELLED	All	The request was cancelled by the CANCEL call.
1010	EIBMTHREADFAIL	All	Returned by the offload function when a beginthread failure occurs.
1011	EIBMADTCPNAME	All	A TCP/IP name that is not valid was detected.
1012	EIBMADREQUESTCODE	All	A request code that is not valid was detected.
1013	EIBMADCONNECTIONSTATE	All	A connection token that is not valid was detected; bad state.

Table 16. System Error Return Codes (continued)

Error Number	Message Name	Socket Type	Error Description
1014	EIBMUNAUTHORIZEDCALLER	All	An unauthorized caller specified an authorized keyword.
1015	EIBMADCONNECTIONMATCH	All	A connection token that is not valid was detected. There is no such connection.
1016	EIBMTCPABEND	All	An abend occurred when TCP/IP was processing this request.
1017	EIBMADMSGID	All	A message ID that is not valid was detected.
1018	EIBMNOCCB	All	A CCB could not be obtained for this request.
1019	EIBMNORESPONSEFOUND	All	No response was found for the request issued previously.
1020	EIBMINVALIDRESPONSE	All	The expected response was not received.
1021	EIBMNOACB	All	An ACB could not be obtained for this request.
1022	EIBMINVALIDKEYWORD	All	A keyword combination that is not valid was used.
1023	EIBMTERMERROR	All	A terminating error was encountered.
1024	EIBMNOADDRSPACENAME	All	An API common module could not find the user address space name.
1025	EIBMSRBMODE	All	A call was issued in SRB mode.
1026	EIBMINDDELETE	All	Delete requestor did not create the connection.
1027	EIBMINDSOCKET	All	A connection token that is not valid was detected. No such socket exists.
1028	EBMINDTCPCONNECTION	All	Connection terminated by TCP/IP. The token was invalidated by TCP/IP.
1029	EIBMADATTACH	All	Task failed to attach.
1030	EBMINDGETSTORAGE	All	Failed to obtain storage.
1031	EBMINDFREESTORAGE	All	Failed to free storage.
1032	EIBMCALLINPROGRESS	All	Another call was already in progress.
1033	EIBMCELLPOOLDELETED	All	Cell pool is marked for deletion.
1034	EBMINDSOCKETCALLNUM	All	The request exceeded the maximum number of requests allowed.

Table 16. System Error Return Codes (continued)

Error Number	Message Name	Socket Type	Error Description
1035	EIBMERRORUNKNOWN	All	An unknown reason code was received.
1036	EIBMNOACTIVETCP	All	TCP/IP is not installed or not active.
1037	EIBMINVTSRBUERDATA	All	The request control block contained data that is not valid.
1038	EIBMINVUSERDATA	All	The request control block contained user data that is not valid.
1039	EIBMADTCP	All	The client program was cancelled because TCP/IP is coming down.
1040	EIBMSELECTEXPOST	SELECTEX	SELECTEX passed an ECB that was already posted.
1041	EIBMADPOSTCODE	All	An application was given a bad post code. Received by an outstanding (blocked) socket when TCP/IP is stopped.
1042	EIBMDUPJOBNAME	INITAPI	An application attempted to create a connection to TCP/IP with a duplicate jobname and/or subtask name.
1043	EIBMCLIENVRVY	All	TCP/IP was reinitialized while an application was communicating with it. Recovery processing deleted the previous TCP/IP environmental information for the application. Subsequent INITAPI socket calls will succeed.
1044	EIBMINVALIDTCB	All except INITAPI	Socket call was issued from a TCB other than the TCB that issued the INITAPI call.
2001	EINVALIDRXSOCKETCALL	REXX	A syntax error occurred in the RXSOCKET parameter list.
2002	ECONSOLEINTERRUPT	REXX	A console interrupt occurred.
2003	ESUBTASKINVALID	REXX	The subtask ID is incorrect.
2004	ESUBTASKALREADYACTIVE	REXX	The subtask is already active.
2005	ESUBTASKALNOTACTIVE	REXX	The subtask is not active.
2006	ESOCKETNOTALLOCATED	REXX	The specified socket could not be allocated.
2007	EMAXSOCKETSREACHED	REXX	The maximum number of sockets has been reached.
2008	ESOCKETALREADYDEFINED	REXX	The socket has already been defined.

Table 16. System Error Return Codes (continued)

Error Number	Message Name	Socket Type	Error Description
2009	ESOCKETNOTDEFINED	REXX	The socket is not defined.
2010	ETCPIPSEVEREDPATH	REXX	TCP/IP has severed the IUCV path.
2011	EDOMAINSERVERFAILURE	REXX	A Domain Name Server failure occurred.
2012	EINVALIDNAME	REXX	An incorrect <i>name</i> was received from the TCP/IP server.
2013	EINVALIDCLIENTID	REXX	An incorrect <i>clientid</i> was received from the TCP/IP server.
2015	ENUCEXTFAILURE	REXX	An error occurred during NUCEXT processing.
2051	EFORMATERROR	REXX	A format error has occurred.
2052	ESERVERFAILURE	REXX	A server failure has occurred.
2053	EUNKNOWNHOST	REXX	An unknown host was encountered.
2054	EQUERYTYPE NOT IMPLEMENTED	REXX	The specified query type is not implemented.
2055	EQUERYREFUSED	REXX	The requested query was refused.
3001	EIUCVINVALIDPATH	REXX	An incorrect IUCV path ID was found.
3002	EIUCVPATHQUIESCED	REXX	The IUCV path is quiesced.
3003	EIUCVMGLIMITEXCEEDED	REXX	The IUCV message limit was exceeded.
3004	EIUCVNOPRIORITY	REXX	An IUCV priority message is not allowed on this path.
3005	EIUCVSMALLBUFFER	REXX	The IUCV buffer is too small.
3006	EIUCVBADFETCH	REXX	An IUCV fetch protection exception condition occurred.
3007	EIUCVBADADDRESS	REXX	An IUCV addressing exception condition occurred on an answer buffer.
3008	EIUCVBADMSGCLASS	REXX	An IUCV conflicting message class, path, or message ID was found.
3009	EIUCVPURGEDMSG	REXX	An IUCV message was purged.
3010	EIUCVBADMSGLENGTH	REXX	An IUCV negative message length was encountered.
3011	EIUCVPARTNER NOT LOGGED ON	REXX	The IUCV target user ID is not logged on.
3012	EIUCVPARTNER NOT INITIALIZED	REXX	The IUCV target user ID is not enabled for IUCV.

Table 16. System Error Return Codes (continued)

Error Number	Message Name	Socket Type	Error Description
3013	EIUCVPATHLIMITEXCEEDED	REXX	The IUCV path limit was exceeded.
3014	EIUCVPARTNERPATHLIMIT	REXX	The IUCV partner path limit was exceeded.
3015	EIUCVNOTAUTHORIZED	REXX	IUCV is not authorized.
3016	EIUCVINVALIDCPSYSTEMSERVICE	REXX	An incorrect IUCV system service was requested.
3018	EIUCVINVALIDMSGLIMIT	REXX	The specified IUCV message limit is incorrect.
3019	EIUCVBUFFERALREADYDECLARED	REXX	The requested IUCV buffer has already been declared.
3020	EIUCVPARTNERSEVERED	REXX	The IUCV partner has severed the path.
3021	EIUCVPARTNERNOPRMDATA	REXX	IUCV cannot accept the data in the parameter list.
3022	EIUCVSENDLISTINVALID	REXX	The IUCV SEND buffer list is incorrect.
3023	EIUCVINVALIDBUFFERLENGTH	REXX	A negative IUCV buffer length was found in answer-list.
3024	EIUCVINVALIDLISTLENGTH	REXX	The IUCV total list length is incorrect.
3025	EIUCVPRMMSGANSLISTCONFLICT	REXX	There is a PRMMMSG, or answer-list conflict.
3026	EIUCVBUFFERLISTNOTALIGNED	REXX	The IUCV buffer list is not aligned.
3027	EIUCVANSWERLISTNOTALIGNED	REXX	The IUCV answer-list is not aligned.
3028	EIUCVNOCONTROLBUFFER	REXX	The IUCV control buffer was not found.
3048	EIUCVFUNCTIONNOTSUPPORTED	REXX	The requested IUCV function is not supported.

Sockets Extended Return Codes

This section covers the error condition codes that are returned by the sockets extended interface. These errors are due to a problem detected by the sockets extended API.

Table 17. Sockets Extended Return Codes

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.

Table 17. Sockets Extended Return Codes (continued)

Error Code	Problem Description	System Action	Programmer's Response
10101	A STORAGE OBTAIN failed. Insufficient TCP/IP storage available.	End the call.	Increase MVS storage in the TCP/IP address space.
10102	Interface could not find the first subsystem.	End the call.	Call your MVS system programmer.
10103	VMCF or IUCV is not available. The VMCF CVT address is 0.	End the call.	Check VMCF or IUCV specification and IPL MVS.
10104	The requested function is not valid.	End the call.	Correct the call and retry.
10105	The number of parameters is incorrect or the return code address is negative.	End the call.	Correct the parameter list.
10106	The high-order bit in the parameter list is not set for the last parameter.	End the call.	Correct the parameter list.
10107	An incorrect subtask ID was obtained from a GETCLIENTID call.	End the call.	Correct the method used to get the subtask ID.
10108	The first call from TCP/IP was not INITAPI or TAKESOCKET.	End the call.	Change the first TCP/IP call to INITAPI or TAKESOCKET.
10109	A duplicate subtask ID occurred in the path ownership table.	End the call.	Correct the subtask ID on INITAPI or TAKESOCKET.
10110	LOAD of EZASOH03 failed.	End the call.	Call the IBM Software Support Center.
10111	The IUCVMINI SET macro detected errors in IUCVMULT or EZACICMT.	End the call.	Call the IBM Software Support Center.
10112	The IUCVMCOM CONNECT macro detected errors.	End the call.	Start TCP/IP or, if TCP/IP is running, ensure that the startup name matches the TCP/IP job name. You might have used a duplicate job name.
10113	The IUCVMCOM SEND macro for the initial message detected errors.	End the call.	Call the IBM Software Support Center.
10114	During an IUCV CONNECT, IUCV sent an unsolicited message to an active task.	Disable the subtask and return an error to the user.	Retry the task.

Table 17. Sockets Extended Return Codes (continued)

Error Code	Problem Description	System Action	Programmer's Response
10115	The option length for a GETSOCKOPT call is incorrect.	Return an error to the user.	Correct the GETSOCKOPT call.
10116	After a CONNECT, IUCV sent an unsolicited message to an active task.	Disable the interface.	Retry the task.
10117	The return code address in your parameter list is zero.	No action.	Correct the return code address.
10118	IUCV sent an unexpected interrupt.	Wait for TRUE to post the subtask.	Retry the task.
10119	TCP/IP severed the IUCV connection, IPUSER=IUCVCHECKRC is set.	End the call.	Call your system programmer.
10120	TCP/IP severed the IUCV connection, IPUSER=SHUTTINGDOWN is set.	End the call.	Change the program to handle the SHUTDOWN condition.
10121	TCP/IP severed the IUCV connection, IPUSER=BAD PATHID is set.	End the call.	Check the path ID.
10122	TCP/IP severed the IUCV connection, IPUSER=NULL SAVED NAME is set.	End the call.	Call the IBM Software Support Center.
10123	TCP/IP severed the IUCV connection, IPUSER=BAD INIT MSG LEN is set.	End the call.	If your program issues an INIT command (0), check the parameters.
10124	TCP/IP severed the IUCV connection, IPUSER=REQUIREDCONSTANT is set.	End the call.	Correct parameter 3 in your INIT call.
10125	TCP/IP severed the IUCV connection, IPUSER=BAD API TYPE is set.	End the call.	Correct parameter 5 in your INIT call.
10126	TCP/IP severed the IUCV connection, IPUSER=RESTRICTED is set.	End the call.	Call your system programmer.
10127	TCP/IP severed the IUCV connection, IPUSER=NO MORE CCBS is set.	End the call.	Call your system programmer.

Table 17. Sockets Extended Return Codes (continued)

Error Code	Problem Description	System Action	Programmer's Response
10128	TCP/IP severed the IUCV connection, IPUSER=NO CCB is set.	End the call.	Call your system programmer.
10129	TCP/IP severed the IUCV connection, IPUSER=KILL nnn is set.	End the call.	Call your system programmer.
10130	TCP/IP severed the IUCV connection, IPUSER=UNKNOWN REASON CODE is set.	End the call.	Call your system programmer.
10131	During a quiesce, IUCV sent an unsolicited message to an active task.	End the call.	Call your system programmer.
10132	During a quiesce, IUCV sent an unsolicited message to an inactive task.	Wait for the TRUE to post the subtask.	Call your system programmer.
10133	IUCV sent an unsolicited priority completion message interrupt to an active task.	End the call.	Call your system programmer.
10134	IUCV sent an unsolicited priority completion message interrupt to an inactive task.	Wait for TRUE to post the subtask.	Call your system programmer.
10135	An error occurred in an IUCV SEVER function.	End the call.	Call your system programmer.
10136	IUCV sent an unsolicited nonpriority completion message interrupt to an inactive task.	Wait for TRUE to post the subtask.	Call your system programmer.
10137	IUCV sent an unsolicited priority pending message interrupt to an active task.	End the call.	Call your system programmer.
10138	IUCV sent an unsolicited priority pending message interrupt to an inactive task.	Wait for TRUE to post the subtask.	Call your system programmer.
10139	IUCV sent an unsolicited external interrupt to the subtask.	Write message EZY1281E to the system console. Return an error code to the caller.	Call your system programmer.

