# Contents

**Figures** .......................................................... xiii

**Tables** ................................................................... xvii

**About this document** .............................................. xix
- Who should read this document .................................... xix
- How this document is organized .................................... xix
- How to use this document ........................................... xx
  - Determining whether a publication is current .................. xx
- How to contact IBM service ........................................... xxi
- Conventions and terminology that are used in this document ... xxi
- How to read a syntax diagram ..................................... xxii
- Prerequisite and related information ............................. xxvi

**Summary of changes.** ............................................. xxxi

## Chapter 1. Introduction to CICS TCP/IP

- TCP/IP internets ...................................................... 2
- TCP/IP Services Telnet support ................................... 2
- CICS TCP/IP client and server processing ...................... 2
- TCP/IP TCP, UDP, and IP protocols ............................ 3
- The socket API communication functions ...................... 3
- Programming with sockets ......................................... 4
- A typical client-server program flow chart ..................... 8
- Concurrent and iterative servers ................................ 9
- Basic socket calls ................................................... 10
- Server TCP/IP calls ................................................ 11
- Client TCP/IP calls ................................................ 13
- Other socket calls used for servers .............................. 14
- CICS TCP/IP requirements ........................................ 19
- CICS TCP/IP components ......................................... 19

**Summary of what CICS TCP/IP provides** ...................... 19
- The socket calls ..................................................... 20
- The IBM listener ..................................................... 20
- CICS TCP/IP conversion routines ............................... 21
- Rules for configuring the IBM-supplied listener for IPv6 ...... 21

## Chapter 2. Setting up and configuring CICS TCP/IP

- Modifications to the startup of CICS ........................... 23
- Modifying CICS startup (MVS JCL) .............................. 25
- Defining CICS TCP/IP resources .................................. 26
  - Transaction definitions for CICS ................................ 27
  - Using storage protection when running with CICS 3.3.0 or later 27
  - Required program definitions to support CICS TCP/IP ....... 28
  - Updates to file definitions for CICS TCP/IP .................. 36
  - Defining the TCPI transient data queue for CICS TCP/IP .... 38
  - CICS monitoring ................................................... 39
  - CICS program list table .......................................... 47
  - System recovery table .......................................... 47
  - CICS TCP/IP security considerations .......................... 49
- Modifying data sets for TCP/IP services. ................. 50
  - hlq.PROFILE.TCPIP data set .................................. 50
  - hlq.TCPIP.DATA data set ..................................... 51
- Adding a z/OS UNIX System Services segment ................ 52

© Copyright IBM Corp. 2000, 2013
Chapter 3. Configuring the CICS Domain Name Server cache

CICS DNS cache function components
- VSAM cache file
- EZACICR macro
- EZACIC25 module

How the DNS cache handles requests

Using the DNS cache
- Step 1: Create the initialization module
- Step 2: Define the cache file to CICS
- Step 3: Issue EZACIC25
- HOSTENT structure

Chapter 4. Managing IP CICS sockets

Starting and stopping CICS automatically

IP CICS socket interface management
- Using the INQUIRE function
- Using the SET function
- Using the START function
- Using the STOP function
- Abbreviating the EZAO transaction parameters

Starting and stopping CICS TCP/IP with program link

Handling task hangs

Chapter 5. Writing your own listener

Prerequisites for writing your own listener

Using IBM environmental support for user-written listeners

Chapter 6. Writing applications that use the IP CICS sockets API

Writing CICS TCP/IP applications
- The client-listener-child-server application set
- Writing your own concurrent server
- The iterative server CICS TCP/IP application
- The client CICS TCP/IP application

Defining socket addresses
- Address family (domain) support
- IP address allocation
- Port number identification
- Address structures

MVS address spaces relationship between TCP/IP and CICS

TCP/IP network byte ordering convention

GETCLIENTID, GIVESOCKET, and TAKESOCKET

CICS application transaction (IBM listener)
- IBM listener input format
- Examples of client input and the listener processing
- IBM listener output format
- Writing your own security or transaction link modules for the listener

Threadsafe considerations for IP CICS sockets applications

How CICS selects an L8 mode TCB

Data conversion routines

Application Transparent Transport Layer Security
- Example of inbound AT-TLS support
- Example of outbound AT-TLS support

Chapter 7. C language application programming

C socket library
C socket compilation ................................ 166
Changes to DFHYITDL ................................ 166
Compile your program ................................ 166
Structures used in socket calls ....................... 167
The ERRNO variable ................................ 170
C socket call guidance ................................ 170
accept() call ..................................... 171
bind() call ........................................ 173
bind2addrsel() call ................................ 176
close() call ........................................ 178
connect() call ...................................... 179
cfcntl() call ....................................... 181
freeaddrinfo() call ................................ 182
gai_strerror() call ................................ 183
getaddrinfo() call .................................. 184
getclientid() call .................................. 191
gethostbyaddr() call ................................ 191
gethostbyname() call ................................ 192
gethostid() call .................................... 193
ghostname() call ................................... 194
getipv4sourcefilter() call ......................... 195
getnameinfo() call .................................. 197
getpeername() call .................................. 199
getsocket() call .................................... 201
getsocketopt(), setsocketopt() calls ............... 203
getsourcefilter() call ................................ 213
givesocket() call ................................... 215
if_freenameindex() call ............................. 216
if_indexoname() call ................................ 217
if_nameindex() call .................................. 218
if_namequery() call .................................. 219
inet_ntop() call .................................... 220
inet6_is_srcaddr() call .............................. 222
initapi() call ...................................... 224
ioctl() call ........................................ 225
listen() call ........................................ 229
read() call ......................................... 229
recv() call ......................................... 230
recvfrom() call ..................................... 231
select() call ........................................ 233
send() call ......................................... 236
sendto() call ....................................... 238
setipv4sourcefilter() call ......................... 239
setsocketopt() call ................................ 239
setsourcefilter() call ................................ 240
shutdown() call .................................... 241
socket() call ........................................ 242
takesocket() call .................................... 244
write() call ........................................ 245
Address Testing Macros .............................. 245

Chapter 8. Sockets extended API .................. 249
Environmental restrictions and programming requirements for the Callable Socket API .......................... 249
CALL instruction API ................................ 249
Understanding COBOL, assembler, and PL/I call formats .................................................. 250
  COBOL language call format ......................... 250
  Assembler language call format ..................... 251
  PL/I language call format ........................... 252
Converting parameter descriptions ...................... 253
Error messages and return codes ..................... 254

Contents  V
Appendix A. Original COBOL application programming interface (EZACICAL) . . . 421

Using the EZACICAL or Sockets Extended API . . . 421
COBOL compilation . . . 421
The EZACICAL API . . . 423
EZACICAL call format for COBOL . . . 423
EZACICAL call format for PL/I . . . 423
EZACICAL call format for assembler language . . . 424
COBOL and assembler language socket calls . . . 424
COBOL call for ACCEPT . . . 425
COBOL call for BIND . . . 426
COBOL call for CLOSE . . . 427
COBOL call for CONNECT . . . 428
COBOL call for FCNTL . . . 429
COBOL call for GETCLIENTID . . . 429
COBOL call for GETHOSTID . . . 431
Using data translation programs for socket call interface . . . 401
Data translation from ASCII and EBCDIC data notation . . . 401
Bit string processing . . . 402
COBOL call for GETHOSTNAME .................................. 431
COBOL call for GETPEERNAME .................................. 433
COBOL call for GETHOSTNAME .................................. 433
COBOL call for GETSOCKNAME .................................. 435
COBOL call for GIVESOCKET .................................. 436
COBOL call for INITAPI ...................................... 437
COBOL call for IOCTL ...................................... 439
COBOL call for LISTEN ...................................... 440
COBOL call for READ ...................................... 441
COBOL call for RECVFROM .................................... 442
COBOL call for SELECT ...................................... 443
COBOL call for SEND ...................................... 445
COBOL call for SENDTO ..................................... 446
COBOL call for SETSOCKOPT ................................... 447
COBOL call for SHUTDOWN .................................... 449
COBOL call for SOCKET ..................................... 450
COBOL call for TAKESOCKET ................................... 451
COBOL call for WRITE ...................................... 452

Appendix B. Return codes ...................................... 455
Sockets return codes (ERRNOs) .................................. 455
Sockets extended ERRNOs ..................................... 465

Appendix C. GETSOCKOPT/SETSOCKOPT command values .................... 471

Appendix D. CICS sockets messages .................................. 475

EZY1218—EZY1366 ............................................. 475
EZY1218E: mm/dd/yy hh:mm:ss PROGRAM programname DISABLED TRANID= transactionid PARTNER INET
ADDR=inetaddress PORT=portnumber .................................. 475
EZY1219E: mm/dd/yy hh:mm:ss UNEXPECTED eventtype EVENT IN LISTENER transactionid FROM CLIENT IP
ADDRESS ipaddress PORT portnumber .................................. 476
EZY1220E: mm/dd/yy hh:mm:ss READ FAILURE ON CONFIGURATION FILE PHASE=phase EIBRESP2=response
EZY1221E: mm/dd/yy hh:mm:ss CICS SOCKETS ENABLE FAILURE EIBRCODE BYTE2 = resp_code .......... 477
EZY1222E: mm/dd/yy hh:mm:ss CICS/SOCKETS REGISTRATION FAILURE RETURN code= return_code ..... 478
EZY1223E: mm/dd/yy hh:mm:ss CICS/SOCKETS ATTACH FAILURE RETURN CODE = return_code REASON
CODE = reason_code ............................................. 478
EZY1224E: mm/dd/yy hh:mm:ss CICS/SOCKETS INITIALIZATION SUCCESSFUL USING tasking_method .... 479
EZY1225E: mm/dd/yy hh:mm:ss STARTBR FAILURE ON CICS/SOCKETS CONFIGURATION FILE PHASE=xx
EIBRESP2=rrrrrr ............................................. 480
EZY1226E: mm/dd/yy hh:mm:ss READNEXT FAILURE ON CICS/SOCKETS CONFIGURATION FILE
PHASE=xx EIBRESP2=rrrrrr ..................................... 480
EZY1227E: mm/dd/yy hh:mm:ss CICS/SOCKETS INVALID LISTENER TRANID = tran .................. 481
EZY1228E: mm/dd/yy hh:mm:ss CICS/SOCKETS LISTENER TRANSACTION tran DISABLED ............. 482
EZY1229E: mm/dd/yy hh:mm:ss CICS SOCKETS LISTENER TRANSACTION tran NOT AUTHORIZED .... 482
EZY1230E: mm/dd/yy hh:mm:ss CICS SOCKETS LISTENER PROGRAM ID mmmmmmmmmmm INVALID .... 483
EZY1231E: mm/dd/yy hh:mm:ss CICS SOCKETS LISTENER PROGRAM ID mmmmmmmmmmm DISABLED .... 483
EZY1232E: mm/dd/yy hh:mm:ss CICS SOCKETS LISTENER tran NOT ON CONFIGURATION FILE .... 484
EZY1235I: module ENTRY POINT IS address ................................... 487
EZY1235E: mm/dd/yy hh:mm:ss CICS SOCKETS MODULE mmmmmmmmm ABEND xxxx .......... 484
EZY1252E: mm/dd/yy hh:mm:ss CICS SOCKETS LISTENER TRANID = tran NOT ON CONFIGURATION FILE ...... 485
EZY1253E: mm/dd/yy hh:mm:ss CICS SOCKETS LISTENER tran NOT ON CONFIGURATION FILE ........ 485
EZY1254E: mm/dd/yy hh:mm:ss CACHE FILE ERROR RESP2 VALUE = message CALL # * .......... 486
EZY1255E: mm/dd/yy hh:mm:ss TEMPORARY STORAGE ERROR RESP2 VALUE = message CALL # * ...... 486
EZY1256E: mm/dd/yy hh:mm:ss CICS SOCKETS INTERFACE NOT ENABLED PRIOR TO LISTENER STARTUP 487
EZY1258I: module ENTRY POINT IS address ................................... 487
EZY1259E: mm/dd/yy hh:mm:ss IOCTL CALL FAILURE TRANSACTION=transactionid TASKID=tasknumber
ERRNO=errno ................................................. 488
EZY1260E: mm/dd/yy hh:mm:ss EZACIC03 ATTACH FAILED GPR15=xxxxxxx ERRNO=errno TRANS=tran
TASK=cicstask .............................................. 488

Contents vii
EZY1301E: mm/dd/yy hh:mm:ss CONNECTION CLOSED BY CLIENT TRANSACTION= transactionid PARTNER INET ADDR= ipaddr PORT= port
EZY1302I: mm/dd/yy hh:mm:ss READ TIMEOUT PARTNER INET ADDR= inetaddress PORT= portnumber
LISTENER TRANID= tran_id TASKID= task_id
EZY1303I: mm/dd/yy hh:mm:ss EZACIC02 GIVESOCKET TIMEOUT TRANS transactionid PARTNER INET ADDR= inetaddress PORT= portnumber
EZY1304I: mm/dd/yy hh:mm:ss UNEXPECTED INPUT EVENT TRANSACTION transactionid PARTNER INET ADDR= inetaddress PORT= portnumber
EZY1305E: mm/dd/yy hh:mm:ss UNEXPECTED EXCEPTION EVENT TRANS transactionid PARTNER INET ADDR= inetaddress PORT= portnumber
EZY1306E: mm/dd/yy hh:mm:ss SECURITY EXIT mmmmmmmmmm IS NOT DEFINED TRANID= tran TASKID=xxxxxxx
EZY1307E: mm/dd/yy hh:mm:ss MAXIMUM # OF SOCKETS USED TRANS= tran TASKID= cicstask ERRNO=errno
EZY1308E: mm/dd/yy hh:mm:ss ACCEPT CALL FAILURE TRANSACTION= tran TASKID= cicstask ERRNO=errno
EZY1309E: mm/dd/yy hh:mm:ss GIVESOCKET FAILURE TRANS transactionid TASKID=tasknumber ERRNO=errno INET ADDR=inetaddress PORT=portnumber
EZY1310E: mm/dd/yy hh:mm:ss IC VALUE NOT NUMERIC TRANID=transactionid PARTNER INET ADDR=inetaddress PORT=portnumber
EZY1311E: mm/dd/yy hh:mm:ss CICS TRANID transactionid NOT AUTHORIZED PARTNER INET ADDR=inetaddress PORT=portnumber
EZY1312E: mm/dd/yy hh:mm:ss SECURITY EXIT mmmmmmmm CANNOT BE LOADED TRANID= tran TASKID=xxxxxxx
EZY1313E: mm/dd/yy hh:mm:ss LISTENER NOT AUTHORIZED TO ACCESS SECURITY EXIT mmmmmmmm TRANID= tran TASKID=xxxxxxx
EZY1314E: mm/dd/yy hh:mm:ss SECURITY EXIT mmmmmmmm IS DISABLED TRANID= tran TASKID=xxxxxxx
EZY1315E: mm/dd/yy hh:mm:ss INVALID TRANSID transactionid PARTNER INET ADDR=inetaddress PORT=portnumber
EZY1316E: mm/dd/yy hh:mm:ss TRANSID transactionid IS DISABLED PARTNER INET ADDR=inetaddress PORT=portnumber
EZY1317E: mm/dd/yy hh:mm:ss TRANSID transactionid IS NOT AUTHORIZED PARTNER INET ADDR=inetaddress PORT=portnumber
EZY1318E: mm/dd/yy hh:mm:ss TD START SUCCESSFUL QUEUEID= que
EZY1319E: mm/dd/yy hh:mm:ss QIDERR FOR TD DESTINATION queuename PARTNER INET ADDR=inetaddress PORT=portnumber
EZY1320E: mm/dd/yy hh:mm:ss I/O ERROR FOR TD DESTINATION queuename PARTNER INET ADDR=inetaddress PORT=portnumber
EZY1321E: mm/dd/yy hh:mm:ss LENGTH ERROR FOR TD DESTINATION queuename PARTNER INET ADDR=inetaddress PORT=portnumber
EZY1322E: mm/dd/yy hh:mm:ss TD DESTINATION queuename DISABLED PARTNER INET ADDR=inetaddress PORT=portnumber
EZY1323E: mm/dd/yy hh:mm:ss TD DESTINATION queuename OUT OF SPACE PARTNER INET ADDR=inetaddress PORT=portnumber
EZY1324E: mm/dd/yy hh:mm:ss TD START FAILED QUEUE ID=queuename PARTNER INET ADDR=inetaddress PORT=portnumber
EZY1325I: mm/dd/yy hh:mm:ss START SUCCESSFUL TRANID=transactionid PARTNER INET ADDR=inetaddress PORT=portnumber
EZY1326E: mm/dd/yy hh:mm:ss START I/O ERROR TRANID=transactionid PARTNER INET ADDR=inetaddress PORT=portnumber
EZY1327E: mm/dd/yy hh:mm:ss START TRANSACTION ID transactionid INVALID PARTNER INET ADDR=inetaddress PORT=portnumber
EZY1328E: mm/dd/yy hh:mm:ss START TRANSACTION ID transactionid NOT AUTHORIZED PARTNER INET ADDR=inetaddress PORT=portnumber
EZY1329E: mm/dd/yy hh:mm:ss START FAILED (99) TRANID=transactionid PARTNER INET ADDR=inetaddress PORT=portnumber
EZY1330E: mm/dd/yy hh:mm:ss IC START SUCCESSFUL TRANID=transactionid PARTNER INET ADDR=inetaddress PORT=portnumber
EZY1331E: mm/dd/yy hh:mm:ss IC START I/O ERROR TRANID=transactionid PARTNER INET ADDR=inetaddress PORT=portnumber

Contents ix
Figures

1. The use of CICS sockets ............................................. 2
2. TCP/IP protocols compared to the OSI model and SNA ..................... 9
3. A typical client-server session ...................................... 10
4. An iterative server .................................................. 10
5. A concurrent server .................................................. 10
6. The SELECT call .................................................... 15
7. How user applications access TCP/IP networks with CICS TCP/IP (run-time environment) ................. 20
8. JCL for CICS startup with the TCP/IP socket interface (part 1 of 2) ................. 24
9. JCL for CICS startup with the TCP/IP socket interface (part 2 of 2) ................. 25
10. EZAC, transaction to configure the socket interface ....................... 28
11. EZAO, transaction to enable the socket interface ....................... 28
12. EZAP, transaction to disable the socket interface ....................... 28
13. CSKL, Listener task transaction ................................... 30
14. EZACIC00, connection manager program ................................ 30
15. EZACIC01, task related user exit program ................................ 30
16. EZACIC02, listener program ........................................ 30
17. EZACIC20, front-end module for CICS sockets .......................... 30
18. EZACIC21, initialization module for CICS sockets ....................... 31
19. EZACIC22, termination module for CICS sockets ....................... 31
20. EZACIC23, primary module for transaction EZAC ....................... 31
21. EZACIC24, message delivery module for CICS sockets ................... 31
22. EZACIC25, domain name server cache module ........................ 32
23. EZACICM, maps used by the EZAO transaction ........................ 32
24. EZACICMEU.S. English text delivery module ........................ 32
25. EZACICSC, sample IPv4 child server transaction and program definitions ............... 33
26. EZACICSS, sample iterative IPv4 server transaction and program definitions ............... 33
27. EZACIC6C, sample IPv6 child server transaction and program definitions ............... 34
28. EZACIC6S, sample iterative IPv6 server transaction and program definitions ............... 34
29. EZACICAC, sample assembler child server transaction and program definitions ............... 35
30. EZACICAS, sample assembler server transaction and program definitions ............... 35
31. ALTER PROGRAM instructions ....................................... 36
32. DFHCSDUP commands to define EZACONFG .......................... 37
33. DFHCSDUP commands to define EZACACHE .......................... 38
34. CICS TCP/IP Transient Data Queue definitions ........................ 39
35. The Monitor Control Table (MCT) for TRUE ................................ 41
36. The Monitor Control Table (MCT) for listener ......................... 44
37. EZASOKET threadsafe transaction ................................... 46
38. Definition of the hlq.TCP/IP profile ................................ 51
39. The TCPIPJOBNAME parameter in the hlq.TCPIPDATA data set ............... 52
40. EZACICFG configuration file ....................................... 53
41. Example of JCL to define a configuration file .......................... 66
42. EZAC initial screen .................................................. 69
43. EZAC, ALTER screen ............................................... 70
44. EZAC, ALTER, CICS screen ........................................ 71
45. EZAC, ALTER, CICS detail screen .................................. 71
46. EZAC, ALTER, LISTENER screen ................................... 72
47. EZAC, ALTER, LISTENER detail screen 1- Standard listener ............... 72
48. EZAC, ALTER, LISTENER detail screen 2: Standard listener ............... 73
49. EZAC, ALTER, LISTENER detail screen 1- Enhanced listener ............... 73
50. EZAC, ALTER, LISTENER detail screen 2: Enhanced listener ............... 74
51. EZAC, CONVERT, LISTENER screen .................................. 75
52. EZAC, CONVERT, LISTENER detail screen 1- Standard listener ............... 76
53. EZAC, CONVERT, LISTENER detail screen 2: Standard listener ............... 76
54. EZAC, CONVERT, LISTENER detail screen 1- Enhanced listener ............... 77
55. EZAC, CONVERT, LISTENER detail screen 2: Enhanced listener ............... 77
113. Example of PL/I layout of the listener output format - Enhanced listener with an IPv4 socket address structure .................................. 150
114. Example of PL/I layout of the listener output format - Enhanced listener with an IPv6 socket address structure .................................. 151
115. Example of assembler layout of the listener output format - Enhanced listener supporting both an IPv4 and an IPv6 socket address structure .................................. 151
116. Example of C structure of the listener output format - Enhanced listener supporting both an IPv4 and an IPv6 socket address structure .................................. 152
117. Storage definition statement examples ........................................................................................................... 254
118. ACCEPT call instructions example ........................................................................................................... 256
119. BIND call instruction example ........................................................................................................... 258
120. BIND2ADDRSEL call instructions example ........................................................................................................... 261
121. CLOSE call instruction example ........................................................................................................... 264
122. CONNECT call instruction example ........................................................................................................... 266
123. FCNTL call instruction example ........................................................................................................... 269
124. FREEADDRINFO call instruction example ........................................................................................................... 271
125. GETADDRINFO call instruction example ........................................................................................................... 273
126. GETCLIENTID call instruction example ........................................................................................................... 281
127. GETHOSTBYADDR call instruction example ........................................................................................................... 283
128. HOSTENT structure returned by the GETHOSTBYADDR call ........................................................................................................... 284
129. GETHOSTBYNAME call instruction example ........................................................................................................... 286
130. HOSTENT structure returned by the GETHOSTBYNAME call ........................................................................................................... 287
131. GETHOSTID call instruction example ........................................................................................................... 288
132. GETHOSTNAME call instruction example ........................................................................................................... 289
133. GETNAMEINFO call instruction example ........................................................................................................... 291
134. GETPEERNAME call instruction example ........................................................................................................... 295
135. GETSOCKNAME call instruction example ........................................................................................................... 297
136. GETSOCKOPT call instruction example ........................................................................................................... 299
137. GIVESOCKET call instruction example ........................................................................................................... 315
138. INET6_IS_SRCADDR call instruction example ........................................................................................................... 317
139. INITAPI call instruction example ........................................................................................................... 321
140. IOCTL call instruction example ........................................................................................................... 323
141. COBOL language example for SIOCGHOMEIF6 ........................................................................................................... 325
142. COBOL language example for SIOCGIFNAMEINDEX ........................................................................................................... 327
143. COBOL II example for SIOCGIFCONF ........................................................................................................... 334
144. LISTEN call instruction example ........................................................................................................... 335
145. NTOP call instruction example ........................................................................................................... 336
146. PTON call instruction example ........................................................................................................... 339
147. READ call instruction example ........................................................................................................... 341
148. READV call instruction example ........................................................................................................... 343
149. RECV call instruction example ........................................................................................................... 345
150. RECVFROM call instruction example ........................................................................................................... 347
151. RECVMSG call instruction example (Part 1 of 2) ........................................................................................................... 351
152. RECVMSG call instruction example (Part 2 of 2) ........................................................................................................... 351
153. SELECT call instruction example ........................................................................................................... 356
154. SELECTEX call instruction example ........................................................................................................... 360
155. SEND call instruction example ........................................................................................................... 366
156. SENDMSG call instruction example ........................................................................................................... 368
157. SENDTO call instruction example ........................................................................................................... 373
158. SETSOCKOPT call instruction example ........................................................................................................... 376
159. SHUTDOWN call instruction example ........................................................................................................... 393
160. SOCKET call instruction example ........................................................................................................... 394
161. TAKESOCKET call instruction example ........................................................................................................... 396
162. TERMAPI call instruction example ........................................................................................................... 398
163. WRITE call instruction example ........................................................................................................... 399
164. WRITEV call instruction example ........................................................................................................... 400
165. EZACIC04 EBCDIC-to-ASCII table ........................................................................................................... 404
166. EZACIC04 call instruction example ........................................................................................................... 405
167. EZACIC05 ASCII-to-EBCDIC ........................................................................................................... 406
168. EZACIC05 call instruction example ........................................................................................................... 407
169. EZACIC06 call instruction example ........................................................................................................... 408
170. EZAZIC08 call instruction example ........................................ 410
171. EZACIC09 call instruction example (Part 1 of 2) .................. 414
172. EZACIC09 call instruction example (Part 2 of 2) .................. 415
173. EZACIC14 EBCDIC-to-ASCII table ........................................ 417
174. EZACIC14 call instruction example ........................................ 418
175. EZACIC15 ASCII-to-EBCDIC table ........................................ 419
176. EZACIC15 call instruction example ........................................ 420
177. Modified JCL for COBOL compilation ................................ 422
178. EZACICSC IPv4 child server sample ..................................... 571
179. EZACICSS IPv4 iterative server sample ................................ 578
180. EZACIC6C IPv6 child server sample ..................................... 595
181. EZACIC6S IPv6 iterative server sample ................................ 604
182. EZACICAC assembler child server sample ......................... 624
183. EZACICAS assembler iterative server sample ....................... 631
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>First fullword passed in a bit string in select</td>
<td>16</td>
</tr>
<tr>
<td>2.</td>
<td>Second fullword passed in a bit string in select</td>
<td>16</td>
</tr>
<tr>
<td>3.</td>
<td>Security/Transaction Exit program information fields</td>
<td>22</td>
</tr>
<tr>
<td>4.</td>
<td>Configuration options affected by OTE</td>
<td>57</td>
</tr>
<tr>
<td>5.</td>
<td>Listener’s action based on RTYTIME and stack state</td>
<td>63</td>
</tr>
<tr>
<td>6.</td>
<td>Conditions for translation of tranid and user data</td>
<td>64</td>
</tr>
<tr>
<td>7.</td>
<td>Functions supported by the EZAC transaction</td>
<td>68</td>
</tr>
<tr>
<td>8.</td>
<td>Calls for the client application</td>
<td>131</td>
</tr>
<tr>
<td>9.</td>
<td>Calls for the server application</td>
<td>132</td>
</tr>
<tr>
<td>10.</td>
<td>Calls for the concurrent server application</td>
<td>133</td>
</tr>
<tr>
<td>11.</td>
<td>CLIENTID structures</td>
<td>139</td>
</tr>
<tr>
<td>12.</td>
<td>Listener configuration presented to security or transaction exit</td>
<td>142</td>
</tr>
<tr>
<td>13.</td>
<td>Listener output format - Standard listener</td>
<td>144</td>
</tr>
<tr>
<td>14.</td>
<td>Listener output format - Enhanced listener</td>
<td>148</td>
</tr>
<tr>
<td>15.</td>
<td>Security or transaction exit data</td>
<td>153</td>
</tr>
<tr>
<td>16.</td>
<td>Different concurrency attributes for IP CICS sockets task-related user exits</td>
<td>158</td>
</tr>
<tr>
<td>17.</td>
<td>Inbound AT-TLS support</td>
<td>162</td>
</tr>
<tr>
<td>18.</td>
<td>Outbound AT-TLS support</td>
<td>163</td>
</tr>
<tr>
<td>19.</td>
<td>C structures</td>
<td>167</td>
</tr>
<tr>
<td>20.</td>
<td>OPTNAME options for GETSOCKOPT and SETSOCKOPT</td>
<td>300</td>
</tr>
<tr>
<td>21.</td>
<td>IOCTL call arguments</td>
<td>331</td>
</tr>
<tr>
<td>22.</td>
<td>OPTNAME options for GETSOCKOPT and SETSOCKOPT</td>
<td>377</td>
</tr>
<tr>
<td>23.</td>
<td>Effect of SHUTDOWN socket call</td>
<td>391</td>
</tr>
<tr>
<td>24.</td>
<td>Sockets ERRNOs</td>
<td>455</td>
</tr>
<tr>
<td>25.</td>
<td>Sockets extended ERRNOs</td>
<td>465</td>
</tr>
<tr>
<td>26.</td>
<td>GETSOCKOPT/SETSOCKOPT command values for Macro, Assembler, COBOL and PL/I</td>
<td>471</td>
</tr>
<tr>
<td>27.</td>
<td>GETSOCKOPT/SETSOCKOPT optname value for C programs</td>
<td>472</td>
</tr>
</tbody>
</table>
About this document

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

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

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

Who should read this document

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

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

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

How this document is organized

This document contains the following topics:

- Chapter 1, “Introduction to CICS TCP/IP,” on page 1 provides an overview of CICS TCP/IP.
- Chapter 2, “Setting up and configuring CICS TCP/IP,” on page 23 describes the steps required to configure CICS TCP/IP.
- Chapter 3, “Configuring the CICS Domain Name Server cache,” on page 97 describes how to configure the CICS domain name server cache.
- Chapter 4, “Managing IP CICS sockets,” on page 107 explains how to start and stop (enable and disable) CICS TCP/IP.
- Chapter 5, “Writing your own listener,” on page 125 discusses writing your own listener.
- Chapter 6, “Writing applications that use the IP CICS sockets API,” on page 129 describes how to write applications that use the sockets application.
programming interface (API). It describes typical sequences of calls for client, concurrent server (with associated child server processes), and iterative server programs.

- Chapter 7, “C language application programming,” on page 165 describes the C language API provided by CICS TCP/IP.
- Chapter 8, “Sockets extended API,” on page 249 describes the sockets extended API.
- Appendix A, “Original COBOL application programming interface (EZACICAL),” on page 421 describes the EZACICAL API.
- Appendix B, “Return codes,” on page 455 describes system-wide message numbers and codes set by the system calls.
- Appendix C, “GETSOCKOPT/SETSOCKOPT command values,” on page 471 provides the decimal or hexadecimal values associated with the GETSOCKOPT/SETSOCKOPT OPTNAMES supported by the APIs discussed in this information.
- Appendix D, “CICS sockets messages,” on page 475 contains CICS socket interface messages.
- Appendix E, “Sample programs,” on page 571 contains samples of the following programs:
  - EZACICSC - An IPv4 child server
  - EZACICSS - An IPv4 iterative server
  - EZACIC6C - An IPv6 child server
  - EZACIC6S - An IPv6 iterative server
  - EZACICAC - An assembler child server
  - EZACICAS - An assembler iterative server
- Appendix F, “Related protocol specifications,” on page 649 lists the related protocol specifications for TCP/IP.
- Appendix G, “Accessibility,” on page 673 contains information about features that help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully.
- “Bibliography” on page 687 contains descriptions of the information in the z/OS™ Communications Server library.

How to use this document

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

Determining whether a publication is current

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

- At the end of a publication’s order number there is a dash followed by two digits, often referred to as the dash level. A publication with a higher dash level is more current than one with a lower dash level. For example, in the publication order number GC28-1747-07, the dash level 07 means that the publication is more current than previous levels, such as 05 or 04.
• If a hardcopy publication and a softcopy publication have the same dash level, it is possible that the softcopy publication is more current than the hardcopy publication. Check the dates shown in the Summary of Changes. The softcopy publication might have a more recently dated Summary of Changes than the hardcopy publication.

• To compare softcopy publications, you can check the last 2 characters of the publication’s file name (also called the book name). The higher the number, the more recent the publication. Also, next to the publication titles in the CD-ROM booklet and the readme files, there is an asterisk (*) that indicates whether a publication is new or changed.

How to contact IBM service

For immediate assistance, visit this website: http://www.software.ibm.com/network/commserv/support/

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

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

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

If you would like to provide feedback on this publication, see “Communicating your comments to IBM” on page 699.

Conventions and terminology that are used in this document

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

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

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

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

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

Note: In this information, you might see the term RDMA network interface card (RNIC) that is used to refer to the IBM 10GbE RoCE Express feature.
For definitions of the terms and abbreviations that are used in this document, you can view the latest IBM terminology at the IBM Terminology website.

**Clarification of notes**

Information traditionally qualified as Notes is further qualified as follows:

- **Note**: Supplemental detail
- **Tip**: Offers shortcuts or alternative ways of performing an action; a hint
- **Guideline**: Customary way to perform a procedure
- **Rule**: Something you must do; limitations on your actions
- **Restriction**: Indicates certain conditions are not supported; limitations on a product or facility
- **Requirement**: Dependencies, prerequisites
- **Result**: Indicates the outcome

**How to read a syntax diagram**

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

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

**Symbols and punctuation**

The following symbols are used in syntax diagrams:

- **Symbol**: Marks the beginning of the command syntax.
- **►**: Indicates that the command syntax is continued.
- **|**: Marks the beginning and end of a fragment or part of the command syntax.
- **►►**: Marks the end of the command syntax.

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

**Commands**

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

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

The following types of parameters are used in syntax diagrams.

Required
Required parameters are displayed on the main path.

Optional
Optional parameters are displayed below the main path.

Default
Default parameters are displayed above the main path.

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

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

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

Syntax examples

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

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

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

Required operands

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

Required_operand

Optional values

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

Operand

Selecting more than one operand

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

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

Blank spaces in syntax diagrams

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

Default operands

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

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

Syntax fragments

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

Syntax fragment:

1ST_OPERAND, 2ND_OPERAND, 3RD_OPERAND

Prerequisite and related information

z/OS Communications Server function is described in the z/OS Communications Server library. Descriptions of those documents are listed in "Bibliography" on page 687, in the back of this document.

Required information

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

Softcopy information

Softcopy publications are available in the following collection.

<table>
<thead>
<tr>
<th>Titles</th>
<th>Order Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM System z® Redbooks Collection</td>
<td>SK3T-7876</td>
<td>The IBM Redbooks® publications selected for this CD series are taken from the IBM Redbooks inventory of over 800 books. All the Redbooks publications that are of interest to the zSeries® platform professional are identified by their authors and are included in this collection. The zSeries subject areas range from e-business application development and enablement to hardware, networking, Linux, solutions, security, parallel sysplex, and many others. For more information about the Redbooks publications, see <a href="http://www-03.ibm.com/systems/z/os/zos/zfavorites/">http://www-03.ibm.com/systems/z/os/zos/zfavorites/</a></td>
</tr>
</tbody>
</table>
Other documents

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

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

To find the complete z/OS library, including the z/OS Information Center, see www.ibm.com/systems/z/os/zos/zos/bkserv/.

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

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

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNA Formats</td>
<td>GA27-3136</td>
</tr>
<tr>
<td>TCP/IP Tutorial and Technical Overview</td>
<td>GG24-3376</td>
</tr>
<tr>
<td>Understanding LDAP</td>
<td>SG24-4986</td>
</tr>
<tr>
<td>z/OS Cryptographic Services System SSL Programming</td>
<td>SC24-5901</td>
</tr>
<tr>
<td>z/OS IBM Tivoli Directory Server Administration and Use for z/OS</td>
<td>SC23-6788</td>
</tr>
<tr>
<td>z/OS JES2 Initialization and Tuning Guide</td>
<td>SA32-0991</td>
</tr>
<tr>
<td>z/OS Problem Management</td>
<td>SC23-6844</td>
</tr>
<tr>
<td>z/OS MVS Diagnosis: Reference</td>
<td>GA32-0904</td>
</tr>
<tr>
<td>z/OS MVS Diagnosis: Tools and Service Aids</td>
<td>GA32-0905</td>
</tr>
<tr>
<td>z/OS MVS Using the Subsystem Interface</td>
<td>SA38-0679</td>
</tr>
<tr>
<td>z/OS Program Directory</td>
<td>G111-9848</td>
</tr>
<tr>
<td>z/OS UNIX System Services Command Reference</td>
<td>SA23-2280</td>
</tr>
<tr>
<td>z/OS UNIX System Services Planning</td>
<td>GA32-0884</td>
</tr>
<tr>
<td>z/OS UNIX System Services Programming: Assembler Callable Services Reference</td>
<td>SA23-2281</td>
</tr>
<tr>
<td>z/OS UNIX System Services User’s Guide</td>
<td>SA23-2279</td>
</tr>
<tr>
<td>z/OS XL C/C++ Runtime Library Reference</td>
<td>SC14-7314</td>
</tr>
<tr>
<td>zEnterprise 196, System z10, System z9 and eServer zSeries OSA-Express Customer’s Guide and Reference</td>
<td>SA22-7935</td>
</tr>
</tbody>
</table>
Redbooks publications

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

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM z/OS V1R13 Communications Server TCP/IP Implementation, Volume 1: Base Functions, Connectivity, and Routing</td>
<td>SG24-7996</td>
</tr>
<tr>
<td>IBM z/OS V1R13 Communications Server TCP/IP Implementation, Volume 2: Standard Applications</td>
<td>SG24-7997</td>
</tr>
<tr>
<td>IBM z/OS V1R13 Communications Server TCP/IP Implementation, Volume 3: High Availability, Scalability, and Performance</td>
<td>SG24-7998</td>
</tr>
<tr>
<td>IBM z/OS V1R13 Communications Server TCP/IP Implementation, Volume 4: Security and Policy-Based Networking</td>
<td>SG24-7999</td>
</tr>
<tr>
<td>IBM Communication Controller Migration Guide</td>
<td>SG24-6298</td>
</tr>
<tr>
<td>IP Network Design Guide</td>
<td>SG24-2580</td>
</tr>
<tr>
<td>Managing OS/390 TCP/IP with SNMP</td>
<td>SG24-5866</td>
</tr>
<tr>
<td>Migrating Subarea Networks to an IP Infrastructure Using Enterprise Extender</td>
<td>SG24-5957</td>
</tr>
<tr>
<td>SecureWay Communications Server for OS/390 V2R8 TCP/IP: Guide to Enhancements</td>
<td>SG24-5631</td>
</tr>
<tr>
<td>SNA and TCP/IP Integration</td>
<td>SG24-5291</td>
</tr>
<tr>
<td>TCP/IP in a Sysplex</td>
<td>SG24-5235</td>
</tr>
<tr>
<td>TCP/IP Tutorial and Technical Overview</td>
<td>GG24-3376</td>
</tr>
<tr>
<td>Threaddes Considerations for CICS</td>
<td>SG24-6351</td>
</tr>
</tbody>
</table>

Where to find related information on the Internet

z/OS

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

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

z/OS Internet Library

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

www.ibm.com/systems/z/os/zos/bkserv/

IBM Communications Server product

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


IBM Communications Server product support

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

IBM Communications Server performance information

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

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

IBM Systems Center publications

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

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

IBM Systems Center flashes

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

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

RFCs

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

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

Internet drafts

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

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

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

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

DNS websites

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

USENET news groups

comp.protocols.dns.bind

BIND mailing lists

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

BIND Users

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

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

• Subscribe by sending mail to bind9-users-request@isc.org.
• Submit questions or answers to this forum by sending mail to bind9-users@isc.org.
The z/OS Basic Skills Information Center

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

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

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

To access the z/OS Basic Skills Information Center, open your web browser to the following website, which is available to all users (no login required):

http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp
Summary of changes

This section describes the release enhancements that were made.