Table 17. Sockets Extended Return Codes (continued)

Error Code	Problem Description	System Action	Programmer's Response
10140	IUCV sent a solicited nonpriority pending message interrupt to the subtask.	Return an error code to the caller.	Call your system programmer.
10141	IUCV sent an unsolicited, nonpriority pending message interrupt to the subtask.	Wait for TRUE to post the subtask.	Call your system programmer.
10142	Errors were found in the parameter list for an ACCEPT call.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the ACCEPT call. You might have incorrect sequencing of socket calls.
10143	Errors were found in the parameter list for a BIND call.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the BIND call. You might have incorrect sequencing of socket calls.
10144	Errors were found in the parameter list for a CLOSE call.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the CLOSE call. You might have incorrect sequencing of socket calls.
10145	Errors were found in the parameter list for a CONNECT call.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the CONNECT call. You might have incorrect sequencing of socket calls.
10146	Errors were found in the parameter list for an FCNTL call.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the FCNTL call. You might have incorrect sequencing of socket calls.
10147	Errors were found in the parameter list for a GETCLIENTID call.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the GETCLIENTID call. You might have incorrect sequencing of socket calls.
10148	Errors were found in the parameter list for a GETHOSTID call.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the GETHOSTID call. You might have incorrect sequencing of socket calls.

Table 17. Sockets Extended Return Codes (continued)

Error Code	Problem Description	System Action	Programmer's Response
10149	Errors were found in the parameter list for a GETHOSTNAME call.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the GETHOSTNAME call. You might have incorrect sequencing of socket calls.
10150	Errors were found in the parameter list for a GETPEERNAME call.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the GETPEERNAME call. You might have incorrect sequencing of socket calls.
10151	Errors were found in the parameter list for a GETSOCKNAME call.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the GETSOCKNAME call. You might have incorrect sequencing of socket calls.
10152	Errors were found in the parameter list for a GETSOCKOPT call.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the GETSOCKOPT call. You might have incorrect sequencing of socket calls.
10153	Errors were found in the parameter list for a GIVESOCKET call.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the GIVESOCKET call. You might have incorrect sequencing of socket calls.
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 zero.	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.
10157	The parameter list for an IOCTL call is incorrect.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the IOCTL call. You might have incorrect sequencing of socket calls.

Table 17. Sockets Extended Return Codes (continued)

Error Code	Problem Description	System Action	Programmer's Response
10158	The parameter list for a LISTEN call is incorrect.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the LISTEN call. You might have incorrect sequencing of socket calls.
10159	A zero 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.
10160	A READ call specified a data length greater than 32KB.	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 zero.	End the call.	Correct the program.
10162	The parameter list for a READ call is incorrect.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the READ call. You might have incorrect sequencing of socket calls.
10163	A 0 or negative data length was found for a RECV, RECVFROM, or RECVMSG call.	Disable the subtask for interrupts. Sever the IUCV path. Return an error code to the caller.	Correct the data length.
10164	The data length for a RECV, RECVFROM, or RECVMSG call exceeded the maximum.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the data length.
10165	Incorrect data was specified for an IUCV SEVER.	End the call.	Call the IBM Software Support Center.
10166	The parameter list for a RECV, RECVFROM, or RECVMSG call is incorrect.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the READ call. You might have incorrect sequencing of socket calls.

Table 17. Sockets Extended Return Codes (continued)

Error Code	Problem Description	System Action	Programmer's Response
10167	The descriptor set size for a SELECT or SELECTEX call is less than or equal to zero.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the SELECT or SELECTEX call. You might have incorrect sequencing of socket calls.
10168	The descriptor set size <i>in bytes</i> for a SELECT or SELECTEX call is greater than 252. A number greater than the maximum number of allowed sockets (2000 is maximum) has been specified.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the descriptor set size.
10169	The parameter list for a SELECT or SELECTEX call is incorrect.	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.
10170	A zero 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.
10171	The data length for a SEND call exceeded the maximum.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the data length in the SEND call.
10172	An error occurred while processing an IUCV SEVER command.	End the program.	Call your system programmer.
10173	The parameter list for a SEND call is incorrect.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the SEND call. You might have incorrect sequencing of socket calls.
10174	A zero 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.
10175	The data length for a SENDTO call exceeded the maximum.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the data length in the SENDTO call.

Table 17. Sockets Extended Return Codes (continued)

Error Code	Problem Description	System Action	Programmer's Response
10176	A nonzero return code was received from an IUCV SEVER command.	Return an error to the user.	Call your system programmer.
10177	The parameter list for a SENDTO call is incorrect.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the SENDTO call. You might have incorrect sequencing of socket calls.
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.
10180	The parameter list for a SETSOCKOPT call is incorrect.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the SETSOCKOPT call. You might have incorrect sequencing of socket calls.
10181	The parameter list for a SHUTDOWN call is incorrect.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the SHUTDOWN call. You might have incorrect sequencing of socket calls.
10182	The parameter list for a SOCKET call is incorrect.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the SOCKET call. You might have incorrect sequencing of socket calls.
10183	The parameter list for a TAKESOCKET call is incorrect.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the TAKESOCKET call. You might have incorrect sequencing of socket calls.
10184	A data length of zero 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.

Table 17. Sockets Extended Return Codes (continued)

Error Code	Problem Description	System Action	Programmer's Response
10185	The data length for a WRITE call exceeded the maximum length.	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.
10187	The third parameter for IUCV SEVER ALL is incorrect.	End the call.	Call the IBM Software Support Center.
10188	Errors were found in the parameter list for a WRITE call.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the WRITE call. You might have incorrect sequencing of socket calls.
10189	Errors were found in the parameter list for a LASTERRNO call.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the LASTERRNO call. You might have incorrect sequencing of socket calls.
10190	The GETHOSTNAME option length is less than 24 or greater than the maximum length.	Disable the subtask for interrupts. Return an error code to the caller.	Correct the length parameter.
10191	IUCV returned an error code.	End the call.	Notify operations.
10192	The function is available.	No action.	No action.
10193	The GETSOCKOPT option length is less than the minimum or greater than the maximum length.	End the call.	Correct the length parameter.
10194	The IUCV SEVER ALL user data is not binary zeros.	Bypass the call.	Call the IBM Software Support Center.
10195	An IUCV SEVER path ID error was detected.	Bypass the call.	Call the IBM Software Support Center.
10196	An error occurred in the IUCV CLEAR function.	Bypass the call.	Call the IBM Software Support Center.
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.

Table 17. Sockets Extended Return Codes (continued)

Error Code	Problem Description	System Action	Programmer's Response
10198	The maximum number of sockets specified for an INITAPI exceeds 2000.	Return to the user.	Correct the INITAPI call.
10199	EZACICMT cannot be loaded.	Bypass the call.	Correct the SYSLIB concatenation and retry.
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 information in <i>TCP/IP for MVS: Application Programming Interface Reference</i> .
10201	The seventh parameter, ERRNO address, for an IOCTL call is incorrect.	End the call.	Correct the parameter list.
10202	The RETARG parameter in the IOCTL call is zero.	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.
10204	The requested socket number exceeds 4095.	End the call.	Correct the requested socket number.
10205	The requested socket number is a duplicate.	End the call.	Correct the requested socket number.
10206	The socket descriptor table is full.	End the call.	Issue an INITAPI call to request more descriptors.
10207	A SYNC call was issued before an ECB was specified.	End the call.	Issue an ECB before the SYNC call.
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.

Table 17. Sockets Extended Return Codes (continued)

Error Code	Problem Description	System Action	Programmer's Response
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.
10216	TCP/IP has terminated.	End the call.	Perform recovery processing for a TCP/IP failure.
10217	The connection to TCP/IP has been severed.	End the call.	Perform recovery processing for the connection. This will include re-establishing communication with TCP/IP.
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.
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.
10220	Trying to close socket that has not been allocated.	End the call.	Correct the S parameter on the CLOSE 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 17. Sockets Extended Return Codes (continued)

Error Code	Problem Description	System Action	Programmer's Response
10224	The IOVCNT parameter is less than or equal to zero, 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	Invalid COMMAND parameter specified for a GETIBMOPT call.	End the call.	Correct the COMMAND parameter of the GETIBMOPT call.
10228	The CANCEL call was issued on a non-asynchronous connection.	End the call.	CANCEL is valid only for connections established with the ASYNC parameter on the INITAPI call.
10228	For CICS, the maximum number of sockets specified for an INITAPI exceeds 255.	End the call.	Correct the INITAPI 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.
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.
10333	An error was detected in the asynchronous exit routine.	End the call.	Perform necessary asynchronous exit recovery where applicable.
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.

Table 17. Sockets Extended Return Codes (continued)

Error Code	Problem Description	System Action	Programmer's Response
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 CICS Sockets Interface is not in operation.	End the call	Start the CICS Sockets Interface before executing this call.

Appendix D. CICS Messages

Messages originating from EZACIC01	277
Messages originating from EZACIC02	280
Messages originating from EZACIC03	289
Messages Originating from EZACIC21	289
Messages Originating from EZACIC22	292
Messages Originating from EZACIC25	293

Messages originating from EZACIC01

This section contains messages originating from EZACIC01.

EZY1260I mm/dd/yy hh:mm:ss EZACIC03
ATTACH FAILED GPR15=xxxxxxx
ERRNO=xxxxxxx TRAN=tran
TASK=cicstask

Explanation: An ATTACH for an MVS subtask has failed. The reason code is in GPR 15.

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)

EZY1262I mm/dd/yy hh:mm:ss GWA ADDRESS
INVALID UEPGAA=xxxxxxx
ERRNO=xxxxxxx 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)

EZY1263I mm/dd/yy hh:mm:ss TIE ADDRESS
INVALID UEPGAA=xxxxxxx
ERRNO=xxxxxxx 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)

EZY1264I mm/dd/yy hh:mm:ss **FLAG WORD ADDRESS INVALID UEPFLAGS=**
xxxxxxx **ERRNO=xxxxxxx TRAN=tran**
TASK=cicstask

Explanation: The Task Related User Exit (TRUE) detected an invalid flag word 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)

EZY1265I mm/dd/yy hh:mm:ss **CICS VERSION UNSUPPORTED GWACIVRM=xxxx**
ERRNO=xxxxxxx 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.

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)

EZY1267I mm/dd/yy hh:mm:ss **ROUTING TASK FUNCTION INVALID UERTIFD=xx**
ERRNO=xxxxxxx TRAN=tran
TASK=cicstask

Explanation: The Task Related User Exit (TRUE) detected an invalid routing task function.

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)

EZY1268I mm/dd/yy hh:mm:ss **SAVE AREA ADDRESS INVALID UEPHMA=**
xxxxxxx **ERRNO=xxxxxxx TRAN=tran**
TASK=cicstask

Explanation: The Task Related User Exit (TRUE) detected an invalid save area address.

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)

EZY1269I mm/dd/yy hh:mm:ss **PARAM LIST ADDRESS INVALID GPR1= xxxxxxxx**
ERRNO=xxxxxxx 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.

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)

EZY1270I mm/dd/yy hh:mm:ss **PARAM nn ADDRESS INVALID ADDRESS=**
xxxxxxx **ERRNO=xxxxxxx 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.

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)

EZY1271I mm/dd/yy hh:mm:ss TOKERR=xxxxxxx
ERRNO=xxxxxxx 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.

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)

EZY1272I mm/dd/yy hh:mm:ss INVALID
SOCKET/FUNCTION CALL FUNCTION=
xxxx ERRNO=xxxxxxx TRAN=tran
TASK=cicstask

Explanation: A call to EZASOKET specified in invalid function.

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)

EZY1273I mm/dd/yy hh:mm:ss IUCV SOCK/FUNC
TABLE INVALID FUNCTION= xxxx
ERRNO=xxxxxxx TRAN=tran
TASK=cicstask

Explanation: A call to EZACICAL specified in invalid function.

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)

EZY1274I mm/dd/yy hh:mm:ss INCORRECT
EZASOKET PARM COUNT FUNCTION=
xxxx ERRNO=xxxxxxx TRAN=tran
TASK=cicstask

Explanation: A call to EZASOKET specified in invalid number of parameters.

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)

EZY1275I mm/dd/yy hh:mm:ss MONITOR CALLS
NOT SUPPORTED UERTFID=xx
ERRNO=xxxxxxx TRAN=tran
TASK=cicstask

Explanation: The Task Related User Exit (TRUE) detected a monitor call which is not supported for this version of CICS.

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)

EZY1276I mm/dd/yy hh:mm:ss EDF CALLS NOT
SUPPORTED UERTFID=xx
ERRNO=xxxxxxx TRAN=tran
TASK=cicstask

Explanation: The Task Related User Exit (TRUE) detected a EDF (Execute Diagnostic Facility) call. This TRUE does not support EDF calls.

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=xxxxxxx
ERRNO=xxxxxxx 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)

EZY1278I mm/dd/yy hh:mm:ss EZACIC03
DETACH SUCCESSFUL TCB
ADDRESS= xxxxxxxx ERRNO=xxxxxxx
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)

EZY1279I 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.

Messages originating from EZACIC02

EZY1218I mm/dd/yy hh:mm:ss PROGRAM
progname DISABLED TRANID= xxxx
PARTNER INET ADDR=xxx.xxx.xxx.xxx
PORT=xxxxxx

Explanation: The Listener checked the status of the program associated with the transaction. It was not enabled.

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

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)

EZY1280I 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

Destination:

EZY1278I 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.

Module: EZACIC02

Destination: LISTENER

EZY1279I mm/dd/yy hh:mm:ss GETCLIENTID
CALL FAILURE TRANSACTION=*tran*
TASKID= *cicstask* ERRNO=xxxxxx

Explanation: Listener transaction *tran* experienced a failure on the GETCLIENTID call.

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

EZY1285I 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

EZY1286I 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.

Module: EZACIC02

Destination: LISTENER

EZY1288I 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: Contact the IBM Software Support Center.

System Programmer Response: None.

Module: EZACIC02

Destination: LISTENER

EZY1289I 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 *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

EZY1292I 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.

System Programmer Response: None.

Module: EZACIC02

Destination: LISTENER

EZY1293I mm/dd/yy hh:mm:ss INITAPI CALL FAILURE TRANSACTION=*tran* TASKID= *cicstask* ERRNO=xxxxxx

Explanation: Listener transaction *tran* experienced a failure on the INITAPI call.

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

EZY1294I mm/dd/yy hh:mm:ss SOCKET CALL FAILURE TRANSACTION= *tran* TASKID= *cicstask* ERRNO= xxxxx

Explanation: Listener transaction *tran* experienced a failure on the SOCKET call.

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

EZY1295I mm/dd/yy hh:mm:ss BIND CALL FAILURE TRANSACTION= *tran* TASKID= *cicstask* ERRNO= xxxxx

Explanation: Listener transaction *tran* experienced a failure on the BIND call.

System Action: Listener transaction *tran* terminates.

User Response: Use the ERRNO value to determine the cause of the failure.

Note: An ERRNO=48 could indicate that the port is not reserved in *hlq*.TCP/IP.PROFILE.

System Programmer Response: None.

Module: EZACIC02

Destination: LISTENER

EZY1296I mm/dd/yy hh:mm:ss LISTEN CALL FAILURE TRANSACTION= *tran* TASKID= *cicstask* ERRNO= xxxxx

Explanation: Listener transaction *tran* experienced a failure on the LISTEN call.

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

EZY1298I mm/dd/yy hh:mm:ss CLOSE FAILURE TRANID= *tran* TASKID= *cicstask* ERRNO= xxxxx

Explanation: Listener transaction *tran* experienced a failure on the CLOSE call.

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

EZY1299I mm/dd/yy hh:mm:ss **SELECT CALL FAILURE TRANSACTION= tran TASKID= xxxxx ERRNO= xxxxx**

Explanation: Listener transaction *tran* experienced a failure on the SELECT call.

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

EZY1300I mm/dd/yy hh:mm:ss **READ FAILURE TRANSID= tran TASKID= cicstran ERRNO= xxxxxx INET ADDR=xxx.xxx.xxx.xxx PORT=xxxxxx**

Explanation: Listener transaction *tran* experienced a failure on the READ call.

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

EZY1301I mm/dd/yy hh:mm:ss **READ CALL RECEIVED NULL DATA TRANSID= tran PARTNER INET ADDR=xxx.xxx.xxx.xxx PORT=xxxxxx**

Explanation: Listener transaction *tran* received null data from the client. Either the client issued a close socket call or it issued a send with a length of zero.

System Action: Listener transaction *xxxx* 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=xxx.xxx.xxx.xxx PORT=xxxxxx**

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.

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 tran PARTNER INET ADDR=xxx.xxx.xxx.xxx PORT=xxxxxx**

Explanation: The started server transaction did not perform the takesocket within the timeout value specified for this listener in the CICS Sockets Configuration File.

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 tran PARTNER INET ADDR=xxx.xxx.xxx.xxx PORT=xxxxxx**

Explanation: The listener received data from the client after the end of the transaction input message.

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

EZY1305I mm/dd/yy hh:mm:ss **UNEXPECTED EXCEPTION EVENT TRANS tran PARTNER INET ADDR=xxx.xxx.xxx.xxx PORT=xxxxxx**

Explanation: The listener received an exception event on this connection other than the event showing a successful takesocket was issued by the server.

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

EZY1306I mm/dd/yy hh:mm:ss SECURITY EXIT
mmmmmmmm IS NOT DEFINED
TRANID= *tran* TASKID=xxxxxxx

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

EZY1307I mm/dd/yy hh:mm:ss MAXIMUM # OF
SOCKETS USED TRANS= *tran* TASKID=
cicstask ERRNO= xxxxx

Explanation: All of the sockets allocated to listener transaction xxxx are in use.

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

EZY1308I mm/dd/yy hh:mm:ss ACCEPT CALL
FAILURE TRANSACTION= *tran*
TASKID= *cicstask* ERRNO= xxxxx

Explanation: Listener transaction *tran* experienced a failure on the ACCEPT call.

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

EZY1309I mm/dd/yy hh:mm:ss GIVESOCKET
FAILURE TRANS *tran* TASKID=
xxxxxxx ERRNO= xxxxxx INET
ADDR=xxx.xxx.xxx.xxx PORT=xxxxxx

Explanation: Listener transaction *tran* experienced a failure on the GIVESOCKET call.

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

EZY1310I mm/dd/yy hh:mm:ss IC VALUE NOT
NUMERIC TRANID= *tran* PARTNER
INET ADDR=xxx.xxx.xxx.xxx
PORT=xxxxxx

Explanation: The interval specified in the transaction input message contains one or more non-numeric characters.

System Action: The interval is ignored, i.e. 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

EZY1311I mm/dd/yy hh:mm:ss CICS TRANID *tran*
NOT AUTHORIZED PARTNER INET
ADDR=xxx.xxx.xxx.xxx PORT=xxxxxx

Explanation: The transaction name specified in the transaction input message is not RSL authorized.

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

EZY1312I 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

EZY1313I mm/dd/yy hh:mm:ss LISTENER NOT
AUTHORIZED TO ACCESS SECURITY
EXIT mmmmmmm TRANID= tran
TASKID=xxxxxxx

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

EZY1314I mm/dd/yy hh:mm:ss SECURITY EXIT
mmmmmmmm IS DISABLED TRANID=
tran TASKID=xxxxxxx

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

EZY1315I mm/dd/yy hh:mm:ss INVALID TRANSID
tran PARTNER INET
ADDR=xxx.xxx.xxx.xxx PORT=xxxxxx

Explanation: The transaction input message from the client specified transaction *tran* but this transaction is not defined to CICS.

System Action: Listener transaction *tran* 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

EZY1316I mm/dd/yy hh:mm:ss TRANSID tran IS
DISABLED PARTNER INET
ADDR=xxx.xxx.xxx.xxx PORT=xxxxxx

Explanation: Transaction *xxxxxxx* 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 server transaction.

System Programmer Response: None.

Module: EZACIC02

Destination: LISTENER

EZY1317I mm/dd/yy hh:mm:ss TRANSID tran IS
NOT AUTHORIZED PARTNER INET
ADDR=xxx.xxx.xxx.xxx PORT=xxxxxx

Explanation: Listener transaction *tran* is not authorized to start the transaction name specified in the transaction input message.

System Action: The transaction is not started.

User Response: Authorize listener transaction *tran* to start the transaction.

System Programmer Response: None.

Module: EZACIC02

Destination: LISTENER

EZY1318I 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

EZY1319I mm/dd/yy hh:mm:ss QIDER FOR TD
DESTINATION qqqq PARTNER INET
ADDR=xxx.xxx.xxx.xxx PORT=xxxxxx

Explanation: The listener transaction was unable to start CICS transaction through transient data queue qqqq. DFHRESP was QIDERR.

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

EZY1320I mm/dd/yy hh:mm:ss I/O ERROR FOR
TD DESTINATION xxxx PARTNER INET
ADDR=xxx.xxx.xxx.xxx PORT=xxxxxx

Explanation: Listener transaction xxxx was unable to start CICS transaction through transient data queue xxxx. DFHRESP was IOERR.

System Action: Listener transaction xxxx continues.

User Response: Contact the CICS Systems Programmer.

System Programmer Response: None.

Module: EZACIC02

Destination: LISTENER

EZY1321I mm/dd/yy hh:mm:ss LENGTH ERROR
FOR TD DESTINATION xxxx PARTNER
INET ADDR=xxx.xxx.xxx.xxx
PORT=xxxxxx

Explanation: Listener transaction xxxx was unable to start CICS transaction through transient data queue xxxx. DFHRESP was LENGERR.

System Action: Listener transaction xxxx 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

EZY1322I mm/dd/yy hh:mm:ss TD DESTINATION
xxxx DISABLED PARTNER INET
ADDR=xxx.xxx.xxx.xxx PORT=xxxxxx

Explanation: Listener transaction xxxx was unable to start CICS transaction through transient data queue xxxx. DFHRESP was DISABLED.

System Action: Listener transaction xxxx continues.

User Response: Use CEMT to enable the destination.

System Programmer Response: None.

Module: EZACIC02

Destination: LISTENER

EZY1323I mm/dd/yy hh:mm:ss TD DESTINATION
xxxx OUT OF SPACE PARTNER INET
ADDR=xxx.xxx.xxx.xxx PORT=xxxxxx

Explanation: Listener transaction xxxx was unable to start CICS transaction through transient data queue xxxx. DFHRESP was NOSPSPACE.

System Action: Listener transaction xxxx continues.

User Response: Contact the CICS Systems Programmer.

System Programmer Response: Allocate space for this Transient Data Queue.

Module: EZACIC02

Destination: LISTENER

EZY1324I mm/dd/yy hh:mm:ss TD START FAILED
QUEUE ID= xxxx PARTNER INET
ADDR=xxx.xxx.xxx.xxx PORT=xxxxxx

Explanation: Listener transaction xxxx was unable to start CICS transaction through transient data queue xxxx. DFHRESP was 99.

System Action: Listener transaction xxxx 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= xxxx
PARTNER INET ADDR=xxx.xxx.xxx.xxx
PORT=xxxxxx

Explanation: Listener transaction xxxx was able to start CICS transaction xxxx transient data queue xxxx.

System Action: Listener transaction xxxx continues.

User Response: None.

System Programmer Response: None.

Module: EZACIC02

Destination: LISTENER

EZY1326I mm/dd/yy hh:mm:ss START I/O ERROR
TRANID= xxxx PARTNER INET
ADDR=xxx.xxx.xxx.xxx PORT=xxxxxx

Explanation: Listener transaction xxxx was unable to start CICS transaction xxxx. DFHRESP was IOERR.

System Action: Listener transaction xxxx 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

EZY1327I mm/dd/yy hh:mm:ss START
TRANSACTION ID xxxx INVALID
PARTNER INET ADDR=xxx.xxx.xxx.xxx
PORT=xxxxxx

Explanation: Listener transaction xxxx was unable to start CICS transaction xxxx. DFHRESP was TRANSIDERR.

System Action: Listener transaction xxxx 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

EZY1328I mm/dd/yy hh:mm:ss START
TRANSACTION ID xxxx NOT
AUTHORIZED PARTNER INET
ADDR=xxx.xxx.xxx.xxx PORT=xxxxxx

Explanation: Listener transaction xxxx was unable to start CICS transaction xxxx. DFHRESP was NOTAUTH.

System Action: Listener transaction xxxx 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 xxxx to start this transaction.

System Programmer Response: None.

Module: EZACIC02

Destination: LISTENER

EZY1329I mm/dd/yy hh:mm:ss START FAILED
(99) TRANSID= xxxx PARTNER INET
ADDR=xxx.xxx.xxx.xxx PORT=xxxxxx

Explanation: Listener transaction xxxx was unable to start CICS transaction xxxx. DFHRESP was 99.

System Action: Listener transaction xxxx continues.

User Response: Contact the CICS Systems Programmer.

System Programmer Response: Check the transaction definition in RDO. Look for associated messages which might indicate why the transaction would not start.

Module: EZACIC02

Destination: LISTENER

EZY1330I mm/dd/yy hh:mm:ss IC START
SUCCESSFUL TRANID= xxxx
PARTNER INET ADDR=xxx.xxx.xxx.xxx
PORT=xxxxxx

Explanation: Listener transaction xxxx was able to start CICS transaction xxxx.

System Action: Listener transaction xxxx continues.

User Response: None.

System Programmer Response: None.