New in z/OS Version 2 Release 1

For specifics on the enhancements for z/OS Version 2, Release 1, see the following publications:

- z/OS Summary of Message and Interface Changes
- z/OS Introduction and Release Guide
- z/OS Planning for Installation
- z/OS Migration
Chapter 1. Introduction to CICS TCP/IP

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

CICS Transaction Server (CICS TS) is an online transaction processing system. Application programs using CICS can handle large numbers of data transactions from large networks of computers and terminals.

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

CICS TCP/IP allows remote users to access CICS client/server applications over TCP/IP Internets. Figure 1 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.
CICS TCP/IP provides a variant of the Berkeley Software Distribution 4.3 sockets interface, which is widely used in TCP/IP networks and is based on the UNIX system and other operating systems. The socket interface consists of a set of calls that your CICS application programs can use to set up connections, send and receive data, and perform general communications control functions. The programs can be written in COBOL, PL/I, assembler language, or the C language.

TCP/IP internets

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

Like SNA, TCP/IP is a communication protocol used between physically separated computer systems. Unlike SNA and most other protocols, TCP/IP is not designed for a particular hardware technology. TCP/IP can be implemented on a wide variety of physical networks, and is specially designed for communicating between systems on different physical networks (local and wide area). This is called Internetworking.

TCP/IP Services Telnet support

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

CICS TCP/IP client and server processing

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

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

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

**TCP/IP TCP, UDP, and IP protocols**

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.

<table>
<thead>
<tr>
<th>SNA</th>
<th>OSI</th>
<th>TCP/IP Family</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>7</td>
<td>Application</td>
</tr>
<tr>
<td>Presentation</td>
<td>6</td>
<td>Presentation</td>
</tr>
<tr>
<td>Data Flow</td>
<td>5</td>
<td>Session</td>
</tr>
<tr>
<td>Transmission</td>
<td>4</td>
<td>Transport</td>
</tr>
<tr>
<td>Path Control</td>
<td>3</td>
<td>Network</td>
</tr>
<tr>
<td>Data Link</td>
<td>2</td>
<td>Data Link</td>
</tr>
<tr>
<td>Physical</td>
<td>1</td>
<td>Physical</td>
</tr>
</tbody>
</table>

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

The protocols implemented by TCP/IP Services and used by CICS TCP/IP are shown in the right hand column in Figure 2.

**Transmission Control Protocol (TCP)**

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

**User Datagram Protocol (UDP)**

UDP is also a transport-layer protocol and is an alternative to TCP. It provides an unreliable datagram connection between applications. Data is transmitted link by link; there is no end-to-end connection. The service provides no guarantees. Data can be lost or duplicated, and datagrams can arrive out of order.

**Internet Protocol (IP)**

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

**The socket API communication functions**

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

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

In addition to these basic functions, the APIs enable you to:
• Interrogate the network system to get names and status of relevant resources
• Perform system and control functions as required

CICS TCP/IP provides three TCP/IP socket application program interfaces (APIs), similar to those used on UNIX systems. One interfaces to C language programs, the other two to COBOL, PL/I, and assembler language programs.

• **C language.** Historically, TCP/IP has been linked to the C language and the UNIX operating system. Textbook descriptions of socket calls are usually given in C, and most socket programmers are familiar with the C interface to TCP/IP. For these reasons, TCP/IP Services includes a C language API. If you are writing new TCP/IP applications and are familiar with C language programming, you might prefer to use this interface. See Chapter 7, “C language application programming,” on page 165 for the sockets calls provided by TCP/IP Services.

• **Sockets Extended API (COBOL, PL/I, assembler language).** The Sockets Extended API is for those who want to write in COBOL, PL/I, or assembler language, or who have COBOL, PL/I, or assembler language programs that need to be modified to run with TCP/IP. If you are writing new TCP/IP applications in COBOL, PL/I, or assembler language, you might prefer to use the Sockets Extended API. See Chapter 8, “Sockets extended API,” on page 249 for details of this interface.

• **Version 2.2.1 (COBOL, PL/I, assembler language).** This is the API that was offered to users of the original release of CICS TCP/IP. It is similar in use to the Sockets Extended API. The Version 2.2.1 API is available for those who want to maintain Version 2.2.1 programs. This interface is described in Appendix A, “Original COBOL application programming interface (EZACICAL),” on page 421.

**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.

**MVS socket APIs**

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

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

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

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

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

**Addressing TCP/IP hosts**

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

**Address families supported for TCP/IP:**

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

**Socket addresses in the AF_INET family:**

A socket address in the AF_INET family contains four fields:

- The name of the address family itself (AF_INET)
- A port
- An IPv4 Internet address
- An 8-byte reserved field

In COBOL, an IPv4 socket address looks like this:

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

A socket address in the AF_INET6 family contains five fields:

- The name of the address family itself (AF_INET6)
- A port

---

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.
• Flow information indicating traffic class and flow label
• An IPv6 Internet address
• A scope ID indicating link scope

In COBOL, an IPv6 socket address looks like this:

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

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

```
01 SOCKADDR.
  05 SOCK-FAMILY PIC 9(4) BINARY.
     88 SOCK-FAMILY-IS-AFINET VALUE 2.
     88 SOCK-FAMILY-IS-AFINET6 VALUE 19.
  05 SOCK-DATA REDEFINES SOCK-FAMILY.
     10 SOCK-SIN-PORT PIC 9(4) BINARY.
     10 SOCK-SIN-ADDR PIC 9(8) BINARY.
     10 FILLER PIC X(8).
     10 FILLER PIC X(12).
  05 SOCK-SIN6 REDEFINES SOCK-DATA.
     10 SOCK-SIN6-PORT PIC 9(4) BINARY.
     10 SOCK-SIN6-FLOWINFO PIC 9(8) BINARY.
     10 SOCK-SIN6-ADDR.
     15 FILLER PIC 9(16) BINARY.
     15 FILLER PIC 9(16) BINARY.
     10 SOCK-SIN6-SCOPEID PIC 9(8) BINARY.
```

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

This structure contains the following fields:

**FAMILY**

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

**PORT**

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

**FLOWINFO**

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

**IP-ADDRESS**

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

---

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

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

SCOPE-ID

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

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

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

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

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

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

Multiple sockets can share the same port and, for CICS, all server applications and the listener share the same port. For client applications, the bind (or connect) socket calls assign a port to the socket that is different from the listener or server port or any other client ports. Normally, client applications do not share ports, but they can if you specify the SO_REUSEADDR socket option. In the case of IMS, an IMS MPR region would normally have a single port number; that port would provide access to one of a number of sockets associated with that IMS system. If an MVS system contains multiple IMS subsystems, each IMS subsystem would have a unique port number.

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

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

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

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

A typical client-server program flow chart

Stream-oriented socket programs generally follow a prescribed sequence. See Figure 3 on page 9 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.

A typical client-server session

Step 1: Server and client create a stream socket s with the socket() call.
Step 2: (Optional for client) Server bind socket s to a local address with the bind() call.
Step 3: Server uses the listen() call to alert the TCP/IP machine of the willingness to accept connections.
Step 4: Client connects socket s to a foreign host with the connect() call.
Step 5: Server accepts the connection and receives a second socket, for example ns, with the accept() call.
Step 6 and 7: Server reads and writes data on socket ns, client reads and writes data on socket s, by using send() and recv() calls, until all data has been exchanged.
Step 8: Server closes socket ns with the close() call. Client closes socket s and end the TCP/IP session with the close() call. Go to step 5.
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 10, after Client A starts a transaction with the server, Client B cannot make a call until A has finished.
So, for lengthy transactions, a different sort of server is needed — the concurrent server, as shown in Figure 5. Here, Client A has already established a connection with the server, which has then created a child server process to handle the transaction. This allows the server to process Client B’s request without waiting for A’s transaction to complete. More than one child server can be started in this way.

TCP/IP provides a concurrent server program called the CICS listener. It is described in “CICS application transaction (IBM listener)” on page 141.

Figure 4. An iterative server

Figure 5. A concurrent server

Figure 3 on page 9 illustrates a concurrent server at work.

Basic socket calls

This topic contains 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.

TAKE_SOCKET
  Accepts a socket from a parent server task.

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

The following calls are used by the client:

SOCKET
  Allocates a socket to read from or write to.

CONNECT
  Allows a client to open a connection to a server’s port.

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

WRITE
  Sends data to the process on the other host.

READ
  Receives data from the other host.

CLOSE
  Terminates a connection, deallocating the socket.

For full discussion and examples of these calls, see Chapter 8, “Sockets extended
API,” on page 249.

Server TCP/IP calls

To understand Socket programming, the client program and the server program
must be considered separately. In this topic, the call sequence for the server is
described; “SOCKET server TCP/IP call” discusses the typical call sequence for a
client. This is the logical presentation sequence because the server is usually
already in running before the client is started. The step numbers (such as 5) in this
topic refer to the steps in Figure 3 on page 9.

SOCKET server TCP/IP call

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

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

The socket function specifies the address family of AF_INET or AF_INET6, the
type of socket (STREAM), and the particular networking protocol (PROTO) to use.
(When PROTO is set to zero, the TCP/IP address space automatically uses the
appropriate protocol for the specified socket type). Upon return, the newly
allocated socket's descriptor is returned in RETCODE.

For an example of the SOCKET call, see “SOCKET call” on page 393.

BIND server TCP/IP call

At this point 2, an entry in the table of communications has been reserved for the
application. However, the socket has no port or IP address associated with it until
the BIND call is issued. The BIND function requires three parameters:
• The socket descriptor that was just returned by the SOCKET call
• The number of the port on which the server wants to provide its service
• The IP address of the network connection on which the server is listening

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

For an example of the BIND call, see “BIND call” on page 257.

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

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

For an example of the LISTEN call, see “LISTEN call” on page 334.

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

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

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

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

---

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.
For an example of the ACCEPT call, see “ACCEPT call” on page 254.

**GIVESOCKET and TAKESOCKET server TCP/IP call**

The GIVESOCKET and TAKESOCKET functions are not supported with the IMS TCP/IP OTMA Connection server. A server handling more than one client simultaneously acts like a dispatcher at a messenger service. A messenger dispatcher gets telephone calls from people who want items delivered, and the dispatcher sends out messengers to do the work. In a similar manner, the server receives client requests, and then spawns tasks to handle each client.

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

Because of architectural differences, CICS sockets does not implement the *fork()* system call. Tasks use the GIVESOCKET and TAKESOCKET functions to pass sockets from parent to child. The task passing the socket uses GIVESOCKET, and the task receiving the socket uses TAKESOCKET. See “GIVESOCKET and TAKESOCKET calls” on page 17 for more information about these calls.

**READ and WRITE server TCP/IP call**

After a client has been connected with the server, and the socket has been transferred from the main task (parent) to the subtask (child), the client and server exchange application data, using various forms of READ/WRITE calls. See “READ/WRITE client TCP/IP calls (the conversation)” on page 14 for details about these calls.

**Client TCP/IP calls**

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

**SOCKET client TCP/IP calls**

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

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

See “SOCKET call” on page 393 for a sample of the SOCKET call.

**CONNECT client TCP/IP calls**

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

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

See "CONNECT call" on page 264 for an example of the CONNECT call.

READ/WRITE client TCP/IP calls (the conversation)

A variety of I/O calls is available to the programmer. The READ and WRITE, READV and WRITEV, and SEND6 and RECV6 calls can be used only on sockets that are in the connected state. The SENDTO and RECVFROM, and SENDMSG and RECVMSG calls can be used regardless of whether a connection exists.

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

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

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

- For an example of the READ call, see "READ call" on page 340.
- For an example of the WRITE call, see "WRITE call" on page 398.

CLOSE TCP/IP 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 call" on page 262.

Other socket calls used for servers

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

SELECT call

Applications such as concurrent servers often handle multiple sockets at simultaneously. In such situations, the SELECT call can be used to simplify the determination of which sockets have data to be read, which are ready for data to be written, and which have pending exceptional conditions. An example of how the SELECT call is used can be found in Figure 6 on page 15.
In this example, the application sends bit sets (the xSNDMASK sets) to indicate which sockets are to be tested for certain conditions, and receives another set of bits (the xRETMASK sets) from TCP/IP to indicate which sockets meet the specified conditions.

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

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

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

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

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

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

The first rule is that the length of a bit string is always expressed as a number of fullwords. If the highest socket descriptor you want to test is socket descriptor 3, you have to pass a 4-byte bit string, because this is the minimum length. If the highest number is 32, you must pass 8 bytes (2 fullwords).

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

\[
\text{INT}\left(\frac{\text{highest socket descriptor}}{32}\right) + 1
\]

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

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

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

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

If you want to test socket descriptor number 32, you must pass two fullwords, where the numbering scheme for the second fullword resembles that of the first. Socket descriptor number 32 is bit 7 in byte 3 of the second fullword. If you want to test socket descriptors 5 and 32, you pass two fullwords with the following content: X'0000000000000001'.

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

<table>
<thead>
<tr>
<th>Socket descriptor numbers represented by byte</th>
<th>Bit 0</th>
<th>Bit 1</th>
<th>Bit 2</th>
<th>Bit 3</th>
<th>Bit 4</th>
<th>Bit 5</th>
<th>Bit 6</th>
<th>Bit 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte 4</td>
<td>63</td>
<td>62</td>
<td>61</td>
<td>60</td>
<td>59</td>
<td>58</td>
<td>57</td>
<td>56</td>
</tr>
<tr>
<td>Byte 5</td>
<td>55</td>
<td>54</td>
<td>53</td>
<td>52</td>
<td>51</td>
<td>50</td>
<td>49</td>
<td>48</td>
</tr>
</tbody>
</table>
If you develop your program in COBOL or PL/I, the EZACIC06 routine, which is provided as part of TCP/IP Services, makes it easier to build and test these bit strings. This routine translates between a character string mask (1 byte per socket) and a bit string mask (1 bit per socket).

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

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

### IOCTL and FCNTL calls

In addition to SELECT, applications can use the IOCTL or FCNTL calls to help perform asynchronous (nonblocking) socket operations. An example of the use of the IOCTL call is shown in “IOCTL call” on page 322.

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

### GIVESOCKET and TAKESOCKET calls

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

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

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

---

5. If a task does not know its address space name, it can use the GETCLIENTID function call to determine its unique name.
If the server does not know the name of the subtask that receives the socket, it blanks out the name of the subtask. The first subtask calling TAKESOCKET with the server's unique name receives the socket.

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

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

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

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

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

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 IMS Message Queue. the CICS START/RETRIEVE commands.
- The concurrent server issues the SELECT call to determine when the GIVESOCKET has successfully completed.
• The subtask calls TAKESOCKET with the concurrent server's ID and socket descriptor and uses the resulting socket descriptor for communication with the client.

• When the GIVESOCKET has successfully completed, the concurrent server issues the CLOSE call to complete the handoff.

An example of a concurrent server is the CICS listener. It is described in “CICS application transaction (IBM listener)” on page 141. Figure 5 on page 10 shows a concurrent server.

CICS TCP/IP requirements

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

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

CICS TCP/IP components

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

CICS TCP/IP has the following main components:

• The stub program is link-edited to each application program that wants to use it. It intercepts requests issued by the calling application program and causes CICS to pass control to the TRUE.

• The TRUE mechanism enables programs to pass calls to the subtask and to the TCP/IP address space.

• CICS TCP/IP supports two methods for accessing TCP/IP
  – The MVS subtask translates commands for accessing TCP/IP into a form acceptable to the TCP/IP resource manager and then passes control to the resource manager. The subtask also handles the MVS waits incurred during socket calls.
  – Using CICS Open Transaction Environment (OTE). The TRUE mechanism accesses TCP/IP directly, not requiring an MVS subtask for blocking commands.

• The Administration Routine contains the EXEC CICS ENABLE and DISABLE commands that are used to install and withdraw the TRUE program.

• The Configuration System configures the interface and its listeners.

Summary of what CICS TCP/IP provides

Figure 7 on page 20 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:

<table>
<thead>
<tr>
<th>Call type</th>
<th>IP CICS TCP API function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic calls:</td>
<td>ACCEPT, BIND, CLOSE, CONNECT, LISTEN, SHUTDOWN</td>
</tr>
<tr>
<td>Read/Write calls:</td>
<td>READ, READV, RECV, RECVFROM, RECVMSG, SEND, SENDMSG, SENDTO, WRITE, WRITEV</td>
</tr>
<tr>
<td>Advanced calls:</td>
<td>FCNTL, FREEADDRINFO, GETADDRINFO, GETHOSTBYADDR, GETHOSTBYNAME, GETHOSTNAME, GETNAMEINFO, GETPEERNAME, GETSOCKNAME, GETSOCKOPT, IOCTL, NTOP, PTON, SELECT, SELECTEX, SETSOCKOPT</td>
</tr>
<tr>
<td>IBM-specific calls:</td>
<td>GETCLIENTID, GIVESOCKET, INITAPI, INITAPIX, TAKESOCKET</td>
</tr>
</tbody>
</table>

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

The IBM listener

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

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

- An EBCDIC-to-ASCII conversion routine that converts EBCDIC data to the ASCII format used in TCP/IP networks and workstations. The routine is run by calling module EZACIC04, which uses an EBCDIC-to-ASCII translation table as described in z/OS Communications Server: IP Configuration Reference.
- A corresponding ASCII-to-EBCDIC conversion routine, EZACIC05, which uses an ASCII-to-EBCDIC translation table as described in z/OS Communications Server: IP Configuration Reference.
- An alternative EBCDIC-to-ASCII conversion routine. It is run by calling EZACIC14, which uses the translation table listed in “EZACIC14 program” on page 416.
- A corresponding alternate ASCII-to-EBCDIC conversion routine, EZACIC15, which uses the translation table listed in “EZACIC15 program” on page 418.
  Tip: A sample translation routine is also supplied in the EZACICTR member of the SEZAINST library. You can modify this member to use alternate EBCDIC-to-ASCII and ASCII-to-EBCDIC translations, including custom translations. See comments in the EZACICTR member for more details.
- A module that converts COBOL character arrays into bit-mask arrays used in TCP/IP. This module, which is run by calling EZACIC06, is used with the socket SELECT or SELECTEX call.
- A routine that decodes the indirectly addressed, variable-length list (hostent structure) returned by the GETHOSTBYADDR and GETHOSTBYNAME calls. This function is provided by calling module EZACIC08.
- A routine that decodes the indirectly addressed, variable-length list (addrinfo structure) returned by the GETADDRINFO call. This function is provided by calling module EZACIC09.

Rules for configuring the IBM-supplied listener for IPv6

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

- You must enable the z/OS system that the IPv6 listener uses for IPv6. See z/OS Communications Server: IP Configuration Reference for information on IPv6 system configuration.
- Because an IPv6 enabled listener uses the GIVESOCKET API function to give an IPv6 socket to a child server transaction, you must enable that child server transaction program to use IPv6 sockets. This requires that all API functions that use a socket address structure be changed to use the larger IPv6 socket address structure. See Chapter 7, “C language application programming,’’ on page 165 or Chapter 8, “Sockets extended API,” on page 249 for more information.
  If the listener gives the accepted socket to the child server program, the child server program must be able to take that socket. If the listener is defined as an INET6 listener, the EBADF errno is issued if the child server’s TAKESOCKET is AF_INET. If the listener is defined as an INET listener, the EBADF errno is issued if the child server’s TAKESOCKET is AF_INET6.
- The Security/Transaction Exit program allows the user to examine and change certain pieces of data that are passed to the child server program by the listener. Table 3 on page 22 illustrates the listener configuration in contrast with the connected client's address family and indicates the contents of the IPv4 and IPv6 IP address fields presented to the Security/Transaction Exit.
Table 3. Security/Transaction Exit program information fields

<table>
<thead>
<tr>
<th>Listener's AF configuration</th>
<th>Connected Client's AF</th>
<th>Exit's Address Family</th>
<th>Exit's Client's IPv4 address</th>
<th>Exit's Client's IPv6 address</th>
<th>Exit's Listener's IPv4 address</th>
<th>Exit's Listener's IPv6 address</th>
</tr>
</thead>
<tbody>
<tr>
<td>not specified</td>
<td>AF_INET</td>
<td>AF_INET</td>
<td>IPv4 addr</td>
<td>zeros</td>
<td>IPv4 addr</td>
<td>zeros</td>
</tr>
<tr>
<td></td>
<td>AF_INET</td>
<td>AF_INET</td>
<td>IPv4 addr</td>
<td>zeros</td>
<td>IPv4 addr</td>
<td>zeros</td>
</tr>
<tr>
<td></td>
<td>AF_INET6</td>
<td>AF_INET6</td>
<td>zeros</td>
<td>IPv4 mapped IPv6 addr</td>
<td>zeros</td>
<td>IPv4 mapped IPv6 addr</td>
</tr>
<tr>
<td></td>
<td>AF_INET6</td>
<td>AF_INET6</td>
<td>zeros</td>
<td>IPv6 addr</td>
<td>zeros</td>
<td>IPv6 addr</td>
</tr>
</tbody>
</table>
Chapter 2. Setting up and configuring CICS TCP/IP

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

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

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

<table>
<thead>
<tr>
<th>Task</th>
<th>See</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modify the CICS job stream to enable CICS TCP/IP startup.</td>
<td>“Modifying CICS startup (MVS JCL)” on page 25</td>
</tr>
<tr>
<td>Define additional files, programs, maps, and transient data queues to CICS using resource definition online (RDO) and the CICS resource management utility DFHCSDUP commands.</td>
<td>“Defining CICS TCP/IP resources” on page 26</td>
</tr>
<tr>
<td>Modify TCP/IP Services data sets.</td>
<td>“Modifying data sets for TCP/IP services” on page 50</td>
</tr>
<tr>
<td>Use the configuration macro (EZACICD), to build the TCP Configuration data set.</td>
<td>“Building the configuration data set with EZACICD” on page 52</td>
</tr>
<tr>
<td>Use the configuration transaction (EZAC) to customize the Configuration data set.</td>
<td>“Customizing the configuration transaction (EZAC)” on page 68</td>
</tr>
</tbody>
</table>

**Note:** You can modify the data set while CICS is running by using EZAC. See “Customizing the configuration transaction (EZAC)” on page 68.

Modifications to the startup of CICS

Figure 8 on page 24 illustrates the modifications required in the CICS startup job stream to enable CICS TCP/IP startup. The numbers in the right margin of the JCL correspond to the modifications that follow.
Figure 8. JCL for CICS startup with the TCP/IP socket interface (part 1 of 2)
The z/OS Communication Server TCP/IP data set prefix names might have been modified during installation. When you see the prefix `hlq` in this information, substitute the prefix used in your installation.

**Modifying CICS startup (MVS JCL)**

These steps provide the minimum information that you need to modify CICS startup.

**Procedure**

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

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

2. **Add a TCPDATA DD entry for the IP CICS sockets output messages (see "Defining the TCPM transient data queue for CICS TCP/IP" on page 38).**

3. **The SYSTCPD DD explicitly identifies which data set is to be used to obtain the parameters defined by TCPIPDATA.** This is used to select the stack you want to use if there are multiple TCP/IP stacks on this system. See [z/OS Communications Server: IP Configuration Guide](https://www.ibm.com/servers/resourcelink/) for further information.
4. The CICS System Initialization Table (SIT) override might contain the following information. See the CICS system initialization information at http://www-01.ibm.com/software/http/cics/library/ for more details about setting CICS SIT parameters:
   - GMTXT= WELCOME TO CICS/TS WITH z/OS CS TCP/IP SOCKETS INTERFACE
   - MCT=SO
     If you want IP CICS sockets to provide performance data then include the IP CICS Sockets Monitor Control Table (MCT) entries in your MCT along with any appropriate monitor SIT controls.
   - PLTPI=SI
     If you want IP CICS sockets to start at Program Load Table (PLT) phase 2 then include EZACIC20 in an appropriate startup PLT.
   - PLTSD=SD
     If you want IP CICS sockets to shutdown at PLT phase 1, then include EZACIC20 in an appropriate shutdown PLT.
   - PLTP1USR=PLTUSER
     PLT User ID. Specify the appropriate user ID to start the IP CICS socket interface and listeners.

5. The following CICS SIT parameters affect the IP CICS socket interface when it is configured to use the CICS Open Transaction Environment. CICS/TS V2R2 or later is required for this support.
   - MAXOPENTCBS=50
     When specifying the EZACICD TYPE=CICS,OTE=YES configuration option, carefully consider this value; it is the size of the CICS managed open API, L8, TCB pool. This pool is used by the IP CICS socket interface and other open API-enabled task-related user exits such as DB2®. Use the CEMT SET DISPATCHER command to dynamically alter this value.
   - FORCEQR
     User programs that are defined to CICS as THREADSAFE are executed on the quasi-reentrant TCB. Use the CEMT SET SYSTEM command to dynamically alter this value.

6. Write the Resolver trace to either a dataset or JES spool.
7. The information is used by IP CICS C Sockets API programs for user messages.

---

**Defining CICS TCP/IP resources**

Make the following CICS definitions:
- Transactions
- Programs (see “Required program definitions to support CICS TCP/IP” on page 28)
- Basic Mapping Support (BMS) mapset (EZACICM, shown in Figure 23 on page 32)
- Files (see “Updates to file definitions for CICS TCP/IP” on page 36)
- Transient data queues (see “Defining the TCPM transient data queue for CICS TCP/IP” on page 38)

To ensure that the CICS system definition (CSD) file contains all necessary socket-related resource definitions, you should execute a CSD upgrade.
(DFHCS Dup) using member EZACICCT in SEZAINST. For information about DFHCS Dup, visit this website: [http://www-01.ibm.com/software/htp/cics/library/](http://www-01.ibm.com/software/htp/cics/library/)

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

### Transaction definitions for CICS

Figures [Figure 10 on page 28](#), [Figure 11 on page 28](#), [Figure 12 on page 28](#), and [Figure 13 on page 28](#) show the CICS CSD update (DFHCS Dup) commands to define the four transactions. These commands can be found in hlq.SEZAINST(EZACICCT).

- **EZAC** Configure the socket interface
- **EZAO** Enable the socket interface
- **EZAP** Internal transaction that is invoked during termination of the socket interface
- **CSKL** Listener task. This is a single listener. Each listener in the same CICS region needs a unique transaction ID.

In the definitions in "Using storage protection when running with CICS 3.3.0 or later," a priority of 255 is suggested. This ensures timely transaction dispatching, and (in the case of CSKL) maximizes the connection rate of clients requesting service.

### Using storage protection when running with CICS 3.3.0 or later

When running with CICS 3.3.0 or later 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 contains more information about storage protection with task-related user exits (TRUEs).

In [Figure 11 on page 28](#), [Figure 12 on page 28](#), and [Figure 13 on page 28](#) 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.
Guidelines:
- Use of the IBM-supplied listener is not required.
- You can use a transaction name other than CSKL.
- The TASKDATALOC values for EZAO and EZAP and the TASKDATALOC value for CSKL must all be the same.
- The user ID invoking the EZAO transaction to activate or deactivate the IP CICS socket interface requires the UPDATE access to the EXITPROGRAM resource when CICS command security is active. The user ID invoking the EZAC transaction requires the UPDATE access to the EXITPROGRAM resource to allow the EZAC transaction to perform an IPv6 run-time check when the AF is changed to INET6. Failure to have at least the UPDATE access to the EXITPROGRAM resource causes the IP CICS socket interface and listener to not start or not stop.

Required program definitions to support CICS TCP/IP

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

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

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

**EZACICME**
The U.S. English text delivery module.

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

**EZACIC01**
The task related user exit (TRUE).

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

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

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

**EZACIC21**
The initialization module for CICS sockets.

**EZACIC22**
The termination module for CICS sockets.

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

**EZACIC24**
The message delivery module for transactions EZAC and EZAO.

**EZACIC25**
The domain name server (DNS) cache module.

Using storage protection when running CICS 3.3.0 or later

When running with CICS 3.3.0 or higher on a storage-protection-enabled machine, all the required CICS TCP/IP programs (EZACIC00, EZACIC01, and EZACIC02) must have EXECKEY(CICS) as part of their definitions. See [http://www-01.ibm.com/software/hta/cics/library/](http://www-01.ibm.com/software/hta/cics/library/) for more information about storage protection with TRUEs.

Figures [Figure 14 on page 30](#), [Figure 15 on page 30](#), and [Figure 16 on page 30](#) show EZACIC00, EZACIC01, and EZACIC02 defined with EXECKEY(CICS). Note that, if the machine does not support storage protection or is not enabled for storage protection, EXECKEY(CICS) is ignored and does not cause an error.
DEFINE PROGRAM(EZACIC00)
DESCRIPTION(PRIMARY PROGRAM FOR TRANSACTION EZAO)
GROUP(SOCKETS)
CEDF(YES) DATALLOCATION(ANY) EXECKEY(CICS)
RELOAD(NO) RESIDENT(NO) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(TRANSIENT)

Figure 14. EZACIC00, connection manager program

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

Figure 15. EZACIC01, task related user exit program

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

Figure 16. EZACIC02, listener program

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

Figure 17. EZACIC20, front-end module for CICS sockets
DEFINE PROGRAM(EZACIC21)
DESCRIPTION(INITIALIZATION MODULE FOR CICS SOCKETS)
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(CICS)
RELOAD(NO) RESIDENT(YES) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(TRANSIENT)

Figure 18. EZACIC21, initialization module for CICS sockets

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

Figure 19. EZACIC22, termination module for CICS sockets

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

Figure 20. EZACIC23, primary module for transaction EZAC

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

Figure 21. EZACIC24, message delivery module for CICS sockets
Optional programs, CICS transaction and program definition needed

The six programs in this topic are optional. They are the supplied samples, and they are also in SEZAINST.

**EZACICSC**
A sample IPv4 child server that works with the IPv4 listener (EZACIC02). See “EZACICSC” on page 571.

**EZACICSS**
A sample IPv4 iterative server. EZACICSS establishes the connection between CICS and TCP/IP stacks, and receives client requests from workstations. See “EZACICSS” on page 577.

**EZACIC6C**
A sample IPv6 child server that works with either a standard or enhanced IPv6 listener (EZACIC02). See “EZACIC6C” on page 595.

**EZACIC6S**
A sample IPv6 iterative server. EZACIC6S establishes the connection between CICS and TCP/IP stacks, and receives client requests from workstations. See “EZACIC6S” on page 604.

**EZACICAC**
A sample assembler child server that works with either a standard or enhanced, IPv4 or IPv6 listener (EZACIC02). See “EZACICAC” on page 623.

**EZACICAS**
A sample assembler iterative server that establishes the connection between CICS and TCP/IP stacks, and accepts either ASCII or EBCDIC, IPv4 or IPv6 (if IPv6 is enabled on the system) client connection requests. See “EZACICAS” on page 630.
If these sample programs are used, they require DFHCSDUP definitions as shown in Figure 25, Figure 26, Figure 27 on page 34, Figure 28 on page 34, Figure 29 on page 35, and Figure 30 on page 35.

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

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

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

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

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

Figure 26. EZACICSS, sample iterative IPv4 server transaction and program definitions
DEFINE TRANSACTION(SRV3)
DESCRIPTION(SAMPLE IPV6 CHILD SERVER)
GROUP(SOCKETS)
PROGRAM(EZACIC6C)
TASKDATALOC(ANY) TASKDATAKEY(USER)

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

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

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

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

Figure 28. EZACIC6S, sample iterative IPv6 server transaction and program definitions
The following programs do not need to be defined to CICS:

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

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

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

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

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

### Threadsafe enablement for to support CICS

The programs in this topic can be defined to CICS as threadsafe. This is particulary important when the IP CICS socket interface is using the CICS Open Transaction Environment. See “TYPE parameter for EZACICD” on page 54 for more information about configuring the IP CICS socket interface to use CICS Open Transaction Environment.