Module: EZACIC02

Destination: LISTENER

EZY1331I mm/dd/yy hh:mm:ss IC START I/O
ERROR TRANID= xxxx PARTNER INET
ADDR=xxx.xxx.xxx.xxx PORT=xxxxxx

Explanation: Listener transaction xxxx was unable to start CICS transaction xxxx. DFHRESP was IOERR.

System Action: Listener transaction xxxx continues.

User Response: Contact the CICS Systems Programmer.

System Programmer Response: Look for other messages that provide specific information on the I/O error and correct the problem.

Module: EZACIC02

Destination: LISTENER

EZY1332I mm/dd/yy hh:mm:ss IC START INVALID
REQUEST TRANID= xxxx PARTNER
INET ADDR=xxx.xxx.xxx.xxx
PORT=xxxxxx

Explanation: Listener transaction xxxx was unable to start CICS transaction xxxx. DFHRESP was INVREQ.

System Action: Listener transaction xxxx continues.

User Response: Contact the IBM Software Support Center.

System Programmer Response: None.

Module: EZACIC02

Destination: LISTENER

EZY1333I mm/dd/yy hh:mm:ss IC START FAILED
(99) TRANID= xxxx PARTNER INET
ADDR=xxx.xxx.xxx.xxx PORT=xxxxxx

Explanation: Listener transaction xxxx was unable to start CICS transaction xxxx. DFHRESP was 99.

System Action: Listener transaction xxxx continues.

User Response: Contact the CICS Systems Programmer.

System Programmer Response: Check the RDO definition of the transaction. Contact the IBM Software Support Center.

Module: EZACIC02

Destination: LISTENER

EZY1334E mm/dd/yy hh:mm:ss INVALID USER
TRANID=xxxxx PARTNER INET ADDR =
xxx.xxx.xxx.xxx PORT = xxxxxx

Explanation: This message is issued only for CICS 4.1 and above. It indicates that the user security exit has given the Listener an invalid USERID field.

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

EZY1335I mm/dd/yy hh:mm:ss WRITE FAILED
ERRNO= xxxxxxxx TRANID= xxxxx.
PARTNER INET ADDR=xxx.xxx.xxx.xxx
PORT=xxxxxx

Explanation: Listener transaction xxxx had a failure on a WRITE command.

System Action: Listener transaction xxxx continues.

User Response: Use the ERRNO value to determine the cause of the failure.

System Programmer Response: None.

Module: EZACIC02

Destination: LISTENER

EZY1336I mm/dd/yy hh:mm:ss **TAKESOCKET FAILURE TRANS** xxxx **TASKID=** *cicstran*
ERRNO= xxxxxx **INET ADDR=**xxx.xxx.xxx.xxx **PORT=**xxxxxx

Explanation: Listener transaction xxxx had a failure on a TAKESOCKET command.

System Action: Listener transaction xxxx continues.

User Response: Use the ERRNO value to determine the cause of the failure.

System Programmer Response: None.

Module: EZACIC02

Destination: LISTENER

EZY1337I 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

EZY1338I mm/dd/yy hh:mm:ss **PROGRAM xxxxxxxx NOT FOUND TRANID=** xxxx
PARTNER INET ADDR=xxx.xxx.xxx.xxx **PORT=**xxxxxx

Explanation: The Listener checked the status of the program associated with the transaction. It was not found.

Messages originating from EZACIC03

EZY1282I 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.

System Action: Listener continues.

User Response: If the transaction id 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

Destination:

EZY1334E mm/dd/yy hh:mm:ss **INVALID USERID TRANID=**xxxx **PARTNER INET ADDR=**xxx.xxx.xxx.xxx **PORT=**xxxxxx

Explanation: The user exit returned a USERID that the security system does not recognize.

System Action: The transaction does not start.

User Response: Do one of the following:

- Correct the security exit so it returns a valid user ID or
- Update the security system to accept the user ID

System Programmer Response: None

Module: EZACIC02

Messages Originating from EZACIC21

EZY1220I 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.

Module: EZACIC21
Destination: INITIALIZATION

EZY1221I 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.

Module: EZACIC21
Destination: INITIALIZATION

EZY1222I mm/dd/yy hh:mm:ss CICS/SOCKETS
REGISTRATION FAILURE RETURN
code= *codexx*

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: None.

Module: EZACIC21
Destination: INITIALIZATION

EZY1223I mm/dd/yy hh:mm:ss CICS/SOCKETS
ATTACH FAILURE RETURN CODE =
retcod REASON CODE = *rc1*

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: Examine the return and reason codes 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

EZY1225I 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.

Module: EZACIC21
Destination: INITIALIZATION

EZY1226I 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.

Module: EZACIC21
Destination: INITIALIZATION

EZY1227I 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

EZY1228I 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

EZY1229I 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

EZY1246I 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

EZY1247I 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 not 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

EZY1250I 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

EZY1251I 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

EZY1252I mm/dd/yy hh:mm:ss UNABLE TO
LOAD EZASOH03 ERROR CODE= *code*
REASON CODE= *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: Using the Error Code and Reason Code it can be determined why the module would not load. Also, look for associated MVS messages.

Module: EZACIC21

Destination:

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

Messages Originating from EZACIC22

EZY1253I 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

System Action: Ignore the second request.

User Response: None.

System Programmer Response: None.

Module: EZACIC22

Destination: TERMINATION

EZY1339I 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

EZY1341I 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 in progress.

System Action: Ignore the second request.

User Response: None.

System Programmer Response: None.

Module: EZACIC22

Destination: TERMINATION

EZY1340I mm/dd/yy hh:mm:ss API ALREADY
QUIESCING DUE TO PREVIOUS REQ.
EZAO IGNORED TERM=*term* TRAN=*tran*

Explanation: A request for a quiesce of the CICS Sockets interface has been made but one is already in progress.

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

Destination:

Messages Originating from EZACIC25

EZY1254I mm/dd/yy hh:mm:ss CACHE FILE
ERROR RESP2 VALUE ***** CALL # *

Explanation: An error occurred on a cache file operation. name.

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.

Module: EZACIC25

Destination: DOMAIN NAME SERVER FUNCTION

EZY1255I 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.

System Programmer Response: None.

Module: EZACIC25

Destination: DOMAIN NAME SERVER FUNCTION

Appendix E. Sample Programs

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
*       5655-HAL (C) COPYRIGHT IBM CORP. 1993
*       This module is restricted materials of IBM
*       REFER TO IBM COPYRIGHT INSTRUCTIONS.
*
*   Status : Version 3, Release 0, Mod 0
*-----*
*
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 '.
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-RCV        PIC X(16) VALUE 'RCV'
  02 SOKET-RCVFROM     PIC X(16) VALUE 'RCVFROM'
  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 'TCP/IPBITMASKCOBL'.
77 TOEBCDIC-TOKEN    PIC X(16) VALUE 'TCP/IPTOEBCDICXLT'.
77 TOASCII-TOKEN     PIC X(16) VALUE 'TCP/IPTOASCIIXLAT'.
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.
      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)

```


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
    ELSE
        MOVE WRITE-SUCCESS TO CICS-MSG-AREA
        PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
TAKESOCKET-SEC-EXIT.
EXIT.

```

```
CLIENT-TASK.
```

```

*-----*
*
* Issue 'READV' socket to receive input data from client
*
*-----*

```

```

MOVE LOW-VALUES TO TCP-BUF.
MOVE 200 TO TCPLENG.

```

```

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.

```

```

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
*-----*

```

```
PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
```

```
*
```

```
* REMOVE FOLLOWING STATEMENT IF USING EBCDIC CLIENT
```

```
*
```

```

CALL 'EZACIC05' USING TOEBCDIC-TOKEN TCP-BUF TCPLENG.
MOVE TCP-BUF TO CICS-MSG-AREA.

```

```
MOVE RETCODE TO TCPLENG.
```

```
*
```

```
* REMOVE FOLLOWING STATEMENT IF USING EBCDIC CLIENT
```

```

*
CALL 'EZACIC04' USING TOASCII-TOKEN TCP-BUF TCPLNG.
CALL 'EZASOKET' USING SOKET-WRITE SOCKID TCPLNG
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 TCPLNG.
*
* REMOVE FOLLOWING STATEMENT IF USING EBCDIC CLIENT
*
CALL 'EZACIC04' USING TOASCII-TOKEN TCP-BUF TCPLNG.
CALL 'EZASOKET' USING SOKET-WRITE SOCKID TCPLNG
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.
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.

```

EZACICSS

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

```

*****
*
*          TCP/IP for MVS
*
*  Licensed Materials - Property of IBM
*  This product contains "Restricted Materials of IBM"
*  5735-FAL (C) Copyright IBM Corp. 1991
*  5655-HAL (C) Copyright IBM Corp. 1992, 1994.
*  All rights reserved.
*  US Government Users Restricted Rights -
*  Use, duplication or disclosure restricted by GSA ADP
*  Schedule
*  Contract with IBM Corp.
*  See IBM Copyright Instructions.
*
*****
* $SEG(EZACICSS)
*-----*

```

```

*
* Module Name      EZACICSS
*
* Description      This is a sample server program. It
*                  establishes a connection between
*                  CICS and TCP/IP 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 an 'END' token to the
*                  server. All processing is terminated
*                  when an 'TRM' token is received from a
*                  client.
*
*-----*
*
* LOGIC            1. Establish server setup
*                  a). TRUE Active
*                  b). CAF Active
*                  2. Assign user specified port at
*                  start up or use the program
*                  declared default.
*                  3. Initialize the Socket.
*                  4. Bind the port.
*                  5. Set Bit Mask to accept incoming
*                  read request.
*                  6. Process request from clients.
*                  a) Wait for connection
*                  b) Process request until 'END'
*                  token is receive from client.
*                  c) Close connection.
*                  note The current client request
*                  ends when the client closes
*                  the connection or sends an
*                  'END' token to the server.
*                  d) If the last request received by
*                  the current client is not a
*                  request to the server to
*                  terminate processing ('TRM'),
*                  continue at step 6A.
*                  7. Close the server's connection.
*
*-----*
IDENTIFICATION DIVISION.
PROGRAM-ID. EZACICSS.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
*
*-----*
* MESSAGES
*-----*
77 BITMASK-ERR          PIC X(30)
   VALUE IS 'BITMASK CONVERSION - FAILED '.
77 ENDDATA-ERR         PIC X(30)
   VALUE IS 'RETRIEVE DATA CAN NOT BE FOUND'.
77 INIT-MSG            PIC X(30)
   VALUE IS 'INITAPI COMPLETE '.
77 IOERR-ERR           PIC X(30)
   VALUE IS 'IOERR OCCURRS '.
77 ITEMERR-ERR        PIC X(30)
   VALUE IS 'ITEMERR ERROR '.

```

```

77 KEYWORD-ERR PIC X(30)
   VALUE IS 'INPUT KEYWORD ERROR' .
77 LENGERR-ERR PIC X(30)
   VALUE IS 'LENGERR ERROR' .
77 NOSPACE-ERR PIC X(30)
   VALUE IS 'NOSPACE CONDITION' .
77 NULL-DATA PIC X(30)
   VALUE IS 'READ NULL DATA' .
77 QIDERR-ERR PIC X(30)
   VALUE IS 'TRANSIENT DATA QUEUE NOT FOUND' .
77 START-MSG PIC X(30)
   VALUE IS 'SERVER PROGRAM IS STARTING' .
77 TCP-EXIT-ERR PIC X(30)
   VALUE IS 'SERVER STOPPED:TRUE NOT ACTIVE' .
77 TCP-SERVER-OFF PIC X(30)
   VALUE IS 'SERVER IS ENDING' .
77 TS-INVREQ-ERR PIC X(30)
   VALUE IS 'WRITE TS FAILED - INVREQ' .
77 TS-NOTAUTH-ERR PIC X(30)
   VALUE IS 'WRITE TS FAILED - NOTAUTH' .
77 TS-IOERR-ERR PIC X(30)
   VALUE IS 'WRITE TS FAILED - IOERR' .
77 WRITETS-ERR PIC X(30)
   VALUE IS 'WRITE TS FAILED' .
01 ACCEPT-ERR.
   05 ACCEPT-ERR-M PIC X(25)
      VALUE IS 'SOCKET CALL FAIL - ACCEPT' .
   05 FILLER PIC X(9)
      VALUE IS ' ERRNO = ' .
   05 ACCEPT-ERRNO PIC 9(8) DISPLAY.
   05 FILLER PIC X(13)
      VALUE IS SPACES.
01 BIND-ERR.
   05 BIND-ERR-M PIC X(25)
      VALUE IS 'SOCKET CALL FAIL - BIND' .
   05 FILLER PIC X(9)
      VALUE IS ' ERRNO = ' .
   05 BIND-ERRNO PIC 9(8) DISPLAY.
   05 FILLER PIC X(13)
      VALUE IS SPACES.
01 CLOSE-ERR.
   05 CLOSE-ERR-M PIC X(30)
      VALUE IS 'CLOSE SOCKET DESCRIPTOR FAILED' .
   05 FILLER PIC X(9)
      VALUE IS ' ERRNO = ' .
   05 CLOSE-ERRNO PIC 9(8) DISPLAY.
   05 FILLER PIC X(8)
      VALUE IS SPACES.
01 DB2END.
   05 FILLER PIC X(16)
      VALUE IS 'DB2 PROCESS ENDS' .
   05 FILLER PIC X(39)
      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 DAINSET 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.
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 'TCPIPTOEBCDIXLT'.
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)  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 '????'.
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 9(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).
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).
   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).
   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.
-----*
*
* 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
  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.
  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
                        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. *
* *
*-----*
      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.
*-----*
*
* 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-RCV          THRU ACCEPT-RCV-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 'RCVFROM'
* COMMAND.
*
*-----*
ACCEPT-RCV.
  MOVE 'T'                      TO TCP-INDICATOR.
  MOVE BUFFER-LENG             TO TCPLENG.
  MOVE LOW-VALUES              TO TCP-BUF.
  CALL 'EZACICAL' USING TCP-TOKEN RCVFROM-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 RCVFROM-ERRNO
    MOVE RCVFROM-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
        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-RCV-EXIT.

```



```

EXIT.
*****
** PROCESSES TALKING TO CLIENT THAT WILL UPDATE DB2 **
** TABLES. **
*****
** DATA PROCESS: **
** **
** INSERT REC - INS,X81,TEST DEPT,A0213B,Y94 **
** UPDATE REC - UPD,X81,,A1234C, **
** DELETE REC - DEL,X81,, **
** END CLIENT - END,{end client connection } **
** END SERVER - TRM,{terminate server } **
** **
*****
TALK-CLIENT.
UNSTRING TCP-BUF DELIMITED BY DEL-ID OR ALL '*'
INTO IN-ACT
      IN-DEPTNO
      IN-DEPTN
      IN-MGRNO
      IN-ADMRDEPT.
IF IN-ACT EQUAL 'END'
  THEN
    MOVE '1' TO TASK-FLAG
  ELSE
    IF IN-ACT EQUAL 'U' OR EQUAL 'UPD'
      THEN
        EXEC SQL UPDATE TCPCICS.DEPT
          SET MGRNO = :IN-MGRNO
          WHERE DEPTNO = :IN-DEPTNO
        END-EXEC
        MOVE 'UPDATE' TO DB2-ACT
        MOVE 'UPDATED: ' TO DB2M-VAR
      ELSE
        IF IN-ACT EQUAL 'I' OR EQUAL 'INS'
          THEN
            EXEC SQL INSERT
              INTO TCPCICS.DEPT (DEPTNO, DEPTNAME,
                MGRNO, ADMRDEPT)
              VALUES (:IN-DEPTNO, :IN-DEPTN,
                :IN-MGRNO, :IN-ADMRDEPT)
            END-EXEC
            MOVE 'INSERT' TO DB2-ACT
            MOVE 'INSERTED: ' TO DB2M-VAR
          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
    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
  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
      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

```

```

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.

```

Appendix F. Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make them available in all countries in which IBM operates. 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. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

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
500 Columbus Avenue
Thornwood, NY 10594
USA

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, NC 27709-2195
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement.

This document is not intended for production use and is furnished as is without any warranty of any kind, and all warranties are hereby disclaimed including the warranties of merchantability and fitness for a particular purpose.

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.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

ACF/VTAM	LANStreamer
AD/Cycle	Library Reader
AIX	MVS/ESA
AIX/ESA	MVS/SP
BookManager	MVS/XA
C/370	NetView
CICS	OpenEdition
DB2	OS/2
DFSMS	OS/390
DFSMS/MVS	PS/2
ESCON	RACF
ES/9000	RISC System/6000
EtherStreamer	RS/6000
Extended Services	SAA
GDDM	System/370
Hardware Configuration Definition	System/390
IBM	VTAM
	3090

The following terms are trademarks of other companies:

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Microsoft, Windows, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

Bibliography

This bibliography lists the publications for IBM TCP/IP products.

IBM TCP/IP Publications

The following sections describe the books associated with IBM TCP/IP products.

OS/390 TCP/IP OpenEdition Publications

- *OS/390 TCP/IP OpenEdition Configuration Guide*, SC31-8304-00.

This book is for people who want to configure, customize, administer, and maintain OS/390 TCP/IP OpenEdition. Familiarity with MVS operating system, TCP/IP protocols, and IBM Time Sharing Option (TSO) is recommended.

- *OS/390 TCP/IP OpenEdition Diagnosis Guide*, SC31-8492-00.

This book explains how to diagnose TCP/IP problems and how to determine whether a specific problem is in the OS/390 TCP/IP OpenEdition product code. It explains how to gather information for and describe problems to the IBM Software Support Center.

- *OS/390 TCP/IP OpenEdition Messages and Codes*, SC31-8307-00.

This book explains the informational and error messages issued by OS/390 TCP/IP OpenEdition. It can help users, operators, or system programmers to diagnose and fix problems identified by error messages.

- *OS/390 TCP/IP OpenEdition Planning and Release Guide*, SC31-8303-00.

This book is intended to help you plan for OS/390 TCP/IP OpenEdition whether you are migrating from a previous version or installing TCP/IP for the first time. This book also identifies the suggested and required modifications needed to enable you to use the enhanced functions provided with OS/390 TCP/IP OpenEdition.

- *OS/390 TCP/IP OpenEdition Programmer's Reference*, SC31-8308-00

This book 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.

This book is for people who want to use the supplied interfaces while writing application programs that access OS/390 TCP/IP OpenEdition. Familiarity with the MVS operating system, TCP/IP protocols, and IBM Time Sharing Option (TSO) is recommended.

- *OS/390 TCP/IP OpenEdition User's Guide*, GC31-8305-00.

This book is for people who want to use OS/390 TCP/IP OpenEdition for data communication. Familiarity with MVS operating system and IBM Time Sharing Option (TSO) is recommended.

TCP/IP for MVS Publications

- *TCP/IP Version 3 for OpenEdition MVS: Applications Feature Guide*, SC31-8069-00.

This book explains how to plan for, install, customize, and use the OpenEdition MVS Applications Feature. The Feature consists of applications and interfaces for direct access to the OpenEdition MVS environment. For example, users of the Feature can use MVS, UNIX, or AIX commands to transfer files, log in to the OpenEdition environment without going through TSO, and run commands remotely. This book also explains how to improve performance and diagnose problems when using the Feature.

- *TCP/IP for MVS: Application Programming Interface Reference*, SC31-7187-02.

This book 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 book to adapt your existing applications to communicate with each other using sockets over TCP/IP.

- *TCP/IP for MVS: CICS TCP/IP Socket Interface Guide and Reference*, SC31-7131-03.

This book is for people who want to set up, write application programs for, and diagnose problems with the socket interface for CICS using TCP/IP for MVS.

- *TCP/IP for MVS: Customization and Administration Guide*, SC31-7134-03.

This book is for people who want to customize, administer, and maintain TCP/IP for MVS. Familiarity with MVS operating system, TCP/IP protocols, and IBM Time Sharing Option (TSO) is recommended.

- *TCP/IP for MVS: Diagnosis Guide*, LY43-0105-02.

This book explains how to diagnose TCP/IP problems and how to determine whether a specific problem is in the IBM TCP/IP for MVS product code. It explains how to gather information for and describe problems to the IBM Software Support Center.

- *TCP/IP for MVS: IMS TCP/IP Application Development Guide and Reference*, SC31-7186-03.

This book is for programmers who want application programs that use the IMS TCP/IP application development services provided by IBM TCP/IP for MVS.

- *TCP/IP for MVS: Messages and Codes*, SC31-7132-03.

This book explains the informational and error messages issued by IBM TCP/IP for MVS. It can help users, operators, or system programmers to diagnose and fix problems identified by TCP/IP for MVS error messages.

- *TCP/IP for MVS: Network Print Facility*, SC31-8074-03.

This book is for system programmers and network administrators who need to prepare their network to route VTAM, JES2, or JES3 printer output to remote printers using TCP/IP for MVS.

- *TCP/IP for MVS: Offloading TCP/IP Processing*, SC31-7133-02.

This book is for people who want to install and configure the Offload feature on IBM 3172 Model 3 Interconnect Controllers. This book is also for people who want to use and customize the Offload feature of TCP/IP for MVS.

- *TCP/IP for MVS: Planning and Migration Guide*, SC31-7189-01.

This book is intended to help you plan for TCP/IP for MVS whether you are migrating from a previous version or installing TCP/IP for MVS for the first time. This book also identifies the suggested and required modifications needed to enable you to use the enhanced functions provided with TCP/IP for MVS.

- *TCP/IP: Performance Tuning Guide*, SC31-7188-02.

This book describes how to improve the performance of your network operations.

- *TCP/IP for MVS: Programmer's Reference*, SC31-7135-02.

This book 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.

This book is for people who want to use the supplied interfaces while writing application programs that access TCP/IP for MVS. Familiarity with the MVS operating system, TCP/IP protocols, and IBM Time Sharing Option (TSO) is recommended.

- *TCP/IP for MVS: User's Guide*, SC31-7136-02.

This book is for people who want to use TCP/IP for MVS for data communication. Familiarity with MVS operating system and IBM Time Sharing Option (TSO) is recommended.

TCP/IP for VM Publications

The following list describes books in the IBM TCP/IP for VM library.

- *IBM TCP/IP Version 2 Release 4 for VM: Messages and Codes*, SC31-6151-03.

This book is for system programmers who want to diagnose and fix problems identified by TCP/IP for VM error messages.

- *IBM TCP/IP Version 2 Release 4 for VM: Planning and Customization*, SC31-6082-03.

This book is for system programmers who want to plan and customize the TCP/IP for VM environment.

- *IBM TCP/IP Version 2 Release 4 for VM: Programmer's Reference*, SC31-6084-03.

This book is for application and system programmers who want to write application programs that use TCP/IP for VM. Application programmers should know the VM operating system.

- *IBM TCP/IP Version 2 Release 4 for VM: User's Guide*, SC31-6081-03.

This book is for people who want to use TCP/IP for VM for data communication. Familiarity with VM operating system, IBM Command

Processor (CP), and IBM Conversational Monitor System (CMS) is recommended.

TCP/IP for OS/2 Publication

IBM TCP/IP Version 3.0 for OS/2: Programmer's Reference, SC31-6077.

This book provides application and system programmers with the information required to write application programs that use TCP/IP for OS/2. Programmers should know the OS/2 operating system.

TCP/IP for DOS Publications

The following list describes books in the IBM TCP/IP for DOS library.

- *IBM TCP/IP Version 2.1.1 for DOS: Command Reference*, SX75-0083.

This book is for people who use a workstation with TCP/IP for DOS, such as end users and system programmers. The people who use this book should be familiar with DOS and the workstation, understand DOS operating system concepts, and be familiar with the *IBM TCP/IP Version 2.1.1 for DOS: User's Guide*

- *IBM TCP/IP Version 2.1.1 for DOS: Installation and Administration*, SC31-7047.

This book provides system programmers, network administrators, and workstation users responsible for installing TCP/IP for DOS with the information required to plan and implement the installation of TCP/IP for DOS. The topics include hardware and software requirements, pre-installation system performance considerations, instructions for installing TCP/IP for DOS, instructions for customizing the TCP/IP for DOS environment, and installation examples.

- *IBM TCP/IP Version 2.1.1 for DOS: Programmer's Reference*, SC31-7046.

This book is for application and system programmers to aid them in writing application programs that use TCP/IP for DOS on a workstation. Application programmers should know the DOS operating system and multitasking operating system concepts. Application programmers should be knowledgeable in the C programming language.

- *IBM TCP/IP Version 2.1.1 for DOS: User's Guide*, SC31-745.

This book is for people who use a workstation with TCP/IP for DOS, such as end users and system programmers. The people who use this book should be familiar with DOS and the workstation, and also understand DOS operating system concepts.

TCP/IP for AIX (RS/6001, PS/2, RT, 370) Publications

The following list shows books in the TCP/IP for AIX library.

- *AIX Operating System TCP/IP User's Guide*, SC23-2309.
- *AIX PS/2 TCP/IP User's Guide*, SC23-2047.
- *TCP/IP for IBM X-Windows on DOS 2.1*, SC23-2349.

TCP/IP for AS/400 Publications

The following list shows books in the TCP/IP for AS/400 library.

- *IBM AS/400 Communications: TCP/IP Guide*, SC41-9875.
- *IBM AS/400 Communications: User's Guide*, SC21-9601.

Other IBM TCP/IP Publications

The following list shows other available IBM TCP/IP books.

- *IBM Local Area Network Technical Reference*, SC30-3383.
- *IBM TCP/IP for VM and MVS: Diagnosis Guide*, LY43-0013.
- *TCP/IP and National Language Support*, GG24-3840.
- *TCP/IP Introduction*, GC31-6080.
- *TCP/IP Tutorial and Technical Overview*, GG24-3376.

IBM Operating System Publications

The following lists show books about various IBM operating systems.

AIX Publications

- *AIX Communications Concepts and Procedures for IBM RISC System/6001*, GC23-2203.