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

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

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

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

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

Figure 30. EZACICAS, sample assembler server transaction and program definitions
```
EZACIC02
Enables the listener to initially execute on an open API TCB. Some TCB switching still occurs because CICS commands that are not threadsafe are used.

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

Sample programs: EZACICSC, EZACIC6C, EZACICAC

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

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

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

Figure 31. ALTER PROGRAM instructions

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

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

Updates to file definitions for CICS TCP/IP

The updates to CICS TCP/IP include two files:
- EZACONFG, the sockets configuration file
- EZACACHE, which is required if you want to use the domain name server cache function (EZACIC25)
EZACONFG

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

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

REMTESYSTEM(....) REMOTENAME(........)
RECORDSIZE(....) KEYLENGTH(... 2

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

Figure 32. DFHCSDUP commands to define EZACONFG

**Note:**

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

EZACACHE

**Tip:** You can use the caching function provided by the z/OS Communications Server system resolver as an alternative to EZACACHE. For more information, see Chapter 3, “Configuring the CICS Domain Name Server cache,” on page 97 for more details.

If you want to use the domain name server Cache function (EZACIC25) instead of the system resolver, this definition is required.

**Guidelines:** The following guidelines apply when you define EZACACHE:

- If you require improved performance for domain name server lookups for both IPv4 and IPv6 resources, you should use the system resolver caching function to obtain the best performance results.
- Using the system resolver caching function provides the following benefits:
  - After a host name is resolved, it is cached locally. All other applications that run in the system can retrieve this information without increasing the network communications.
  - The system resolver caching function honors the time to live (TTL) value, which indicates when the information for a resource record expires.
- The system resolver can cache IPv4 and IPv6 resources.

Use the following DFHCSDUP commands to define EZACACHE file:

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

Figure 33. DFHCSDUP commands to define EZACACHE

Note:
1. Choose a DSName to fit installation standards.
2. For strings, specify the maximum number of concurrent users.
3. Databuffers should equal strings multiplied by two.
4. Indexbuffers equals the number of records in the index set.
5. Although it is optional, you should specify TABLE(USER) because it makes the process run faster. For more information about data tables, visit this website: http://www-01.ibm.com/software/htp/cics/library/
6. Maxnumrecs equals the maximum number of destinations queried.

Defining the TCPM transient data queue for CICS TCP/IP

Figure 34 on page 39 shows the DFHCSDUP commands required to define the TCPM transient data queue for CICS TCP/IP. For more information about DFHCSDUP commands, visit this website: http://www-01.ibm.com/software/htp/cics/library/

The destination TCPM can be changed. If it is changed, it must match the name specified in the ERRORTD parameter of the EZAC DEFINE CICS, the EZACICD TYPE=CICS, or both (see “Building the configuration data set with EZACICD” on page 52).
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. Define an extrapartition transient data queue as shown in Figure 34.

The CICS startup JCL must include a DD statement for the extrapartition transient data queue being defined (as in Modifying CICS startup (MVS JCL) on line 3).

The listener transaction can start a server using a transient data queue, as described in “IBM listener input format” on page 142. The intrapartition transient data queue definition in Figure 34 shows an entry for an application that is started using the trigger-level mechanism of destination control.

**CICS monitoring**

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

*EXC* - Invalid monitoring point

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

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

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

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

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

Figure 34. CICS TCP/IP Transient Data Queue definitions
Event monitoring points for the TRUE

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

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

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

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

The following Monitor Control Table (MCT) entries use the event monitoring points in the performance class used by the Task Related User Exit (TRUE). These entries are in hlq.SEZAINST(EZACIMCT).
DFHMCT TYPE=INITIAL,SUFFIX=S0

* ENTRIES FOR IP CICS SOCKETS TASK-RELATED USER EXIT

* DFHMCT TYPE=EMP,ID=(EZA01.01),CLASS=PERFORM, PERFORM=SCLOCK(1), CLOCK=(1,INIT,READ,WRITE,SELECT,OTHER)

* DFHMCT TYPE=EMP,ID=(EZA01.02),CLASS=PERFORM, PERFORM=PCLOCK(1)

* SOCKET FUNCTIONS READING DATA

* DFHMCT TYPE=EMP,ID=(EZA01.03),CLASS=PERFORM, PERFORM=SCLOCK(2)

* DFHMCT TYPE=EMP,ID=(EZA01.04),CLASS=PERFORM, PERFORM=PCLOCK(2)

* SOCKET FUNCTIONS WRITING DATA

* DFHMCT TYPE=EMP,ID=(EZA01.05),CLASS=PERFORM, PERFORM=SCLOCK(3)

* DFHMCT TYPE=EMP,ID=(EZA01.06),CLASS=PERFORM, PERFORM=PCLOCK(3)

* SOCKET FUNCTIONS SELECTING SOCKETS

* DFHMCT TYPE=EMP,ID=(EZA01.07),CLASS=PERFORM, PERFORM=SCLOCK(4)

* DFHMCT TYPE=EMP,ID=(EZA01.08),CLASS=PERFORM, PERFORM=PCLOCK(4)

* OTHER SOCKET FUNCTIONS

* DFHMCT TYPE=EMP,ID=(EZA01.09),CLASS=PERFORM, PERFORM=SCLOCK(5)

* DFHMCT TYPE=EMP,ID=(EZA01.10),CLASS=PERFORM, PERFORM=PCLOCK(5)

* CICS TASK TERMINATION

* DFHMCT TYPE=EMP,ID=(EZA01.13),CLASS=PERFORM, PERFORM=(MLTCNT(1,5)), COUNT=(1,TINIT,TREAD,TWRITE,TSELECT,OTHER)

* REUSABLE SUBTASK POOL

* DFHMCT TYPE=EMP,ID=(EZA01.11),CLASS=PERFORM, PERFORM=ADDCNT(6,1), COUNT=(6,REUSABLE,ATTACHED,OPENAPI,TCBLIM)

* DYNAMICALLY DEFINED SUBTASKS

* DFHMCT TYPE=EMP,ID=(EZA01.12),CLASS=PERFORM, PERFORM=ADDCNT(7,1)

* OPEN API

* DFHMCT TYPE=EMP,ID=(EZA01.15),CLASS=PERFORM, PERFORM=ADDCNT(8,1)

* TCBLIM

* DFHMCT TYPE=EMP,ID=(EZA01.16),CLASS=PERFORM, PERFORM=ADDCNT(9,1)

* CICS TASK INTERFACE TERMINATION

* DFHMCT TYPE=EMP,ID=(EZA01.14),CLASS=PERFORM, PERFORM=(MLTCNT(10,4)), COUNT=(10,TREUSABLE,TATTACHE,OPENAPI,TCBLIM)

In the ID parameter, the following specifications are used:
Event monitoring points for the listener

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

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

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

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

* NUMBER OF TIMES LENGTH ERROR ON CHILD SERVER TD QUEUE
Figure 36. 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

Open TCB measurements

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

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

- Task startup
- Processing a non-threadsafe CICS command
• Processing application code when switched back to the QR TCB
• Processing non-threadsafe subprograms
• Final task processing

The time spent on the L8 TCB can be used on the following:
• OPEN TCB processing
• Processing the EZASOKET call
• Running the application code
• Processing threadsafe CICS commands
• Processing threadsafe subprograms
• TCP/IP processing the socket call

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

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

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

Figure 37 on page 46 shows a EZASOKET threadsafe transaction. The numbers correspond to the list that follows the figure.
1. Represents the task startup and the application until it issues the first EZASOKET call.
2. Actual time spent in Sockets Extended, processing the first EZASOKET call.
3. Time spent in the resource manager interface (RMI), processing the EZASOKET call.
4. Threadsafe application code and EXEC CICS commands running.
5. Time spent in Sockets Extended, processing the second EZASOKET call.
6. Time spent in the RMI, processing the second request.
7. Final application code, which issues a non-threadsafe EXEC CICS WRITEQ TD command causing a change_mode back to the QR TCB.
8. Final task processing on the QR TCB.
CICS program list table

You can enable automatic startup or shutdown of the CICS socket interface through updates to the program list table (PLT). Put the EZACIC20 module in the appropriate PLT to enable automatic startup and shutdown.

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

* 
* PLT USED TO SUPPORT IP CICS SOCKETS STARTUP

* 
  DFHPLT TYPE=INITIAL,SUFFIX=SI
  DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
  DFHPLT TYPE=ENTRY,PROGRAM=EZACIC20

* Add other IP CICS Socket PLT startup programs here...
* 
  DFHPLT TYPE=FINAL
  END

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

* 
* PLT USED TO SUPPORT IP CICS SOCKETS SHUTDOWN

* 
  DFHPLT TYPE=INITIAL,SUFFIX=SD

* Add other IP CICS Socket PLT shutdown programs here...
* 
  DFHPLT TYPE=ENTRY,PROGRAM=EZACIC20
  DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
  DFHPLT TYPE=FINAL
  END

System recovery table

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

You can modify the default recovery action by writing your own recovery program. You do this using the XSRAB global user exit point within the system recovery program (SRP). For programming information about the XSRAB exit, see [http://www-01.ibm.com/software/htp/cics/library/](http://www-01.ibm.com/software/htp/cics/library/).

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

DFHSRT macroinstruction types

You can code the following macroinstructions in a system recovery table:

- DFHSRT TYPE=INITIAL establishes the control section.
- DFHSRT TYPE=SYSTEM or DFHSRT TYPE=USER specifies the abend codes that are to be handled.
Control section:

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

```
--DFHSRT--TYPE=INITIAL
     "-SUFFIX="xx
```

For general information about TYPE=INITIAL macroinstructions, including the use of the SUFFIX operand, visit this website: [http://www-01.ibm.com/software/htp/cics/library/](http://www-01.ibm.com/software/htp/cics/library/)

Abend codes:

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

```
--DFHSRT--TYPE="SYSTEM"
  "-ABCODE="(codes)

--DFHSRT--TYPE="USER"
  "-ABCODE="(codes)
```

**SYSTEM**

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

**USER**

The abend code is a user (including CICS) abend code corresponding to an MVS Unnnn abend code. The abend code must be a decimal number (nnnn) representing the user part of the MVS abend code Unnnn. This is usually the same number as the CICS message that is issued before CICS tries to terminate abnormally.

**ABCODE=(codes)**

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

**RECOVER**

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

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

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

Abend code 0F3 covers various machine check conditions. It also covers the Alternate Processor Retry condition that can occur only when running on a multiprocessor. CICS-supplied recovery code attempts to recover from instruction-failure machine checks on the assumption that they are not permanent. It also attempts to recover from Alternate Processor Retry conditions.
CICS tries to recover from these standard abend codes if you code the system recovery table simply as follows. There is no need to list the standard codes individually.

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

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

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

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

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

**DFHSRT example**

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

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

**CICS TCP/IP security considerations**

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

- **EZAC** Configure sockets interface.
- **EZAO** Enable sockets interface.
- **EZAP** Disable socket interface started by the EZAO, STOP, and YES transactions.
- **CSKL** Listener. Also, any user defined transactions that execute EZACIC02.

The EZAC and EZAO transactions are designed to be run with a terminal. If you want a user to administer the IP CICS sockets configuration then you must grant the user authorization to the EZAC transaction. If you want a user to manually start and stop the IP CICS socket interface then you must grant the user authorization to the EZAO and EZAP transactions. If you want a user to manually start and stop the listener then you must grant the user authorization to the EZAO and CSKL (and any user defined transaction defined to execute EZACIC02) transactions.
For terminal tasks where a user has not signed on, the user ID is the CICS user ID associated with the terminal and is either:

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

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

User ID's used to manage the starting and stopping of the CICS socket interface (EZAO), the listener (CSKL or user defined transactions executing EZACICO2) and user application programs linking to the IP CICS domain name server module, EZACICxx should at least be granted UPDATE access to the EXITPROGRAM resource.

For more information about RACF security management in the CICS environment, see z/OS Security Server RACF Security Administrator’s Guide.

### Modifying data sets for TCP/IP services

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

**hlq.PROFILE.TCPIP data set**

You define the CICS region to TCP/IP on z/OS in the hlq.PROFILE.TCPIP data set (described in z/OS Communications Server: IP Configuration Reference and z/OS Communications Server: IP Configuration Guide). In it, you must provide entries for the CICS region in the PORT statement, as shown in Figure 38 on page 51.

The format for the PORT statement is:

```
port_number TCP CICS_jobname
```

Write an entry for each port that you want to reserve for an application. Figure 38 on page 51 shows two entries, allocating port number 3000 for SERVA, and port number 3001 for SERVB. SERVA and SERVB are the job names of our CICS regions.

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

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

---

6. Note that in this information, the abbreviation hlq stands for high level qualifier. This qualifier is installation dependent.
Two different CICS listeners running on the same host can share a port. See the discussion on port descriptions in z/OS Communications Server: IP Configuration Reference for more information about ports.

**hlq.TCP/IP.DATA data set**

For CICS TCP/IP, you do not have to make any extra entries in `hlq.TCPIP.DATA`. However, you need to check the `TCPIPJOBNAME` parameter that was entered during TCP/IP Services setup. This parameter is the name of the started procedure used to start the TCP/IP Services address space.

You need it when you initialize CICS TCP/IP (see Chapter 4, “Managing IP CICS sockets,” on page 107). In Figure 39 on page 52, `TCPIPJOBNAME` is set to TCPV3. The default name is TCPIP.
Adding a z/OS UNIX System Services segment

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

Configuring the CICS TCP/IP environment

You need to create data for configuring the CICS TCP/IP environment.

Procedure

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

2. Initialize the data set using the program generated by the EZACICD macro. The first two steps are described in “JCL for the configuration macro” on page 65.
3. Add to or modify the data set using the configuration transaction EZAC. This step is described in “Customizing the configuration transaction (EZAC)” on page 68.

Building the configuration data set with EZACICD

The configuration macro (EZACICD) is used to build the configuration data set. This data set can then be incorporated into CICS using resource definition online (RDO) and can be modified using the configuration transactions (see “Customizing the configuration transaction (EZAC)” on page 68). The macro is keyword driven; the TYPE parameter controls the specific function request. The data set contains one record for each instance of CICS that it supports, and one record for each listener. The following is an example of the macros required to create a configuration file for two instances of the CICS socket interface listeners each:

7. The EZAC transaction is modeled after the CEDA transaction used by CICS Resource Definition Online (RDO).
Figure 40. EZACICFG configuration file

EZACICD TYPE=INITIAL, Start of macro assembly input
FILNAME=EZACICDF, DD name for configuration file
PRGNAME=EZACICDF, Name of batch program to run

EZACICD TYPE=CICS, CICS record definition
APPLID=CICSPROD, APPLID of CICS region not using OTE
TCPADDR=TCP/IP, Job/Step name for TCP/IP
PLTSDI=YES, PLT shutdown method is immediately
MTASKS=20, Number of subtasks
DPRTY=0, Subtask dispatch priority difference
CACHMIN=15, Minimum refresh time for cache
CACHMAX=30, Maximum refresh time for cache
CACHRES=10, Maximum number of resident resolvers
ERRORTD=CSMT, Transient data queue for error msgs
TCBLIM=0, Open API TCB Limit
OTE=YES, Use Open Transaction Environment
TRACE=NO, Trace CICS Sockets
APPLDAT=YES, Register Application Data
SMSGSUP=NO, STARTED Messages Suppressed?
TERMLIM=100, Subtask Termination Limit

EZACICD TYPE=CICS, CICS record definition
APPLID=CICSPRDB, APPLID of CICS region using OTE
TCPADDR=TCP/IP, Job/Step name for TCP/IP
PLTSDI=NO, PLT shutdown method is deferred
CACHMIN=15, Minimum refresh time for cache
CACHMAX=30, Maximum refresh time for cache
CACHRES=10, Maximum number of resident resolvers
ERRORTD=CSMT, Transient data queue for error msgs
TCBLIM=12, Open API TCB Limit
OTE=NO, Use Open Transaction Environment
TRACE=NO, Trace CICS Sockets
APPLDAT=NO, No Application Data
SMSGSUP=NO, STARTED Messages Suppressed?

EZACICD TYPE=LISTENER, Listener record definition
FORMAT=STANDARD, Standard Listener
APPLID=CICSPROD, Applid of CICS region
TRANID=CSKL, Transaction name for Listener
PORT=3010, Port number for Listener
AF=INET, Listener Address Family
IMMED=YES, Listener starts up at initialization?
BACKLOG=20, Backlog value for Listener
NUMSOCK=50, # of sockets supported by Listener
MINMSG=4, Minimum input message length
ACCTIME=30, Timeout value for Accept
GIVTIME=30, Timeout value for Givessocket
REATIME=30, Timeout value for Read
RTRYTIME=10, Wait 10 seconds for TCP to come back
LAPPLD=YES, Register Application Data
TRANTRN=YES, Is TRANUSR=YES conditional?
TRANUSR=YES, Translate user data?
SECEXIT=EZACICSE, Name of security exit program

EZACICD TYPE=LISTENER, Listener record definition
FORMAT=ENHANCED, Enhanced Listener
APPLID=CICSPROD, Applid of CICS region
TRANID=CSKM, Transaction name for Listener
PORT=3011, Port number for Listener
AF=INET, Listener Address Family
IMMED=YES, Listener starts up at initialization?
BACKLOG=20, Backlog value for Listener
NUMSOCK=50, # of sockets supported by Listener
ACCTIME=30, Timeout value for Accept
GIVTIME=30, Timeout value for Givessocket
REATIME=30, Timeout value for Read
RTRYTIME=20, Wait 20 seconds for TCP to come back
LAPPLD=INHERIT, Inherit interface setting
CSTRAN=TNN1, Name of child IPv4 server transaction
CSTSYP=KC, Child server startup type
CSDelay=000000, Child server delay interval
MSGLEN=0, Length of input message
PEEKDAT=NO, Peek option
MSGFORM=ASCII, Output message format
SECEXIT=EZACICSE, Name of security exit program

EZACICD TYPE=LISTENER, Listener record definition
FORMAT=STANDARD, Standard listener
APPLID=CICSPROD, Applid of CICS region

Chapter 2. Setting up and configuring CICS TCP/IP  53
TRANID=CS6L, Transaction name for listener X
PORT=3012, Port number for listener X
AF=INET6, Listener Address Family X
IMMED=YES, Listener starts up at initialization? X
BACKLOG=20, Backlog value for listener X
NUMSOCK=50, # of sockets supported by listener X
MINMSGLEN=4, Minimum input message length X
ACCTIME=30, Timeout value for Accept X
GIVTIME=30, Timeout value for Givesocket X
READTIME=30, Timeout value for Read X
RTTYTIME=0, Listener will end when TCP ends X
LAPPLD=NO, No Application Data X
TRANUSR=YES, Translate user data? X
TRANTRN=YES, Is TRANUSR=YES conditional? X
SECEXIT=EZACICSE Name of security exit program

EZACICD TYPE=LISTENER, Listener record definition X
FORMAT=ENHANCED, Enhanced listener X
TRANID=CS6M, Transaction name for listener X
PORT=3013, Port number for listener X
AF=INET6, Listener Address Family X
IMMED=YES, Listener starts up at initialization? X
BACKLOG=20, Backlog value for listener X
NUMSOCK=50, # of sockets supported by listener X
ACCTIME=30, Timeout value for Accept X
GIVTIME=30, Timeout value for Givesocket X
READTIME=30, Timeout value for Read X
RTTYTIME=0, Listener will end when TCP ends X
LAPPLD=INHERIT, Inherit interface setting X
CSTRAN=TRN6, Name of IPv6 child server transaction X
CSSTYP=KC, Child server startup type X
CSDELAY=000000, Child server delay interval X
MSGLEN=0, Length of input message X
PEEKDAT=NO, Peek option X
MSGFORM=ASCII, Output message format X
USERID=USER0001, Listener User ID X
SECEXIT=EZACICSE Name of security exit program

EZACICD TYPE=FINAL End of assembly input

**TYPE parameter for EZACICD**

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>INITIAL</td>
<td>Initialize the generation environment. This value should be used only once per generation and it should be in the first invocation of the macro. For subparameters, see “TYPE=INITIAL setting for the TYPE parameter.”</td>
</tr>
<tr>
<td>CICS</td>
<td>Identify a CICS object. This value corresponds to a specific instance of CICS. Specifying this value creates a configuration record. For subparameters, see “TYPE=CICS setting for the TYPE parameter” on page 55.</td>
</tr>
<tr>
<td>LISTENER</td>
<td>Identify a listener object. This value creates a listener record. For subparameters, see “TYPE=LISTENER setting for the TYPE parameter” on page 59.</td>
</tr>
<tr>
<td>FINAL</td>
<td>Indicates the end of the generation. There are no subparameters.</td>
</tr>
</tbody>
</table>

**TYPE=INITIAL setting for the TYPE parameter:**

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
</table>

---

54  z/OS V2R1.0 Communications Server: IP CICS Sockets Guide
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 setting for the TYPE parameter:
When TYPE=CICS is specified, the following parameters apply:

Value  Meaning

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

The IBM listener's optional security exit can override this setting for each accepted connection that is to be given to a child server. Overriding the setting enables application data that is specific to the child server to be registered against the accepted connections. For more information about using the security exit to register application data, see Chapter 6, “Writing applications that use the IP CICS sockets API,” on page 129 and Application data in z/OS Communications Server: IP Programmer’s Guide and Reference. For more information about programming applications, see Application data in z/OS Communications Server: IP Programmer’s Guide and Reference. The associated application data is made available on the Netstat ALL/-A, ALLConn/-a and CONn/-c reports, in the SMF 119 TCP connection termination records, and through the network management interface (NMI) on the GetTCPListeners and GetConnectionDetail poll requests. The Netstat and NMI interfaces support new filters for selecting sockets based on wildcard comparisons of the application data. This support can assist in locating application sockets during problem determination and can aid capacity planning and accounting applications to correlate TCP/IP SMF resource records with other applications records. It is the responsibility of the using applications to record the content, format, and meaning of the associated data.

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

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

**CACHMIN**
The minimum refresh time for the domain name server cache in minutes. This value depends on the stability of your network, that is, the time you would expect a domain name to have the same Internet address. Higher values improve performance but increase the risk of getting an incorrect (expired) address when resolving a name. The value must be less than CACHMAX. The default value is 15.

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

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

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

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

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

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

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

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

Table 4. Configuration options affected by OTE

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

PLTSDI

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

SMSGSUP

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

For detailed information about CICS sockets messages, see Appendix D, “CICS sockets messages,” on page 475.
TCBLIM
Specifies the maximum number of open API (L8) TCBs that can be used by the IP CICS socket interface to support socket calls, which, in turn, limits the maximum number of concurrently supported socket calls.

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

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

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

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

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

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

TRACE
The value for TRACE is either YES (the default) or NO. A value of NO will
direct the TRUE and the listener to not generate CICS AP trace records even if CICS trace is active. The value of YES will direct the TRUE and the listener to generate CICS AP trace records which also requires that CICS Trace be active. Trace records are generated only if CICS tracing is active and TRACE=YES. See the CICS Transaction Server for z/OS CICS Supplied Transactions publication for guidance on enabling and disabling the CICS trace. See the CICS Transaction Server for z/OS CICS Operations and Utilities Guide for guidance printing the CICS trace. Use the EZAO,START|STOP,TRAce to dynamically enable or disable tracing. Suppressing the generation of trace records after IP CICS sockets application programs are tested and debugged or for normal operations can improve performance.

**TYPE=LISTENER setting for the TYPE parameter:**

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

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

**AF**
Determines whether the listener being defined supports IPv6 partners and is able to give an IPv6 socket descriptor to an IPv6 child server program. INET6 indicates that the listener gives an IPv6 socket to the child server program. INET, the default, indicates that the listener gives an IPv4 socket to the child server program. Ensure that the child server program performing the TAKESOCKET command matches the domain of the socket being given by the listener.

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

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

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

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

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

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

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

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

**CSTRAN**

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

**FORMAT**

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

**GETTID**

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

The GETTID values have the following meaning for the listener:

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

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

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

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

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

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


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

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

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

The IBM listener’s optional security exit can override this setting for each accepted connection that is to be given to a child server. Overriding the setting enables application data that is specific to the child server to be registered against the accepted connections to be given. For more information about programming applications, see Chapter 6, “Writing applications that use the IP CICS sockets API,” on page 129 and Application data in z/OS Communications Server: IP Programmer’s Guide and Reference. The associated application data is made available on the Netstat ALL/-A, ALLConn/-a and CONn/-c reports, in the SMF 119 TCP connection termination records and through the network management interface (NMI) on the GetTCPListeners and GetConnectionDetail poll requests. The Netstat and NMI interfaces support new filters for selecting sockets based on wildcard comparisons of the application data. This support can assist in locating application sockets during problem determination and can aid capacity planning and accounting applications to correlate TCP/IP SMF resource records with other applications records. It is the responsibility of the using applications to record the content, format, and meaning of the associated data.

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

**MINMSGL**

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

**MSGFORM**

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

**MSGLEN**

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

**NUMSOCK**

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

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

**PEEKDAT**

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

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

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

Result: If REATIME=0 is specified when either the MINMSGL byte value or the MSGLEN byte value is greater than 0, then the listener will wait indefinitely for that number of bytes to arrive before starting a child server task.

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

<table>
<thead>
<tr>
<th>Listener</th>
<th>RTYTIME</th>
<th>TCP down</th>
<th>TCP up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initially started</td>
<td>0</td>
<td>Listener ends</td>
<td>Listener initializes</td>
</tr>
<tr>
<td></td>
<td>&gt;0</td>
<td>Listener waits</td>
<td></td>
</tr>
<tr>
<td>Previously active</td>
<td>0</td>
<td>Listener ends</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&gt;0</td>
<td>Listener waits</td>
<td></td>
</tr>
</tbody>
</table>

**SECEXIT**
The name of the user written security exit used by this listener. The default is no security exit. The listener uses the EXEC CICS LINK command to give control to the security exit. If OTE=YES then it should be expected that the security exit program is defined to CICS as threadsafe, implying it is coded to threadsafe standards. A flag which indicates that the IP CICS socket interface is using CICS’s Open Transaction Environment is passed to the security exit. This flag enables the security exit to decide which child server transaction to use and if it should possibly limit its use of
non-threadsafe resources or commands. See “Writing your own security or transaction link modules for the listener” on page 152 for a thorough discussion on the data passed to the exit. See “Threadsafe considerations for IP CICS sockets applications” on page 156 for more information about coding threadsafe programs. A check is made to ensure the specified security exit program is defined to CICS and enabled for use when the listener is started by the EZAO Operator transaction.

TRANID

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

TRANTRN

This parameter is specific to the standard version of the listener. Specify YES or NO. YES indicates that the translation of the user data is based on the character format of the transaction code. That is, with YES specified for TRANTRN, the user data is translated if and only if TRANUSR is YES and the transaction code is not uppercase EBCDIC. If NO specified for TRANTRN, the user data is translated if and only if TRANUSR is YES. The default value for TRANTRN is YES. See Table 6 for more information.

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

TRANUSR

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

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

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

Table 6. Conditions for translation of tranid and user data

<table>
<thead>
<tr>
<th>TRANTRN</th>
<th>TRANUSR</th>
<th>Tranid format</th>
<th>Translate tranid?</th>
<th>Translate user data?</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>YES</td>
<td>EBCDIC</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>YES</td>
<td>NO</td>
<td>EBCDIC</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>NO</td>
<td>YES</td>
<td>EBCDIC</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>NO</td>
<td>NO</td>
<td>EBCDIC</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>YES</td>
<td>YES</td>
<td>ASCII</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>YES</td>
<td>NO</td>
<td>ASCII</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>NO</td>
<td>YES</td>
<td>ASCII</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>NO</td>
<td>NO</td>
<td>ASCII</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>
USERID

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

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

JCL for the configuration macro

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

The following sample illustrates a job stream used to define a configuration file.
//CONFIG JOB (accounting,information),programmer.name,
//MGLEVEL=(1,1),MSGCLASS=A,CLASS=A
//#
//* z/OS Communications Server
//* SMP/E distribution name: EZACICFG
//* Licensed Materials - Property of IBM
//* "Restricted Materials of IBM"
//* 5694-A01
//* Copyright IBM Corp. 2000,2009
//* Status = CSV1R11
//*
//* Function: This job defines and then loads the VSAM
//* file used for the CICS TCP configuration. The job stream
//* has the following steps:
//*
//* 1. Delete a configuration file if one exists
//* 2. Define the VSAM configuration file to VSAM
//* 3. Assemble the initialization program
//* 4. Link the initialization program
//* 5. Execute the initialization program to load the
//* VSAM configuration file
//*
//* THIS STEP DELETES AN OLD COPY OF THE FILE
//* IF ONE IS THERE.
//*
//DELE 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) -
// RECORDSIZE(150 150) FREESPACE(0 15) -
// INDEXED -
// SHAREOPTIONS(2,3)) -
// DATA ( -
// NAME(CICS.TCP.CONFIG.DATA) -
// KEYS (16 0) ) -
// INDEX ( -
// NAME(CICS.TCP.CONFIG.INDEX) )
//*
//* THIS STEP ASSEMBLES THE INITIALIZATION PROGRAM
//*
//ASM EXEC PGM=ASMA90,PARM='OBJECT,TERM',REGION=1024K
//SYSLIB DD DISP=SHR,DSNAME=SYS1.MACLIB
// DD DISP=SHR,DSNAME=TCPIP.SEZACMAC
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(5,1))
//SYSUT2 DD UNIT=SYSDA,SPACE=(CYL,(2,1))
//SYSUT3 DD UNIT=SYSDA,SPACE=(CYL,(2,1))
//SYSPUNCH DD DISP=SHR,DSNAME=NULLFILE
//SYSLIN DD DSNNAME=&&OBJSET,DISP=(MOD,PASS),UNIT=SYSDA,
// SPACE=(400,(500,50)),
// DCF=(RECFM=FB,BLKSIZE=400,LRECL=80)
//SYSTERM DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
//EZACICD TYPE=INITIAL, Start of macro assembly input X
// FILNAME=EZACICDF, DD name for configuration file X
// PRGNAME=EZACICDF Name of batch program to run
// EZACICD TYPE=CICS, CICS record definition
// APPLID=CICSPROD, APPLID of CICS region not using OTE X
// TCPADDR=TCPIP, Job/Step name for TCP/IP X
// NTASKS=20, Number of subtasks X

Figure 41. Example of JCL to define a configuration file

66 z/OS V2R1.0 Communications Server: IP CICS Sockets Guide
Chapter 2. Setting up and configuring CICS TCP/IP

**DPRTY=0, Subtask dispatch priority difference**

**CACHMIN=15, Minimum refresh time for cache**

**CACHMAX=30, Maximum refresh time for cache**

**CACHRES=10, Maximum number of resident resolvers**

**ERRORTD=CSMT, Transient data queue for error msgs**

**TCBLIM=0, Open API TCB Limit**

**OTE-NO, Open Transaction Environment**

**TRACE-NO, No CICS Trace records**

**SMSGSUP=NO, STARTED Messages Suppressed?**

**EZACICD TYPE=CICS, CICS record definition**

**APPLID=CICSPROD, APPLID of CICS region using OTE**

**TCPADDR=TCPIP, Job/Step name for TCP/IP**

**APPLID=CICSPRDB, APPLID of CICS region with OTE**

**TRANID=CSKL, Transaction name for Listener**

**PORT=3010, Port number for Listener**

**AF=INET, Listener Address Family**

**IMMED=YES, Listener starts up at initialization?**

**BACKLOG=20, # of sockets supported by Listener**

**NUMSOCK=50, Minimum input message length**

**ACCTIME=30, Timeout value for Accept**

**GIVTIME=30, Timeout value for Givesocket**

**REATIME=30, Timeout value for Read**

**TRANTRN=YES, Is TRANUSR=YES conditional?**

**TRANUSR=YES, Translate user data?**

**SECEXIT=EZACICSE, Name of security exit program**

**EZACICD TYPE=LISTENER, Listener record definition**

**FORMAT=STANDARD, Standard Listener**

**APPLID=CICSPROD, APPLID of CICS region**

**TRANID=CSKL, Transaction name for Listener**

**PORT=3010, Port number for Listener**

**AF=INET, Listener Address Family**

**IMMED=YES, Listener starts up at initialization?**

**BACKLOG=20, Backlog value for Listener**

**NUMSOCK=50, # of sockets supported by Listener**

**MINMSGLEN=4, Minimum input message length**

**ACCTIME=30, Timeout value for Accept**

**GIVTIME=30, Timeout value for Givesocket**

**REATIME=30, Timeout value for Read**

**CSSTRAN=TRN1, Name of child IPv4 server transaction**

**CSSTTYP=KC, Child server startup type**

**CSDELAY=000000, Child server delay interval**

**MSGLEN=0, Length of input message**

**PEEKDAT=NO, Peek option**

**MSGFORM=ASCII, Output message format**

**SECEXIT=EZACICSE, Name of security exit program**

**EZACICD TYPE=LISTENER, Listener record definition**

**FORMAT=ENHANCED, Enhanced Listener**

**APPLID=CICSPROD, APPLID of CICS region**

**TRANID=CSKL, Transaction name for Listener**

**PORT=3010, Port number for Listener**

**AF=INET, Listener Address Family**

**IMMED=YES, Listener starts up at initialization?**

**BACKLOG=20, Backlog value for Listener**

**NUMSOCK=50, # of sockets supported by Listener**

**MINMSGLEN=4, Minimum input message length**

**ACCTIME=30, Timeout value for Accept**

**GIVTIME=30, Timeout value for Givesocket**

**REATIME=30, Timeout value for Read**

**TRANTRN=YES, Is TRANUSR=YES conditional?**

**TRANUSR=YES, Translate user data?**

**SECEXIT=EZACICSE, Name of security exit program**

**EZACICD TYPE=LISTENER, Listener record definition**

**FORMAT=STANDARD, Standard Listener**

**APPLID=CICSPROD, APPLID of CICS region**

**TRANID=CS6L, Transaction name for Listener**

**PORT=3012, Port number for Listener**

**AF=INET6, Listener Address Family**

**IMMED=YES, Listener starts up at initialization?**

**BACKLOG=20, Backlog value for Listener**

**NUMSOCK=50, # of sockets supported by Listener**

**MINMSGLEN=4, Minimum input message length**

**ACCTIME=30, Timeout value for Accept**

**GIVTIME=30, Timeout value for Givesocket**

**REATIME=30, Timeout value for Read**

**TRANTRN=YES, Is TRANUSR=YES conditional?**

**TRANUSR=YES, Translate user data?**

**SECEXIT=EZACICSE, Name of security exit program**

**EZACICD TYPE=LISTENER, Listener record definition**

**FORMAT=ENHANCED, Enhanced Listener**

**APPLID=CICSPROD, APPLID of CICS region**

**TRANID=CS6M, Transaction name for Listener**

**PORT=3013, Port number for Listener**

**AF=INET6, Listener Address Family**

**IMMED=YES, Listener starts up at initialization?**

**BACKLOG=20, Backlog value for Listener**

**NUMSOCK=50, # of sockets supported by Listener**

**MINMSGLEN=4, Minimum input message length**

**ACCTIME=30, Timeout value for Accept**

**GIVTIME=30, Timeout value for Givesocket**

**REATIME=30, Timeout value for Read**

**TRANTRN=YES, Is TRANUSR=YES conditional?**

**TRANUSR=YES, Translate user data?**

**SECEXIT=EZACICSE, Name of security exit program**
Customizing the configuration transaction (EZAC)

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

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

The EZAC transaction is a panel-driven interface that lets you add, delete, or modify the configuration file. Table 7 lists and describes the functions supported by the EZAC transaction.

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

Table 7. Functions supported by the EZAC transaction

<table>
<thead>
<tr>
<th>Command</th>
<th>Object</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALTER</td>
<td>CICS/listener</td>
<td>Modifies the attributes of an existing resource definition</td>
</tr>
<tr>
<td>CONVERT</td>
<td>Listener</td>
<td>Converts listener from the standard listener that requires the standard header to the enhanced listener that does not require the header.</td>
</tr>
</tbody>
</table>

NUMSOCK=50, # of sockets supported by listener
ACCTIME=30, Timeout value for Accept
GIVTIME=30, Timeout value for GIVESOCKET
REATIME=30, Timeout value for READ
CSTRAN=TRN6, Name of child IPv6 server transaction
CSSTYP=KC, Child server startup type
CSDELAY=000000, Child server delay interval
MSGLEN=0, Length of input message
PEEKDAT=NO, Peek option
MSGFORM=ASCII, Output message format
SECEXIT=EZACICSE, Name of security exit program
Table 7. Functions supported by the EZAC transaction (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Object</th>
<th>Function</th>
</tr>
</thead>
</table>
| COPY    | CICS/listener | • CICS - Copies the CICS object and its associated listeners to create another CICS object. COPY fails if the new CICS object already exists.  
• Listener - Copies the listener object to create another listener object. COPY fails if the new listener object already exists. |
| DEFINE  | CICS/listener | Creates a new resource definition                                          |
| DELETE  | CICS/listener | • CICS - Deletes the CICS object and all of its associated listeners.  
• Listener - Deletes the listener object. |
| DISPLAY | CICS/listener | Shows the parameters specified for the CICS/listener object.               |
| RENAME  | CICS/listener | Performs a COPY followed by a DELETE of the original object.               |

If you enter EZAC, the following screen is displayed:

![EZAC initial screen](image)

**ALTER function for EZAC**

The ALTER function is used to change CICS objects or their listener objects. If you specify ALter on the EZAC Initial Screen or enter EZAC,ALT on a blank screen, the following screen is displayed:
Note: You can skip this screen by entering either EZAC,ALTER,CICS or EZAC,ALTER,LISTENER.

ALTER,CICS:

For alteration of a CICS object, the following screen is displayed:
After the APPLID is entered, the following screen is displayed:

```
EZAC,ALTER,CICS

Enter all fields

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

Press ENTER or PF3 to exit
```

**Figure 44. EZAC,ALTER,CICS screen**

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

**ALTER,LISTENER:**

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

```
EZAC,ALTer,CICS

Overtype to Enter

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

Press ENTER or PF3 to exit
```

**Figure 45. EZAC,ALTER,CICS detail screen**
If you are altering a standard listener, the first screen shows the attributes of the standard listener:

**Figure 46. EZAC,ALTER,LISTENER screen**

EZAC,ALTER,LISTENER (standard listener) APPLID = ........

Enter all fields

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

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

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

Pressing PF8 displays the screen used to manage the unique attributes of the standard listener
Pressing PF7 displays the screen used to manage the common attributes of the standard listener.

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

Pressing PF8 displays the screen used to manage the unique attributes of the enhanced listener.
Pressing PF7 displays the screen used to manage the common attributes of the enhanced listener.

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

CONVERT function for EZAC

The CONVERT function is used to convert between standard and enhanced versions of the listener. If you specify CONvert on the EZAC Initial Screen or enter EZAC,CON on a blank screen, the following screen is displayed:
After the names and format type are entered, one of the following screens is displayed. The first screen is displayed for the standard version.

If converting to a standard listener, then the first screen shows the attributes of the standard listener.
Pressing PF8 displays the screen used to manage the unique attributes of the standard listener.

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

OvertypetoeEnter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLID</td>
<td>APPLID of CICS System</td>
</tr>
<tr>
<td>TRANID</td>
<td>Transaction Name of listener</td>
</tr>
<tr>
<td>PORT</td>
<td>Port Number of listener</td>
</tr>
<tr>
<td>AF</td>
<td>Listener Address Family</td>
</tr>
<tr>
<td>IMMEDIATE</td>
<td>Immediate Startup Yes</td>
</tr>
<tr>
<td>BACKLOG</td>
<td>Backlog Value for listener</td>
</tr>
<tr>
<td>NUMSOCK</td>
<td>Number of Sockets in listener</td>
</tr>
<tr>
<td>ACCTIME</td>
<td>Timeout Value for ACCEPT</td>
</tr>
<tr>
<td>GIVTIME</td>
<td>Timeout Value for GIVESOCKET</td>
</tr>
<tr>
<td>REATIME</td>
<td>Timeout Value for READ</td>
</tr>
<tr>
<td>RTYTIME</td>
<td>Stack Connection Retry Time</td>
</tr>
<tr>
<td>LAPPLD</td>
<td>Register Application Data</td>
</tr>
</tbody>
</table>

Verify parameters, press PF8 to go to screen 2

```
PF 3 END 8 NEXT 12 CNCL
```

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

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

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

OvertypetoeEnter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINMSGL</td>
<td>Minimum Message Length</td>
</tr>
<tr>
<td>TRAPTRN</td>
<td>Translate TRNID Yes</td>
</tr>
<tr>
<td>TRANUSR</td>
<td>Translate User Data Yes</td>
</tr>
<tr>
<td>SECEXIT</td>
<td>Name of Security Exit</td>
</tr>
<tr>
<td>GETTID</td>
<td>Get TTLS ID (YES</td>
</tr>
<tr>
<td>USERID</td>
<td>Listeners User ID</td>
</tr>
</tbody>
</table>

Verify parameters, press PF7 to go back to screen 1

or ENTER if finished making changes

```
PF 3 END 7 PREV 12 CNCL
```

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

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

If converting to an enhanced listener, the first screen shows the attributes of the enhanced listener.
Pressing PF8 displays the screen used to manage the unique attributes of the enhanced listener.

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

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

Verify parameters, press PF8 to go to screen 2
PF 3 END 8 NEXT 12 CNCL
```

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

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

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

Overtypeto Enter
CSTRAN ===> .... Child Server Transaction Name
CSSSTYP ===> ... Startup Method (KC|IC|TD)
CSDELAY ===> .... Delay Interval (hhmmss)
MSGLENth ===> ... Message Length (0-999)
PEEKDATA ===> ... Enter Y|N
MSGFORMAT ===> ...... Enter ASCII|EBCDIC
USEREXIT ===> .... Name of User/Security exit
GETTID ===> ...... Get TTLS ID (YES|NO)
USERID ===> ........ Listeners User ID

Verify parameters, press PF7 to go back to screen 1
or ENTER if finished making changes
PF 3 END 7 PREV 12 CNCL
```

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

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

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

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

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

Enter One of the Following

CICS
LISTENER

PF 3 END   12 CNCL
```

Figure 56. EZAC,COPY screen

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

**COPY,CICS:**

If you specify CICS on the previous screen, the following screen is displayed:
After the APPLIDs of the source CICS object and the target CICS object are entered, confirmation is requested. When confirmation is entered, the copy is performed.

**COPY,LISTENER:**

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

```
EZAC,COPY,CICS

Enter all fields

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

PF 3 END 12 CNCL

Figure 57. EZAC,COPY,CICS screen
```
After the APPLIDs of the source and target CICS objects and the names of the source and target listeners are entered, confirmation is requested. When the confirmation is entered, the copy is performed.

**DEFINE function for EZAC**

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

**DEFINE,CICS:**

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

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

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

APPLID ===> ........ APPLID of CICS System
TRANID ===> .... Transaction Name of listener
Format ===> ........ Enter STANDARD|ENHANCED

PF 3 END 12 CNCL
```

Figure 62. EZAC,DEFINE,LISTENER screen

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

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

Overtype to Enter

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

Verify parameters, press PF8 to go to screen 2

PF 3 END 8 NEXT 12 CNCL
```

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

Pressing PF8 displays the screen used to manage the unique attributes of the standard listener.
Pressing PF7 displays the screen used to manage the common attributes of the standard listener.