- *AIX Communications Programming Concepts* , SC23-2206.
- *IBM AIX Operating System Technical Reference, Volume 1* , SC23-2300.
- *IBM AIX Operating System Technical Reference, Volume 2* , SC23-2301.

AS/400 Publications

- *IBM AS/400 CL Reference Manual Volume 1* , SC21-9775.
- *IBM AS/400 CL Reference Manual Volume 2* , SC21-9776.
- *IBM AS/400 CL Reference Manual Volume 3* , SC21-9777.
- *IBM AS/400 CL Reference Manual Volume 4* , SC21-9778.
- *IBM AS/400 CL Reference Manual Volume 5* , SC21-9779.
- *IBM AS/400 Communications: APPN Network User's Guide* , SC21-8188.
- *IBM AS/400 Communications: Programmer's Guide* , SC21-9590.
- *IBM AS/400 Communications: User's Guide* , SC21-9601.
- *IBM AS/400 Device Configuration Guide* , SC21-8106.
- *IBM AS/400 Programming: Command Reference Summary* , SC21-8076.
- *IBM AS/400 Programming: Data Management Guide* , SC21-9658.
- *IBM AS/400 System Operations: Database Coordinator' Guide* , SC21-8086.
- *IBM AS/400 System Operations: Operator's Guide* , SC21-8082.

DOS Publications

- *DOS Getting Started Version 5.00* , SA40-0637.
- *DOS 5.02 Technical Reference* , S16G-4559.
- *DOS/Windows Client Getting Started* , SC09-3001.
- *PC DOS 6.1 Command Reference* , S71G-3634.

MVS Publications

For a complete description of the library for MVS/ESA Version 5, see *OS/390 Information Roadmap*, GC28-1727-03. See also "JES Publications" on page 328.

OS/2 Publications

- *IBM OS/2 Warp Server Up and Running!* , S25H-8004
- *IBM Official Guide to Using OS/2 Warp* , ISBN 1-56884-466-2 (Karla Stagra and Linda S. Rogers; Foster City, CA: An IBM Press Book published by IDG Books Worldwide, Inc., 1995)
- *IBM OS/2 Warp Internet Connection: Your Key to Cruising the Internet and the World Wide Web* , ISBN 1-56884-465-4 (Deborah Morrison; Foster City, CA: An IBM Press Book published by IDG Books Worldwide, Inc., 1995)

OS/390 Publications

- *OS/390 Information Roadmap*, GC28-1727-03
This book describes the documentation for the specific elements included in OS/390.
- *OS/390 Release 4 Planning for Installation*, GC28-1726-03
This book is intended to help you plan for the installation of OS/390. It describes migration, installation, hardware and software requirements, and coexistence considerations.
- *OS/390 OpenEdition Introduction*, GC28-1889-01.
- *OS/390 OpenEdition Planning*, SC28-1890-03.
- *OS/390 OpenEdition User's Guide*, SC28-1891-03.
- *OS/390 OpenEdition Command Reference*, SC28-1892-03.
- *OS/390 OpenEdition Messages and Codes*, SC28-1908-03.
- *OS/390 Language Environment Programming Guide*, SC28-1939-03.
- *OS/390 Language Environment Programming Reference*, SC28-1940-03.
- *OS/390 OpenEdition Programming: Assembler Callable Services Reference*, SC28-1899-03.
- *OS/390 Open Systems Adapter Support Facility Users's Guide*, SC28-1855.
- *Planning for the System/390 Open Systems Adapter Feature*, GC23-3870.

VM Publications

- *VM/ESA CMS Command Reference Summary* , SX24-5249.
- *VM/ESA CP Planning and Administration for 370* , SC24-5430.
- *VM/ESA CP Programming Services for 370* , SC24-5435.

- *VM/ESA Group Control System Reference for 370* , SC24-5426.
- *VM/ESA: Library Guide and Master Index* , GC23-0367.
- *VM/ESA: Master Index for 370* , GC24-5436.
- *VM/ESA Service Introduction and Reference* , SC24-5444.
- *VM/SP CMS Command Reference* , ST00-1981.
- *VM/SP Group Control System Macro Reference* , SC24-5250.
- *VM/SP Installation Guide* , SC24-5237.
- *VM/SP High Performance Option: Library Guide and Master Index* , GC23-0187.
- *VM/SP System Facilities for Programming* , SC24-5288.
- *VM/XA CP Programming Services* , SC23-0370.
- *VM/XA Diagnosis Reference* , LY27-8054.
- *VM/XA Installation and Service* , SC23-0364.
- *VM/XA SP Group Control System Command and Macro Reference*, SC23-0433.
- *VTAM Programming for CSM*, SC31-8420-00.
- *VTAM CMIP Services and Topology Agent Programming Guide*, SC31-8365-00.
- *VTAM Diagnosis*, LY43-0078-00.
- *VTAM Data Areas for MVS/ESA Volume 1*, LY43-0076-00.
- *VTAM Data Areas for MVS/ESA Volume 2*, LY40-0077-00.
- *APPC Application Suite User's Guide*, SC31-6532-00.
- *APPC Application Suite Administration*, SC31-6533-00.
- *APPC Application Suite Programming*, SC31-6534-00.
- *VTAM AnyNet Guide to Sockets over SNA*, SC31-8371-00.
- *VTAM AnyNet Guide to SNA over TCP/IP*, SC31-8376-00.
- *VTAM Glossary*, GC31-8366-00.
- *Planning for NetView, NCP, and VTAM*, SC31-8063-00.
- *Planning for Integrated Networks*, SC31-8062-00.
- *VTAM Licensed Program Specifications*, GC31-8379-00.
- *VTAM Operation Quick Reference*, SX75-0208-00.

IBM Software Publications

The following sections describe the books associated with IBM software products.

ACF/VTAM Publications

The following list shows books in the VTAM Version 4 Release 4 library.

- *VTAM Installation and Migration Guide*, GC31-8367-00.
- *VTAM Release Guide*, GC31-6545-00.
- *VTAM Network Implementation Guide*, SC31-8370-00.
- *VTAM Resource Definition Reference*, SC31-8377-00.
- *VTAM Resource Definition Samples*, SC31-8378-00.
- *VTAM Customization*, LY43-0075-00.
- *VTAM Operation*, SC31-8372-00.
- *VTAM Messages*, GC31-8368-00.
- *VTAM Codes*, GC31-8369-00.
- *VTAM Programming*, SC31-8373-00.
- *VTAM Guide to Programming for LU 6.2*, SC31-8374-00.
- *VTAM Programming Reference for LU 6.2*, SC31-8375-00.

DATABASE 2 Publications

The following lists show books in the DATABASE 2 library.

DATABASE 2 Version 2

- *IBM DATABASE 2 Version 2: Administration Guide* , SC26-4374.
- *IBM DATABASE 2 Version 2: Application Programming and SQL Guide* , SC26-4377.
- *IBM DATABASE 2 Version 2: Messages and Codes* , SC26-4379.
- *IBM DATABASE 2 Version 2: Reference Summary* , SX26-3771.
- *IBM DATABASE 2 Version 2: SQL Reference* , SC26-4380.

DATABASE 2 Version 3

- *IBM DATABASE 2 Version 3: DB2 Administration Guide*, SC26-4888.
- *IBM DATABASE 2 Version 3: DB2 Application Programming and SQL Guide* , SC26-4889.

- *IBM DATABASE 2 Version 3: DB2 Messages and Codes* , SC26-4892.
- *IBM DATABASE 2 Version 3: DB2 Reference Summary* , SX26-3801.
- *IBM DATABASE 2 Version 3: DB2 SQL Reference* , SC26-4890.

ISPF Publication

ISPF Dialog Management Guide and Reference , SC34-4266.

JES Publications

- *MVS/ESA Library Guide with JES2*, GC28-1423.
- *MVS/ESA Library Guide with JES3*, SC28-1424

MVS/DFP Publications

- *MVS/DFP Version 3 Release 3: Customizing and Operating the Network File System Server* , SC26-4832.
- *MVS/DFP Version 3 Release 3: Macro Instructions for Data Sets* , S26-4747.
- *MVS/DFP Version 3 Release 3: Using Data Sets* , SC26-4749.
- *MVS/DFP Version 3 Release 3: Using the Network File System Server* , SC26-4732.

Network Control Program (NCP) Publications

- *NCP V7R5 Migration Guide*, SC30-3833-00
- *NCP and EP Reference*, LY43-0029-04.
- *NCP, SSP, and EP Generation and Loading Guide*, SC31-6221-01.
- *NCP, SSP, and EP Resource Definition Guide*, SC31-6223-05.
- *NCP, SSP, and EP Resource Definition Reference*, SC31-6224-05.
- *NCP, SSP, and EP Messages and Codes*, SC31-6222-04
- *NTuneMON V2R4 User's Guide*, SC31-6266-03
- *NCP, SSP, and EP Trace Analysis Handbook*, LY43-0037-03

TME 10 NetView for OS/390 Publications

For a complete description of the TME 10 NetView for OS/390 library, see the *TME 10 NetView for OS/390 Library Reference*, SC31-8249.

OpenEdition MVS Publications

The following list shows selected books in the OpenEdition MVS library.

- *OS/390 OpenEdition Introduction*, GC28-1889-01
- *OS/390 OpenEdition Planning*, SC28-1890-03

Programming Publications

The following list shows books about various programming applications.

- *IBM C/370 Diagnosis Guide and Reference* LY09-1804 (feature 8082).
- *IBM C/370 General Information Manual* GC09-1386.
- *IBM C/370 Installation and Customization Guide Version 2 Release 1.0* , GC09-1387.
- *IBM C/370 Programming Guide* , SC09-1384.
- *IBM C/370 Reference Summary* , SX09-1211.
- *IBM C/370 User's Guide* , SC09-1264.
- *OS/390 C/C++ Run-Time Library Reference*, SC28-1663-02.
- *IBM TSO Extensions CLISTs* , SC28-1876.
- *IBM TSO Extensions Command Language Reference* GX23-0015.
- *IBM TSO Extensions Interactive Data Transmission Facility: User's Guide* , SC28-1104.
- *IMS/ESA V3R1 Application Programming: DL/I Calls* SC26-4274.
- *HiPPI User's Guide and Programmer's Reference* , SA23-0369.
- *Parallel I/O Access Methods Programmer's Guide* , SC26-4648.
- *VS Pascal Application Programming Guide* SC26-4319.
- *VS Pascal Diagnosis Guide and Reference* LY27-9525.
- *VS Pascal General Information* , GT00-2664.
- *VS Pascal Installation and Customization for MVS* SC26-4321.
- *VS Pascal Installation and Customization for VM* SC26-4342.
- *VS Pascal Language Reference* , SC26-4320.

RACF Publications

The following list shows books in the RACF library.

- *IBM Resource Access Control Facility (RACF): General Information Manual* , GT00-2820.
- *IBM Resource Access Control Facility (RACF): User's Guide*, SC28-1341.
- *External Security Interface (RACROUTE) Macro Reference*, GC28-1366.
- *RACF Publications Order Guide* , GX22-0012.
- *Resource Access Control Facility (RACF) Security Administrator's Guide*, SC28-1340.
- *System Programming Library: RACF*, SC28-1343.

SMP/E Publications

The following list shows books in the SMP/E Release 8 library.

- *SMP/E Diagnosis Guide* , SC23-3130.
- *SMP/E Messages and Codes* , SC28-1107.
- *SMP/E Reference* , SC28-1107.
- *SMP/E Reference Summary* , SX22-0016.
- *SMP/E User's Guide* , SC28-1302.

VSAM Publication

MVS/370 VSAM Administration Guide , GC26-4066.

X.25 NPSI Publications

The following list shows books in the X.25 NPSI library.

- *X.25 Network Control Program Packet Switching Interface Diagnosis, Customization, and Tuning Version 3* , LY30-5610.
- *X.25 Network Control Program Packet Switching Interface Host Programming* , SC30-3502.
- *X.25 Network Control Program Packet Switching Interface Planning and Installation* , SC30-3470.

IBM Hardware Publications

The following sections describe the books associated with IBM hardware products.

System/370 and System/390 Publications

The following list shows the principles of operation manuals for the System/370 and System/390 processors.

- *IBM ESA/370 Principles of Operation*, SA22-7200.
- *IBM ESA/390 Principles of Operation*, SA22-7201.
- *IBM System/370 Extended Architecture Principles of Operation*, SA22-7085.
- *IBM System/370 Principles of Operation*, GA22-7001.
- *S/360, S/370, and S/390 I/O Interface Channel to Channel Control Unit OEMI*, GA22-6974.

3172 Interconnect Controller Publications

The following list shows books in the IBM 3172 Interconnect Controller library.

- *IBM Interconnect Controller Program User's Guide*, SC30-3525.
- *IBM 3172 Interconnect Controller Installation and Service Guide*, GA27-3861.
- *IBM 3172 Interconnect Controller Operator's Guide*, GA27-3860.
- *IBM 3172 Interconnect Controller Planning Guide*, GA27-3867.
- *IBM 3172 Interconnect Controller Status Codes*, GA27-3951.

3270 Information Display System Publication

3270 Information Display System: 3270 Data Stream Programmer's Reference , GA23-0059.

8232 LAN Channel Station Publications

The following list shows books in the IBM 8232 LAN Channel Station library.

- *IBM LAN Channel Support Program: Version 1.0 User's Guide*, SC30-3458.
- *IBM 8232 LAN Channel Station: Installation and Testing*, GA27-3796.
- *IBM 8232 LAN Channel Station: Operating Guide*, GA27-3785.

9370 Publications

The following list shows books in the 9370 library.

- *IBM 9370 Information System: Using the X.25 Communications Subsystem* , SA09-1742.
- *IBM 9370 Information System X.25 Communications Subsystem Description* , SA09-1743.
- *VM/ESA: Connectivity Planning, Administration, and Operation Release 1*, SC24-5448.

Other TCP/IP-Related Publications

The following sections describe other books associated with TCP/IP.

- *The Art of Distributed Application: Programming Techniques for Remote Procedure Calls* John R. Corbin, Springer-Verlog, 1991.
- *CAE Specification: X/Open Transport Interface (XTI)*, X/Open Company Ltd., U. K., 1992, SC31-8005.
- *IEEE Network Magazine* , July 1990.
- *TCP/IP Illustrated Volume I: The Protocols* , W. Richard Stevens, Addison-Wesley Publishing Company, Inc., 1994, SR28-5586.
- *TCP/IP Illustrated Volume II: The Implementation* , Gary R. Wright and Richard Stevens, Addison-Wesley Publishing Company, Inc., 1995, SR28-5630.
- *TCP/IP Illustrated Volume III* , W. Richard Stevens, Addison-Wesley Publishing Company, Inc., 1996, SR23-7289
- *Interoperability Report* , Volume 3, No. 3, March 1989.
- "MIB II Extends SNMP Interoperability," C. Vanderberg, *Data Communications* , October 1990.
- "Network Management and the Design of SNMP," J.D. Case, J.R. Davin, M.S. Fedor, M.L. Schoffstall.
- "Network Management of TCP/IP Networks: Present and Future," A. Ben-Artzi, A. Chandna, V. Warriar.
- *The Simple Book: An Introduction to Management of TCP/IP-based Internets* , Marshall T Rose, Prentice Hall, Englewood Cliffs, New Jersey, 1993.

- "Special Issue: Network Management and Network Security," *ConneXions-The Interoperability Report* Volume 4, No. 8, August 1990.
- *UNIX Programmer's Reference Manual* (4.3 Berkeley Software Distribution, Virtual VAX-11 Version). Department of Electrical Engineering and Computer Science. University of California, Berkeley, 1988.

OSF/Motif Publications

The following list shows OSF/Motif books.

- *OSF/Motif Application Environment Specifications* (AES), Open Software Foundation, Prentice Hall, Inc., 1990, ISBN 0-13-640483-9.
- *OSF/Motif Programmer's Guide* Open Software Foundation, Prentice Hall, Inc., 1990, ISBN 0-13-640509-6.
- *OSF/Motif Programmer's Reference* Open Software Foundation, Prentice Hall, Inc., 1990, ISBN 0-13-640517-7.
- *OSF/Motif Style Guide* Open Software Foundation, Prentice Hall, Inc., 1990, ISBN 0-13-640491-X.
- *OSF/Motif User's Guide* Open Software Foundation, Prentice Hall, Inc., 1990, ISBN 0-13-640525-8.

Sun (RPC) Publications

The following list shows Sun Microsystems books.

- *Networking on the Sun Workstation: Remote Procedure Call Programming Guide* (800-1324-03), Sun Microsystems, Inc.
- *Network Programming* (800-1779-10), Sun Microsystems, Inc.

X Window System Publications

The following list shows X Window System books.

- *Introduction to the X Window System* , Oliver Jones, Prentice-Hall, 1988, ISBN 0-13-499997-5.
- *PEXlib Specification and C Language Binding* Jeff Stevenson, Hewlett-Packard Company, 1992, SR28-5116.
- *The X Window System Series* (6 volumes), O'Reilly & Associates, 1988, 1989, 1990, ISBN 0-937175-40-4, 0-937175-27-7, 0-937175-28-5, 0-937175-35-6, 0-937175-33-1, 0-937175-35-8.

- *X Protocol Reference Manual* Adrian Nye, ed. O'Reilly & Associates, Inc., 1990, ISBN 0-937175-50-1.
- *X Window System: C Library and Protocol Reference* Robert Scheifler, James Gettys, and Ron Newman, DEC Press, 1988, ISBN 1-55558-012-2.
- *X Window System: Programming and Applications with Xt*, Douglas A. Young, Prentice-Hall, 1989, ISBN 0-13-972167-3.
- *X Window System: Programming and Applications with Xt, OSF/Motif Edition* Douglas A. Young, Prentice-Hall, 1990, ISBN 0-13-497074-8.
- *X Window System Technical Reference* , Steven Mikes, Addison-Wesley, 1990, ISBN 0-201-52370-1.
- *X Window System User's Guide* Valerie Quercia and Tim O'Reilly, O'Reilly & Associates, Inc., 1990, ISBN 0-937175-14-5.

Network Architecture Publications

The following sections list books associated with network architecture.

Open Systems Interconnection (OSI) Publication

Open Systems Interconnection , Z320-9757.

Systems Network Architecture (SNA) Publications

The following list shows books in the SNA library.

- *Systems Network Architecture: Sessions between Logical Units* , GC20-1868.
- *Systems Network Architecture Format and Protocol Reference Manual: Architecture Logic* , SC30-3112.
- *Systems Network Architecture Format and Protocol Reference Manual: Management Services* , SC30-3346.
- *Systems Network Architecture Formats* GA27-3136.
- *Systems Network Architecture Network Product Formats*, LY43-0081.

Index

A

- abend codes
 - AEY9 84
- ACCEPT (call) 148
- accept system call
 - C language 115
 - EZACICAL call 226
 - use in server 96
- adapter 18
- address
 - family (domain) 99
 - MVS address spaces 100
 - structures 99
- AF_INET domain parameter 99, 141
- AF parameter on call interface, on SOCKET 198
- ALTER 51
- ASCII data format 107
- automatic startup 79

B

- BACKLOG parameter on call interface, LISTEN call 174
- big endian 101
- BIND (call) 149
- bind system call
 - C language 117
 - EZACICAL call 227
 - use in server 96
- bit-length on call interface, on EZACIC06 call 206
- bit-mask on call interface, on EZACIC06 call 206
- blocking/nonblocking option 121, 130
- broadcast option 126
- BUF parameter on call socket interface 146
 - on READ 175
 - on RECV 177
 - on RECVFROM 179
 - on SEND 190
 - on SENDTO 194
 - on WRITE 201

C

- C language
 - API 111, 124
 - basic calls 19
 - C structures
 - clientid 114
 - ifconf 114
 - ifreq 114
 - linger 114
 - sockaddr_in 115
 - timeval 115
 - calls
 - accept() 115
 - bind() 117
 - close() 118
 - connect() 119

C language (continued)

- calls (continued)
 - fcntl() 120
 - getpeername() 123
 - getclientid() 121
 - gethostid() 122
 - gethostname() 122
 - getsockname() 124
 - getsockopt() 124
 - givesocket() 127
 - initapi() 128
 - ioctl() 129
 - listen() 131
 - read() 132
 - recv() 133
 - recvfrom() 134
 - select() 135
 - send() 137
 - sendto() 138
 - setsockopt() 124
 - shutdown() 140
 - socket() 140
 - takesocket() 141
 - write() 142
- compiling and linking 111
- header files needed 111
- cache file, VSAM 69
- CALL Instruction Interface for Assembler, PL/1, and COBOL 145
- Call Instructions for Assembler, PL/1, and COBOL Programs 146
 - ACCEPT 148
 - BIND 149
 - CLOSE 151
 - CONNECT 152
 - EZACIC04 204
 - EZACIC05 205
 - EZACIC06 206
 - EZACIC08 208
 - FCNTL 154
 - GETCLIENTID 155
 - GETHOSTBYADDR 156
 - GETHOSTBYNAME 158
 - GETHOSTID 160
 - GETHOSTNAME 160
 - GETPEERNAME 161
 - GETSOCKNAME 162
 - GETSOCKOPT 163
 - GIVESOCKET 165
 - INITAPI 167
 - IOCTL 169
 - LISTEN 173
 - READ 174
 - READV 175
 - RECV 176
 - RECVFROM 178
 - RECVMSG 180
 - SELECT 183

Call Instructions for Assembler, PL/1, and COBOL

Programs *(continued)*

- SELECTEX 187
- SENDMSG 190
- SENDTO 193
- SETSOCKOPT 195
- SHUTDOWN 197
- SOCKET 198
- TAKESOCKET 199
- TERMAPI 200
- WRITE 200
- WRITEV 201

CH-MASK parameter on call interface, on EZACIC06 206

child server 95, 9

CICS 79

- starting automatically 79
- starting manually 79
- starting with program link 85

CICS transaction processing system

- defining resources in setup 23
- operation with CICS TCP/IP 18

client

- definition 3
- socket calls used in 94

CLIENT parameter on call socket interface 146

- on GETCLIENTID 155

- on GIVESOCKET 166

- on TAKESOCKET 200

client/server processing 3

clientid

- C structure 114

close system call

- C language 118

- EZACICAL call 228

- use in child server 96

- use in client 95

- use in server 97

COBOL language

- basic calls 19

- call format 226

- choosing EZACICAL or Sockets Extended API 223

- compilation JCL 223

- EZACICAL API 225, 251

- socket API calls (EZACICAL, SOKETS)

- ACCEPT 226

- BIND 227

- CLOSE 228

- CONNECT 229

- FCNTL 230

- GETCLIENTID 231

- GETHOSTID 232

- GETHOSTNAME 232

- GETPEERNAME 233

- GETSOCKNAME 234

- GETSOCKOPT 235

- GIVESOCKET 236

- INITAPI 237

- IOCTL 238

- LISTEN 239

- READ 240

COBOL language *(continued)*

- socket API calls (EZACICAL, SOKETS) *(continued)*

- RECVFROM 241

- SELECT 242

- SEND 244

- SENDTO 245

- SETSOCKOPT 246

- SHUTDOWN 247

- SOCKET 248

- TAKESOCKET 249

- WRITE 250

COMMAND parameter on call interface, IOCTL call 170

COMMAND parameter on call socket interface 146

- on EZACIC06 206

- on FCNTL 154

COMMAREA 220

COMP (COBOL USAGE) 226

concurrent server 92

- defined 8

- illustrated 8, 9

- writing your own 96

configuration file, JCL 48

configuration macro 44

configuration transaction 50

configuring CICS TCP/IP 21, 44

connect system call

- C language 119

- EZACICAL call 229

- use in client 95

conversion routines 107

COPY 54

CSKL transaction 103, 107

- defining in CICS 25

D

data conversion 107

data structures, external

- configuration data set 211

- global work area 212

- listener control area 221

- parameter list for EZACIC20 220

- task interface element 215

data translation, socket interface 146, 203

- ASCII to EBCDIC 205

- bit-mask to character 206

- character to bit-mask 206

- EBCDIC to ASCII 204

datasets, modifying 50

DEFINE 56

DELETE 59

Destination Control Table 37

DISPLAY 61

DNS

- EZACIC25, adding to RDO 26

domain

- address family 99

- parameter in socket call 141

Domain Name Server Cache 69

- cache file 69

Domain Name Server Cache (*continued*)

- EZACICR macro 70
- initialization module, creating 71

E

- EBCDIC data format 107
- environmental support 87
- ERETMSK parameter on call interface, on SELECT 186
- ERRNO parameter on call socket interface 146
 - on ACCEPT 149
 - on BIND 150
 - on CLOSE 151
 - on CONNECT 153
 - on FCNTL 154
 - on GETCLIENTID 156
 - on GETHOSTNMAE 160
 - on GETPEERNAME 162
 - on GETSOCKNAME 163
 - on GETSOCKOPT 165
 - on GIVESOCKET 167
 - on INITAPI 168
 - on IOCTL 172
 - on LISTEN 174
 - on READ 175
 - on READV 176
 - on RECV 177
 - on RECVFROM 179
 - on RECVMMSG 182
 - on SELECT 186
 - on SELECTEX 188
 - on SEND 190
 - on SENDMSG 192
 - on SENDTO 194
 - on SETSOCKOPT 197
 - on SHUTDOWN 198
 - on SOCKET 199
 - on TAKESOCKET 200
 - on WRITE 201
 - on WRITEV 202
- errno variable 115
- error check option 126
- ESDNMASK parameter on call interface, on SELECT 186
- event monitoring
 - for Listener 40
 - for TRUE 38
- EWOULDBLOCK error return, call interface calls
 - RECV 176
 - RECVFROM 178
- EXEC CICS LINK 85
- EXEC CICS RETRIEVE 101
- EXEC CICS START 101
- EZAC (configuration transaction) 50
- EZAC start screen 81
- EZACACHE, defining to RDO 35
- EZACIC04, call interface, EBCDIC to ASCII translation 204
- EZACIC05, call interface, ASCII to EBCDIC translation 205