If defining an enhanced listener, the first screen shows the attributes of the enhanced listener.
Pressing PF8 displays the screen used to manage the unique attributes of the enhanced listener.

After the definition is entered, confirmation is requested. When confirmation is entered, the object is created on the configuration file.
DELETE function for EZAC

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

```
EZAC,DELETE, APPLID = ........
Enter One of the Following
CICS
LISTENER
```

Figure 67. 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 entered, the CICS object is deleted.

DELETE,LISTENER:

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

<table>
<thead>
<tr>
<th>EZAC,DELETE,CICS</th>
<th>APPLID = ........</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter all fields</td>
<td></td>
</tr>
<tr>
<td>APPLID ===&gt; ........</td>
<td>APPLID of CICS System</td>
</tr>
</tbody>
</table>

Figure 68. EZAC,DELETE,CICS screen

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

DELETE,LISTENER:

If you specify DELETE,LISTENER, the following screen is displayed:
After the APPLID and listener name are entered, confirmation is requested. When confirmation is entered, the listener object is deleted.

**DISPLAY function for EZAC**

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

```
EZAC,DELETE LISTENER

Enter all fields

APPLID ===> ........ APPLID of CICS System
TRANID ===> .... Transaction Name of listener

PF 3 END 12 CNCL
```

Figure 69. EZAC,DELETE,LISTENER screen
Note: You can skip this screen by entering either EZAC,DISPLAY,CICS or EZAC,DISPLAY,LISTENER.

DISPLAY,CICS:

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

```plaintext
EZAC,DISPLAY,CICS
APPLID = ........

Enter all fields

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

Press ENTER or PF3 to exit

PF 3 END 12 CNCL
```

Figure 71. EZAC,DISPLAY,CICS screen

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

```
EZAC,DISPLAY,CICS
APPLID = ........

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

Press ENTER or PF3 to exit

PF 3 END 12 CNCL
```

Figure 72. EZAC,DISPLAY,CICS detail screen

**DISPLAY,LISTENER:**

If you specify DISPLAY,LISTENER, the following screen is displayed:
If displaying a standard listener, the first screen shows the attributes of the standard listener.

Pressing PF8 displays the screen used to manage the unique attributes of the standard listener.
Pressing PF7 displays the screen used to manage the common attributes of the standard listener.

If displaying an enhanced listener, the first screen shows the attributes of the enhanced listener.
Pressing PF8 displays the screen used to manage the unique attributes of the enhanced listener.

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,DISPLAY,LISTENER (enhanced listener. screen 1 of 2)  APPLID = .........

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLID</td>
<td>APPLID of CICS System</td>
</tr>
<tr>
<td>TRANID</td>
<td>Transaction Name of listener</td>
</tr>
<tr>
<td>PORT</td>
<td>Port Number of listener</td>
</tr>
<tr>
<td>AF</td>
<td>Listener Address Family</td>
</tr>
<tr>
<td>IMMEDIATE</td>
<td>Immediate Startup Yes</td>
</tr>
<tr>
<td>BACKLOG</td>
<td>Backlog Value for listener</td>
</tr>
<tr>
<td>NUMSOCK</td>
<td>Number of Sockets in listener</td>
</tr>
<tr>
<td>ACCTIME</td>
<td>Timeout Value for ACCEPT</td>
</tr>
<tr>
<td>GVTIME</td>
<td>Timeout Value for GIVESOCKET</td>
</tr>
<tr>
<td>REALTIME</td>
<td>Timeout Value for READ</td>
</tr>
<tr>
<td>RTYTIME</td>
<td>Stack Connection Retry Time</td>
</tr>
<tr>
<td>LAPPLD</td>
<td>Register Application Data</td>
</tr>
</tbody>
</table>

Verify parameters, press PF8 to go to screen 2

PF 3 END 8 NEXT 12 CNCL

---

### EZAC,DISPLAY,LISTENER (enhanced listener. screen 2 of 2)  APPLID = .........

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSTRAN</td>
<td>Child Server Transaction Name</td>
</tr>
<tr>
<td>CSSTYP</td>
<td>Startup Method (KC</td>
</tr>
<tr>
<td>CSDELAY</td>
<td>Delay Interval (hmmms)</td>
</tr>
<tr>
<td>MSGLENth</td>
<td>Message Length (0-999)</td>
</tr>
<tr>
<td>PEEKDATA</td>
<td>Enter Y</td>
</tr>
<tr>
<td>MSGFORMAT</td>
<td>Enter ASCII</td>
</tr>
<tr>
<td>USEREXIT</td>
<td>Name of User/Security exit</td>
</tr>
<tr>
<td>GETTID</td>
<td>Get TTLS ID (YES</td>
</tr>
<tr>
<td>USERID</td>
<td>Listeners User ID</td>
</tr>
</tbody>
</table>

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

PF 3 END 7 PREV 12 CNCL

---

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

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

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

### RENAME function for EZAC

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

RENAME,CICS:

If you specify CICS on the previous screen, the following screen is displayed:
After the APPLIDs of the source CICS object and the target CICS object are entered, confirmation is requested. When confirmation is entered, the rename is performed.

**RENAME,LISTENER:**

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

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

Enter all fields

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

PF 3 END 12 CNCL
```

*Figure 79. EZAC,RENAME,CICS screen*

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

**RENAME,LISTENER:**

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

### z/OS UNIX System Services environment effects on IP CICS sockets

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

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

For more information about how MAXFILEPROC affects tuning applications, see [z/OS UNIX System Services Planning](https://www.ibm.com/support/docview.zhtml?c=103331&d=315723). The z/OS configuration tool, called Managed System Infrastructure (msys), contains additional information about the impacts of the UNIX MAXFILEPROC parameter settings.

For more information about the FILEPROCMAX specification, see the documentation provided for the SAF product in use on your system. If using RACF, this can be found in the [z/OS Security Server RACF Security Administrator's Guide](https://www.ibm.com/support/docview.zhtml?c=103331&d=315723).

CICS/TS V2R3 and later does a set_dub_default causing each CICS Sockets task to run as its own OMVS process. Therefore, the MAXPROCSYS parameter must be large enough to accomodate the largest possible number of CICS Sockets tasks plus any other OMVS processes (CICS/TS itself always has at least 2 OMVS processes).
Chapter 3. Configuring the CICS Domain Name Server cache

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

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

Although the CICS DNS cache function is optional, it is useful in a highly active CICS client environment. It combines the GETHOSTBYNAME() call that is supported in CICS sockets, and a cache that saves results from GETHOSTBYNAME() for future reference. If your system receives repeated requests for the same set of domain names, using the DNS can improve performance significantly. If you have specified that IP CICS sockets should use the Open Transaction Environment, and you link to the domain name service module, EZACIC25, your threadsafe program is switched to the QR TCB.

Guideline: If the system resolver caching function is enabled, CICS DNS caching should not be configured. Resolver caching (when enabled) provides a significant performance improvement over the CICS DNS cache. For more information about resolver caching, visit this website: http://www-01.ibm.com/software/htp/cics/library/

See z/OS Communications Server: IP Configuration Reference for information about caching issues, and see z/OS Communications Server: IP Configuration Guide for more information about system resolver caching.

Rules:

- DNS caching does not support the caching of IPv6 addresses because the gethostbyname() function is not IPv6 enabled.
- Using the system resolver caching function provides the following benefits:
  - After a host name is resolved, it is cached locally. Locally caching a host name enables all other applications that run in the system to retrieve this information without increasing the network communications.
  - The system resolver caching function honors the time to live (TTL) value, which indicates when the information for the resource record expires.
  - The system resolver can cache IPv4 and IPv6 resources.
- DNS caching supports the caching of IPv4 addresses. You can use the system resolver for both IPv4 and IPv6 name resolution. IPv6 clients use unique host names and you must enable DNS entries to allow unique host names to exist in different DNS zones. An IPv6 client gets an AAAA address to use when connecting to an IPv6-enabled listener.
CICS DNS cache function components

The function consists of three parts.
- A VSAM file which is used for the cache.

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

VSAM cache file

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

EZACICR macro

The EZACICR macro builds an initialization module for the cache file, because the cache file must start with at least one record to permit updates by the EZACIC25 module. To optimize performance, you can preload dummy records for the host names which you expect to be used frequently. This results in a more compact file and minimizes the I/O required to use the cache. If you do not specify at least one dummy record, the macro builds a single record of binary zeros. See “Step 1: Create the initialization module” on page 100.

EZACIC25 module

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

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

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

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

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

Maximum resolver requests - CACHRES
The maximum number of concurrent requests to the resolver. It is set at 10. See “How the DNS cache handles requests” on page 99.
If the transaction program is executing in the Open Transaction Environment, expect a TCB switch to occur for each call to EZACIC25.

**How the DNS cache handles requests**

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

1. Attempt to retrieve this entry from the cache. If unsuccessful, issue the GETHOSTBYNAME call unless request specifies cache only.

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

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

**Using the DNS cache**

These steps provides the minimum information that you need to use the DNS cache.

**Procedure**

Perform the following steps to use the DNS cache:

1. Create the initialization module, which in turn defines and initializes the file and the EZACIC25 module. See “Step 1: Create the initialization module” on page 100.

2. Define the cache files to CICS. See “Step 2: Define the cache file to CICS” on page 102.

Results

Step 1: Create the initialization module

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

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

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

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

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

The specifications for the EZACICR macro are as follows:

<table>
<thead>
<tr>
<th>Operand</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>There are three acceptable values:</td>
</tr>
<tr>
<td>Value</td>
<td>Meaning</td>
</tr>
<tr>
<td>INITIAL</td>
<td>Indicates the beginning of the generation input. This value should appear only once and should be the first entry in the input stream.</td>
</tr>
<tr>
<td>RECORD</td>
<td>Indicates a dummy record the user wants to generate. There can be from 0 to 4096 dummy records generated and each of them must have a unique name. Generating dummy records for frequently used host names improves the performance of the cache file. A TYPE=INITIAL must precede a TYPE=RECORD statement.</td>
</tr>
<tr>
<td>FINAL</td>
<td>Indicates the end of the generation input. This value should appear only once and should be the last entry in the input stream. A TYPE=INITIAL must precede a TYPE=FINAL.</td>
</tr>
<tr>
<td>AVGREC</td>
<td>The length of the average cache record. This value is specified on the TYPE=INITIAL macro and has a default value of 500. Use the default value until you have adequate statistics to determine a better value. This parameter is the same as the first subparameter in the RECORDSIZE parameter of the IDCAMS DEFINE statement. Accurate definition of this parameter along with use of dummy records minimizes control interval and control area splits in the cache file.</td>
</tr>
<tr>
<td>NAME</td>
<td>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.</td>
</tr>
</tbody>
</table>
The macro can be used in conjunction with IDCAMS to define and load the file. The following example shows a sample job to define and initialize a cache file:

```
//CACHEDEF JOB (accounting,information),programmer.name,
// MSGLEVEL=(1,1),MSGCLASS=A,CLASS=A
//
//* z/OS Communications Server
//* SMP/E distribution name: EZACICFG
//* Licensed Materials - Property of IBM
//* "Restricted Materials of IBM"
//* Copyright IBM Corp. 2000,2008
//* Status = CSV1R10
//*
//* Function: This job defines and then loads the VSAM
//* file used for the CICS TCP cache. The job stream
//* has the following steps:
//* 1. Delete a cache file if one exists
//* 2. Define the VSAM cache file to VSAM
//* 3. Assemble the initialization program
//* 4. Link the initialization program
//* 5. Execute the initialization program to load the
//*    VSAM cache file
//*
//* 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(CICVOL) -
  CYL(1) -
  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
//
//ASM EXEC PGM=ASMA90,PARM='OBJECT,TERM',REGION=1024K
//SYSLIB DD DISP=SHR,DSNAME=SYS1.MACLIB
// DD DISP=SHR,DSNAME=TCPV34.SEZACMAC
//SYST1 DD UNIT=SYSDA,SPACE=(CYL,(5,1))
//SYST2 DD UNIT=SYSDA,SPACE=(CYL,(2,1))
//SYST3 DD UNIT=SYSDA,SPACE=(CYL,(2,1))
//SYSPUNCH DD DISP=SR,DSNAME=NULLFILE
//SYSLIN DD DSNNAME=&OBJSET,DISP=(MOD,PASS),UNIT=SYSDA,
//    SPACE=(400,(500,50)),
//    DCB=(RECFM=FB,BLKSIZE=400,LRECL=80)
//SYSTEM DD SYSDA
// SYSPRINT DD SYSOUT=* SYSIN DD *
// EZACICR TYPE=INITIAL
// EZACICR TYPE=RECORD,NAME=RALVM12
// EZACICR TYPE=FINAL
```
After the cache file has been created, it has the following layout:

Field name
Description

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

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

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

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

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

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

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

Step 2: Define the cache file to CICS

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

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

RDO keyword
Value
File
EZACACHE

Group
Name of group you are placing this function in.

DSName
Must agree with name defined in the IDCAMS in “Step 1: Create the initialization module” on page 100 (for example, CICS.USER.CACHE).
STRings
Maximum number of concurrent users.

Opertime
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: Issue EZACIC25

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

Field name
Description

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

Value Meaning
-1 ERRNO value returned from GETHOSTBYNAME() call. Check ERRNO field.
0 Host name could not be resolved either within the cache or by use of the GETHOSTBYNAME call.

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

1 Host name was resolved using cache.
2 Host name was resolved using GETHOSTBYNAME call.
ERRNO
A fullword binary field specifying the ERRNO returned from the GETHOSTBYNAME call.

HOSTENT address
The address of the returned HOSTENT structure.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>GHBN</td>
<td>GETHOSTBYNAME. This is the only function supported.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Attempt query using cache. If unsuccessful, attempt using GETHOSTBYNAME() call.</td>
</tr>
<tr>
<td>1</td>
<td>Attempt query using GETHOSTBYNAME() call. This forces a cache refresh for this entry.</td>
</tr>
<tr>
<td>2</td>
<td>Attempt query using cache only.</td>
</tr>
</tbody>
</table>

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

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

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

**HOSTENT structure**

The returned HOSTENT structure is shown in Figure 82 on page 105.
Figure 82. The DNS HOSTENT
Chapter 4. Managing IP CICS sockets

Use the CICS TCP/IP interface to:

- Customize your system so that CICS TCP/IP starts and stops automatically. See “Starting and stopping CICS automatically.”
- Manually start and stop CICS TCP/IP after CICS has been initialized. An operator can also query and change specific CICS TCP/IP interface attributes after CICS has been initialized. See “IP CICS socket interface management” on page 108.
- Start and stop CICS TCP/IP from a CICS application program. See “Starting and stopping CICS TCP/IP with program link” on page 122.
- Handle task hangs for TCP/IP CICS socket applications. See “Handling task hangs” on page 123.

Restriction: The IP CICS Socket Operator transaction, EZAO, is not designed to be run from the CICS terminal associated with the MVS system console.

Starting and stopping CICS automatically

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

- Startup (PLTPI)
  To start the IP CICS socket interface automatically, make the following entry in PLTPI after the DFHDELIM entry:
  *
  /* PLT USED TO SUPPORT IP CICS SOCKETS STARTUP */
  /*
  DFHPLT TYPE=INITIAL,SUFFIX=SI
  DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
  DFHPLT TYPE=ENTRY,PROGRAM=EZACIC20
  *
  */
  Add other IP CICS Socket PLT startup programs here...
  *
  DFHPLT TYPE=FINAL
  END

- Shutdown (PLTSD)
  To shut down the IP CICS socket interface automatically (including all other IP CICS sockets enabled programs), make the following entry in the PLTSD before the DFHDELIM entry:
  *
  /* PLT USED TO SUPPORT IP CICS SOCKETS SHUTDOWN */
  /*
  DFHPLT TYPE=INITIAL,SUFFIX=SD
  *
  */
  Add other IP CICS Socket PLT shutdown programs here...
  *
  DFHPLT TYPE=ENTRY,PROGRAM=EZACIC20
  DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
  DFHPLT TYPE=FINAL
  END

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

IP CICS socket interface management

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

This operational transaction has the following functions:

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

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

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

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

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

**Set Interface**
Alter some attributes of the IP CICS socket interface and listener.

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

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

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

When you enter EZAO, the following screen is displayed:
Using the INQUIRE function

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

```
EZAO, INQUIRE

Enter one of the following

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

PF 3 END 12 CNCL
```

Figure 84. EZAO INQUIRE screen

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

```
EZAO, INQUIRE, CICS

Enter one of the following

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

PF 3 END 12 CNCL
```

Figure 85. EZAO INQUIRE CICS screen

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

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

Figure 86. EZAO INQUIRE LISTENER selection screen

If you select a listener transaction, the following screen is displayed:
The LAPPLD entry indicates whether the IP CICS socket interface automatically registers socket application data for the listener.

**Using the SET function**

Use the SET function to dynamically change certain attributes of the IP CICS socket interface and listener. Changes made in this way are not reflected in the configuration options contained in the EZACONFG dataset. Use the EZAO,INQUIRE command to query some values. If you enter SET in the screen shown in [Figure 83 on page 109](#) or if you enter EZAO,SET on a blank screen, the following screen is displayed:
If you enter SET CICS, the following screen is displayed:

This screen displays the following information:

- **TRACE** is the current IP CICS sockets CICS tracing flag. Specify YES or NO to dynamically enable or disable IP CICS sockets CICS tracing.
- **TCBLIM** is the current IP CICS sockets-imposed TCB limit. Specify a value in the range 0 to the value specified by the MAXOPENTCBS option to dynamically change the TCB limiting factor.
• APPLDAT is the current IP CICS socket interface socket application data registration flag. Specify YES or NO to dynamically enable or disable the registration of socket application data.

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

*Figure 90. EZAO SET LISTENER selection screen*

If you select a listener transaction, the following screen is displayed:
The LAPPLD entry indicates whether the IP CICS socket interface registers socket application data for the listener.

**Using the START function**

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

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

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

Enter one of the following

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

PF 3 END              12 CNCL
```

**Figure 92. EZAO START screen**

**EZAO START CICS**

If you type START CICS, the following screen is displayed:

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

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

CICS socket interface Startup Complete

PF 3 END              12 CNCL
```

**Figure 93. EZAO START CICS response screen**

**EZAO START LISTENER**

If you type START LISTENER, the following screen is displayed:
After you type the listener name, the listener starts. The following screen is displayed, and the results appear in the message area:

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

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

PF 3 END 12 CNCL
```

*Figure 94. EZAO START LISTENER screen*

After you type the listener name, the listener starts. The following screen is displayed, and the results appear in the message area:

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

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

CICS socket interface listener CSKL is Started

PF 3 END 12 CNCL
```

*Figure 95. EZAO START LISTENER result screen*

**EZAO START TRACE**

If you type START TRACE, the following screen is displayed:
Issue the EZAO,START,TRACE command on the CICS region where APPLID matches the IP CICS socket interface and where CICS tracing is to be started.

Using the STOP function

The STOP function is used to stop the CICS socket interface or a listener within the interface. If the interface is stopped, all listeners are stopped before the interface is stopped. The STOP function also disables CICS tracing for the CICS socket interface and the listener. If you type STO in the screen shown in Figure 83 on page 109 or enter EZAO STO on a blank screen, the following screen is displayed:
If you specify STOP CICS, the following screen is displayed:

**EZAO STOP CICS**

The following options are available to stop CICS TCP/IP:

**IMMEDIATE=NO**

Used this option in most cases because it gracefully terminates the interface. This option has the following effects on applications using this API:

- If no other socket applications are active or suspended, the listener transaction (CSKL) quiesces after a maximum wait of 3 minutes.
• If active or suspended sockets applications exist, the listener allows them to continue processing. When all of these tasks are complete, the listener terminates.
• This option denies access to this API for all new CICS tasks. Tasks that start after CICS TCP/IP has been stopped END with the CICS abend code AEY9.

**IMMEDIATE=**YES
This option is reserved for unusual situations and abruptly terminates the interface. It has the following effect on applications using this API:
• Purges the master server (listener) CSKL.
• Denies access to the API for all CICS tasks. Tasks that have successfully called the API previously abend with the AETA abend code on the next socket call. New tasks that have started are denied by the AEY9 abend code.

After you choose an option, the stop is attempted. The screen is displayed again, and the results appear in the message area.

**EZAO STOP LISTENER**

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

```
EZAO,STOP,LISTENER

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

PF 3 END 12 CNCL
```

*Figure 99. EZAO STOP LISTENER screen*

When you input the listener named, that listener is stopped. The screen is displayed again, and the results appear in the message area.

**EZAO STOP TRACE**

If you specify STOP TRACE, the following screen is displayed:
Issue the EZAO,STOP,TRACE command on the CICS region where APPLID matches the IP CICS socket interface and where CICS tracing is to be stopped.

### Abbreviating the EZAO transaction parameters

It is possible to abbreviate the parameters of the EZAO transaction, but a minimum of three characters must be specified. This capability allows the command to be issued using minimal keystrokes. The following list of commands shows the abbreviated parameters:

- **EZAO,STArt,CICs**
  - Starts the interface

- **EZAO,STOp,CICs**
  - Stops the interface

- **EZAO,STArt,LIStener**
  - Starts a listener

- **EZAO,STOp,LIStener**
  - Stops a listener

- **EZAO,STArt,TRAce**
  - Enables CICS tracing

- **EZAO,STOp,TRAce**
  - Disables CICS tracing

**Note:**

- The values in uppercase characters are the minimal acceptable value for parameters.
- You can use spaces instead of commas as a parameter delimiter. This is shown in the following example:
  
  ```
  EZAO STArt CICs
  ```

  This is the same as the following:
Starting and stopping CICS TCP/IP with program link

Issue an EXEC CICS LINK to program EZACIC20 to start or stop the CICS socket interface. You need to follow these steps in the LINKing program.

Procedure

Perform the following steps to start or stop the CICS socket interface with program link:

1. Define the COMMAREA for EZACIC20 by including the following instruction in your DFHEISTG definition:
   
   ```
   EZACICA AREA=P20,TYPE=CSECT
   ```
   
   The length of the area is equated to P20PARML, and the name of the structure is P20PARMS.

2. Initialize the COMMAREA values as follows:

   - **P20TYPE**
     - I Initialization
     - T Immediate termination
     - D Deferred termination
     - Q Quiesce the CICS socket interface by querying the PLT shutdown immediate configuration option and performing the shutdown based on the results of that query

   - **P20OBJ**
     - C CICS sockets interface
     - L Listener

   - **P20LIST**
     Name of listener (if this is listener initialization or termination)

3. Issue the EXEC CICS LINK to program EZACIC20. EZACIC20 does not return until the function is complete.

4. Check the P20RET field for the response from EZACIC20. See the P20RET field of the P20PARMS structure in the hlq.SEZACMAC(EZACICA) macro for the meanings of the return values from calling EZACIC20.

Results

EZACIC20 can issue the following user abend codes:

- Abend code E20L is issued if the CICS socket interface is not in startup or termination and no COMMAREA was provided.
- Abend code E20T is issued if CICS is not active or if you run the EZACIC20 program at the wrong PLT phase. See "CICS program list table" on page 47 for more information about setting CICS TCP sockets to automatically startup or shutdown by using updates to the PLT.
Handling task hangs

TCP/IP CICS socket applications might encounter hangs when they are using sockets API blocking calls. The most common scenario occurs when the remote peer fails to send data for the read or receive functions that are issued by the CICS socket application. When this situation occurs, get the read data from the socket before using a select or selectex function call. However, even when you use these functions to get the read data, you must end the hung transactions. The external symptom of this kind of hang in CICS is that the transactions are in an external wait in the TCP/IP CICS TRUE (module EZACIC01).

Perform one of the following two tasks to terminate a transaction that is in an external wait in EZACIC01:

- Set the APPLDAT value to YES in the TYPE=CICS configuration (EZAC transaction). You can use the NETSTAT CONN APPLDATA (CLIENT CICSjobname command to correlate the connection IDs to the associated hung transactions. The following sample shows the Netstat output when you use the appldata keyword:

<table>
<thead>
<tr>
<th>User Id</th>
<th>Conn</th>
<th>Local Socket</th>
<th>Foreign Socket</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>CICS</td>
<td>00006BF0</td>
<td>0.0.0.0..3010</td>
<td>0.0.0.0..0</td>
<td>Listen</td>
</tr>
<tr>
<td>Application Data: EZACIC00 CSKL 0000037 ........</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  The data that is returned consists of the transaction name (CSKL in the sample) and the CICS transaction number (0000037 in the sample).

  By using this data with the TCP/IP Conn ID (00006BF0 in the sample), you can issue a Netstat drop command to take the following actions:
  - Stop the connection from a TCP/IP perspective.
  - Cause the outstanding blocking function call to fail.
  - Return control to the application.

- Use CEMT force purge from CICS.

Note: CEMT purge or DTIMEOUT do not have an effect because the TCP/IP CICS TRUE is defined as non-purgeable.
Chapter 5. Writing your own listener

The IP CICS socket interface provides a structure that supports multiple listeners. These listeners can be multiple copies of the IBM-supplied listener, user-written listeners, or a combination of the two. You can also run without a listener.

For each listener (IBM-supplied or user-written), there are certain basic requirements that enable the interface to manage the listeners correctly, particularly during initialization and termination. They are:

- Each listener instance must have a unique transaction name, even if you are running multiple copies of the same listener.
- Each listener should have an entry in the CICS sockets configuration data set. Even if you do not use automatic initiation for your listener, the lack of an entry would prevent correct termination processing and could prevent CICS from completing a normal shutdown.

For information on the IBM-supplied listener, see "CICS application transaction (IBM listener)" on page 141.

Prerequisites for writing your own listener

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

1. Determine what capability is required that is not supplied by the IBM-supplied listener. Is this capability a part of the listener or a part of the server?
2. Knowledge of the CICS-Assembler environment is required.
3. Knowledge of multi-threading applications is required. A listener must be able to perform multiple functions concurrently to achieve good performance.
4. Knowledge of the CICS socket interface is required.
5. Knowledge of how to use compare and swap logic for serially updating shared resources.

Using IBM environmental support for user-written listeners

A user-written listener can use the environmental support supplied and used by the IBM-supplied listener. To employ this support, the user-written listener must do the following in addition to the requirements described in "Prerequisites for writing your own listener”:

- The user-written listener must be written in Assembler.
- The RDO definitions for the listener transaction and program should be identical to those for the IBM-supplied listener with the exception of the transaction/program names. Reference the program definition for the IBM-supplied listener, EZACIC02, in SEZAINST(EZACICCT).
In the program, define an input area for the configuration file records. If you are going to read the configuration file using MOVE mode, you can define the area by making the following entry in your DFHEISTG area:

```
EZACICA AREA=CFG,TYPE=CSECT
```

If you are going to read the configuration file using LOCATE mode you can define a DSECT for the area as follows:

```
EZACICA AREA=CFG,TYPE=DSECT
```

In either case, the length of the area is represented by the EQUATE label CFGLEN. The name of the area/DSECT is CFG0000.

In the program, define a DSECT for mapping the Global Work Area (GWA). This is done by issuing the following macro:

```
EZACICA AREA=GWA,TYPE=DSECT
```

The name of the DSECT is GWA0000.

In the program, define a DSECT for mapping the Task Interface Element (TIE). This is done by issuing the following macro:

```
EZACICA AREA=TIE,TYPE=DSECT
```

The name of the DSECT is TIE0000.

In the program define a DSECT for mapping the listener Control Area (LCA). This is done by issuing the following macro:

```
EZACICA AREA=LCA,TYPE=DSECT
```

The name of the DSECT is LCA0000.

Obtain address of the GWA. This can be done using the following CICS command:

```
EXEC CICS EXTRACT EXIT PROGRAM(EZACIC01) GASET(ptr) GALEN(len)
```

where `ptr` is a register and `len` is a halfword binary variable. The address of the GWA is returned in `ptr` and the length of the GWA is returned in `len`. Use of the Extract Exit command requires UPDATE access to the EXITPROGRAM resource. Failure to have at least the UPDATE access to the EXITPROGRAM resource causes the IP CICS socket interface and listener to either not start when starting or not stop when stopping.

**Guideline:** As of CICS/TS 2.3, the EXEC CICS EXTRACT command is not threadsafe. If the interface is using the CICS Open Transaction Environment, you should issue this command with other non-threadsafecommands to prevent excessive TCB switching.

Read the configuration file during initialization of the listener. The configuration file is identified as EZACONFG in the CICS Configuration file. The record key for the user-written listener is as follows:

- APPLID
An 8-byte character field set to the APPLID value for this CICS. This value can be obtained from the field GWACAPPL in the GWA or by using the following CICS command:

EXEC CICS ASSIGN APPLID(applid)

where applid is an 8-byte character field.

– Record Type
   A 1-byte character field set to the record type. It must have the value L.
– Reserved Field
   A 3-byte hex field set to binary zeros.
– Transaction
   A 4-byte character field containing the transaction name for this listener. It can be obtained from the EIBTRNID field in the Execute Interface Block.

The configuration record provides the information entered by either the EZACICD configuration macro or the EZAC Configuration transaction. The user-written listener can use this information selectively, but it is preferred because it contains the values specified for PORT, BACKLOG, and NUMSOCK. See Chapter 2, “Setting up and configuring CICS TCP/IP,” on page 23 for more information about the configuration data set with EZACICD TYPE parameter subsection.

For shared files: If the user-written listener reads the configuration file, it must first issue an EXEC CICS SET command to enable and open the file. When the file operation is complete, the user-written listener must issue an EXEC CICS SET command to disable and close the file. Failure to do so results in file errors in certain shared-file situations.

Requirement: Use of the EXEC CICS ENABLE command requires UPDATE access to EXITPROGRAM resources. Failure to have at least the UPDATE access to the EXITPROGRAM resource causes the IP CICS socket interface and listener to either not start when starting or not stop when stopping.

• The user-written listener should locate its listener Control Area (LCA). The LCAs are located contiguously in storage with the first one pointed to by the GWALCAAD field in the GWA. The correct LCA has the transaction name of the listener in the field LCATRAN.

• The user-written listener should set the LCASTAT field to a value specified by LCASTATP so that the IP CICS socket interface is aware that the listener is active. Otherwise, the IP CICS sockets listener termination logic bypasses the posting of the listeners termination ECB.

• The user-written listener should monitor either the LCASTAT field in the LCA or the GWATSTAT field in the GWA for shutdown status. If either field shows an immediate shutdown in progress, the user-written listener should terminate by issuing the EXEC CICS RETURN command and allow the interface to clean up any socket connections. If either field shows a deferred termination in progress, the user-written listener should do the following:
  1. Accept any pending connections, and close the passive (listen) socket.
  2. Complete the processing of any sockets involved in transaction initiation (that is, processing the GIVESOCKET command). When processing is complete, close these sockets.
  3. When all sockets are closed, issue the EXEC CICS RETURN command.

• The user-written listener should avoid socket calls which imply blocks dependent on external events such as ACCEPT or READ. These calls should be preceded by a single SELECTEX call that waits on the ECB LCATECB in the
LCA. This ECB is posted when an immediate termination is detected, and its posting causes the SELECTEX to complete with a RETCODE of 0 and an ERRNO of 0. The program should check the ECB when the SELECTEX completes in this way as this is identical to the way SELECTEX completes when a timeout happens. The ECB can be checked by looking for a X'40' in the first byte (post bit).

This SELECTEX should also specify a timeout value. This provides the listener with a way to periodically check for a deferred termination request. Without this, CICS sockets Deferred Termination or CICS Deferred Termination cannot complete.

- The user-written listener should use a non-reusable subtask. Issue the INITAPI command or an INITAPIX command with the letter L in the last byte of the subtask name. The user-written listener implements the termination and detach logic in the same way that the IBM-supplied listener does.

- The user-written listener should update LCASTAT with one of the following:

```
LCASTAT DS X Status of this listener
LCASTAT0 EQU B'00000000' Listener not in operation
LCASTAT1 EQU B'00000001' Listener in initialization
LCASTAT5 EQU B'00000010' Listener in SELECT
LCASTATP EQU B'00000100' Listener processing
LCASTATL EQU B'00001000' Listener had initialization error
LCASTATQ EQU B'00100000' Deferred termination in progress
LCASTATL2 EQU B'01000000' Listener is active
LCASTATR EQU B'10000000' Listener is CICS delayed retry
```

**Rule:** If IP CICS sockets is configured to use CICS's Open Transaction Environment, then ensure that you serially update the LCASTAT value. The Listener Control Area (LCA) is part of the global work area (GWA), and is considered to be a shared resource. An appropriate value to move into LCASTAT would be LCASTATP (B'00000100') when the user-written listener starts. This value enables the CICS socket logic to correctly post the LCATECB during both deferred and immediate termination.

- User-written listener programs can use the LCASTAT2A status flag to determine whether this listener should register application data. The user-written listener should update LCASTAT2 with one of the following:

```
LCASTAT2 DS X Listener status byte 2
LCASTAT2C EQU B'00000001' Listener can now connect to TCP
LCASTAT2A EQU B'00000010' Register Application Data
LCASTAT2H EQU B'00000100' LAPPLD inherits APPLDAT
LCASTAT2S EQU B'00100000' This is a STANDARD listener
LCASTAT2E EQU B'01000000' This is an ENHANCED listener
LCASTAT26 EQU B'10000000' Listeners AF is AF_INET6
```
Chapter 6. Writing applications that use the IP CICS sockets API

This topic describes how to write applications that use the IP CICS sockets API. It describes typical sequences of calls for client, concurrent server (with associated child server processes), and iterative server programs. The contents of the topic are:

- The following setups for writing CICS TCP/IP applications are available:
  - Concurrent server (the supplied listener transaction) and child server processes run under CICS TCP/IP.
  - The same as 1 but with a user-written concurrent server.
  - An iterative server running under CICS TCP/IP.
  - A client application running under CICS TCP/IP.
- Socket addresses
- MVS address spaces
- GETCLIENTID, GIVESOCKET, and TAKESOCKET commands
- The listener program
- CICS Open Transaction Environment considerations
- Application Transparent Transport Layer Security (AT-TLS)

Chapter 7, “C language application programming,” on page 165 describes the C language calls that can be used with CICS.

Chapter 8, “Sockets extended API,” on page 249 provides reference information on the Sockets Extended API for COBOL, PL/I, and Assembler language. The Sockets Extended API is the preferred interface for new application development.

Note: Appendix A, “Original COBOL application programming interface (EZACICAL),” on page 421 provides reference information on the EZACICAL API for COBOL and assembler language. This interface was made available in a prior release of TCP/IP Services and is being retained in the current release for compatibility. For the best results, however, use the Sockets Extended API whenever possible. It is described in Chapter 8, “Sockets extended API,” on page 249.

Writing CICS TCP/IP applications

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

The setups are:

- The client-listener-child server application set. The concurrent server and child server processes run under CICS TCP/IP. The concurrent server is the supplied listener transaction. The client might be running TCP/IP under one of the various UNIX operating systems such as AIX®.
• Writing your own concurrent server. This is the same setup as the first except that a user-written concurrent server is being used instead of the IBM listener.

• The iterative server CICS TCP/IP application. This setup is designed to process one socket at a time.

• The client CICS TCP/IP application. In this setup, the CICS application is the client and the server is the remote TCP/IP process.

For details of how the CICS TCP/IP calls should be specified, see Chapter 7, “C language application programming,” on page 165, Chapter 8, “Sockets extended API,” on page 249, and Appendix A, “Original COBOL application programming interface (EZACICAL),” on page 421.

The client-listener-child-server application set

Figure 102 on page 131 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.
Table 8 explains the functions of each of the calls listed in Figure 102.

**Table 8. Calls for the client application**

<table>
<thead>
<tr>
<th>Call</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) INITAPI</td>
<td>Connect the CICS application to the TCP/IP interface. (This call is used only by applications written in Sockets Extended or the EZACICAL interface). Use the MAXSOC parameter on the Sockets Extended INITAPI or the MAX-SOCK parameter on the EZACICAL interface to specify the maximum number of sockets to be used by the application.</td>
</tr>
</tbody>
</table>
Table 8. Calls for the client application (continued)

<table>
<thead>
<tr>
<th>Call</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2) SOCKET</td>
<td>This obtains a socket. You define a socket with three parameters:</td>
</tr>
<tr>
<td></td>
<td>• The domain, or addressing family</td>
</tr>
<tr>
<td></td>
<td>• The type of socket</td>
</tr>
<tr>
<td></td>
<td>• The protocol</td>
</tr>
<tr>
<td></td>
<td>For CICS TCP/IP, the domain can be only one of the TCP/IP Internet domains, either AF_INET (2) for IPv4 or AF_INET6 (19) for IPv6. The type can be SOCK_STREAM (1) for stream sockets (TCP) or SOCK_DGRAM (2) for datagram sockets (UDP). The protocol can be either TCP or UDP. Passing 0 for the protocol selects the default protocol.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>(3) CONNECT</td>
<td>Client applications use this to establish a connection with a remote server. You must define the local socket S to be used in this connection and the address and port number of the remote socket. The system supplies the local address, so on successful return from CONNECT, the socket is completely defined, and is associated with a TCP connection (if stream) or UDP connection (if datagram).</td>
</tr>
<tr>
<td>(4) WRITE</td>
<td>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.</td>
</tr>
<tr>
<td>(5) READ/WRITE</td>
<td>These calls continue the conversation with the server until it is complete.</td>
</tr>
<tr>
<td>(6) CLOSE</td>
<td>This closes a specified socket and so ends the connection. The socket resources are released for other applications.</td>
</tr>
</tbody>
</table>

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 “Writing your own concurrent server” on page 133, where the listener calls in Figure 102 on page 131 are explained.

Child server call sequence

Table 9 explains the functions of each of the calls listed in Figure 102 on page 131.

Table 9. Calls for the server application

<table>
<thead>
<tr>
<th>Call</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>(7) EXEC CICS RETRIEVE</td>
<td>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.</td>
</tr>
<tr>
<td>(8) TAKESOCKET</td>
<td>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. <strong>Note:</strong> If TAKESOCKET is the first call, it issues an implicit INITAPI with default values.</td>
</tr>
</tbody>
</table>
Table 9. Calls for the server application (continued)

<table>
<thead>
<tr>
<th>Call</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>(9) READ/WRITE</td>
<td>The conversation with the client continues until complete.</td>
</tr>
<tr>
<td>(10) CLOSE</td>
<td>Terminates the connection and releases the socket resources when finished.</td>
</tr>
</tbody>
</table>

Writing your own concurrent server

The overall setup is the same as the first scenario, but your concurrent server application performs many of the functions performed by the listener. Obviously, the client and child server applications have the same functions.

Concurrent server call sequence

Table 10 explains the functions of each of the steps listed in Figure 102 on page 131.

Table 10. Calls for the concurrent server application

<table>
<thead>
<tr>
<th>Call</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>(11) INITAPI</td>
<td>Connects the application to TCP/IP, as in Table Table 8 on page 131.</td>
</tr>
<tr>
<td>(12) SOCKET</td>
<td>This obtains a socket, as in Table Table 8 on page 131.</td>
</tr>
<tr>
<td>(13) BIND</td>
<td>After a socket has been obtained, a concurrent server uses this call to attach itself to a specific port at a specific address so that the clients can connect to it. The socket descriptor and a local address and port number are passed as arguments. On successful return of the BIND call, the socket is bound to a port at the local address, but not (yet) to any remote address.</td>
</tr>
<tr>
<td>(14) LISTEN</td>
<td>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.</td>
</tr>
<tr>
<td>(15) GETCLIENTID</td>
<td>This command returns the identifiers (MVS address space name and subtask name) by which the concurrent server is known by TCP/IP. This information is needed by the EXEC CICS START call.</td>
</tr>
<tr>
<td>(16) SELECTEX</td>
<td>The SELECTEX call monitors activity on a set of sockets. In this case, it is used to interrogate the queue (created by the LISTEN call) for connections. It returns when an incoming CONNECT call is received or when LCATECB was posted because immediate termination was detected, or else times out after an interval specified by one of the SELECTEX parameters.</td>
</tr>
<tr>
<td>(17) ACCEPT</td>
<td>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.</td>
</tr>
<tr>
<td>(18) RECV</td>
<td>A RECV is not issued if the FORMAT parameter is ENHANCED and MSGLENTH is 0. If FORMAT is ENHANCED, MSGLENTH is not 0, and PEEKDATA is YES, the listener peeks the number of bytes specified by MSGLENTH. If FORMAT is STANDARD, the listener processes the client data as in earlier releases.</td>
</tr>
</tbody>
</table>
### Table 10. Calls for the concurrent server application  (continued)

<table>
<thead>
<tr>
<th>Call</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>(19) CICS INQ</td>
<td>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.</td>
</tr>
<tr>
<td>(20) GIVESOCKET</td>
<td>This makes the socket obtained by the ACCEPT call available to a child server program.</td>
</tr>
<tr>
<td>(21) CICS START</td>
<td>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 “IBM listener output format” on page 144, the parameters LSTN-NAME and LSTN-SUBNAME define the listener.</td>
</tr>
<tr>
<td>(22) SELECTEX 8</td>
<td>Again, the SELECTEX call is used to monitor TCP/IP activity. This time, SELECTEX returns when the child server issues a TAKESOCKET call.</td>
</tr>
<tr>
<td>(23) CLOSE</td>
<td>This releases the new socket to avoid conflicts with the child server.</td>
</tr>
</tbody>
</table>

### 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.

### The iterative server CICS TCP/IP application

Figure 103 on page 135 shows the sequence of socket calls involved in a simple client-iterative server setup.

---

8. This SELECTEX is the same as the SELECTEX call in Step 16. They are shown as two calls to clarify the functions being performed.
The setup with an iterative server is much simpler than the previous cases with concurrent servers.

**Iterative server use of sockets**

The iterative server needs to obtain only two 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. After this completed, socket 1 closes.
4. Control returns to the SELECT call, which then waits for the next connection request.

The disadvantage of an iterative server is that it remains blocked for the duration of a transaction, as described in Chapter 1, “Introduction to CICS TCP/IP,” on page 1.

**The client CICS TCP/IP application**

Figure 104 on page 136 shows the sequence of calls in a CICS client-remote server setup. The calls are similar to the previous examples.
Figure 104 shows that the server can be on any processor and can run under any operating system, provided that the combined software-hardware configuration supports a TCP/IP server.

For simplicity, the figure shows an iterative server. A concurrent server would need a child server in the remote processor and an adjustment to the calls according to the model in Figure 102 on page 131.

A CICS server issues a READ call to read the client’s first message, which contains the CICS transaction name of the required child server. When the server is in a non-CICS system, application design must specify how the first message from the CICS client indicates the service required (in Figure 104, the first message is sent by a WRITE call).

If the server is a concurrent server, this indication is typically the name of the child server. If the server is iterative, as in Figure 104, and all client calls require the same service, this indication might not be necessary.

### Defining 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) support

CICS TCP/IP supports the AF_INET and AF_INET6 TCP/IP addressing family (or domain, as it is called in the UNIX system). This is the Internet domain, denoted by AF_INET or AF_INET6 in C. Many of the socket calls require you to define the domain as one of their parameters.

A socket address is defined by the IP address of the socket and the port number allocated to the socket.

IP address allocation

IP addresses are allocated to each TCP/IP services address on a TCP/IP Internet. Each address is a unique 32-bit (an IPv4 Internet Address) or a unique 128-bit (an IPv6 Internet Address) quantity defining the host’s network and the particular host. A host can have more than one IP address if it is connected to more than one network (a so-called multihomed host).

Port number identification

A host can maintain several TCP/IP connections at one time. One or more applications using TCP/IP on the same host are identified by a port number. The port number is an additional qualifier used by the system software to get data to the correct application. Port numbers are 16-bit integers; some numbers are reserved for particular applications and are called well-known ports (for example, 23 is for TELNET).

Address structures

The address structure depends on the IP addressing family. An IPv4 socket address in an IP addressing family is comprised of the following four fields:

- **Address family**: Set to AF_INET in C, or to a decimal 2 in other languages.
- **Port**: Port used by the application, in network byte order (which is explained in “TCP/IP network byte ordering convention” on page 139).
- **IPv4 address**: The IPv4 address of the network interface used by the application. It is also in network byte order.
- **Character array**: Should always be set to all zeros.

An IPv6 socket address in an IP addressing family is comprised of the following five fields:

- **Address family**: Set to AF_INET6 in C or to a decimal 19 in other languages.
- **Port**: Port used by the application, in network byte order (which is explained in “TCP/IP network byte ordering convention” on page 139).
- **Flow Information**: Four bytes in binary format indicating traffic class and flow label. This field is currently not implemented.
- **IPv6 address**: The IPv6 address of the network interface used by the application. It is in network byte order.
Scope ID
Used to specify link scope for an IPv6 address as a interface index. If specified, and the destination is not link local, the socket call fails.

Address structure for COBOL, PL/I, and assembler language programs

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Assembler</th>
<th>COBOL</th>
<th>PL/I</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAMILY</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
<td>FIXED BIN(15)</td>
</tr>
<tr>
<td>PORT</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
<td>FIXED BIN(15)</td>
</tr>
<tr>
<td>ADDRESS</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
<td>FIXED BIN(31)</td>
</tr>
<tr>
<td>ZEROS</td>
<td>XL8</td>
<td>PIC X(8)</td>
<td>CHAR(8)</td>
</tr>
</tbody>
</table>

The address structure of an IPv6 Internet socket address should be defined as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Assembler</th>
<th>COBOL</th>
<th>PL/I</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAMILY</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
<td>FIXED BIN(15)</td>
</tr>
<tr>
<td>PORT</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
<td>FIXED BIN(15)</td>
</tr>
<tr>
<td>FLOWINFO</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
<td>FIXED BIN(31)</td>
</tr>
<tr>
<td>ADDRESS</td>
<td>XL16</td>
<td>two PIC 9(16) BINARY</td>
<td>CHAR(16)</td>
</tr>
<tr>
<td>SCOPE ID</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
<td>FIXED BIN(31)</td>
</tr>
</tbody>
</table>

Address structure for C programs

The structure of an IPv4 Internet socket address is defined by the `sockaddr_in` structure, which is found in the IN.H header file. The structure of an IPv6 Internet socket address structure is defined by the `sockaddr_in6` structure, which is found in the IN.H header file. The format of these structures is shown in Table 19 on page 167.

MVS address spaces relationship between TCP/IP and CICS

Figure 105 on page 139 shows the relationship between TCP/IP and CICS in terms of MVS address spaces.
Within each CICS region, server and client processes are allocated subtask
numbers. TCP/IP treats each CICS region together with its application programs as
a client application. Because of this, the address space and subtask of each CICS
TCP/IP application is called its CLIENTID. This applies to CICS TCP/IP servers as
well as to clients.

A single task can support up to 65535 sockets. However, the maximum number of
sockets that the TCP/IP address space can support is determined by the value of
MAXSOCKETS. Therefore, using multiple tasks, a single CICS region can support a
number of sockets up to the setting of MAXSOCKETS, which has a maximum
possible value of 16,777,215.

MAXFILEPROC limits the number of sockets per process. Because CICS is
considered a process, MAXFILEPROC can limit the number of files allocated for
the CICS region. Ensure that MAXFILEPROC is set to accommodate the total
number of sockets used by all tasks running in the region.

The structure of CLIENTID is shown in Table 11. With CICS TCP/IP, the domain is
always AF_INET, so the name (that is, address space) and subtask are the items of
interest.

Table 11. CLIENTID structures

<table>
<thead>
<tr>
<th>C structure</th>
<th>COBOL structure</th>
</tr>
</thead>
</table>
| struct clientid {
  int domain;
  char name[8];
  char subtaskname[8];
  char reserved[20];
}; | CLIENTID STRUCTURE:
  01 CLIENTID.
  02 DOMAIN PIC 9(8) BINARY.
  02 NAME PIC X(8).
  02 TASK PIC X(8).
  02 RESERVED PIC X(20). |

TCP/IP network byte ordering convention

Ports and addresses are specified using the TCP/IP network byte ordering
convention, which is known as big endian.

In a big endian system, the most significant byte comes first. By contrast, in a little
endian system, the least significant byte comes first. MVS uses the big endian
convention; because this is the same as the network convention, CICS TCP/IP
applications do not need to use any conversion routines, such as htonl, htons,
ntohl, and ntohs.
Note: The socket interface does not handle differences in data byte ordering within application data. Sockets application writers must handle these differences themselves.

GETCLIENTID, GIVESOCKET, and TAKESOCKET

The socket calls GETCLIENTID, GIVESOCKET, and TAKESOCKET are unique to the IBM implementation of the socket interface. In CICS TCP/IP, they are used with the EXEC CICS START and EXEC CICS RETRIEVE commands to make a socket available to a new process. This is shown in Figure 106.

Figure 106 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 (CLIENT-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 on page 141.

2. The listener calls GIVESOCKET, specifying a socket descriptor and the CLIENTID of the child server.
   
   If the listener and child server processes are in the same CICS region (and so in the same address space), the MVS address space identifier in CLIENTID can be set to blanks. This means that the listener’s address space is also the child’s address space.
   
   If the listener and child server processes are in different CICS regions, enter the new address space and subtask.
   
   In the CLIENTID structure, the supplied listener sets the address space name and subtask identifier to blanks. This makes the socket available to a TAKESOCKET command from any task in the same MVS image, but only the child server receives the socket descriptor number, so the exposure is minimal. For total integrity, the subtask identifier of the child server should be entered.

Figure 106 shows the calls used to make a listener socket available to a child server process. It shows the following steps:

1. The listener calls GETCLIENTID. This returns the listener’s own CLIENTID (CLIENT-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 on page 141.

2. The listener calls GIVESOCKET, specifying a socket descriptor and the CLIENTID of the child server.
   
   If the listener and child server processes are in the same CICS region (and so in the same address space), the MVS address space identifier in CLIENTID can be set to blanks. This means that the listener’s address space is also the child’s address space.
   
   If the listener and child server processes are in different CICS regions, enter the new address space and subtask.
   
   In the CLIENTID structure, the supplied listener sets the address space name and subtask identifier to blanks. This makes the socket available to a TAKESOCKET command from any task in the same MVS image, but only the child server receives the socket descriptor number, so the exposure is minimal. For total integrity, the subtask identifier of the child server should be entered.
3. The listener performs an EXEC CICS START. In the FROM parameter, the "CLIENTID-L", obtained by the previous GETCLIENTID, is specified. The listener is telling the new child server where to retrieve its socket from in step 5.

4. The child server performs an EXEC CICS RETRIEVE. In the INTO parameter, "CLIENTID-L" is retrieved.

5. The child server calls TAKE SOCKET, specifying "CLIENTID-L" as the process from which it wants to take a socket.

### CICS application transaction (IBM listener)

In a CICS system based on SNA terminals, the CICS terminal management modules perform the functions of a concurrent server. Because the TCP/IP interface does not use CICS terminal management, CICS TCP/IP provides these functions in the form of a CICS application transaction, the listener. The CICS transaction ID of the IBM distributed listener is CSKL. This transaction is defined at installation to execute the EZACIC02 program and is to be further referenced as the listener. This transaction ID can be configured to a transaction ID suitable for the user's requirements through the use of the EZACICD macro or the EZAC CICS transaction and the accompanying RDO transaction definition.

The listener performs the following functions:

- It issues appropriate TCP/IP calls to listen on the port specified in the configuration file and waits for incoming connection requests issued by clients. The port number must be reserved in the hlq.TCPIP.PROFILE to the CICS region using the TCP/IP CICS sockets interface.
- When an incoming connection request arrives, the listener accepts it and obtains a new socket to pass to the CICS child server application program.
- The standard listener starts the CICS child server transaction based on information in the first message on the new connection. The format of this information is given in ["IBM listener input format" on page 142] for the enhanced listener, it starts the CICS child server transaction based on information in the TCP/IP CICS configuration file, EZACONFG.
- It waits for the child server transaction to take the new socket and then issues the close call. When this occurs, the receiving application assumes ownership of the socket and the listener has no more interest in it.

The listener program is written so that some of this activity goes on in parallel. For example, while the program is waiting for a new server to accept a new socket, it listens for more incoming connections. The program can be in the process of starting 49 child servers simultaneously. The starting process begins when the listener accepts the connection and ends when the listener closes the socket it has given to the child server.

[Table 12 on page 142] illustrates the listener configuration in contrast with the connected clients address family and indicates the contents of the IPv4 and IPv6 IP address fields presented to the security or transaction exit.
Table 12. Listener configuration presented to security or transaction exit

<table>
<thead>
<tr>
<th>Listeners AF configuration</th>
<th>Connected client's AF</th>
<th>Exits address family</th>
<th>Exits client's IPv4 address</th>
<th>Exits client's IPv6 address</th>
<th>Exits listener's IPv4 address</th>
<th>Exits listener's IPv6 address</th>
</tr>
</thead>
<tbody>
<tr>
<td>not specified</td>
<td>AF_INET</td>
<td>AF_INET</td>
<td>IPv4 addr zeros</td>
<td>IPv4 addr zeros</td>
<td>IPv4 addr zeros</td>
<td>IPv4 addr zeros</td>
</tr>
<tr>
<td>AF_INET</td>
<td>AF_INET</td>
<td>AF_INET</td>
<td>IPv4 addr zeros</td>
<td>IPv4 addr zeros</td>
<td>IPv4 addr zeros</td>
<td>IPv4 addr zeros</td>
</tr>
<tr>
<td>AF_INET6</td>
<td>AF_INET6</td>
<td>AF_INET6</td>
<td>zeros</td>
<td>IPv4 mapped IPv6 addr zeros</td>
<td>IPv4 mapped IPv6 addr</td>
<td>IPv4 mapped IPv6 addr</td>
</tr>
<tr>
<td>AF_INET6</td>
<td>AF_INET6</td>
<td>AF_INET6</td>
<td>zeros</td>
<td>IPv6 addr zeros</td>
<td>IPv4 addr zeros</td>
<td>IPv4 addr zeros</td>
</tr>
</tbody>
</table>

IBM listener input format

The standard listener requires the following input format from the client in its first transmission. The client should then wait for a response before sending any subsequent transmissions. Input can be in uppercase or lowercase. The commas are required.

**Note:** Because the listener cannot distinguish between a comma used as a delimiter in the listener's initial message and a comma that is part of the client-in-data format, the client-in-data format should not contain a comma. In text such as x'2C' in ASCII data or such as '6B' in EBCDIC data, the single quote can be interpreted as a comma.
**tran**
The CICS transaction ID (in uppercase) that the listener is going to start. This field can be one to four characters.

**client-in-data**
Optional. Application data, used by the optional security exit or the server transaction. The maximum length of this field is a 40-byte character (35 bytes, plus 1 byte filler and 4 bytes for startup type).

**/ic/td/kc**
Optional. The startup type that can be either KC for CICS task control, IC for CICS interval control or TD for CICS transient data. These can also be entered in lowercase (kc, ic, or td). If this field is left blank, startup is immediate using CICS task control (KC). KC or kc can be specified to indicate that the child server task is started using EXEC CICS START with no delay interval. This is the same as specifying IC,000000.

**hhmmss**
Optional. Hours, minutes, and seconds for interval time if the transaction is started using interval control. All six digits must be given.

**Note:** TD ignores the timefield.

### Examples of client input and the listener processing

The following are examples of client input and the listener processing that results from them. The data fields referenced can be found in "IBM listener output format" on page 144.

**Note:** Parameters are separated by commas.

<table>
<thead>
<tr>
<th>Example</th>
<th>Listener response</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRN1,userdataishere</td>
<td>It starts the CICS transaction TRN1 using task control, and passes to it the data userdataishere in the field CLIENT-IN-DATA.</td>
</tr>
<tr>
<td>TRN2,,IC,000003</td>
<td>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.</td>
</tr>
</tbody>
</table>

---

9. See "Writing your own security or transaction link modules for the listener" on page 152

---

Chapter 6. Writing applications that use the IP CICS sockets API  143
Example | Listener response
--- | ---
TRN3,,userdataishere,TD | It writes a message to the transient data queue named TRN3 in the format described by the structure TCP_SOCKET-PARM, described in IBM listener output format. The data contained in userdataishere is passed to the field CLIENT-IN-DATA. This queue must be an intrapartition queue with trigger-level set to 1. It causes the initiation of transaction TRN3 if it is not already active. This transaction should be written to read the transient data queue and process requests until the queue is empty. This mechanism is provided for those server transactions that are used very frequently and for which the overhead of initiating a separate CICS transaction for each server request could be a performance concern.

TRN3,,TD | It causes data to be placed on transient data queue TRN3, which in turn causes the start or continued processing of the CICS transaction TRN3, as described in the TRN3 previous example. There is no user data passed.

TRN4 | It starts the CICS transaction TRN4 using task control. There is no user data passed.

### IBM listener output format

There are two different formats for the listener output; one for child server tasks started through a standard listener and one for child server tasks started through the enhanced listener.

**Guidelines:** The listener output format now supports an IPv6 socket address structure for both the standard and the enhanced listener. The size of the standard listener output format has increased. Child server programs should consider the following:

- A child server transaction program, using the EXEC CICS RETRIEVE function to get the data passed to it by the listener, should expand the storage it has allocated to contain the IPv6 socket address structure. The LENGTH specified on the EXEC CICS RETRIEVE function should reflect the amount of storage allocated to contain the listener output format. The LENGERR flag is raised if the LENGTH is smaller than the amount of data sent. Coding a HANDLE condition allows you to contain this.

- A child server transaction program, using the EXEC CICS READQ TD function to get the data placed on a CICS Transient Data Queue by the listener, should expand the storage it has allocated to contain the IPv6 socket address structure. The LENGTH specified on the EXEC CICS READQ TD function should reflect the amount of storage allocated to contain the listener output format.

Table 13 shows the format of the listener output data area passed to the child server through a standard listener.

#### Table 13. Listener output format - Standard listener

<table>
<thead>
<tr>
<th>Description</th>
<th>Offset</th>
<th>Format</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socket descriptor being given to the child subtask</td>
<td>0</td>
<td>Fullword binary</td>
<td>Socket number to be specified on the TAKESOCKET command by the child subtask</td>
</tr>
</tbody>
</table>
Table 13. Listener output format - Standard listener  (continued)

<table>
<thead>
<tr>
<th>Description</th>
<th>Offset</th>
<th>Format</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVS address space identifier</td>
<td>+4</td>
<td>8-byte character</td>
<td>Name of the listener’s address space</td>
</tr>
<tr>
<td>TCP/IP task identifier</td>
<td>+12</td>
<td>8-byte character</td>
<td>The listener’s task identifier</td>
</tr>
<tr>
<td>Data area</td>
<td>+20</td>
<td>35-byte character</td>
<td>Either the CLIENT-IN-DATA from the listener (if FORMAT is STANDARD) or the first 35 bytes data that was read by the listener (if FORMAT is ENHANCED)</td>
</tr>
<tr>
<td>OTE</td>
<td>+55</td>
<td>1-byte character</td>
<td>Indicates that the IP CICS socket interface is using CICS Open Transaction Environment.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 Using OTE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 Using MVS subtasks</td>
</tr>
<tr>
<td>Filler</td>
<td>+55</td>
<td>1-byte character</td>
<td>Unused byte for fullword alignment</td>
</tr>
<tr>
<td>Socket address structure</td>
<td>+56</td>
<td>28 bytes</td>
<td></td>
</tr>
<tr>
<td>Addressing family</td>
<td>+56</td>
<td>Halfword binary</td>
<td>Is 2 to indicate AF_INET or 19 to indicate AF_INET6</td>
</tr>
<tr>
<td>IPv4 portion of the socket address structure</td>
<td>+58</td>
<td>26 bytes</td>
<td>See the next three fields</td>
</tr>
<tr>
<td>Port number</td>
<td>+58</td>
<td>Halfword binary</td>
<td>The client's port number</td>
</tr>
<tr>
<td>32-bit IPv4 address</td>
<td>+60</td>
<td>Fullword binary</td>
<td>The IPv4 address of the client's host</td>
</tr>
<tr>
<td>Unused portion</td>
<td>+64</td>
<td>8 bytes</td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td>+72</td>
<td>12 bytes</td>
<td>For alignment with the IPv6 socket address structure</td>
</tr>
<tr>
<td>IPv6 portion of the socket address structure</td>
<td>+58</td>
<td>26 bytes</td>
<td>See the next four fields</td>
</tr>
<tr>
<td>Port number</td>
<td>+58</td>
<td>Halfword binary</td>
<td>The client's port number</td>
</tr>
<tr>
<td>Flow Information</td>
<td>+60</td>
<td>Fullword binary</td>
<td>Indicates traffic class and flow label</td>
</tr>
<tr>
<td>128-bit IPv6 address</td>
<td>+64</td>
<td>16 bytes</td>
<td>The IPv6 address of the client's host</td>
</tr>
<tr>
<td>Scope ID</td>
<td>+80</td>
<td>Fullword binary</td>
<td>Indicates link scope</td>
</tr>
<tr>
<td>Reserved</td>
<td>+84</td>
<td>17 fullwords</td>
<td>Reserved for future use</td>
</tr>
</tbody>
</table>

For a standard listener, the following COBOL definition is used:
01 TCPSOCKET-PARM.
  05 GIVE-TAKE-_SOCKET PIC 9(8) COMP.
  05 LSTN-NAME PIC X(8).
  05 LSTN-SUBNAME PIC X(8).
  05 CLIENT-IN-DATA PIC X(35).
  05 OTE PIC X(1).
  05 SOCKADDR-IN-PARM.
    10 SOCK-SIN REDEFINES SOCK-DATA.
      15 SOCK-SIN-PORT PIC 9(4) BINARY.
      15 SOCK-SIN-ADDR PIC 9(8) BINARY.
      15 FILLER PIC X(8).
      15 FILLER PIC X(12).
    10 SOCK-SIN6 REDEFINES SOCK-DATA.
      15 SOCK-SIN6-PORT PIC 9(4) BINARY.
      15 SOCK-SIN6-FLOWINFO PIC 9(8) BINARY.
      15 SOCK-SIN6-ADDR.
        20 FILLER PIC 9(16) BINARY.
        20 FILLER PIC 9(16) BINARY.
      15 SOCK-SIN6-SCOPEID PIC 9(8) BINARY.
    05 FILLER PIC X(68).

Figure 107. Example of COBOL layout of the listener output format - Standard listener

DCL 1 TCPSOCKET_PARM,
   2 GIVE_TAKE_SOCKET FIXED BIN(31),
   2 LSTN_NAME CHAR(8),
   2 LSTN_SUBNAME CHAR(8),
   2 CLIENT_IN_DATA CHAR(35),
   2 OTE CHAR(1),
   2 FILLER_1 CHAR(1),
   2 SOCK_FAMILY FIXED BIN(15),
   2 SOCK_SIN_PORT FIXED BIN(15),
   2 SOCK_SIN_ADDR FIXED BIN(31),
   2 SOCK_SIN_RESERVED CHAR(8),
   2 SOCK_SIN_FILLER CHAR(12),
   2 FILLER_68 CHAR(68);

Figure 108. Example of PL/I layout of the listener output format - Standard listener with an IPv4 socket address structure
DCL 1 TCPSOCKET_PARM,
  2 GIVE_TAKE_SOCKET FIXED BIN(31),
  2 LSTN_NAME CHAR(8),
  2 LSTN_SUBNAME CHAR(8),
  2 CLIENT_IN_DATA CHAR(35),
  2 OTE CHAR(1),
  2 SOCK_FAMILY FIXED BIN(15),
  2 SOCK_SIN6_PORT FIXED BIN(15),
  2 SOCK_SIN6_FLOWINFO FIXED BIN(31),
  2 SOCK_SIN6_ADDR CHAR(16),
  2 SOCK_SIN6_SCOPEID FIXED BIN(31),
  2 FILLER_68 CHAR(68);

Figure 109. Example of PL/I layout of the listener output format - Standard listener with an IPv6 socket address structure

TCPSOCKET_PARM DS OC
GIVE_TAKE_SOCKET DS F
LSTN_NAME DS CL8
LSTN_SUBNAME DS CL8
CLIENT_IN_DATA DS CL35
OTE DS CL1
SOCKADDR DS OF
SOCK_FAMILY DS H
SOCK_DATA DS OC
SOCK_ADDR EQU *-SOCKADDR
ORG SOCK_DATA
SOCK_SIN DS OC
SOCK_SIN_PORT DS H
SOCK_SIN_ADDR DS CL4
  DS CL8
  DS 20F
SOCK_SIN#LEN EQU *-SOCK_SIN
  ORG SOCK_DATA
SOCK_SIN6 DS OC
SOCK_SIN6_PORT DS H
SOCK_SIN6_FLOWINFO DS CL4
SOCK_SIN6_ADDR DS CL16
SOCK_SIN6_SCOPE_ID DS CL4
SOCK_SIN6#LEN EQU *-SOCK_SIN6
  ORG SOCK_DATA
DS CL68

Figure 110. Example of Assembler layout of the listener output format - Standard listener supporting both an IPv4 and an IPv6 socket address structure
Table 14 shows the format of the listener output data area passed to the child server through the enhanced listener.

**Note:** With the enhanced listener, no CLIENT-IN-DATA is extracted from the initial client data. The child server program must either read the initial client data itself (if PEEKDATA is YES) or obtain it from DATA-AREA-2 (if PEEKDATA is NO). If a listener is converted from a standard listener to an enhanced listener, its corresponding child server applications must be changed to handle the larger transaction initial message (TIM) by specifying a large enough length on the EXEC CICS RETRIEVE command or on the EXEC CICS READQ TD command. Otherwise, the command fails with a LENGERR response and the child server task could abend.

**Table 14. Listener output format - Enhanced listener**

<table>
<thead>
<tr>
<th>Description</th>
<th>Offset</th>
<th>Format</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socket descriptor being given to the child subtask</td>
<td>0</td>
<td>Fullword binary</td>
<td>Socket number to be specified on the TAKESOCKET command by the child subtask</td>
</tr>
<tr>
<td>MVS address space identifier</td>
<td>+4</td>
<td>8-byte character</td>
<td>Name of the listener’s address space</td>
</tr>
<tr>
<td>TCP/IP task identifier</td>
<td>+12</td>
<td>8-byte character</td>
<td>The listener’s task identifier</td>
</tr>
<tr>
<td>Data area</td>
<td>+20</td>
<td>35-byte character</td>
<td>Either the CLIENT-IN-DATA from listener (if FORMAT is STANDARD) or the first 35 bytes of data read by the listener (if FORMAT is ENHANCED)</td>
</tr>
<tr>
<td>OTE</td>
<td>+55</td>
<td>1-byte character</td>
<td>Indicates that the IP CICS socket interface is using CICS’s Open Transaction Environment.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 Using OTE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 Using MVS subtasks</td>
</tr>
<tr>
<td>Socket address structure</td>
<td>+56</td>
<td>28 bytes</td>
<td></td>
</tr>
<tr>
<td>Addressing family</td>
<td>+56</td>
<td>Halfword binary</td>
<td>Is 2 to indicate AF_INET or 19 to indicate AF_INET6</td>
</tr>
<tr>
<td>IPv4 portion of the socket address structure</td>
<td>+58</td>
<td>26 bytes</td>
<td>See the next three fields</td>
</tr>
</tbody>
</table>

Figure 111. Example of C structure of the listener output format - Standard listener supporting both an IPv4 and an IPv6 socket address structure
Table 14. Listener output format - Enhanced listener (continued)

<table>
<thead>
<tr>
<th>Description</th>
<th>Offset</th>
<th>Format</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port number</td>
<td>+58</td>
<td>Halfword binary</td>
<td>The client's port number</td>
</tr>
<tr>
<td>32-bit IPv4 address</td>
<td>+60</td>
<td>Fullword binary</td>
<td>The IPv4 address of the client's host</td>
</tr>
<tr>
<td>Unused portion</td>
<td>+64</td>
<td>8 bytes</td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td>+72</td>
<td>12 bytes</td>
<td>For alignment with the IPv6 socket address structure</td>
</tr>
<tr>
<td>IPv6 portion of the socket address structure</td>
<td>+58</td>
<td>26 bytes</td>
<td>See the next four fields</td>
</tr>
<tr>
<td>Port number</td>
<td>+58</td>
<td>Halfword binary</td>
<td>The client's port number</td>
</tr>
<tr>
<td>Flow Information</td>
<td>+60</td>
<td>Fullword binary</td>
<td>Indicates traffic class and flow label</td>
</tr>
<tr>
<td>128-bit IPv6 address</td>
<td>+64</td>
<td>16 bytes</td>
<td>The IPv6 address of the client's host</td>
</tr>
<tr>
<td>Scope ID</td>
<td>+80</td>
<td>Fullword binary</td>
<td>Indicates link scope</td>
</tr>
<tr>
<td>Reserved</td>
<td>+84</td>
<td>17 fullwords</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>Data length</td>
<td>+152</td>
<td>Halfword binary</td>
<td>The length of the data received from the client. If the PEEKDATA option was configured, Data length is zero with no data in Data area-2.</td>
</tr>
<tr>
<td>Data area - 2</td>
<td>+154</td>
<td>Length determined by the previous field</td>
<td>The data received from the client starting at position 1</td>
</tr>
</tbody>
</table>

For the enhanced listener, the following COBOL definition is used:
The value of xxx is at least equal to the largest MSGLENgth parameter for the listeners that can start this application.

Figure 112. Example of COBOL layout of the listener output format - Enhanced listener

```
01 TCPSOCKET-PARM.
  05 GIVE-TAKE-SOCKET PIC 9(8) COMP.
  05 LSTN-NAME PIC X(8).
  05 LSTN-SUBNAME PIC X(8).
  05 CLIENT-IN-DATA PIC X(35).
  05 OTE PIC X(1).
  05 SOCKADDR-IN-PARM.
    10 SOCK-FAMILY PIC 9(4) BINARY.
    10 SOCK-DATA PIC X(26).
    10 SOCK-SIN REDEFINES SOCK-DATA.
      15 SOCK-SIN-PORT PIC 9(4) BINARY.
      15 SOCK-SIN-ADDR PIC 9(8) BINARY.
      15 FILLER PIC X(8).
      15 FILLER PIC X(12).
    10 SOCK-SIN6 REDEFINES SOCK-DATA.
      15 SOCK-SIN6-PORT PIC 9(4) BINARY.
      15 SOCK-SIN6-FLOWINFO PIC 9(8) BINARY.
      15 SOCK-SIN6-ADDR.
      20 FILLER PIC 9(16) BINARY.
      20 FILLER PIC 9(16) BINARY.
      15 SOCK-SIN6-SCOPEID PIC 9(8) BINARY.
    05 FILLER PIC X(68).
    05 CLIENT-IN-DATA-LENGTH PIC 9(4) BINARY.
    05 CLIENT-IN-DATA_2 PIC X(xxx).
```

Figure 113. Example of PL/I layout of the listener output format - Enhanced listener with an IPv4 socket address structure

```
DCL 1 TCPSOCKET_PARM,
  2 GIVE_TAKE_SOCKET FIXED BIN(31),
  2 LSTN_NAME CHAR(8),
  2 LSTN_SUBNAME CHAR(8),
  2 CLIENT_IN_DATA CHAR(35),
  2 OTE CHAR(1),
  2 SOCK_FAMILY FIXED BIN(15),
  2 SOCK_SIN_PORT FIXED BIN(15),
  2 SOCK_SIN_ADDR FIXED BIN(31),
  2 SOCK_SIN_RESERVED CHAR(8),
  2 SOCK_SIN_FILLER CHAR(12),
  2 FILLER_68 CHAR(68),
  2 CLIENT_IN_DATA_LENGTH FIXED BIN(15),
  2 CLIENT_IN_DATA_2 CHAR(xxx);"
The value of xxx is at least equal to the largest MSGLENgth parameter for the listeners that can start this application.

Figure 114. Example of PL/I layout of the listener output format - Enhanced listener with an IPv6 socket address structure

Figure 115. Example of assembler layout of the listener output format - Enhanced listener supporting both an IPv4 and an IPv6 socket address structure
The value of \textit{xxx} is at least equal to the largest MSGLEN\textit{g}th parameter for the listeners that can start this application.

**Writing your own security or transaction link modules for the listener**

The listener process provides an exit point for those users who want to write and include a module that performs the following:

- Check to indicate whether the expanded security or transaction input format is used
- Security check before a CICS transaction is initiated

The exit point is implemented so that if a module is not provided, all valid transactions are initiated.

If you write a security or transaction module, you can name it anything you want, as long as you define it in the configuration data set. In previous releases, you needed to name the module EZACICSE; you can still use that module name. You can write this program in COBOL, PL/I, or assembler language, and you must provide an appropriate CICS program definition.

**Note:** Specify the name of the security or transaction module in the SECEXIT field in Alter or Define. If you do not name the module, CICS assumes you do not have this module. See Figure 63 on page 83 for more information about this process.

Just before the child server task creation process, the listener invokes the security or transaction module by a conditional CICS LINK passing a COMMAREA. The listener passes a data area to the module that contains information for the module to use for security checking and a 1-byte switch. Your security or transaction module should perform a security check and set the switch accordingly. Included in this data is the OTE indicator which indicates when the IP CICS socket interface is using CICS’s open transaction environment. The security exit should follow threadsafe programming practices to ensure that CICS continues to execute the listener on an open API TCB.

When the security or transaction module returns, the listener checks the state of the switch and initiates the transaction if the switch indicates security clearance. The module can perform any function that is valid in the CICS environment. Excessive processing, however, could cause performance degradation.
A field is supplied to indicate if the expanded security or transaction input format is used. If used, fields also exist for the listener’s IP address and port number, a data length field, and a second data area (up to MSGLENTH in length). Table 15 shows the data area used by the security or transaction module.

**Table 15. security or transaction exit data**

<table>
<thead>
<tr>
<th>Description</th>
<th>Offset</th>
<th>Format</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CICS transaction identifier</td>
<td>0</td>
<td>4-byte character</td>
<td>CICS transaction requested by the client or supplied by the CSTRANID parameter.</td>
</tr>
<tr>
<td>Data area</td>
<td>+4</td>
<td>35-byte character</td>
<td>If the FORMAT parameter value is STANDARD, then this contains the 35-byte application data that was extracted from the client’s initial data. Otherwise, it contains up to the first 35 bytes of data sent by the client (The MSGLENTH value determines the limit).</td>
</tr>
<tr>
<td>security or transaction exit data</td>
<td>+39</td>
<td>1-byte character</td>
<td>Indicates whether or not this data area is in the expanded format: 1 Expanded format (the area in green is included) 0 Not expanded (the area in green is not included)</td>
</tr>
<tr>
<td>OTE indicator</td>
<td>+40</td>
<td>1-byte character</td>
<td>Indicates whether the IP CICS socket interface is using CICS’s open transaction environment. 1 Using OTE 0 Using MVS subtasks</td>
</tr>
<tr>
<td>TTLS indicator</td>
<td>+41</td>
<td>1-byte character</td>
<td>Indicates whether this connection is secured using AT-TLS. 1 This connection is secured using AT-TLS 0 This connection is not secured using AT-TLS</td>
</tr>
<tr>
<td>Register Application Data</td>
<td>+42</td>
<td>1-byte character</td>
<td>Indicates that application data is registered against the accepted connection to be given. This flag has the value 1 when either the LAPPLD value is yes or the LAPPLD parameter inherited the APPLDAT=YES specification. 1 Application data is registered 0 Application data is not registered</td>
</tr>
<tr>
<td>Reserved</td>
<td>+43</td>
<td>1-byte character</td>
<td>Reserved for IBM use.</td>
</tr>
<tr>
<td>Description</td>
<td>Offset</td>
<td>Format</td>
<td>Value</td>
</tr>
<tr>
<td>------------------------------</td>
<td>--------</td>
<td>-------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>Action</td>
<td>+44</td>
<td>2-byte character</td>
<td>Method of starting the task:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IC    Interval control</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>KC    Task control</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TD    Transient data</td>
</tr>
<tr>
<td>Interval control time</td>
<td>+46</td>
<td>6-byte character</td>
<td>Interval requested for IC start. Has the form hhmmss.</td>
</tr>
<tr>
<td>Address family</td>
<td>+52</td>
<td>Halfword binary</td>
<td>Network address family. The value contains a 2 to indicate AF_INET and a 19 to indicate AF_INET6.</td>
</tr>
<tr>
<td>Client's port</td>
<td>+54</td>
<td>Halfword binary</td>
<td>The number of the requestor's port.</td>
</tr>
<tr>
<td>Client's IPv4 address</td>
<td>+56</td>
<td>Fullword binary</td>
<td>The IPv4 address of the requestor's host.</td>
</tr>
<tr>
<td>Switch</td>
<td>+60</td>
<td>1-byte character</td>
<td>1    Permit the transaction</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Not 1 Prohibit the transaction</td>
</tr>
<tr>
<td>Switch-2</td>
<td>+61</td>
<td>1-byte character</td>
<td>1    Listener sends message to the client</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Not 1 security or transaction exit sends message to client</td>
</tr>
<tr>
<td>Terminal identification</td>
<td>+62</td>
<td>4-byte character</td>
<td>Return binary zeroes if no terminal is to be associated with the new task. Otherwise, return the CICS terminal ID to be associated with the new task.</td>
</tr>
<tr>
<td>Socket descriptor</td>
<td>+66</td>
<td>Halfword binary</td>
<td>Current socket descriptor.</td>
</tr>
</tbody>
</table>
Table 15. security or transaction exit data (continued)

<table>
<thead>
<tr>
<th>Description</th>
<th>Offset</th>
<th>Format</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>User ID</td>
<td>+68</td>
<td>8-byte character</td>
<td>A user ID can be returned so that it is associated with the new task. This is mutually exclusive from terminal ID.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• If the GETTID value is YES in the listener definition and the listener is able to obtain the user ID that is associated with the connection client’s certificate, then this field is initialized using that user ID. Otherwise, it is initialized as binary zeroes. The security exit can use that user ID to identify the client.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• If the security exit permits the transaction and does not overwrite this field, then the child server task inherits this user ID (unless the start type is TD).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• If the security exit overwrites this field with nulls or blanks, then the child server inherits the listener task’s user ID (unless the start type is TD).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• If the security exit overwrites this field with another user ID, then the child server task inherits that user ID (unless the start type is TD). The user ID under which the listener executes must have RACF surrogate authority to use any user ID that can be specified by this field.</td>
</tr>
<tr>
<td>Listener’s IPv4 address</td>
<td>+76</td>
<td>Fullword binary</td>
<td>The local IPv4 address associated with this new TCP/IP connection.</td>
</tr>
<tr>
<td>Listener’s port</td>
<td>+80</td>
<td>Halfword binary</td>
<td>The listener’s port number.</td>
</tr>
<tr>
<td>Listener’s IPv6 address</td>
<td>+82</td>
<td>16 bytes binary</td>
<td>The local IPv6 address associated with this new TCP/IP connection.</td>
</tr>
<tr>
<td>Listener’s scope ID</td>
<td>+98</td>
<td>Fullword binary</td>
<td>The scope ID of the listener’s IPv6 address.</td>
</tr>
<tr>
<td>Client’s IPv6 address</td>
<td>+102</td>
<td>16 bytes binary</td>
<td>The IPv6 address of the requestor’s host.</td>
</tr>
<tr>
<td>Client’s scope ID</td>
<td>+118</td>
<td>Fullword binary</td>
<td>The scope ID of the listener’s IPv6 address.</td>
</tr>
</tbody>
</table>
Table 15. security or transaction exit data (continued)

<table>
<thead>
<tr>
<th>Description</th>
<th>Offset</th>
<th>Format</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client’s certificate length</td>
<td>+122</td>
<td>Halfword binary</td>
<td>Indicates whether the client’s certificate exists.</td>
</tr>
<tr>
<td>Client’s certificate address</td>
<td>+124</td>
<td>Fullword binary</td>
<td>The address of the client’s certificate.</td>
</tr>
<tr>
<td>Reserved</td>
<td>+128</td>
<td>34 bytes</td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>Data length</td>
<td>+162</td>
<td>Halfword binary</td>
<td>The length of the data received from the client.</td>
</tr>
<tr>
<td>Data area - 2</td>
<td>+164</td>
<td>Length determined by the previous field</td>
<td>The data received from the client starting at position 1. If this is the enhanced listener, the first 35 bytes are the same as Data Area-1.</td>
</tr>
</tbody>
</table>

Note:

1. The security/user exit can change the value of the following fields:
   - CICS transaction identifier
   - Data area
   - Action
   - Register Application Data
   - Interval control time
   - Address family
   - Client’s port
   - Client’s IPv4 address
   - Switch
   - Terminal identification (output only)
   - User ID
   - Client’s IPv6 address
   - Client’s Scope ID
   - Data length
   - Data area -2

2. Although the security exit can alter the contents of the Data area, Data length, and Data area -2 fields when PEEK=YES, the changed values are not reflected to the child server in the listener input data. The child server must read the data itself if the listener is configured with PEEK=YES.

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

Threadsafe considerations for IP CICS sockets applications

This topic describes how to enable IP CICS sockets applications to exploit the Open Transaction Environment (OTE) through threadsafe programming.

The IP CICS socket interface includes the IP CICS sockets task-related user exit, EZACIC01, which is invoked when an application program makes an EZASOKET request. This includes the following programs:

- EZASOKET
- EZACICSO
• EZACICAL
• using any of the IP CICS C sockets functions that are provided through
  EZACIC17 (Programs using IP CICS sockets functions that are provided through
  EZACIC07 are not considered threadsafe due to not being re-entrant.)

The IP CICS socket interface manages the process of transferring to TCP/IP and
returning control to the application program when EZASOKET processing is
complete.

When the IP CICS sockets configuration option is specified as OTE=NO, then the
IP CICS sockets task-related user exit operates as a quasi-reentrant task-related
user exit program. It runs on the CICS main TCB (the QR TCB) and uses its own
MVS subtask TCB to process the EZASOKET request. However, when the IP CICS
sockets configuration option is specified as OTE=YES, then the IP CICS socket
interface exploits the Open Transaction Environment (OTE) to enable the IP CICS
sockets task-related user exit to invoke and return from TCP/IP without switching
TCBs. In the OTE, the IP CICS sockets task-related user exit operates as a
threadsafe and open API task-related user exit program; it is automatically enabled
using the OPENAPI option on the ENABLE PROGRAM command during
connection processing. This enables it to receive control on an open L8 mode TCB.

In the OTE, if the user application program that invoked the task-related user exit
conforms to threadsafe coding conventions and is defined to CICS as threadsafe, it
can also run on the L8 TCB. Before its first EZASOKET request, the application
program runs on the CICS main TCB, the QR TCB. When it makes an EZASOKET
request and invokes the task-related user exit, control passes to the L8 TCB, and IP
CICS sockets processing is carried out. On return from TCP/IP, if the application
program is threadsafe, it continues to run on the L8 TCB.

When the correct conditions are met, the use of open TCBs for IP CICS sockets
applications decreases usage of the QR TCB, and avoids TCB switching. An ideal
IP CICS sockets application program for the open transaction environment is a
threadsafe program, containing only threadsafe EXEC CICS commands, and using
only threadsafe user exit programs. An application like this moves to an L8 TCB
when it makes its first EZASOKET request, and then continues to run on an L8
TCB through any amount of IP CICS sockets requests and application code,
requiring no TCB switching. This situation produces a significant performance
improvement where an application program issues multiple EZASOKET calls. The
gains are also significant when making a DB2 request because the DB2 task-related
user exit also operates as threadsafe and exploits the open transaction
environment. If the application program does not issue many EZASOKET calls, the
performance benefits might not be as significant.

If the execution of a user application involves any actions that are not threadsafe,
CICS switches back to the QR TCB. Such actions are non-threadsafe CICS requests
issued by the program, the use of non-threadsafe task-related user exits, and the
involvement of non-threadsafe global user exits. Switching back and forth between
the open TCB and the QR TCB is detrimental to the application’s performance.

Requirements: In order to gain the performance benefits of the OTE for IP CICS
sockets applications, you must meet the following conditions:

• IP CICS sockets must be configured to use the Open Transaction Environment
  with the OTE=YES configuration option.
• The system initialization parameter FORCEQR must be set to NO. FORCEQR
  forces programs defined as threadsafe to run on the QR TCB; it can be set to
YES as a temporary measure while problems connected with threadsafe-defined programs are investigated and resolved. FORCEQR applies to all programs defined as threadsafe that are not invoked as task-related user exits, global user exits, or user-replaceable modules.

- The IP CICS sockets application must have threadsafe application logic (that is, the native language code in between the EXEC CICS commands must be threadsafe), use only threadsafe EXEC CICS commands, and be defined to CICS as threadsafe. Only code that has been identified as threadsafe is permitted to execute on open TCBs. If your IP CICS sockets application is not defined as threadsafe, or if it uses EXEC CICS commands that are not threadsafe, TCB switching occurs and some or all of the performance benefits of OTE exploitation are lost. If your IP CICS sockets application is defined as threadsafe and it contains non-threadsafe code between the EXEC CICS commands, unpredictable results can occur.

- Any global user exits on the execution path used by the application must be coded to threadsafe standards and defined to CICS as threadsafe.

- Any other task-related user exits used by the application must be defined to CICS as threadsafe or enabled as OPENAPI.

See [http://www-01.ibm.com/software/htp/cics/library/](http://www-01.ibm.com/software/htp/cics/library/) for information about how to make application programs and user exit programs threadsafe. By defining a program to CICS as threadsafe, you are specifying that only the application logic is threadsafe, not that all the EXEC CICS commands included in the program are threadsafe. CICS can ensure that EXEC CICS commands are processed safely by switching to the QR TCB for those commands not yet converted that must be quasi-reentrant. To permit your program to run on an open TCB, CICS requires you to verify that your application logic is threadsafe.


If a user application program in the open transaction environment is not threadsafe, the IP CICS sockets task-related user exit still runs on an L8 TCB, but the application program runs on the QR TCB throughout the task. Every time the program makes an EZASOKET request, CICS switches from the QR TCB to the L8 TCB and back again, so the performance benefits of the open transaction environment are negated.

Table 16 shows what happens when application programs with different concurrency attributes invoke the IP CICS sockets task-related user exit.

<table>
<thead>
<tr>
<th>Program's concurrency attribute</th>
<th>IP CICS sockets task-related user exit's operation</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUASIRENT or THREADSAFE</td>
<td>Quasi-reentrant when OTE=NO</td>
<td>Application program and task-related user exit run under the CICS QR TCB. The task-related user exit manages its own TCBs, switching to and from them for each EZASOKET request.</td>
</tr>
</tbody>
</table>
Table 16. Different concurrency attributes for IP CICS sockets task-related user exits (continued)

<table>
<thead>
<tr>
<th>Program's concurrency attribute</th>
<th>IP CICS sockets task-related user exit's operation</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUASIRENT</td>
<td>Threadsafe and open API (when OTE=YES)</td>
<td>Application program runs under the CICS QR TCB. Task-related user exit runs under an L8 TCB, and EZASOKET calls are executed under the L8 TCB. CICS switches to and from the CICS QR and the L8 TCB for each EZASOKET call.</td>
</tr>
<tr>
<td>THREADSAFE</td>
<td>Threadsafe and open API (when OTE=YES)</td>
<td>OTE exploitation. Task-related user exit runs under an open API, L8 TCB, and EZASOKET calls are executed under the open API, L8, TCB. The application program also runs on the open API, L8, TCB when control is returned to it. No TCB switches are needed until the task terminates, or the program issues a non-threadsafe CICS command, which forces a switch back to the QR TCB for CICS to ensure resource integrity.</td>
</tr>
</tbody>
</table>

If you define a program with CONCURRENCY(THREADSAFE), then all routines that are statically or dynamically called from that program (for example, COBOL routines) must also be coded to threadsafe standards.

When an EXEC CICS LINK command is used to link from one program to another, the program link stack level is incremented. However, a routine that is statically called, or dynamically called, does not involve passing through the CICS command level interface, and does not cause the program link stack level to be incremented. With COBOL routines, for a static call, a simple branch and link is used when an address is resolved by the Linkage Editor. For a dynamic call, although there is a program definition involved, this is required only so Language Environment® can load the program. After the load, a simple branch and link is executed. When a routine is called by either of these methods, CICS does not regard this as a change of program. The program that called the routine is still considered to be executing, and the program definition for that program is still considered to be the current one.

If the program definition for the calling program states CONCURRENCY(THREADSAFE), then the called routine must also comply with this specification. Programs with the CONCURRENCY(THREADSAFE) attribute remain on an open API TCB until they return from a EZASOKET call, and this is not appropriate for a program that is not threadsafe. For example, consider the situation where the initial program of a transaction, program A, issues a dynamic call to program B, which is a COBOL routine. Because the CICS command level
interface was not involved, CICS is unaware of the call to program B, and considers the current program to be program A. Program B further issues a EZASOKET call. On return from the EZASOKET call, CICS needs to determine whether the program can remain on the open API TCB, or whether the program must switch back to the QR TCB to ensure thread-safe processing. To do this, CICS examines the CONCURRENCY attribute of what it considers to be the current program, which is program A. If program A is defined as CONCURRENCY(THREADSAFE), then CICS allows processing to continue on the open API TCB. In fact program B is executing, so if processing is to continue safely, program B must be coded to thread-safe standards.

In summary, to gain the performance benefits of the open transaction environment:

1. IP CICS sockets must be configured to use the open transaction environment by the use of the OTE=YES configuration option.
2. FORCEQR must be set to NO.
3. The IP CICS sockets application must have thread-safe application logic (that is, the native language code in between the EXEC CICS commands must be thread-safe), use only thread-safe EXEC CICS commands, and be defined to CICS as thread-safe. If the application program is not defined as thread-safe, and so must operate on the CICS QR TCB, TCB switching occurs for every EZASOKET request, even if the task-related user exit is running on an open TCB. If the application program is defined as thread-safe but uses non-thread-safe EXEC CICS commands, TCB switching occurs for every non-thread-safe EXEC CICS commands.
4. The IP CICS sockets application must use only thread-safe task-related user exits and global user exits. If any non-thread-safe exits are used, this forces a switch back to the QR TCB. If application programs are defined to CICS as CONCURRENCY(THREADSAFE) and they contain non-thread-safe code, unpredictable results can occur.

How CICS selects an L8 mode TCB

The CICS dispatcher manages the pool of L8 mode TCBs up to the limit set by the MAXOPENTCBS system initialization parameter. At any one time, the pool can consist of some TCBs that are allocated to tasks, and others that are free. For example, if the maximum number of L8 mode TCBs is set to 10, at a particular time the pool can consist of 5 TCBs, not all of which are allocated to running tasks. The CICS dispatcher attaches a new TCB when it cannot find a free TCB that is suitable. The process of allocating an L8 mode TCB is summarized in the following steps:

1. If the transaction already has an L8 mode TCB allocated, it is used.
2. If there is a free L8 mode TCB for the current subspace, it is allocated and used.
3. If the number of open TCBs is less than the MAXOPENTCBS limit, a new L8 mode TCB is created, and associated with the task's subspace.
4. If the number of open TCBs is at the MAXOPENTCBS limit and there is a free L8 mode TCB with the wrong subspace, then the CICS dispatcher destroys it and creates a new one for the required subspace. This technique avoids suspending the task until the number of TCBs is less than the pool limit, and is called stealing. This action is recorded in the CICS dispatcher TCB mode statistics under the count of TCB steals.
5. If the number of open TCBs is at the MAXOPENTCBS limit and there is no free open TCB to steal, the task is suspended (with an OPENPOOL wait) until one becomes free, or the MAXOPENTCBS limit is increased.
The various events that can occur during the TCB allocation process are recorded in the dispatcher TCB pool statistics, and these are reported by the DFH0STAT statistics program.

Data conversion routines

CICS uses the EBCDIC data format, whereas TCP/IP networks use ASCII. When moving data between CICS and the TCP/IP network, your application programs must initiate the necessary data conversion. Sockets for CICS programs can use routines provided by TCP/IP Services for:

- Converting data from EBCDIC to ASCII and back (when sending and receiving data to and from the TCP/IP network) with the SEND, SENDMSG, SENDTO, READ, READV, RECV, RECVFROM, RECVMSG, WRITE, and WRITEV calls.
- Converting between bit arrays and character strings when using the SELECT or SELECTEX call.

For details of these routines, see EZACIC04, EZACIC05, and EZACIC06, EZACIC14, and EZACIC15 in Chapter 8, “Sockets extended API,” on page 249.

Application Transparent Transport Layer Security

Before reading this topic, first read the Application Transparent Transport Layer Security (AT-TLS) topic of the z/OS Communications Server: IP Configuration Guide.

The z/OS Communications Server TCP/IP stack provides Application Transparent Transport Layer Security (AT-TLS). This allows socket applications that use the TCP protocol to transparently use the Secure Socket Layer protocol (TLS/SSL) to communicate with partners in the network. IP CICS sockets enabled applications can take advantage of this support. This requires the following:

- The TCP/IP stack must support AT-TLS. This can be determined by the TTLS parameter on the TCPCONFIG statement.
- An AT-TLS Policy configuration that matches identifiers of the CICS applications that use it. Examples of identifiers that can be used are whether the application is a listener or client, the IP addresses, and the ports that are used for communication. Note that for CICS applications, the AT-TLS identity associated with the AT-TLS environment is always the user ID of the CICS region. This is the case even if individual CICS transactions are running under their own identity.
- SSL key rings and certificates must be created for these applications. For CICS applications using SSL, the user ID that is associated with the keyring is that of the CICS region. See the z/OS Communications Server: IP Configuration Guide for the RACF commands necessary for creating SSL keyrings and certificates. See the z/OS Security Server RACF Security Administrator’s Guide for more information about setting up and managing digital certificates.
- For policy level or application level (such as GETTID) support that requires mapping SSL Certificates to RACF user IDs see the z/OS Communications Server: IP Configuration Guide for more information.

Careful consideration must be given for IP CICS sockets-enabled applications that act as clients connecting outbound because the AT-TLS policy might not be specific enough to restrict individual CICS users from logging on to and invoking these clients. Additional CICS security controls such as transaction security and resource...
security can be considered in order to limit users’ access to remote hosts. See “Example of outbound AT-TLS support” on page 163 for more information.

If a CICS listener is AT-TLS enabled but the client does not use SSL, there is a mismatch; AT-TLS receives unencrypted data when it is expecting encrypted data. In this case, AT-TLS resets the connection. See the Application Transparent Transport Layer Security (AT-TLS) topic in the z/OS Communications Server: IP Configuration Guide for information regarding defining keyrings, client certificates, mapping them to user IDs, permitting users access to keyrings, and other AT-TLS details.

When taking advantage of AT-TLS support, CICS application programmers and TCP/IP administrators must work together to provide the required support. This can also require communication with RACF administrators.

Example of inbound AT-TLS support

No inbound AT-TLS support is needed for listener port 3010, inbound AT-TLS support needed for listener port 3011.

Table 17. Inbound AT-TLS support

<table>
<thead>
<tr>
<th>AT-TLS Definitions</th>
<th>CICS listener Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTLRule CSKLrule</td>
<td>TRANID ====&gt; CSKL</td>
</tr>
<tr>
<td></td>
<td>PORT ====&gt; 03010</td>
</tr>
<tr>
<td></td>
<td>GETTID ====&gt; NO</td>
</tr>
<tr>
<td></td>
<td>TRANID ====&gt; CSKM</td>
</tr>
<tr>
<td></td>
<td>PORT ====&gt; 03011</td>
</tr>
<tr>
<td></td>
<td>GETTID ====&gt; YES</td>
</tr>
<tr>
<td></td>
<td>LocalPortRange 3010</td>
</tr>
<tr>
<td></td>
<td>Direction Inbound</td>
</tr>
<tr>
<td></td>
<td>TTLSGlobalActionRef NOTTLSGR</td>
</tr>
<tr>
<td></td>
<td>TTLSEnabled OFF</td>
</tr>
<tr>
<td></td>
<td>TTLSGlobalAction NOTTLSGR</td>
</tr>
<tr>
<td></td>
<td>TTLSEnabled OFF</td>
</tr>
<tr>
<td></td>
<td>TTLSGlobalRule CSKMrule</td>
</tr>
<tr>
<td></td>
<td>LocalPortRange 3011</td>
</tr>
<tr>
<td></td>
<td>Direction Inbound</td>
</tr>
<tr>
<td></td>
<td>TTLSGlobalActionRef TTLSGRP1</td>
</tr>
<tr>
<td></td>
<td>TTLSEnvironmentActionRef TTLSENVI</td>
</tr>
<tr>
<td></td>
<td>TTLSEnvironmentAction TTLSENVI</td>
</tr>
<tr>
<td></td>
<td>HandshakeRole ServerWithClientAuth</td>
</tr>
<tr>
<td></td>
<td>EnvironmentUserInstance 1</td>
</tr>
<tr>
<td></td>
<td>TTLSEnvironmentAdvancedParmsRef TTLSADV1</td>
</tr>
<tr>
<td></td>
<td>TTLSEnvironmentAdvancedParms TTLSADV1</td>
</tr>
<tr>
<td></td>
<td>ClientAuthType SAFcheck</td>
</tr>
<tr>
<td></td>
<td>TTLSGlobalAction TTLSGRP1</td>
</tr>
<tr>
<td></td>
<td>TTLSEnabled ON</td>
</tr>
</tbody>
</table>
Example of outbound AT-TLS support

No outbound AT-TLS support is needed for remote port 3010, outbound AT-TLS support needed for remote port 3011

Table 18. Outbound AT-TLS support

<table>
<thead>
<tr>
<th>AT-TLS Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTLSRule ClientRule1</td>
</tr>
<tr>
<td>RemotePortRange 3010</td>
</tr>
<tr>
<td>UserId CICS1</td>
</tr>
<tr>
<td>Direction Outbound</td>
</tr>
<tr>
<td>TTLSGroupActionRef NOTTLSGR</td>
</tr>
<tr>
<td>TTLSGroupAction NOTTLSGR</td>
</tr>
<tr>
<td>TTLSEnabled OFF</td>
</tr>
<tr>
<td>TTLSRule ClientRule2</td>
</tr>
<tr>
<td>RemotePortRange 3011</td>
</tr>
<tr>
<td>Direction Outbound</td>
</tr>
<tr>
<td>TTLSGroupActionRef TTLSGRP2</td>
</tr>
<tr>
<td>TTLSEnvironmentActionRef TTLSENV2</td>
</tr>
<tr>
<td>TTLSEnvironmentAction TTLSENV2</td>
</tr>
<tr>
<td>HandshakeRole Client</td>
</tr>
<tr>
<td>EnvironmentUserInstance 1</td>
</tr>
<tr>
<td>TTLSGroupAction TTLSGRP2</td>
</tr>
<tr>
<td>TTLSEnabled ON</td>
</tr>
</tbody>
</table>
Chapter 7. C language application programming

This topic describes the C language API provided by CICS TCP/IP and contain the following topics:

- **“C socket library”** lists the required header files and explains how to make them available to your programs.
- **“C socket compilation” on page 166** shows how to compile a C socket program that contains calls to sockets for CICS.
- **“Structures used in socket calls” on page 167** lists data structures used in C language socket calls.
- **“The ERRNO variable” on page 170** describes the use of a global variable used by the socket system to report errors.
- **“C socket call guidance” on page 170** 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.
- **“Address Testing Macros” on page 245** describes the macros that is used to test special IPv6 addresses.

C socket library

To use the socket routines described in this topic, you must include these header files:

```
bsdt ime.h  
bsdtype s.h  
cmanifes.h (reentrant programs only)  
erro h (reentrant programs only)  
ezacichd.h (non-reentrant programs only)  
ezbpinfc.h (if using the SIOCGPARTNERINO or SIOCSPARTNERINFO IOCTL calls)  
ezbzt1sc.h (if using IOCTL calls related to AT-TLS)  
fcntl1.h  
if.h  
in.h  
inet.h  
ioctl.h  
manifest.h (non-reentrant programs only)  
netd b.h  
rtrouteh.h  
socket.h  
uio.h
```

The files are in the SEZACMAC, SEZAINST, and SEZANMAC data sets, which must be concatenated to the SYSLIB DD in the compilation JCL (as described in Step 1 of “Changes to DFHYITDL” on page 166). These files contain a .h extension in this text to distinguish them as header files.

In the IBM implementation, you must include either `manifest.h` (if the program is non-reentrant) or `cmanifes.h` (if the program is reentrant) to remap function long names to 8-character names. To reference `manifest.h` or `cmanifes.h`, you need to include one of the following statements as the first `#include` at the beginning of each program:
Include the following definition to expose the required IPv6 structures, macros, and definitions in the header files in "C socket library" on page 165:

```c
#define __CICS_IPV6
```

Include the following definition to expose structures, macros and definitions in the TCP C header files previously listed:

```c
#define __CICS_SOCKETS
```

Include the in.h header before the socket.h header because the socket.h header needs structure types that are defined by in.h.

---

**C socket compilation**

To compile a C socket program that contains calls to CICS TCP/IP, you must change the standard procedure for C socket compilation that is provided with CICS. The CICS sample compile procedures are in SDFHSAMP. To compile a C sockets program, modify the DFHYITDL procedure to the version of CICS and the C Compiler that you have installed on your system.

**Restriction:** The IP CICS C sockets API does not support C++ programs.


**Changes to DFHYITDL**

1. In the C step (running the C socket compiler) you must concatenate the SEZACMAC, SEZAINST, and SEZANMAC data sets to the SYSLIB DD.

2. In the PLKED step you must concatenate the SEZARNT1 data set to the SYSLIB DD if and only if the program is to be compiled as reentrant (that is, with the RENT option). Ensure that the system administrator has performed the actions listed for Program Reentrancy in [Restrictions for Using MVS TCP/IP API with z/OS Unix](https://www.ibm.com/support/docview.wss?uid=swg27015818) in [z/OS XL C/C++ Programming Guide](https://www.ibm.com/support/docview.wss?uid=swg27015818).

3. In the LKED step you must concatenate the SEZATCP and SEZACMTX data sets to the SYSLIB DD.

**Compile your program**

```c
//PROCJOB
//
// FOR NON-REENTRANT PROGRAMS CODE NORENT ON THE
// CPARMS=() STATEMENT, AND ADD THE FOLLOWING INCLUDE
// STATEMENT TO THE LKED.SYSIN DD * STATEMENT:
// * INCLUDE SYSLIB(EZACICO7)
// */
// FOR REENTRANT PROGRAMS CODE RENT ON THE
// CPARMS=() STATEMENT, AND ADD THE FOLLOWING INCLUDE
// STATEMENT TO THE LKED.SYSIN DD * STATEMENT:
// * INCLUDE SYSLIB(EZACICO7)
```
Requirements:

- If the program is non-reentrant, you must perform the following actions:
  - Add an INCLUDE statement for module EZACIC07 and use EZACIC07 in place of CMIUCSOC.
  - Specify the compiler option of NORENT (non-reentrant) when you include the module EZACIC07 and &lt;ezacichd.h&gt;.

- If the program is reentrant, you must perform the following actions:
  - Add an INCLUDE statement for module EZACIC17 and use EZACIC17 in place of CMIUCSOC.
  - Specify the compiler option of RENT (reentrant) when you include the module EZACIC17 and &lt;errno.h&gt;.

- You must specify the NOSEARCH C/C++ compiler option to direct the compiler preprocessor to search only those data sets that are specified in the SYSLIB statement. For more information about the NOSEARCH compiler option, see z/OS XL C/C++ User’s Guide.

Structures used in socket calls

The parameter lists for some C language socket calls include a pointer to a data structure defined by a C structure. The structures are defined in the header files in.h, socket.h, and if.h. [Table 19] shows the C structure calls.

<table>
<thead>
<tr>
<th>C structure</th>
<th>Format</th>
</tr>
</thead>
</table>
| clientid    | struct clientid {
  | int domain; |
  | char name[8]; |
  | char subtaskname[8]; |
  | char reserved[20]; |
|             | }       |

| ifconf      | struct ifconf {
  | int ifc_len;
  | union {
    | caddr_t ifcu_buf;
    | struct ifreq *ifcu_req;
  } ifc_ifcu;
|             | }       |

[Table 19]
<table>
<thead>
<tr>
<th>C structure</th>
<th>Format</th>
</tr>
</thead>
</table>
| ifreq        | 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;
|             | } ifru_ifru;
|             | };
|             | NetConfHdr
|             | struct HomeIf {
|             | struct in6_addr HomeIfAddress;
|             | };
|             | struct NetConfHdr {
|             | char NchEyeCatcher[4];
|             | uint32_t NchIOCTL;
|             | int32_t NchBufferLength;
|             | union {
|             | struct HomeIf * __ptr32 NchIfHome;
|             | struct GRT6RtEntry * __ptr32 NchGRT6RtEntry;
|             | } NchBufferPtr;
|             | int32_t NchNumEntryRet;
|             | };
| If_NameIndex | struct if_nameindex {
|             | unsigned int if_index;
|             | char * if_name;
|             | };
| linger       | struct linger {
|             | int l_onoff;
|             | int l_linger;
|             | };
| ip_mreq      | struct ip_mreq {
|             | struct in_addr imr_multiaddr;
|             | struct in_addr imr_interface;
|             | };
| ipv6_mreq    | struct ipv6_mreq {
|             | struct in6_addr ipv6mr_multiaddr;
|             | unsigned int ipv6mr_interface;
|             | };

Table 19. C structures (continued)
<table>
<thead>
<tr>
<th>C structure</th>
<th>Format</th>
</tr>
</thead>
</table>
| sockaddr_in     | 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];         
| sockaddr_in6    | struct in6_addr {           
|                 |     union {              
|                 |         uint8_t _S6_u8[16]; 
|                 |         uint32_t _S6_u32[4]; 
|                 |     } _S6_un;              
|                 |   };                         
|                 | struct sockaddr_in6 {       
|                 |     uint8_t sin6_len;       
|                 |     sa_family_t sin6_family; 
|                 |     in_port_t sin6_port;    
|                 |     uint32_t sin6_flowinfo;  
|                 |     struct in6_addr sin6_addr; 
|                 |     uint32_t sin6_scope_id;  
| addrinfo        | struct addrinfo {           
|                 |     int ai_flags;           
|                 |     int ai_family;          
|                 |     int ai_socktype;        
|                 |     int ai_protocol;        
|                 |     socklen_t ai_addrlen;   
|                 |     char *ai_canonname;     
|                 |     struct sockaddr *ai_addr; 
|                 |     struct addrinfo *ai_next;  
|                 |     int ai_eflags;          
| timeval         | struct timeval {            
|                 |     time_t tv_sec;          
|                 |     long tv_usec;           
| ip_mreq_source  | struct ip_mreq_source {     
|                 |     struct in_addr imr_multiaddr;    
|                 |     struct in_addr imr_sourceaddr;    
|                 |     struct in_addr imr_interface;    
| group_req       | struct group_req {          
|                 |     uint32_t gr_interface;    
|                 |     uint32_t __gr_01;         
|                 |     struct sockaddr_storage gr_group;    
|               | };                             
|               |                           |
**Table 19. C structures (continued)**

<table>
<thead>
<tr>
<th>C structure</th>
<th>Format</th>
</tr>
</thead>
</table>
| group_source_req     | struct group_source_req {
|                      |    uint32_t gsr_interface;
|                      |    uint32_t __gsr_01;
|                      |    struct sockaddr_storage gsr_group;
|                      |    struct sockaddr_storage gsr_source;
|                      | }                         |
| SetApplData          | #define SetAD_eye1 "SETAPPLD"
|                      | #define SETADVER 1
|                      | struct {
|                      |    char SetAD_eye1[8];
|                      |    short SetAD_ver;
|                      |    short SetAD_len;
|                      |    char SetAD_rsv[4];
|                      | #ifndef _LP64
|                      |    int SetAD_ptrHW;
|                      | #endif
|                      |    SetADcontainer *SetAD_ptr;
|                      | } SetApplData;             |
| SetADcontainer        | #define SETADYE2 "APPLDATA"
|                      | typedef struct {
|                      |    char SetAD_eye2[8];
|                      |    char SetAD_buffer[40];
|                      | } SetADcontainer;          |

**The ERRNO variable**

The global variable `errno` is used by the socket system calls to report errors. If a socket call results in an error, the call returns a negative value, and an error value is set in `errno`. To be able to access these values, you must add one of the following include statements:

- **Non-reentrant programs:**
  ```c
  #include <ezacichd.h>
  ```

- **Reentrant programs:**
  ```c
  #include <errno.h>
  ```

**Note:**
- Do not use `tcperror()`.
- A copy of `EZACICHD.H` can be found in dataset `hlq.SEZAINST`.

**C socket call guidance**

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

For syntax, parameters, and other reference information for each C socket call, see [z/OS Communications Server: IP Programmer's Guide and Reference](https://www.ibm.com).
accept() call

A server issues the accept() call to accept a connection request from a client. The call uses a socket already created with a socket() call and marked by a listen() call.

An accept() call
1. Accepts the first connection on its queue of pending connections.
2. Creates a new socket with the same properties as the socket used in the call.
3. Returns the new socket descriptor to the server.

The new socket cannot be used to accept new connections, but is used by the client for application purposes. The server issues a givesocket() call and a CICS START command to enable a child server to communicate with the client for application purposes. The original socket remains available to the server to accept more connection requests.

The accept() call optionally saves the connection requester’s address for use by the server.

Note:
- 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().
- accept() calls are the only way to screen clients. The application cannot predetermine clients from which it accepts connections, but it can close a connection immediately after discovering the identity of the client.
- The select() call checks a socket for incoming connection requests.

accept() call format

This call has the following format:
accept() call parameters

s  The s parameter is a stream socket descriptor that has already been created with the socket() call. It is usually bound to an address with the bind() call. The listen() call marks the socket as one that accepts connections and allocates a queue to hold pending connection requests. The listen() call allows the caller to place an upper boundary on the size of the queue.

name  The pointer to a sockaddr structure into which the address of a client requesting a connection is placed on completion of the accept() call. If the server application does not need the client address, set the name parameter to the NULL pointer before making the accept() call.

The format of the name buffer is expected to be sockaddr_in, for an IPv4 socket address, or sockaddr_in6, for an IPv6 socket address, as defined in the header file in.h. The format of the structure is shown in Table 19 on page 167.

Use the following fields to define the IPv4 socket address structure for the socket that is to be accepted:

sin_family  Field must be set to AF_INET.

sin_port  Field contains the client’s port number.

in_addr.sin_addr  Field contains the 32-bit IPv4 Internet address, in network byte order, of the client’s host machine.

sin_zero  Field is not used and is set to all zeros.

Use the following fields to define the IPv6 socket address structure for the socket that is to be accepted:

sin6_family  Field must be set to AF_INET6.

sin6_port  Field contains the client’s port number.

sin6_flowinfo  Field contains the traffic class and flow label. The value of this field is undefined.

in6_addr.sin6_addr  Field contains the 128-bit IPv6 Internet address, in network byte order, of the client’s host machine.

sin6_scope_id  Field identifies a set of interfaces as appropriate for the scope of the address carried in the in6_addr.sin6_addr field. For a link scope

#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <bsdtypsetes.h>
#include <in.h>
#include <socket.h>
int accept(int s, struct sockaddr *name, int *namelen)
in6_addr.sin6_addr, sin6_scope_id contains the link index for the
in6_addr.sin6_addr. For all other address scopes, sin6_scope_id is
undefined.

namelen
The size, in bytes, of the buffer pointed to by name. For an IPv4 socket
address, the namelen parameter should contain a decimal 16. For an IPv6
socket address, the namelen parameter should contain a decimal 28.

accept() call return values
A nonnegative socket descriptor indicates success; the value -1 indicates an error.
To determine which error occurred, check the errno global variable, which is set to
a return code. Possible codes include:

EBADF
The s parameter is not a valid socket descriptor.

EFAULT
Using name and namelen results in an attempt to copy the address into a
portion of the caller’s address space into which information cannot be
written.

EINVAL
Listen() was not called for socket s.

ENOBFS
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() call
The bind() call binds a unique local port to an existing socket. Note that, on
successful completion of a socket() call, the new socket descriptor does not have an
associated port.

The bind() call can specify the required port or let the system choose. A listener
application should always bind to the same well-known port, so that clients can
know which port to use.

Even if an application specifies a value of 0 for the IP address on the bind(), the
system administrator can override that value by specifying the BIND parameter on
the PORT reservation statement in the TCP/IP profile. This has an effect similar to
the application specifying an explicit IP address on the bind() function. For more
information, see z/OS Communications Server: IP Configuration Reference.

bind() format
This call has the following format:
bind() parameters

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

name

The pointer to a socket address structure that contains the name that is to be bound to s. The format of the name buffer is expected to be sockaddr_in for an IPv4 socket address or sockaddr_in6 for an IPv6 socket address, as defined in the header file in.h. The format of the structure is shown in Table 19 on page 167.

Use the following fields to specify the IPv4 socket address structure for the socket that is to be bound:

sin_family  
Field must be set to AF_INET.

sin_port  
Field is set to the port to which the application must bind. It must be specified in network byte order. If sin_port is set to 0, the caller expects the system to assign an available port. The application can call getsockname() to discover the port number assigned.

in_addr.sin_addr  
Field is set to an IPv4 IP address and must be specified in network byte order. On hosts with more than one network interface (called multihomed hosts), you can select the interface to which it is to bind. Subsequently, only TCP connection requests from this interface are routed to the application.

If you set this field to the constant INADDR_ANY, as defined in in.h, the socket is bound to all network interfaces on the host. By leaving the address unspecified with INADDR_ANY, the server can accept all TCP connection requests made for its port, regardless of the network interface on which the requests arrived. Set INADDR_ANY for servers that offer a service to multiple networks.

sin_zero  
Field is not used and must be set to all zeros.

Use the following fields to specify the IPv6 socket address structure for the socket that is to be bound:

sin6_family  
Field must be set to AF_INET6.

sin6_port  
Field is set to the port to which the application must bind. It must be specified in network byte order. If sin6_port is set to 0, the caller expects the system to assign an available port. The application can call getsockname() to discover the port number assigned.
sin6_flowinfo
Field is used to specify the traffic class and flow label. This field must be set to zero.

in6_addr.sin6_addr
Field is set to an IPv6 address and must be specified in network byte order. On hosts with more than one network interface (called multihomed hosts), you can select the interface to which it is to bind. Subsequently, only TCP connection requests from this interface are routed to the application.