- EZACIC06 16
- EZACIC06, call interface, bit-mask translation 206
- EZACIC08, HOSTENT structure interpreter utility 208
- EZACIC20, parameter list 220
- EZACICAL 223
- EZACICAL API 225, 251
- EZACICAL program 225
- EZACICD (configuration macro) 44
- EZACICR macro 70, 71
- EZACICSE program 105
- EZACICxx programs
 - defining in CICS 26
 - EZACIC00 27
 - EZACIC01 27
 - EZACIC02 28
 - EZACIC03 33
 - EZACIC07 33
 - EZACIC20 28
 - PLT entries 42
 - EZACIC21 29
 - EZACIC22 29
 - EZACIC23 30
 - EZACIC24 30
 - EZACIC25
 - defining in RDO 31
 - Domain Name Server Cache 70
 - EZACICAL 33
 - EZACICM 31
 - EZACICME 31
 - EZACICSC 33
 - EZACICSS 32
 - summary 26
- EZACONFG, defining to RDO 33
- EZAO transaction
 - defining in CICS 24
 - manual startup/shutdown 79
- EZAP transaction
 - defining in CICS 25

F

- FCNTL (call) 154
- fcntl system call
 - C language 120
 - EZACICAL call 230
- files, defining to RDO 33
 - EZACACHE 35
 - EZACONFG 33
- FLAGS parameter on call socket interface 146
 - on RECV 177
 - on RECVFROM 179
 - on RECVMMSG 182
 - on SEND 189
 - on SENDMSG 192
 - on SENDTO 194
- FNDELAY flag on call interface, on FCNTL 154
- Functions
 - ALTER 51
 - COPY 54
 - DEFINE 56
 - DELETE 59

G

GETCLIENTID (call) 155
getclientid system call
 C language 121
 EZACICAL call 231
 use in server 96, 101
GETHOSTBYADDR (call) 156
GETHOSTBYNAME (call) 158
GETHOSTID (call) 160
gethostid system call
 C language 122
 EZACICAL call 232
GETHOSTNAME (call) 160
gethostname system call
 C language 122
 EZACICAL call 232
GETPEERNAME (call) 161
getpeername system call
 C language 123
 EZACICAL call 233
GETSOCKNAME (call) 162
getsockname system call
 C language 124
 EZACICAL call 234
GETSOCKOPT (call) 163
getsockopt system call
 C language 124
 EZACICAL call 235
GIVESOCKET (call) 165
givesocket system call
 C language 127
 EZACICAL call 236
 use in server 97, 101

H

hlq.PROFILE.TCPIP data set 42
hlq.TCPIP.DATA data set 43
HOSTADDR parameter on call interface, on
 GETHOSTBYADDR 156
HOSTENT parameter on socket call interface
 on GETHOSTBYADDR 156
 on GETHOSTBYNAME 158
HOSTENT structure interpreter parameters, on
 EZACIC08 209
HOW parameter on call interface, on SHUTDOWN
 197

I

IDENT parameter on call interface, INITAPI call 168
ifconf C structure 114
ifreq C structure 114
immediate=no 84
immediate=yes 84
IN-BUFFER parameter on call interface, EZACIC05 call
 205
INITAPI(call) 167
initapi system call
 C language 128
 EZACICAL call 237

initapi system call (*continued*)
 use in client 94
 use in server 96
installing CICS TCP/IP 21
internets, TCP/IP 2
interval control 104
IOCTL (call) 169
ioctl system call
 C language 129
 EZACICAL call 238
IOV parameter on call socket interface 146
 on READV 175
 on WRITEV 202
IOVCNT parameter on call socket interface 146
 on READV 176
 on RECVMSG 182
 on SENDMSG 192
 on WRITEV 202
IP protocol 4
iterative server
 defined 8
 illustrated 92, 9
 socket calls in 97

J

JCL jobs
 for C compilation 111
 for CICS startup 22
 for CICS/TCP configuration 48
 for COBOL compilation 223
 for DNS cache file 73

L

LENGTH parameter on call socket interface 146
 on EZACIC04 204
 on EZACIC05 205
linger C structure 114
linger on close option 126
link, program 85
LISTEN (call) 173
listen system call
 C language 131
 EZACICAL call 239
 use in server 96
Listener
 control area 221
 input format 103
 monitor control table 40
 output format 104
 security module 105
 starting and stopping 103, 107
 user-written 87
listener/server, socket call (general) 96
listener/server call sequence 95
little endian 101

M

macro, EZACICR 70
manifest.h C header 111

- manual startup 79
- MAXSNO parameter on call interface, INITAPI call 168
- MAXSOC parameter on call socket interface 146
 - on INITAPI 168
 - on SELECT 185
 - on SELECTEX 187
- modifying datasets 50
- Monitor Control Table
 - for Listener 41
 - for TRUE 38
- monitoring, event
 - for Listener 40
 - for TRUE 38
- MSG parameter on call socket interface 146
 - on RECVMSG 182
 - on SENDMSG 192
- MVS address spaces 100

N

- NAME parameter on socket call interface
 - on ACCEPT 149
 - on BIND 150
 - on CONNECT 153
 - on GETHOSTBYNAME 158
 - on GETHOSTNAME 160
 - on GETPEERNAME 161
 - on GETSOCKNAME 162
 - on RECVFROM 179
 - on SENDTO 194
- NAMELEN parameter on socket call interface
 - on GETHOSTBYNAME 158
 - on GETHOSTNAME 160
- NBYTE parameter on call socket interface 146
 - on READ 175
 - on RECV 177
 - on RECVFROM 179
 - on SEND 190
 - on SENDTO 194
 - on WRITE 201
- network byte order 101

O

- OPTLEN parameter on socket call interface
 - on GETSOCKOPT 165
 - on SETSOCKOPT 196
- OPTNAME parameter on call socket interface 146
 - on GETSOCKOPT 164
 - on SETSOCKOPT 195
- OPTVAL parameter on call socket interface 146
 - on GETSOCKOPT 165
 - on SETSOCKOPT 196
- original COBOL application programming interface (API) 223, 251
- OSI 3
- OUT-BUFFER parameter on call interface, on EZACIC04 204
- out-of-band data
 - options in get/setsockopt call 126
 - sending with send call 138

P

- passing sockets 97
- pending activity 15
- pending exception 16
- pending read 16
- PL/I programs, required statement 147
- PLT 79
- PLT entry 42
- port numbers
 - definition 99
 - reserving port numbers 42
- ports
 - compared with sockets 6
 - numbers 99
 - reserving port numbers 42
- program link 85
- Program List Table 79
- program variable definitions, call interface 146
 - assembler definition 147
 - COBOL PIC 147
 - PL/I declare 147
 - VS COBOL II PIC 147
- programs, defining in CICS 26
- PROTO parameter on call interface, on SOCKET 199
- protocol parameter in socket call 141

R

- RDO
 - configure the socket interface (EZAC) 23
- READ (call) 174
- read system call
 - C language 132
 - EZACICAL call 240
 - use in child server 95
 - use in client 95
- READV (call) 175
- RECV (call) 176
- recv system call, C language 133
- RECVFROM (call) 178
- recvfrom system call
 - C language 134
 - EZACICAL call 241
 - use in server 96
- RECVMSG (call) 180
- RENAME 64
- REQARG and RETARG parameter on call socket interface 154, 146
 - on FCNTL 154
 - on IOCTL 171
- requirements for CICS TCP/IP 18
- resource definition in CICS 23
- RETARG parameter on call interface, on IOCTL 172
- RETCODE parameter on call socket interface 146
 - on ACCEPT 149
 - on BIND 150
 - on CLOSE 152
 - on CONNECT 153
 - on EZACIC06 206
 - on FCNTL 155
 - on GETCLIENTID 156

RETCODE parameter on call socket interface

(continued)

- on GETHOSTBYADDR 156
- on GETHOSTBYNAME 158
- on GETHOSTID 160
- on GETHOSTNAME 161
- on GETPEERNAME 162
- on GETSOCKNAME 163
- on GETSOCKOPT 165
- on GIVESOCKET 167
- on INITAPI 169
- on IOCTL 172
- on LISTEN 174
- on READ 175
- on READV 176
- on RECV 177
- on RECVFROM 179
- on RECVMSG 183
- on SELECT 186
- on SELECTEX 188
- on SEND 190
- on SENDMSG 193
- on SENDTO 194
- on SETSOCKOPT 197
- on SHUTDOWN 198
- on SOCKET 199
- on TAKESOCKET 200
- on WRITE 201
- on WRITEV 203

return codes

call interface 147

reuse local address option 126

RRETMASK parameter on call interface, on SELECT 186

RSNDMSK parameter on call interface, on SELECT 185

S

S, defines socket descriptor on socket call interface

- on ACCEPT 149
- on BIND 150
- on CLOSE 151
- on CONNECT 153
- on FCNTL 154
- on GETPEERNAME 161
- on GETSOCKNAME 162
- on GETSOCKOPT 163
- on GIVESOCKET 166
- on IOCTL 170
- on LISTEN 173
- on READ 174
- on READV 175
- on RECV 177
- on RECVFROM 179
- on RECVMSG 182
- on SEND 189
- on SENDMSG 192
- on SENDTO 194
- on SETSOCKOPT 195
- on SHUTDOWN 197
- on WRITE 201

S, defines socket descriptor on socket call interface

(continued)

- on WRITEV 202
- security module 105
- SELECT (call) 183
- select mask 15
- select system call
 - C language 135
 - EZACICAL call 242
 - use in server 96, 97
- SELECTEX (call) 187
- SEND (call) 189
- send system call
 - C language 137
 - EZACICAL call 244
- SENDMSG (call) 190
- SENDTO (call) 193
- sendto system call
 - C language 138
 - EZACICAL call 245
- server
 - definition 3
 - socket calls in child server 95
 - socket calls in concurrent server 96
 - socket calls in iterative server 97
- SETSOCKOPT (call) 195
- setsockopt system call
 - C language 124
 - EZACICAL call 246
- SEZATCP data set 22
- SHUTDOWN (call) 197
- shutdown, immediate 84
- shutdown, manual 79
- shutdown system call
 - C language 140
 - EZACICAL call 247
- SNA protocols and CICS 1
- SOCK_STREAM type parameter 141
- sockaddr_in C structure
 - format 115
 - use in accept call 116
 - use in bind call 117
 - use in connect call 119
- SOCKET (call) 198
- socket system call 140
 - EZACICAL call 248
 - use in client 95
 - use in server 96
- sockets
 - compared with ports 6
 - introduction 4
 - passing 97
- Sockets Extended API 4
- SOCRECV parameter on call interface, TAKESOCKET call 200
- SOCTYPE parameter on call interface, on SOCKET 198
- startup
 - automatic 79
 - manually 79
 - program link 85

storage protection machines 23, 26
stub program 18
subtask 18
SUBTASK parameter on call interface, INITAPI call
168
support, environmental 87

T

TAKESOCKET (call) 199
takesocket system call
C language 141
EZACICAL call 249
use in child server 95, 101
task control 104
task-related user exit 18
TCP/IP for MVS, modifying data sets 42
TCP/IP protocols 3
TCP protocol 3
TCPIP, compared with SNA 1
TCPIP.DATA data set 43
tcpip.SEZACMAC data set 111
tcpip.SEZATCP data set 22
TCPIPJOBNAME user id 43
TCPM td queue 37
TERMAPI (call) 200
TIE 215
TIMEOUT parameter on call interface, on SELECT
185
TIMEOUT parameter on call socket interface 146
on SELECTEX 187
timeval structure 115
TOKEN parameter on call interface, on EZACIC06 206
transaction identifier 103
transactions, defining in CICS 23
transient data 37
TRUE module
description 18
monitor control table 38
type (of socket) option 127
type parameter 44
TYPE=CICS 45
TYPE=INITIAL 45
TYPE=LISTENER 46
type parameter in socket call 141

U

UDP protocol 4
use of HOSTENT structure interpreter, EZACIC08 208
utility programs 146, 203
EZACIC04 204
EZACIC05 205
EZACIC06 206
EZACIC08 208

V

VSAM cache file 69

W

WRETMSK parameter on call interface, on SELECT
186

WRITE (call) 200
write system call
C language 142
EZACICAL call 250
use in child server 95
use in client 95
WRITEV (call) 201
WSNDMSK parameter on call interface, on SELECT
186

Readers' Comments — We'd Like to Hear from You

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Complete	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to find	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Easy to understand	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Well organized	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Applicable to your tasks	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

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

Address

Company or Organization

Phone No.



Fold and Tape

Please do not staple

Fold and Tape



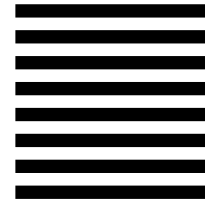
NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Information Development
Department CGMD / Bldg 500
P.O. Box 12195
Research Triangle Park, NC
27709-9990



Fold and Tape

Please do not staple

Fold and Tape



File Number: S370/4300/30xx-50



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC31-7131-03