If you set this field to the constant in6addr_any, as defined in in.h, the socket is bound to all network interfaces on the host. By leaving the address unspecified with in6addr_any, the server can accept all TCP connection requests made for its port, regardless of the network interface on which the requests arrived. Set in6addr_any for servers that offer a service to multiple networks.

sin6_scope_id
Field is used to identify a set of interfaces as appropriate for the scope of the address carried in the in6_addr.sin6_addr field. A value of zero indicates the sin6_scope_id field does not identify the set of interfaces to be used, and might be specified for any address types and scopes. For a link scope in6_addr.sin6_addr field, sin6_scope_id might specify a link index which identifies a set of interfaces. For all other address scopes, sin6_scope_id must be set to zero.

namelen
The size, in bytes, of the buffer pointed to by name. For an IPv4 socket address, the namelen parameter should contain a decimal 16. For an IPv6 socket address, the namelen parameter should contain a decimal 28.

bind() return values
The value 0 indicates success; the value -1 indicates an error. To determine which error occurred, check the errno global variable, which is set to a return code. Possible codes include:

EADDRINUSE
The address is already in use. See the SO_REUSEADDR option described in “sockopt(), setsockopt() calls” on page 203 for more information.

The address is in a timed wait because a LINGER delay from a previous close or another process is using the address. This error also occurs if the port specified in the bind call has been configured as RESERVED on a port reservation statement in the TCP/IP profile.

If you want to reuse the same address, use the SO_REUSEADDR parameter in setsockopt(). If you do not want to reuse the same address, use a different address or port in the socket address structure. If the port has been configured as RESERVED, then the port is unavailable for bind.

EADDRNOTAVAIL
The address specified is not valid on this host. For example, the IP address does not specify a valid network interface.

EAFNOSUPPORT
The address family is not supported (it is not AF_INET or AF_INET6).

EBADF
The s parameter is not a valid socket descriptor.
EFAULT
Using name and namelen results in an attempt to copy the address into a nonwritable portion of the caller’s address space.

EINVAL
The socket is already bound to an address. An example is trying to bind a name to a socket that is in the connected state. This value is also returned if namelen is not the expected length.

bind2addrsel() call

The bind2addrsel() call binds a socket to the local IP address that would be selected by the stack to communicate with the input destination IP address.

In a TCP or UDP application, the bind2addrsel() call usually follows the setsockopt() call with optname IPV6_ADDR_PREFERENCES, and precedes any communication with a remote host. The bind2addrsel() call is used when the application must verify that a local IP address that is assigned by the stack meets its address selection criteria as provided by the IPV6_ADDR_PREFERENCES socket option before sending any packets to the remote host.

Result: The stack attempts to select a local IP address according to your application preferences. However, a successful bind2addrsel() result does not guarantee that all your source IP address selection preferences were met.

Guidelines:

- Use the setsockopt() call to set the IPV6_ADDR_PREFERENCES options to indicate your source IP address selection preferences before binding the socket, and before allowing an implicit bind of the socket to occur.
  
  **Tip:** If a socket has not been explicitly bound to a local IP address with a bind() or bind2addrsel() call when a connect(), sendto(), or sendmsg() call is issued, an implicit bind occurs.

- After you successfully issue the bind2addrsel() call, use the getsockname() call to obtain the local IP address bound to the socket. After the local IP address is obtained, use the inet6_is_srcaddr() call to verify that the local IP address meets your address selection criteria.

bind2addrsel() call format

This call has the following format:

```c
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <bsdtypes.h>
#include <socket.h>
#include <in.h>
#include <netdb.h>

int bind2addrsel (int s, const struct sockaddr *name, socklen_t namelen)
```

bind2addrsel() parameters

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

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

- **name** The pointer to an IPv6 socket address structure that contains the name that is to be bound to the socket descriptor specified by the s parameter. The
The format of the name buffer is expected to be sockaddr_in6 as defined in the header file in.h. The format of the structure is shown in Table 19 on page 167.

Use the following fields to specify the IPv6 socket address structure for the socket that is to be bound:

**sin6_family**
Field must be set to AF_INET6.

**sin6_port**
A halfword binary field. This field is ignored by bind2addrsel() processing.

**Guideline:** To determine the assigned port number, issue the getsockname() call after the bind2addrsel() call completes.

**sin6_flowinfo**
A fullword binary field. This field is ignored by bind2addrsel() processing.

**in6_addr.sin6_addr**
A 16-byte binary field that is set to the 128-bit IPv6 Internet address (network byte order) of the remote host that the socket will communicate with.

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

**sin6_scope_id**
A fullword binary field that identifies a set of interfaces as appropriate for the scope of the address specified in the in6_addr.sin6_addr field. The value 0 indicates that the sin6_scope_id field does not identify the set of interfaces to be used.

**Requirements:** The sin6_scope_id value must be nonzero if the address is link-local. For all other address scopes, the sin6_scope_id value must be set to 0.

**namelen**
The size, in bytes, of the buffer pointed to by the name parameter. The namelen parameter should contain the decimal value 28.

**bind2addrsel() return values**
The value 0 indicates success; the value -1 indicates an error. To determine which error occurred, check the errno global variable, which is set to a return code. Possible codes include:

**EADDRNOTAVAIL**
For the specified destination address, there is no source address that the application can bind to. Possible reasons can be one of the following situations:

* The socket is a stream socket, but the specified destination address is a multicast address.
* No ephemeral ports are available to assign to the socket.

**EAFNOSUPPORT**
The address family is not supported. The address family must be AF_INET6.
EBADF
The s parameter is not a valid socket descriptor.

EFAULT
Using the name and namelen parameters results in an attempt to copy the address into a nonwritable portion of the address space of the caller.

EHOSTUNREACH
There is no route to the host.

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 the namelen value is not the expected length.

EPROTOTYPE
The referenced socket is not a stream (TCP) or datagram (UDP) socket.

close() call
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.

close() call format
This call has the following format:
close() call parameter

$s$  The descriptor of the socket to be closed.

close() call return values

The value 0 indicates success; the value -1 indicates an error. To determine which error occurred, check the $errno$ global variable, which is set to a return code. Possible codes include:

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

connect() call

A connect() call attempts to establish a connection between a local socket and a remote socket. For a stream socket, the call performs two tasks. First, it completes the binding necessary for a stream socket in case it has not been previously bound by a bind() call. Second, it attempts to make a connection to another socket.

The connect() call on a stream socket is used by a client application to establish a connection to a server. To be able to accept a connection with an accept() call, the server must have a passive open pending, which means it must have successfully called bind() and listen() before the client issues connect().

If the socket is in blocking mode, the connect() call blocks the caller until the connection is set up, or until an error is received. If the socket is in nonblocking mode and no errors occurred, the return codes indicate that the connection can be initiated. The caller can test the completion of the connection setup by calling select() and testing for the ability to write to the socket.

Stream sockets can call connect() one time only.

connect() call format

This call has the following format:

```c
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <socket.h>
int close(int $s$)
```
connect() call parameters

`s`  The socket descriptor of the socket that is going to be used as the local endpoint of the connection.

`name`  The pointer to a socket address structure that contains the destination socket address to which a connection is requested.

The format of the `name` buffer is expected to be `sockaddr_in` for an IPv4 socket address or `sockaddr_in6` for an IPv6 socket address, as defined in the header file `in.h`. The format of the structure is shown in Table 19 on page 167.

Use the following fields to specify the IPv4 socket address structure for the socket that is to be bound:

`sin_family`  Field must be set to AF_INET.

`sin_port`  Field is set to the port to which the server is bound. It must be specified in network byte order.

`in_addr.sin_addr`  Field is set to the 32-bit IPv4 Internet address of the server's host machine in network byte order.

`sin_zero`  Field is not used and must be set to all zeros.

Use the following fields to specify the IPv6 socket address structure for the socket that is to be bound:

`sin6_family`  Field must be set to AF_INET6.

`sin6_port`  Field is set to the port to which the server is bound. It must be specified in network byte order.

`sin6_flowinfo`  Field is used to specify the traffic class and flow label. This field must be set to zero.

`in6_addr.sin6_addr`  Field is set to the 128-bit IPv6 Internet address of the server's host machine in network byte order.

`sin6_scope_id`  Field is used to identify a set of interfaces as appropriate for the scope of the address carried in the `in6_addr.sin6_addr` field. A value of zero indicates the `sin6_scope_id` field does not identify the set of interfaces to be used, and might be specified for any address types and scopes. For a link scope `in6_addr.sin6_addr`, `sin6_scope_id` might specify a link index which identifies a set of interfaces. For all other address scopes, `sin6_scope_id` must be set to zero.
**namelen**

The size of the socket address pointed to by `name` in bytes. For an IPv4 socket address the `namelen` parameter should contain a decimal 16 and for an IPv6 socket address the `namelen` parameter should contain a decimal 28.

**connect() call return values**

The value 0 indicates success; the value -1 indicates an error. To determine which error occurred, check the `errno` global variable, which is set to a return code. Possible codes include:

- **EADDRNOTAVAIL**
  The calling host cannot reach the specified destination.

- **EAFNOSUPPORT**
  The address family is not supported.

- **EALREADY**
  The socket `s` is marked nonblocking, and a previous connection attempt has not completed.

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

- **ECONNREFUSED**
  The connection request was rejected by the destination host.

- **EFAULT**
  Using `name` and `namelen` results in an attempt to copy the address into a portion of the caller’s address space to which data cannot be written.

- **EINPROGRESS**
  The socket `s` is marked nonblocking, and the connection cannot be completed immediately. The EINPROGRESS value does not indicate an error condition.

- **EINVAL**
  The `namelen` parameter is not a valid length.

- **EISCONN**
  The socket `s` is already connected.

- **ENETUNREACH**
  The network cannot be reached from this host.

- **ETIMEDOUT**
  The connection establishment timed out before a connection was made.

**fcntl() call**

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.
fcntl() call format

This call has the following format:

```c
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <socket.h>
#include <bsdtypes.h>
#include <fcntl.h>
signed int fcntl(int s, int cmd, int arg)
```

fcntl() call 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.

fcntl() call return values

For the F_GETFL command, the return value is a bit mask that is comprised of the flag settings. For the F_SETFL command, the value 0 indicates success; the value -1 indicates an error. To determine which error occurred, check the `errno` global variable, which is set to a return code. Possible codes include:

- **EBADF**: The `s` parameter is not a valid socket descriptor.
- **EINVAL**: The `arg` parameter is not a valid flag.

freeaddrinfo() call

The `freeaddrinfo()` call receives an input `addrinfo` structure pointer and releases that storage (plus any other chained `addrinfo` structures and related storage) back into the general storage pool.

freeaddrinfo() call format

This call has the following format:
freeaddrinfo() call parameters

ai  A pointer to an addrinfo structure returned by the getaddrinfo() res function variable.

freeaddrinfo() call return values

The value 0 indicates success; the value -1 indicates an error. To determine which error occurred, check the errno global variable, which is set to a return code. Possible codes include:

EAI_AGAIN
The resolver address space has not been started. The request can be retried at a later time.

EAI_FAIL
An unrecoverable error has occurred.

gai_strerror() call

The gai_strerror() function returns a pointer to a text string describing the error value returned by a failure return from either the getaddrinfo() or getnameinfo() function. If the ecode is not one of the EAI_xxx values from the <netdb.h> then gai_strerror() returns a pointer to a string indicating an unknown error. Subsequent calls to gai_strerror() overwrites the buffer that contains the text string.

gai_strerror() call format

This call has the following format:

```c
#include <manifest.h>  (non-reentrant programs only)
#include <cmanifes.h>  (reentrant programs only)
#include <socket.h>
#include <in.h>
#include <netdb.h>

void freeaddrinfo(struct addrinfo *ai) {

  freeaddrinfo() call parameters

  ai  A pointer to an addrinfo structure returned by the getaddrinfo() res function variable.

  freeaddrinfo() call return values

  The value 0 indicates success; the value -1 indicates an error. To determine which error occurred, check the errno global variable, which is set to a return code. Possible codes include:

  EAI_AGAIN
  The resolver address space has not been started. The request can be retried at a later time.

  EAI_FAIL
  An unrecoverable error has occurred.

  gai_strerror() call

  The gai_strerror() function returns a pointer to a text string describing the error value returned by a failure return from either the getaddrinfo() or getnameinfo() function. If the ecode is not one of the EAI_xxx values from the <netdb.h> then gai_strerror() returns a pointer to a string indicating an unknown error. Subsequent calls to gai_strerror() overwrites the buffer that contains the text string.

  gai_strerror() call format

  This call has the following format:
```
gai_strerror() call parameters

ecode    The errno value returned by the getaddrinfo() or getnameinfo() functions.


gai_strerror() call return values

When successful, gai_strerror() returns a pointer to a string describing the error. Upon failure, gai_strerror() returns NULL and set errno to the following:

ENOMEM    Insufficient memory to allocate buffer for text string describing the error.

getaddrinfo() call

The getaddrinfo() call translates the name of a service location (for example, a host name), a service name, or both and returns a set of socket addresses and associated information. This information is used to open a socket with which to address the specified service or to send a datagram to the specified service.

getaddrinfo() call format

This call has the following format:
The addrinfo structure has the following fields:

**ai_flags**
A fullword binary field. The value of this field must be 0 or the bitwise OR of one or more of the following flags:

**AI_PASSIVE**
Specifies how to fill in the ai_addr pointed to by the returned res.

If this flag is specified, the returned address information is suitable for use in binding a socket for accepting incoming connections for the specified service (for example, the bind() call). In this case, if the nodename parameter is null, the IP address portion of the socket address structure pointed to by the returned res is set to INADDR_ANY, for an IPv4 address, or to the IPv6 unspecified address (in6addr_any).

If this flag is not set, the returned address information is suitable for the connect() call (for a connection-mode protocol) or for a connect(), sendto() or sendmsg() call (for
a connectionless protocol). In this case, if the *nodename* parameter is not specified, the *ai_addr* pointed to by the returned *res* is set to the loopback address.

This flag is ignored if the *nodename* parameter is specified.

**AI_CANONNAMEOK**

If this flag is specified and the *nodename* parameter is specified, the getaddrinfo() call attempts to determine the canonical name corresponding to the *nodename* parameter.

**AI_NUMERICHOST**

If this flag is specified, the *nodename* parameter must be a numeric host address in presentation form. Otherwise, an error of host not found [EAI_NONAME] is returned.

**AI_NUMERICSERV**

If this flag is specified, the *servname* parameter must be a numeric port in presentation form. Otherwise, an error [EAI_NONAME] is returned.

**AI_V4MAPPED**

If this flag is specified with the *ai_family* field using the value of AF_INET6, or the value of AF_UNSPEC when IPv6 is supported on the system, the caller accepts IPv4-mapped IPv6 addresses. When the AI_ALL flag is not also specified, if no IPv6 addresses are found, a query is made for IPv4 addresses. If IPv4 addresses are found, they are returned as IPv4-mapped IPv6 addresses. If the *ai_family* field does not have the value of AF_INET6, or the *ai_family* field contains AF_UNSPEC but IPv6 is not supported on the system, then this flag is ignored.

**AI_ALL**

If the *ai_family* field has a value of AF_INET6 and AI_ALL is set, the AI_V4MAPPED flag must also be set to indicate that the caller accepts all addresses: IPv6 and IPv4-mapped IPv6 addresses. If the *ai_family* field has a value of AF_UNSPEC when the system supports IPv6 and AI_ALL is set, the caller accepts both IPv6 and IPv4 addresses. A query is first made for IPv6 addresses and if successful, the IPv6 addresses are returned. Another query is then made for IPv4 addresses, and any IPv4 addresses found are returned as IPv4-mapped IPv6 addresses (if AI_V4MAPPED is also specified) or as IPv4 addresses (if AI_V4MAPPED is not specified). If the *ai_family* field does not have the value of AF_INET6, or does not have the value of AF_UNSPEC when the system supports IPv6, then this flag is ignored.

**AI_ADDRCONFIG**

If this flag is specified, then a query on the name in *nodename* occurs if the resolver determines that one of the following is true:

- If the system is IPv6 enabled and has at least one IPv6 interface, the resolver makes a query for IPv6 (AAAA or A6 DNS records) records.
• If the system is IPv4 enabled and has at least one IPv4 interface, the resolver makes a query for IPv4 (A DNS records) records.

**AI_EXTFLAGS**

If this flag is specified, the addrinfo structure contains an `ai_eflags` field (see the field description of `ai_eflags`).

**ai_family**

Used to limit the returned information to a specific address family. The value of AF_UNSPEC means that the caller accepts any protocol family. The value of a decimal 0 indicates AF_UNSPEC. The value of a decimal 2 indicates AF_INET and the value of a decimal 19 indicates AF_INET6.

**ai_socktype**

Used to limit the returned information to a specific socket type. A value of 0 means that the caller accepts any socket type. If a specific socket type is not given (for example, a value of 0), information about all supported socket types are returned.

The following are the acceptable socket types:

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

Any other socket type fails with a return code of EAI_SOCKTYPE. Note that although SOCK_RAW is accepted, it is valid only when `servname` is numeric (for example, `servname`=23). A lookup for a service name never occurs in the appropriate services file (for example, `hlq.ETC.SERVICES`) using any protocol value other than SOCK_STREAM or SOCK_DGRAM. If `ai_protocol` is not 0 and `ai_socktype` is 0, the only acceptable input values for `ai_protocol` are IPPROTO_TCP and IPPROTO_UDP; otherwise, the getaddrinfo() function fails with a return code of EAI_BADFLAGS. If `ai_socktype` and `ai_protocol` are both specified as 0, getaddrinfo() proceeds as follows:

• If `servname` is null, or if `servname` is numeric, any returned `addrinfo` structures default to a specification of `ai_socktype` as SOCK_STREAM.

• If `servname` is specified as a service name, for example `servname`=FTP, the getaddrinfo() call searches the appropriate services file (for example, `hlq.ETC.SERVICES`) twice. The first search uses SOCK_STREAM as the protocol, and the second search uses SOCK_DGRAM as the protocol. No default socket type provision exists in this case.

If both `ai_socktype` and `ai_protocol` are specified as nonzero, then they should be compatible, regardless of the value specified by the `servname` parameter. In this context, compatibility means one of the following:

• `ai_socktype`=SOCK_STREAM and `ai_protocol`=IPPROTO_TCP
• `ai_socktype`=SOCK_DGRAM and `ai_protocol`=IPPROTO_UDP
• *ai_socktype* is specified as SOCK_RAW. In this case, *ai_protocol* can be anything.

*ai_protocol*

Used to limit the returned information to a specific protocol. A value of 0 means that the caller accepts any protocol.

The following are the acceptable protocols:

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

If *ai_protocol* and *ai_socktype* are both specified as 0, getaddrinfo() proceeds as follows:

• If *servname* is null, or if *servname* is numeric, then any returned addrinfos default to a specification of *ai_socktype* as SOCK_STREAM.

• If *servname* is specified as a service name (for example, *servname*=FTP), getaddrinfo() searches the appropriate services file (for example, hlq.ETC.SERVICES) twice. The first search uses SOCK_STREAM as the protocol, and the second search uses SOCK_DGRAM as the protocol. No default socket type provision exists in this case.

If both *ai_socktype* and *ai_protocol* are specified as nonzero then they should be compatible, regardless of the value specified by servname. In this context, compatibility means one of the following:

• *ai_socktype* = SOCK_STREAM and *ai_protocol* = IPPROTO_TCP
• *ai_socktype* = SOCK_DGRAM and *ai_protocol* = IPPROTO_UDP
• *ai_socktype* = SOCK_RAW. In this case, *ai_protocol* can be anything.

If the lookup for the value specified in *servname* fails [that is, the service name does not appear in the appropriate services file (for example, hlq.ETC.SERVICES) using the input protocol], the getaddrinfo() call fails with return code of EAI_SERVICE.

*ai_addrlen*

On input, this field must be 0.

*ai_canonname*

On input, this field must be 0.

*ai_addr*

On input, this field must be 0.

*ai_next*

On input, this field must be 0.

*ai_eflags*

A fullword binary field that specifies the source IPv6 address selection preferences. This field is required if AI_EXTFLAGS is specified in the *ai_flags* field. The value of this field must be 0 or the bitwise OR of one or more of the following flags:

**IPV6_PREFER_SRC_HOME**

Indicates that home source IPv6 addresses are preferred over care-of source IPv6 addresses.
**IPV6_PREFER_SRC_COA**  
Indicates that care-of source IPv6 addresses are preferred over home source IPv6 addresses.

**IPV6_PREFER_SRC_TMP**  
Indicates that temporary source IPv6 addresses are preferred over public source IPv6 addresses.

**IPV6_PREFER_SRC_PUBLIC**  
Indicates that public source IPv6 addresses are preferred over temporary source IPv6 addresses.

**IPV6_PREFER_SRC_CGA**  
Indicates that cryptographically generated source IPv6 addresses are preferred over non-cryptographically generated source IPv6 addresses.

**IPV6_PREFER_SRC_NONCGA**  
Indicates that non-cryptographically generated source IPv6 addresses are preferred over cryptographically generated source IPv6 addresses.

If contradictory or invalid EFLAGS are specified, the GETADDRINFO call fails with the return code -1 and the errno EAI_BADEXTFLAGS (decimal value 11).

- An example of contradictory EFLAGS is IPV6_PREFER_SRC_TMP and IPV6_PREFER_SRC_PUBLIC.
- An example of invalid EFLAGS is X'00000040', or a decimal value of 64.

**Note:** The field is required only if AI_EXTFLAGS is specified in the `ai_flags` field.

*res*  
Initially a fullword binary field. On a successful return, this field contains a pointer to a chain of one or more addrinfo structures. The structures are allocated in the key of the calling application. The structures returned by getaddrinfo() are serially reusable storage for the z/OS UNIX process. The structures can be used or referenced between process threads, but should not be used or referenced between processes. When you finish using the structures, explicitly release their storage by specifying the returned pointer on a freaddrinfo() call.

The address information structure contains the following fields:

**ai_flags**  
Not used as output.

**ai_family**  
The value returned in this field can be used as the `domain` argument on the socket() call to create a socket suitable for use with the returned socket address pointed to by `ai_addr`.

**ai_socktype**  
The value returned in this field can be used as the `type` argument on the socket() call to create a socket suitable for use with the returned address socket pointed to by `ai_addr`.

**ai_protocol**  
The value returned in this field can be used as the `protocol` argument on the socket() call to create a socket suitable for use with the returned socket address pointed to by `ai_addr`. 
The length of the socket address structure pointed to by the `ai_addr` field. The value returned in this field can be used as the arguments for the `connect()` or `bind()` call with this socket type, according to the AI_PASSIVE flag.

A pointer to the canonical name for the value specified by `nodename`. If the `nodename` argument is specified, and if the AI_CANONNAMEOK flag was specified by the `hints` parameter, the `ai_canonname` field in the first returned address information structure contains the address of storage that contains the canonical name corresponding to the input `nodename` parameter. If the canonical name is not available, the `ai_canonname` field refers to the `nodename` parameter or a string with the same contents.

The address of the returned socket address structure. The value returned in this field can be used as the arguments for the `connect()` or `bind()` call with this socket type, according to the AI_PASSIVE flag.

Contains the address of the next address information structure on the list, or zeros if it is the last structure on the list.

This field is not used as output.

### getaddrinfo() call return values

The value 0 indicates success; the value -1 indicates an error. To determine which error occurred, check the `errno` global variable, which is set to a return code. Possible codes include:

**EAI_AGAIN**

The name specified by the `nodename` parameter could be not be resolved within the configured time interval, or the resolver address space has not been started. The request can be retried at a later time.

**EAI_BADFLAGS**

The flags parameter had a value that is incorrect.

**EAI_BADEXTFLAGS**

The `ai_eflags` parameter had a value that is incorrect.

**EAI_FAMILY**

The family parameter has a value that is incorrect.

**EAI_MEMORY**

Memory allocation failure occurred trying to acquire an addrinfo structure.

**EAI_NONAME**

The name does not resolve for the specified parameters. At least one of the `nodename` or `servname` parameters must be specified. Or the requested nodename parameter is valid but does not have a record at the name server.

**EAI_SERVICE**

The service passed was not recognized for the specified socket type.
EAI_SOCKTYPE
The intended socket type was not recognized.

getclientid() call

A getclientid() call returns the identifier by which the calling application is known to the TCP/IP address space. Do not be confused by the term client in the name of this call; the call always returns the ID of the calling process, be it client or server. For example, in CICS TCP/IP, this call is issued by the IBM listener; the identifier returned in that case is that of the listener (a server). This identifier is used in the givesocket() and takesocket() calls.

getclientid() call format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <bsdtypes.h>
#include <socket.h>
int getclientid(int domain, struct clientid *clientid)
```

getclientid() call parameters

- **domain** The domain must be set to AF_INET when requesting client data from an IPv4 stack and it must be set to AF_INET6 when requesting client data from an IPv6 stack.
- **clientid** Points to a clientid structure to be provided.
  - **domain** Domain associated with the program executing this call. Contains either AF_INET (a decimal 2) or AF_INET6 (a decimal 19).
  - **name** Address space name associated with the program executing this call.
  - **subtaskname** Subtask name associated with the program executing this call.
  - **reserved** Binary zeros.

getclientid() call return values

The value 0 indicates success; the value -1 indicates an error. To determine which error occurred, check the errno global variable, which is set to a return code. Possible codes include:

- **EFAULT** Using the clientid parameter as specified results in an attempt to access storage outside the caller’s address space, or storage not modifiable by the caller.
- **EPFNOSUPPORT** Domain is not AF_INET or AF_INET6.

gethostbyaddr() call

The gethostbyaddr() call tries to resolve the IP address to a host name. The resolution attempted depends on how the resolver is configured and if any local
host tables exist. See z/OS Communications Server: IP Configuration Guide for information about configuring the resolver and using local host tables.

gethostbyaddr() call format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <netdb.h>
struct hostent *gethostbyaddr(char *addr, int addrlen, int domain)
```

gethostbyaddr() call parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>addr</code></td>
<td>The pointer to an unsigned long value that contains the address of the host.</td>
</tr>
<tr>
<td><code>addrlen</code></td>
<td>The size of <code>addr</code> in bytes.</td>
</tr>
<tr>
<td><code>domain</code></td>
<td>The address domain supported (AF_INET).</td>
</tr>
</tbody>
</table>

gethostbyaddr() call return values

The gethostbyaddr() call returns a pointer to a hostent structure for the host address specified on the call. For more information about the hostent structure, see Figure 128 on page 284. A null pointer is returned if the gethostbyaddr() call fails.

There are no errno values for gethostbyaddr().

gethostbyname() call

The gethostbyname() call tries to resolve the host name to an IP address. The resolution attempted depends on how the resolver is configured and if any local host tables exist. See z/OS Communications Server: IP Configuration Guide for information about configuring the resolver and using local host tables.

gethostbyname() call format

This call has the following format:
gethostbyname() call parameters

name    The name of the host being queried. The name has a maximum length of 255 characters.

gethostbyname() call return values

The gethostbyname() call returns a pointer to a hostent structure for the host name specified on the call. For more information about the hostent structure, see Figure 130 on page 287. A null pointer is returned if the gethostbyname() call fails.

There are no errno values for gethostbyname().

A new part called EZACIC17 has been created. EZACIC17 is like EZACIC07 except it uses the internal C errno function. Also, a new header file called cmanifes.h has been created to remap EZACIC17's long function names into unique 8-character names.

EZACIC07 and EZACIC17 now support the gethostbyaddr() and gethostbyname() functions.

gethostid() call

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.

gethostid() call format

This call has the following format:
gethostid() call parameters

None.

gethostid() call return values

The gethostid() call returns the 32-bit identifier of the current host, which should be unique across all hosts.

gethostname() call

The gethostname() call returns the name of the host processor on which the program is running.

Note: The host name that is returned is the host name that the TCPIP stack learned at startup. For more information about hostname, see HOSTNAME statement in z/OS Communications Server: IP Configuration Reference.

gethostname() call format

This call has the following format:


```c
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <socket.h>

int gethostname(char *name, int namelen)
```

**gethostname() call parameters**

- **name** The character array to be completed with the host name. The name that is returned is NULL-terminated unless truncated to the size of the name array.

- **namelen** The length of the *name* value. The minimum length of the *name* field is 1 character. The maximum length of the *name* field is 24 characters.

**gethostname() call return values**

The value 0 indicates success; the value -1 indicates an error. To determine what error has occurred, check the *errno* global variable, which is set to a return code. Possible codes are:

- **EFAULT** The *name* parameter specified an address outside the caller’s address space.

**getipv4sourcefilter() call**

Obtains a list of the IPv4 source addresses that comprise the source filter, along with the current mode on a given interface and a multicast group for a socket. The source filter can either include or exclude the set of source addresses, depending on the filter mode (MCAST_INCLUDE or MCAST_EXCLUDE).

**getipv4sourcefilter() call format**

This call has the following format:
The socket descriptor.
The local IP address of the interface.
The IP multicast address of the group.
A pointer to an integer that contains the filter mode on a successful return. The value of the filter mode can be MCAST_INCLUDE or MCAST_EXCLUDE.
As an input parameter, a pointer to the number of source addresses that can fit in the array specified by the slist parameter. As an output parameter, a pointer to the total number of source addresses in the filter.
A pointer to an array of IP addresses that is either included or excluded, depending on the filter mode. If the numsrc value was 0 on input, a NULL pointer can be supplied.

If the application does not know the size of the source list before, it can make a reasonable guess (for example, 0). When the process completes, the numsrc value is larger, the operation can be repeated with a larger buffer.

On return, the numsrc value is always updated to be the total number of sources in the filter. The slist value specifies as many source addresses as fit, up to the minimum array size that was specified by the numsrc value and the total number of sources in the filter.

getipv4sourcefilter() call return values

When successful, the value 0 is returned. When an error has occurred, the value -1 is returned and the errno value is one of the following:

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

EINVAL
The interface or group parameter value is not a valid IPv4 address, or the socket s has already requested multicast setsockopt options. For more information, see the z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference.

EPROTOTYPE
The socket protocol type is not correct.

EADDRNOTAVAIL
The tuple consisting of socket, interface, and multicast group values does not exist, or the specified interface address is incorrect for this host, or the specified interface address is not multicast capable.
ENOMEM
Insufficient storage is available to supply the array.

getnameinfo() call

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

getnameinfo() call format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <manifes.h> (reentrant programs only)
#include <in.h>
#include <netdb.h>

int getnameinfo(const struct sockaddr *sa, socklen_t salen,
                 char *host, socklen_t hostlen,
                 char *serv, socklen_t servlen,
                 int flags)
```

getnameinfo() call parameters

- `sa`: The pointer to a socket address structure that is expected to be either `sockaddr_in` for an IPv4 socket address or `sockaddr_in6` for an IPv6 socket address, as defined in the header file `in.h`. Table 19 on page 167 shows the format of the structure.

  The following fields are used to specify the IPv4 socket address structure to be translated.
  - The `sin_family` field must be set to AF_INET.
  - The `sin_port` field is set to a port number, in network byte order.
  - The `in_addr.sin_addr` field is set to an IPv4 address and must be specified in network byte order.
  - The `sin_zero` field is not used and must be set to all zeros.

  The following fields are used to specify the IPv6 socket address structure to be translated.
  - The `sin6_family` field must be set to AF_INET6.
  - The `sin6_port` field is set to the a port number, in network byte order.
  - The `sin6_flowinfo` field is used to specify the traffic class and flow label. This field is currently not implemented.
  - The `in6_addr.sin6_addr` field is set to an IPv6 address and must be specified in network byte order.
  - The `sin6_scope_id` field is used to specify the link scope for an IPv6 address as an interface index. The resolver ignores the `sin6_scope_id` field, unless the input IPv6 address is a link-local address and the `host` parameter is also specified.

- `salen`: The size, in bytes, of the buffer pointed to by `sa`. For an IPv4 socket address, the `salen` parameter should contain a decimal 16, and for an IPv6 socket address, the `salen` parameter should contain a decimal 28.

- `host`: On input, storage capable of holding the returned resolved host name. The host name can be a maximum of 255 bytes for a null terminated string, for
the input socket address. If inadequate storage is specified to contain the resolved host name, then the resolver returns the host name up to the storage amount specified and truncation might occur. If the host name cannot be located, the numeric form of the host address is returned instead of its name. However, if the NI_NAMEREQD option is specified and no host name is located, an error is returned.

If the specified IPv6 address is a link-local address, and the `sin6_scope_id` interface index is a non-zero value, scope information is appended to the resolved host name using the format `host%scope information`. The scope information can be either the numeric form of the interface index, or the interface name associated with the interface index.

Use the NI_NUMERICSCOPE option to select which form should be returned. The combined host name and scope information is always a null-terminated string that is no more than 256 bytes in length. For more information about scope information and getnameinfo() processing, see z/OS Communications Server: IPv6 Network and Application Design Guide.

This is an optional field, but if this field value is not 0, you must also specify the `hostlen` parameter. Specify both the `service` and `servlen` parameters or both the `host` and `hostlen` parameters. An error occurs if both are omitted.

`hostlen` A field that contains the length of the host storage used to contain the resolved host name. The `hostlen` parameter value must be equal to or greater than the length of the longest host name or of the host name and scope information combination, plus one for the null termination character, to be returned. The getnameinfo() call returns the host name, or host name and scope information, up to the length specified by the `hostlen` parameter. If the `hostlen` parameter is 0 on input, then the resolved host name is not returned.

This is an optional field, but if the field value is not 0, you must also specify the `host` parameter. Specify both the `service` and `servlen` parameters or both the `host` and `hostlen` parameters. An error occurs if both are omitted.

`serv` On input, storage capable of holding the returned resolved service name, which can be a maximum of 33 bytes for a null terminated string, for the input socket address. If inadequate storage is specified to contain the resolved service name, the resolver returns the service name up to the storage specified and truncation might occur. If the service name cannot be located, or if NI_NUMERICSERV was specified in the `flags` parameter, then the numeric form of the service address is returned instead of its name.

This is an optional field, but if the value is not 0, then you must also specify the `servlen` parameter. Specify both the `service` and `servlen` parameters or both the `host` and `hostlen` parameters. An error occurs if both are omitted.

`servlen` A field that contains the length of the storage used to contain the returned resolved service name (specified by the `serv` parameter). The `servlen` parameter must be equal to or greater than the length of the longest service name to be returned, plus one for the null termination character. The getnameinfo() call returns the service name up to the length specified by the `servlen` parameter value. If the `servlen` value is 0 on input, the service name information is not returned.
This is an optional field, but if the value is not 0, you must also specify the serv parameter. Specify both the service and servlen parameters or both the host and hostlen parameters. An error occurs if both are omitted.

**flags**

The parameter can be set to 0 or one of the following:

- **NI_NOFQDN**
  - Return the NAME portion of the fully qualified domain name.

- **NI_NUMERICHOST**
  - Return only the numeric form of host's address.

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

- **NI_NUMERICSERV**
  - Return only the numeric form of the service address.

- **NI_DGRAM**
  - Indicates that the service is a datagram service. The default behavior is to assume that the service is a stream service.

- **NI_NUMERICSCOPE**
  - Return only the numeric form of the sin6_scope_id interface index, if applicable.

### getnameinfo() call return values

The value 0 indicates success; the value -1 indicates an error. To determine which error occurred, check the errno global variable, which is set to a return code. Possible codes include:

- **EAI_AGAIN**
  - The host address specified could not be resolved within the configured time interval, or the resolver address space has not been started. The request can be retried at a later time.

- **EAI_BADFLAGS**
  - The flags parameter had an incorrect value.

- **EAI_FAIL**
  - An unrecoverable error has occurred.

- **EAI_FAMILY**
  - The address family was not recognized, or the address length was incorrect for the specified family.

- **EAI_MEMORY**
  - A memory allocation failure occurred.

- **EAI_NONAME**
  - The hostname does not resolve for the supplied parameters.

    NI_NAMEREQD is set and the hostname cannot be located, or both nodename and servname were null. Or the requested address is valid but does not have a record at the name server.

### getpeername() call

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

This call has the following format:

```c
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <socket.h>
#include <bsdtypes.h>
int getpeername(int s, struct sockaddr *name, int *namelen)
```

**getpeername() call parameters**

- **s**  
  The socket descriptor.

- **name**  
  A pointer to a structure that contains the IP address of the connected socket that is filled by getpeername() before it returns. The exact format of name is determined by the domain in which communication occurs.

  The following fields are used to define the IPv4 socket address structure for the remote socket that is connected to the local socket specified in field s.

  - The `sin_family` field is set to AF_INET.
  - The `sin_port` field contains the connection peer’s port number.
  - The `in_addr.sin_addr` field contains the 32-bit IPv4 Internet address, in network byte order, of the connection peer’s host machine.
  - The `sin_zero` field is not used and is set to all zeros.

  The following fields are used to define the IPv6 socket address structure for the remote socket that is connected to the local socket specified in field s.

  - The `sin6_family` field is set to AF_INET6.
  - The `sin6_port` field contains the connection peer’s port number.
  - The `sin6_flowinfo` field contains the traffic class and flow label. The value of this field is undefined.
  - The `in6_addr.sin6_addr` field contains the 128-bit IPv6 Internet address, in network byte order, of the connection peer’s host machine.
  - The `sin6_scope_id` field identifies a set of interfaces as appropriate for the scope of the address carried in the `in6_addr.sin6_addr` field. For a link scope `in6_addr.sin6_addr`, `sin6_scope_id` contains the link index for the `in6_addr.sin6_addr`. For all other address scopes, `sin6_scope_id` is undefined.

- **namelen**  
  A pointer to the structure that contains the size of the address structure pointed to by name in bytes. For an IPv4 socket address the namelen parameter should contain a decimal 16 and for an IPv6 socket address the namelen parameter should contain a decimal 28.

**getpeername() call return values**

The value 0 indicates success; the value -1 indicates an error. To determine which error occurred, check the errno global variable, which is set to a return code. Possible codes include:

- **EBADF**  
  The s parameter is not a valid socket descriptor.
EFAULT
Using the name and namelen parameters as specified results in an attempt to access storage outside of the caller’s address space.

ENOTCONN
The socket is not in the connected state.

getsockname() call

A getsockname() call returns the current name for socket s in the sockaddr structure pointed to by the name parameter. It returns the address of the socket that has been bound. If the socket is not bound to an address, the call returns with family set, and the rest of the structure set to zero. For example, an unbound IPv4 socket causes the name to point to a sockaddr_in structure with the sin_family field set to AF_INET and all other fields set to zero. An unbound IPv6 socket causes the name to point to a sockaddr_in6 structure with the sin6_family field set to AF_INET6 and all other fields set to zero.

Stream sockets are not assigned a name until after a successful call to either bind(), connect(), or accept().

The getsockname() call is often used to discover the port assigned to a socket after the socket has been implicitly bound to a port. For example, an application can call connect() without previously calling bind(). In this case, the connect() call completes the binding necessary by assigning a port to the socket. This assignment can be discovered with a call to getsockname().

getsockname() call format

This call has the following format:
getsockname() call parameters

s The socket descriptor.

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

The following fields are used to define the IPv4 socket address structure returned by the call.

- The sin_family field is set to AF_INET.
- The sin_port field contains the port number bound to this socket. If the socket is not bound, 0 is returned.
- The in_addr.sin_addr field contains the 32-bit IPv4 Internet address, in network byte order, of the local host machine. If the socket is not bound, the address is INADDR_ANY.
- The sin_zero field is not used and is set to all zeros.

The following fields are used to define the IPv6 socket address structure returned by the call.

- The sin6_family field is set to AF_INET6.
- The sin6_port field contains the port number bound to this socket. If the socket is not bound, 0 is returned.
- The sin6_flowinfo field contains the traffic class and flow label. The value of this field is undefined.
- The in6_addr.sin6_addr field contains the 128-bit IPv6 Internet address, in network byte order, of the local host machine. If the socket is not bound, the address is the IPv6 unspecified address (in6addr_any).
- The sin6_scope_id field identifies a set of interfaces as appropriate for the scope of the address carried in the in6_addr.sin6_addr field. For a link scope in6_addr.sin6_addr, sin6_scope_id contains the link index for the in6_addr.sin6_addr. For all other address scopes, sin6_scope_id is undefined.

namelen Must initially point to an integer that contains the size in bytes of the storage pointed to by name. Upon return, that integer contains the size of the data returned in the storage pointed to by name. For an IPv4 socket address the namelen parameter contains a decimal 16 and for an IPv6 socket address the namelen parameter contains a decimal 28.

getsockname() call return values

The value 0 indicates success; the value -1 indicates an error. To determine which error occurred, check the errno global variable, which is set to a return code. Possible codes include:

EBADF

The s parameter is not a valid socket descriptor.
Using the name and namelen parameters as specified results in an attempt to access storage outside of the caller’s address space.

**getsockopt(), setsockopt() calls**

The getsockopt() call gets options associated with a socket; setsockopt() sets the options.

The following options are recognized at the IPPROTO_IP level:
- Joining a multicast group
- Leaving a multicast group or leaving all sources for a given multicast group
- Setting the multicast interface
- Setting the IP time-to-live of outgoing multicast datagrams
- Looping back multicast datagrams
- Joining a source-specific multicast group
- Leaving a source-specific multicast group
- Blocking data from a given source to a given multicast group
- Unblocking a previously blocked source for a given multicast group

The following options are recognized at the IPPROTO_IPV6 level:
- Joining a multicast group
- Leaving a multicast group
- Setting the multicast interface
- Setting multicast hop limit
- Looping back multicast datagrams
- Setting unicast hop limit
- Restricting sockets to AF_INET6 sockets
- Setting source IP address selection preferences
- Retrieving source IP address selection preferences

The following options are recognized at the IPPROTO_IP and IPPROTO_IPV6 level:
- Joining an IPv4 or IPv6 multicast group
- Leaving an IPv4 or IPv6 multicast group or leaving all sources for a given IPv4 or IPv6 multicast group
- Joining an IPv4 or IPv6 source-specific multicast group
- Leaving an IPv4 or IPv6 source-specific multicast group
- Blocking IPv4 or IPv6 data from a given source to a given multicast group
- Unblocking an IPv4 or IPv6 previously blocked source for a given multicast group

The following options are recognized at the socket level:
- Broadcasting messages (IPv4 UDP socket only)
- Toggling the TCP keep-alive mechanism for a stream socket
- Lingering on close if data is present
- Receiving of out-of-band data
- Local address reuse
The following option is recognized at the TCP level (IPPROTO_TCP):

- Prevent infinite blocking for receive and send type functions

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

**getsockopt(), setsockopt() calls format**

The format for getsockopt() is as follows:

```c
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <socket.h>
#include <bsdtypes.h>
#include <bsdtime.h>

int getsockopt(int s, int level, int optname, char *optval, int *optlen)
```

The format for setsockopt() is as follows:

```c
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <socket.h>
#include <bsdtypes.h>
#include <bsdtime.h>

int setsockopt(int s, int level, int optname, char *optval, int optlen)
```

**Note:** This code sample is for getsockopt(). The setsockopt() call requires the same parameters and declarations, except that:

- The socket function name changes; getsockopt() becomes setsockopt().
- int *optlen should be replaced by int optlen (without the asterisk).

**getsockopt(), setsockopt() calls parameters**

- **s** The socket descriptor.
- **level** When manipulating socket options, you must specify the level at which the option resides and the name of the option. To manipulate options at the socket level, the level parameter must be set to SOL_SOCKET as defined in socket.h. For TCP_NODELAY at the TCP level, the level parameter must be set to IPPROTO_TCP. To manipulate other TCP level options or options at any other level, such as the IP level, supply the appropriate protocol number for the protocol controlling the option. Currently, only the IPPROTO_IP, IPPROTO_IPV6, IPPROTO_TCP, and SOL_SOCKET levels are supported.
- **optname** The name of a specified socket option. The options that are available with CICS TCP/IP are shown in [“Possible entries for optname” on page 205](#).
- **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 \textit{optval} and \textit{optlen} parameters are used to pass data used by the particular set command. The \textit{optval} parameter points to a buffer that contains the data needed by the set command. The \textit{optval} parameter is optional and can be set to the NULL pointer, if data is not needed by the command. The \textit{optlen} parameter must be set to the size of the data pointed to by \textit{optval}.

For both calls, all of the socket level options except SO_LINGER expect \textit{optval} to point to an integer and \textit{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 \textit{optval} to point to a \textit{linger} structure as defined in \textit{socket.h}.

This structure is defined in the following example:

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

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

\section*{Possible entries for optname}

The following options are recognized at the IPPROTO_IP level:

\begin{description}
\item[IP_ADD_MEMBERSHIP] Enables an application to join a multicast group on a specific interface. An interface must be specified with this option. Only applications that want to receive multicast datagrams need to join multicast groups. This is an IPv4 only socket option.

For setsockopt(), set the \textit{optval} value to the structure as defined in in.h. The \textit{ip_mreq} structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 interface address.

This option cannot be specified with the getsockopt() call.

\item[IP_ADD_SOURCE_MEMBERSHIP] Enables an application to join a multicast group on a specific interface and a specific source address. An interface and a source address must be specified with this option. Only applications that want to receive multicast datagrams need to join source multicast groups. This socket option applies only to IPv4.

For the setsockopt() function, set the \textit{optval} value to the \textit{ip_mreq_source} structure as defined in the in.h header. The \textit{ip_mreq_source} structure contains the following:

- 4-byte IPv4 multicast address
- 4-byte IPv4 source address
- 4-byte IPv4 interface address

This option cannot be specified with the getsockopt() function.

\item[IP_BLOCK_SOURCE] Enables an application to block multicast packets that have a source address that matches the given IPv4 source address. An interface and a
source address must be specified with this option. The specified multicast group must be joined previously. This socket option applies only to IPv4.

For the setsockopt() function, set the optval value to the ip_mreq_source structure as defined in the in.h header. The ip_mreq_source structure contains the following:

- 4-byte IPv4 multicast address
- 4-byte IPv4 source address
- 4-byte IPv4 interface address

This option cannot be specified with the getsockopt() function.

**IP_DROP_MEMBERSHIP**

Enables an application to exit a multicast group or to exit a multicast group and drop all sources. This is an IPv4-only socket option.

For the setsockopt() function, set the optval value to the ip_mreq structure as defined in the in.h header. The ip_mreq structure contains the following:

- 4-byte IPv4 multicast address
- 4-byte IPv4 interface address

This option cannot be specified with the getsockopt() function.

**IP_DROP_SOURCE_MEMBERSHIP**

Enables an application to exit a source multicast group. This socket option applies only to IPv4.

For the setsockopt() function, set the optval value to the ip_mreq_source structure as defined in the in.h header. The ip_mreq_source structure contains the following:

- 4-byte IPv4 multicast address
- 4-byte IPv4 source address
- 4-byte IPv4 interface address

This option cannot be specified with the getsockopt() function.

**IP_MULTICAST_IF**

Sets or obtains the IPv4 interface address used for sending outbound multicast datagrams from the socket application. This is an IPv4-only socket option.

**Note:** Multicast datagrams can be transmitted only on one interface at a time.

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

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

**IP_MULTICAST_TTL**

Sets or obtains the IP time-to-live of outgoing multicast datagrams. The default value is ‘01’x, meaning that multicast is available only to the local subnet. This is an IPv4-only socket option.

For setsockopt(), set optval to a value in the range X’00’ - X’FF’ specifying the time to live (TTL). optval is a 1-byte field.

For getsockopt(), optval contains a value in the range X’00’ - X’FF’, indicating TTL. optval is a 1-byte field.

**IP_MULTICAST_LOOP**

Controls or determines if a copy of multicast datagrams is looped back for
multicast datagrams sent to a group to which the sending host itself belongs. The default is to loop the datagrams back. This is an IPv4-only socket option.

For setsockopt(), set `optval` to 1 to enable and set to 0 to disable.

For getsockopt(), `optval` contains a 1 when enabled and contains a 0 when disabled.

**IP_UNBLOCK_SOURCE**
Enables an application to unblock a previously blocked source for a given IPv4 source multicast group. An interface and a source address must be specified with this option. This socket option applies only to IPv4.

For the setsockopt() function, set the `optval` value to the ip_mreq_source structure as defined in the in.h header. The ip_mreq_source structure contains the following:
- 4-byte IPv4 multicast address
- 4-byte IPv4 source address
- 4-byte IPv4 interface address

This option cannot be specified with the getsockopt() function.

The following options are recognized at the IPPROTO_IPV6 level:

**Option Description**

**IPV6_ADDR_PREFERENCES**
Sets or retrieves the IPv6 address preferences to be used when selecting the source address for the specified AF_INET6 socket. Possible values are:

- **IPV6_PREFER_SRC_HOME (x'00000001')**
  A home IPv6 address is preferred over a care-of IPv6 address.

- **IPV6_PREFER_SRC_COA (x'00000002')**
  A care-of IPv6 address is preferred over a home IPv6 address.

- **IPV6_PREFER_SRC_TMP (x'00000004')**
  A temporary IPv6 address is preferred over a public IPv6 address.

- **IPV6_PREFER_SRC_PUBLIC (x'00000008')**
  A public IPv6 address is preferred over a temporary IPv6 address.

- **IPV6_PREFER_SRC_CGA (x'00000010')**
  A cryptographically generated IPv6 address is preferred over a non-cryptographically generated IPv6 address.

- **IPV6_PREFER_SRC_NONCGA (x'00000020')**
  A non-cryptographically generated IPv6 address is preferred over a cryptographically generated IPv6 address.

For setsockopt(), contradictory flags such as IPV6_PREFER_SRC_CGA and IPV6_PREFER_SRC_NONCGA result in the return code -1 and the errno EINVAL (121).

**IPV6_JOIN_GROUP**
Controls the reception of multicast packets and specifies that the socket join a multicast group. This is an IPv6-only socket option.

For setsockopt(), set `optval` to the ipv6_mreq structure as defined in in.h. The ipv6_mreq structure contains a 16-byte IPv6 multicast address followed by a 4-byte IPv6 interface index number. If the interface number is 0, the stack chooses the local interface.
This cannot be specified with getsockopt().

**IPV6_LEAVE_GROUP**
Controls the reception of multicast packets and specify that the socket leave a multicast group. This is an IPv6-only socket option.

For setsockopt(), set `optval` to the `ipv6_mreq` structure as defined in `in.h`. The `ipv6_mreq` structure contains a 16-byte IPv6 multicast address followed by a 4-byte IPv6 interface index number. If the interface number is 0, then the stack chooses the local interface.

This cannot be specified with getsockopt().

**IPV6_MULTICAST_HOPS**
Sets or obtains the hop limit used for outgoing multicast packets. This is an IPv6-only socket option.

For setsockopt(), set `optval` to a value in the range 0 - 255, specifying the multicast hops. If `optval` is not specified or is set to 0, the default is 1 hop. If `optval` is set to a -1, the stack default hop is used.

**Rule:** An application must be APF authorized to enable it to set the hop limit value above the system defined hop limit value. The CICS application cannot execute as APF authorized.

For getsockopt(), `optval` contains a value in the range 0 - 255, indicating the number of multicast hops.

**IPV6_MULTICAST_IF**
Sets or obtains the index of the IPv6 interface used for sending outbound multicast datagrams from the socket application. This is an IPv6-only socket option.

For setsockopt(), set `optval` to a value that contains an IPv6 interface index.

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

**IPV6_MULTICAST_LOOP**
Controls or determines whether a multicast datagram is looped back on the outgoing interface by the IP layer for local delivery when datagrams are sent to a group to which the sending host itself belongs. The default is to loop multicast datagrams back. This is an IPv6-only socket option.

For setsockopt(), set `optval` to 1 to enable and set to 0 to disable.

For getsockopt(), `optval` contains a 1 when enabled and contains a 0 when disabled.

**IPV6_UNICAST_HOPS**
Sets or obtains the hop limit used for outgoing unicast IPv6 packets. This is an IPv6-only socket option.

For setsockopt(), set `optval` to a value in the range 0 - 255, specifying the unicast hops. If `optval` is not specified or is set to 0, the default is 1 hop. If `optval` is set to a -1, the stack default hop is used.

**Rule:** An application must be APF authorized to enable it to set the hop limit value above the system defined hop limit value. The CICS application cannot execute as APF authorized.

For getsockopt(), `optval` contains a value in the range 0 - 255 indicating the number of unicast hops.

**IPV6_V6ONLY**
Sets or determines whether the socket is restricted to send and receive only
IPv6 packets. The default is to not restrict the sending and receiving of only IPv6 packets. This is an IPv6-only socket option.

For setsockopt(), set optval to 1 to enable and set to 0 to disable.

For getsockopt(), optval contains a 1 when enabled and contains a 0 when disabled.

The following options are recognized at the IPPROTO_IP and IPPROTO_IPV6 level:

Option Description

MCAST_BLOCK_SOURCE
Enables an application to block multicast packets that have a source address that matches the given source address. An interface index and a source address must be specified with this option. The specified multicast group must have been joined previously.

For the setsockopt() function, set the optval value to the group_source_req structure as defined in the in.h header. The group_source_req structure contains the following:
- 4-byte interface index number
- Socket address structure of the multicast address
- Socket address structure of the source address

This option cannot be specified with the getsockopt() function.

MCAST_JOIN_GROUP
Enables an application to join a multicast group on a specific interface. An interface index must be specified with this option. The stack chooses a default interface if the interface index 0 is specified. Only applications that want to receive multicast datagrams need to join multicast groups.

For the setsockopt() function, set the optval value to the group_req structure as defined in the in.h header. The group_req structure contains the following:
- 4-byte interface index number
- Socket address structure of the multicast address

This option cannot be specified with the getsockopt() function.

MCAST_JOIN_SOURCE_GROUP
Enables an application to join a multicast group on a specific interface and a source address. An interface index and the source address must be specified with this option. The stack chooses a default interface if the interface index 0 is specified. Only applications that want to receive multicast datagrams need to join source multicast groups.
For the setsockopt() function, set the optval value to the group_source_req structure as defined in the in.h header. The group_source_req structure contains the following:

- 4-byte interface index number
- Socket address structure of the multicast address
- Socket address structure of the source address

This option cannot be specified with the getsockopt() function.

**MCAST_LEAVE_GROUP**

Enables an application to exit a multicast group or to exit a multicast group and drop all sources.

For the setsockopt() function, set the optval value to the group_req structure as defined in the in.h header. The group_req structure contains the following:

- 4-byte interface index number
- Socket address structure of the multicast address

This option cannot be specified with the getsockopt() function.

**MCAST_LEAVE_SOURCE_GROUP**

Enables an application to exit a source multicast group on a specific interface and a source address.

For the setsockopt() function, set the optval value to the group_source_req structure as defined in the in.h header. The group_source_req structure contains the following:

- 4-byte interface index number
- Socket address structure of the multicast address
- Socket address structure of the source address

This option cannot be specified with the getsockopt() function.

**MCAST_UNBLOCK_SOURCE**

Enables an application to unblock a previously blocked source for a given multicast group. An interface index and a source address must be specified with this option.

For the setsockopt() function, set the optval value to the group_source_req structure as defined in the in.h header. The group_source_req structure contains the following:

- 4-byte interface index number
- Socket address structure of the multicast address
- Socket address structure of the source address

This option cannot be specified with the getsockopt() function.

The following options are recognized at the TCP level:

**TCP_KEEPALIVE**

For setsockopt, the TCP_KEEPALIVE socket option specifies a socket-specific timer value which remains in effect until specified by SETSOCKOPT or until the socket is closed. Valid values are in the range 0 - 2147460 seconds; if a value greater than the allowed range is specified, 2147460 seconds is used. For the getsockopt call, the TCP_KEEPALIVE socket option returns the specific timer value in seconds in effect for the given socket, or 0 if TCP_KEEPALIVE timing is not active. See [z/OS](z/OS V2R1.0 Communications Server: IP CICS Sockets Guide)
TCP_NODELAY
For setsockopt, toggles the use of the Nagle algorithm (RFC 896) for all
data sent over the socket. Under most circumstances, TCP sends data when
it is presented. However, when outstanding data has not yet been
acknowledged, TCP gathers small amounts of output to be sent in a single
packet after an acknowledgment is received. For interactive applications,
such as ones that send a stream of mouse events which receive no replies,
this gathering of output can cause significant delays. For these types of
applications, disabling the Nagle algorithm improves response time. When
the Nagle algorithm is disabled, TCP can send small amounts of data
before the acknowledgment for previously sent data is received.

For getsockopt, returns the setting of the Nagle algorithm for the socket.
When optval is 0, the Nagle algorithm is enabled and TCP waits to send
small packets of data until the acknowledgment for the previous data is
received. When optval is not 0, the Nagle algorithm is disabled and TCP
can send small packets of data before the acknowledgment for previously
sent data is received.

The following options are recognized at the socket level:

SO_BROADCAST
Toggles the ability to broadcast messages. If this option is enabled, it
allows the application to send broadcast messages over s, if the interface
specified in the destination supports the broadcasting of packets. This
option has no meaning for stream sockets.

SO_ERROR
This cannot be specified with setsockopt(). It returns any pending error on
the socket and clears the error status. It can be used to check for
asynchronous errors on connected datagram sockets or for other
asynchronous errors (errors that are not returned explicitly by one of the
socket calls).

SO_KEEPALIVE
Sets or determines whether the keepalive mechanism periodically sends a
packet on an otherwise idle connection for a stream socket. The default is
disabled. When activated, the keepalive mechanism periodically sends a
packet on an otherwise idle connection. If the remote TCP does not
respond to the packet or to retransmissions of the packet, the connection is
closed with the error ETIMEDOUT.

SO_LINGER
Lingers on close if data is present. When this option is enabled and there is
unsent data present when close() is called, the calling application is
blocked during the close() call until the data is transmitted or the
connection has timed out. If this option is disabled, the TCP/IP address
space waits to try to send the data. Although the data transfer is usually
successful, it cannot be guaranteed, because the TCP/IP address space
waits a finite amount of time trying to send the data. The close() call
returns without blocking the caller.

Note: If you set a 0 linger time, the connection cannot close in an orderly
manner, but stops, resulting in a RESET segment being sent to the
connection partner. Also, if the aborting socket is in nonblocking mode, the
close call is treated as though no linger option had been set.

Chapter 7. C language application programming 211
SO_OOBINLINE
Toggles reception of out-of-band data. When this option is enabled, it causes out-of-band data to be placed in the normal data input queue as it is received, making it available to recvfrom() without having to specify the MSG_OOB flag in the call. When this option is disabled, it causes out-of-band data to be placed in the priority data input queue as it is received, making it available to recvfrom(), and only by specifying the MSG_OOB flag in that call.

SO_RCVTIMEO
Use this option to set or determine the maximum amount of time a receive-type function can wait before it completes. If a receive-type function has blocked for this much time without receiving data, it returns with an errno set to EWOULDBLOCK. The default for this option is 0, which indicates that a receive-type function does not time out.

When the MSG_WAITALL flag (stream sockets only) is specified, the timeout takes precedence. The receive-type function might return the partial count. See the explanation of the MSG_WAITALL flag parameter in “recv() call parameters” on page 231 and “recvfrom() call” on page 231.

For setsockopt(), this value accepts a timeval structure; the number of seconds and microseconds specify the limit on how long to wait for a receive-type function to complete. The timeval structure contains the number of seconds and microseconds specified as fullword binary numbers. The seconds can be a value in the range 0 - 2678400 (equal to 31 days), and the microseconds can be a value in the range 0 - 1000000 (equal to 1 second). Although the timeval structure can be specified using microsecond granularity, the internal TCP/IP timers used to implement this function have a granularity of approximately 100 milliseconds.

The following receive-type functions are included:
- read()
- recv()
- recvfrom()

SO_REUSEADDR
Toggles local address reuse. When enabled, this option allows local addresses that are already in use to be bound. This alters the normal algorithm used in the bind() call. Normally, the system checks at connect time to ensure that the local address and port do not have the same foreign address and port. The error EADDRINUSE is returned if the association already exists. If you require multiple servers to bind to the same port and listen on INADDR_ANY or the IPv6 unspecified address (in6addr_any), see to the SHAREPORT option on the PORT statement in TCPIP.PROFILE.

SO_SNDBUF
Applies to getsockopt() only. Returns the size of the data portion of the TCP/IP send buffer in optval. The size of the data portion of the send buffer is protocol-specific, based on the DATABUFFERPOOLSIZEx statement in the PROFILE.TCPIP data set. The value is adjusted to allow for protocol header information.

SO_SNDTIMEO
Use this option to set or determine the maximum amount of time a send-type function can remain blocked before it completes. If a send-type function has blocked for this time, it returns with a partial count, or it
returns with errno set to EWOULDBLOCK if no data is sent. The default for this option is 0, which indicates that a send-type function does not time out.

For setsockopt(), this value accepts a timeval structure; the number of seconds and microseconds specify the limit on how long to wait for a send-type function to complete. The timeval structure contains the number of seconds and microseconds specified as fullword binary numbers. The seconds can be a value in the range 0 - 2,678,400 (equal to 31 days), and the microseconds can be a value in the range 0 - 1,000,000 (equal to 1 second). Although the timeval structure can be specified using microsecond granularity, the internal TCP/IP timers used to implement this function have a granularity of approximately 100 milliseconds.

The following send type functions are included:
- send()
- sendto()
- write()

**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.

**getsockopt(), setsockopt() calls return values**

The value 0 indicates success; the value -1 indicates an error. To determine which error occurred, check the `errno` global variable, which is set to a return code. Possible codes include:

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

**EFAULT**
Using `optval` and `optlen` parameters results in an attempt to access storage outside the caller’s address space.

**ENOPROTOOPT**
The `optname` parameter is unrecognized, or the `level` parameter is not SOL_SOCKET.

**getsourcefilter() call**

Obtains a list of the IPv4 or IPv6 source addresses that comprise the source filter, along with the current mode on a given interface and a multicast group for a socket. The source filter can either include or exclude the set of source addresses, depending on the filter mode (MCAST_INCLUDE or MCAST_EXCLUDE).

**getsourcefilter() call format**

This call has the following format:
getsourcelfilter() call parameters

s  The socket descriptor.

interface  The interface index of the interface.

group  A pointer to either a sockaddr_in structure for IPv4 addresses or a sockaddr_in6 structure for IPv6 addresses that holds the IP multicast address of the group.

grouplen  The length of the sockaddr_in or sockaddr_in6 structure.

fmode  A pointer to an integer that contains the filter mode on a successful return. The value of the filter mode can be either MCAST_INCLUDE or MCAST_EXCLUDE.

numsrc  On input, a pointer to the number of source addresses that can fit in the array specified by the slist parameter. On output, a pointer to the total number of source addresses in the filter.

slist  A pointer to an array of IP addresses that is either included or excluded, depending on the filter mode. If a numsrc value 0 was specified on input, you can specify a NULL pointer.

On return, the numsrc value is always updated to be the total number of sources in the filter; the slist pointer points to an array that holds as many source addresses as fit, which is the minimum of the array size specified by the input numsrc value and the total number of sources in the filter.

If the application is not aware of the size of the source list before processing, it can make a reasonable guess (for example, 0). When the process completes, if the numsrc is large, the operation can be repeated with a large buffer.

getsourcelfilter() call return values

When successful, the value 0 is returned. When an error has occurred, the value -1 is returned and the errno value is one of the following:

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

EAFNOSUPPORT  The address family of the sockaddr value is not AF_INET or AF_INET6.

EPROTOTYPE  The socket protocol type is not correct.

EADDRNOTAVAIL  The tuple consisting of socket, interface, and multicast group values does not exist, or the specified interface address is not multicast capable.
EINVAL
The socket address family of an input parameter is not correct or the socket specified by the s parameter already requested multicast setsockopt options. For more information, see the z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference.

ENOMEM
Insufficient storage is available to supply the array.

ENXIO
The interface index specified by the interface parameter does not exist.

givesocket() call

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() call format

This call has the following format:
givesocket() call parameters

s
The descriptor of a socket to be given to another application.

clientid
A pointer to a clientid structure specifying the target program to whom the
socket is to be given. You should fill the structure as follows:

domain
Set to either AF_INET (a decimal 2) or AF_INET6 (a decimal 19).

Rule: An AF_INET socket can be given only to an AF_INET
takesocket(). An AF_INET6 socket can be given only to an
AF_INET6 takesocket(). EBADF is set if the domain does not
match.

name
This is the child server’s address space name, left-justified and
padded with blanks. The child server can run in the same address
space as the parent server. In this case, the field is set to the parent
server’s address space.

subtaskname
Blanks.

reserved
Binary zeros.

givesocket() call return Values

The value 0 indicates success; the value -1 indicates an error. To determine which
error occurred, check the errno global variable, which is set to a return code.
Possible codes include:

EBADF
The s parameter is not a valid socket descriptor, the socket has already
been given, or the socket domain is not AF_INET or AF_INET6.

EBUSY
listen() has been called for the socket.

EFAULT
Using the clientid parameter as specified results in an attempt to access
storage outside the caller’s address space.

EINVAL
The clientid parameter does not specify a valid client identifier.

ENOTCONN
The socket is not connected.

EOPNOTSUPP
The socket type is not SOCK_STREAM.

if_free nameindex() call

The if_free nameindex() function is used to release the array storage obtained by
the if_nameindex() function.
if_freenameindex() call format

This call has the following format:

```c
#include <manifest.h> (non-reentrant programs only)
#include <cmannfies.h> (reentrant programs only)
#include <if.h>

void if_freenameindex(struct if_nameindex *ptr)
```

**if_freenameindex() call parameters**

- `ptr` A pointer that contains the address of the array of structures returned by the if_nameindex() function.

**if_freenameindex() call return values**

No return value is defined.

if_indextoname() call

The if_indextoname() function returns an interface name when given an interface index.

**if_indextoname() call format**

This call has the following format:
if_indextoname() call parameters

ifindex
Storage that contains an interface index.

ifname
A buffer that contain the name of the index value specified in the ifindex parameter.

if_indextoname() call return values

Possible return values include:

EINVAL
The ifindex parameter was zero, or the ifname parameter was NULL, or both.

ENOMEM
Insufficient storage is available to obtain the information for the interface name.

ENXIO
The ifindex does not yield an interface name.

if_nameindex() call

The if_nameindex() function is used to obtain a list of interface names and their corresponding indices. The if_nameindex() function is not supported by IPv4-only stacks. However, if a mixture of IPv4-only and IPv4 and IPv6 stacks are active under CINET, CINET assigns a single interface index to the IPv4-only stack. This allows applications using IPv6 sockets to target an IPv4-only stack but does not allow the selection of a particular interface on an IPv4-only stack. Not all interfaces are returned in the output from if_nameindex(). VIPA interfaces are not returned. Interfaces that have never been activated are not returned.

if_nameindex() call format

This call has the following format:
if_nameindex() call parameters

There are no input parameters as the if_nameindex() function returns a pointer to an array of structures that contains information about each system interface. Check the if_nameindex structure in if.h for the format of the returned data.

if_nameindex() call return values

When successful, if_nameindex() returns a pointer to an array of if_nameindex structures. Upon failure, if_nameindex() returns NULL and sets errno to the following:

ENOMEM
Insufficient storage is available to supply the array.

if_nametoindex() call

The if_nametoindex() function returns an interface index when given an interface name.

if_nametoindex() call format

This call has the following format:
if_nametoindex() call parameters

ifname
A pointer to null terminated storage that contains the interface name. If the interface specified by ifname does not exist then 0 is returned.

if_nametoindex() call return values
When successful, if_nametoindex() returns the interface index corresponding to the interface name ifname. Upon failure, if_nametoindex() returns zero and sets errno to one of the following:

EINVAL
A parameter was not specified. The ifname parameter was NULL.

ENOMEM
Insufficient storage is available to obtain the information for the interface name.

ENXIO
The specified interface name provided in the ifname parameter does not exist.

inet_ntop() call
Converts numeric IP addresses to their printable form.

inet_ntop() call format
This call has the following format:
const char * inet_ntop(int af, const void *src, char *dst, socklen_t size)

**inet_ntop() call parameters**

*af*  
The address family of the IP address being converted specified as AF_INET or AF_INET6.

*src*  
A pointer to the IP address, in network byte order, to be converted to presentable form.

*dst*  
A pointer to storage used to contain the converted IP address.

*size*  
The size of the IP address pointed to by the *src* parameter.

**inet_ntop() call return values**

If successful, *inet_ntop()* returns a pointer to the buffer that contains the converted address.

If unsuccessful, *inet_ntop()* returns NULL and sets *errno* to one of the following values:

- **EAFNOSUPPORT**  
The address family specified in *af* is unsupported.

- **ENOSPC**  
The destination buffer *size* is too small.

**inet_pton() call**

Converts IP addresses from presentable text form to numeric form.

**inet_pton() call format**

This call has the following format:
inet_pton() call parameters

*af*  
The address family of the IP address being converted, specified as AF_INET or AF_INET6.

*src*  
A pointer to the IP address, in presentable text form, to be converted to numeric form.

*dst*  
A pointer to storage used to contain the converted IP address. The converted address is in numeric form and network byte order.

inet_pton() call return values

If successful, inet_pton() returns 1 and stores the binary form of the Internet address in the buffer pointed to by *dst*.

If unsuccessful because the input buffer pointed to by *src* is not a valid string, inet_pton() returns 0.

If unsuccessful because the *af* argument is unknown, inet_pton() returns -1 and sets *errno* to the following value:

EAFNOSUPPORT
The address family specified in *af* is unsupported.

inet6_is_srcaddr() call

The inet6_is_srcaddr() call tests whether the input IP address matches an IP address in the node that conforms to all IPV6_ADDR_PREFERENCES flags specified in the call. You can use this call with IPv6 addresses or with IPv4-mapped IPv6 addresses.

You can use this call to test local IP addresses to verify that these addresses have the characteristics required by your application.


inet6_is_srcaddr() call format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanfies.h> (reentrant programs only)
#include <inet.h>
short inet6_is_srcaddr(struct sockaddr_in6 *name, uint32_t flags)
```

inet6_is_srcaddr() parameters

*name*  
Specifies the AF_INET6 socket address structure for the address that is to be tested.
Requirement: You must specify an AF_INET6 address. You can specify an IPv6 address or an IPv4-mapped IPv6 address. The format of the name buffer is expected to be sockaddr_in6 as defined in the header file in.h. The format of the structure is shown in Table 19 on page 167.

The IPv6 socket address structure specifies the following fields:

- **sin6_family**
  - This field must be set to AF_INET6.

- **sin6_port**
  - A halfword binary field. This field is ignored by inet6_is_srcaddr() processing.

- **sin6_flowinfo**
  - A fullword binary field. This field is ignored by inet6_is_srcaddr() processing.

- **in6_addr.sin6_addr**
  - A 16-byte binary field that is set to the 128-bit IPv6 Internet address (network byte order) to be tested.

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

- **sin6_scope_id**
  - A fullword binary field that identifies a set of interfaces as being appropriate for the scope of the address specified in the in6_addr.sin6_addr field. The value 0 indicates that the sin6_scope_id field does not identify the set of interfaces to be used.

  **Requirements:** The sin6_scope_id value must be nonzero if the address is a link-local address. For all other address scopes, sin6_scope_id must be set to 0.

- **flags**
  - A fullword binary field containing one or more IPV6_ADDR_PREFERENCES flags. The following table defines the valid IPV6_ADDR_PREFERENCES flags.

<table>
<thead>
<tr>
<th>Flag name</th>
<th>Binary value</th>
<th>Decimal value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPV6_PREFER_SRC_HOME</td>
<td>x'00000001'</td>
<td>1</td>
<td>Test whether the input IP address is a home address.¹</td>
</tr>
<tr>
<td>IPV6_PREFER_SRC_COA</td>
<td>x'00000002'</td>
<td>2</td>
<td>Test whether the input IP address is a care-of address.²</td>
</tr>
<tr>
<td>IPV6_PREFER_SRC_TMP</td>
<td>x'00000004'</td>
<td>4</td>
<td>Test whether the input IP address is a temporary address.</td>
</tr>
<tr>
<td>IPV6_PREFER_SRC_PUBLIC</td>
<td>x'00000008'</td>
<td>8</td>
<td>Test whether the input IP address is a public address.</td>
</tr>
</tbody>
</table>
Flag name | Binary value | Decimal value | Description
---|---|---|---
IPV6_PREFER_SRC_CGA | x'00000010' | 16 | Test whether the input IP address is cryptographically generated.\(^2\)
IPV6_PREFER_SRC_NONCGA | x'00000020' | 32 | Test whether the input IP address is not cryptographically generated.\(^1\)

**Note:**
1. Any valid IP address that is known to the stack satisfies this flag.
2. z/OS Communications Server does not support this type of address. The call always returns FALSE when this flag is specified with a valid IP address that is known to the stack.

**Tips:**
- The samples SEZAINST(EZACOBOL) and SEZAINST(CBLOCK) contain mappings for these flags.
- Some of these flags are contradictory. For example:
  - The flag IPV6_PREFER_SRC_HOME contradicts the flag IPV6_PREFER_SRC_COA.
  - The flag IPV6_PREFER_SRC_CGA contradicts the flag IPV6_PREFER_SRC_NONCGA.
  - The flag IPV6_PREFER_SRC_TMP contradicts the flags IPV6_PREFER_SRC_PUBLIC.

**Result:** If you specify contradictory flags in the call, the result is FALSE.

**inet6_is_srcaddr() return values**

**Value description:**

0 | FALSE
---|---
The call was successful, and the result is FALSE. The input AF_INET6 address corresponds to an IP address on the node, but does not conform to one or more IPV6_ADDR_PREFERENCES flags specified in the call.

1 | TRUE
---|---
The call was successful, and the result is TRUE. The input AF_INET6 address corresponds to an IP address on the node, and conforms to all IPV6_ADDR_PREFERENCES flags specified in the call.

-1 | Check ERRNO for an error code.
---|---
See Appendix B, “Return codes,” on page 455 for information about ERRNO values.

**initapi() call**

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

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
int initapi(int max_sock, char *subtaskid)
```

initapi() call parameters

`max_sock`

The maximum number of sockets requested. This value cannot exceed 2000. The minimum value is 50.

`subtaskid`

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

Using the letter L as the last character in the subtask parameter causes the tasking mechanism to assume that the CICS transaction is a listener. The task mechanism schedules the transaction using a non-reusable subtask by way of MVS attach processing when OTE=NO. This value has no effect when OTE=YES.

initapi() call return values

A positive value indicates success; a value of -1 indicates an error. To determine which error occurred, check the `errno` global variable, which is set to a return code.

ioctl() call

The ioctl() call controls the operating characteristics of sockets. This call can issue a command to do any of the following:

- Set or clear nonblocking input and output for a socket.
- Get the number of immediately readable bytes for the socket.
- Query whether the current location in the data input is pointing to out-of-band data.
- Get the IPv6 home interface addresses.
- Get the network interface address.
- Get the network interface broadcast address.
- Get the network interface configuration.
- Get the network interface names and indices.
- Control Application Transparent Transport Layer Security (AT-TLS) for a connection
- Retrieve connection routing information and partner security credentials

ioctl() call format

This call has the following format:

```
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <bsdtypes.h>
#include <ioct1.h>
#include <ezbpinfc.h>
```


```c
#include <ezbztlsc.h>
#include <ezbyaplc.h>
#include <rtrouteh.h>
#include <if.h>

int ioctl(int s, unsigned long cmd, char *arg)

ioctl() call parameters
s The socket descriptor.
cmd and arg

cmd is the command to perform; arg is a pointer to the data associated with cmd. The following are valid ioctl() commands:

FIONBIO
Sets or clears nonblocking input and output for a socket. arg is a pointer to an integer. If the integer is 0, the socket is in nonblocking mode. Otherwise, the socket is set for nonblocking input/output.

FIONREAD
Gets the number of immediately readable bytes for the socket. arg is a pointer to an integer. Sets the value of the integer to the number of immediately readable characters for the socket.

SIOCATMARK
Queries whether the current location in the data input is pointing to out-of-band data. The arg parameter is a pointer to an integer. The parameter sets the argument to 1 if the socket points to a mark in the data stream for out-of-band data. Otherwise, it sets the argument to 0.

SIOCGHOMEIF6
Get the IPv6 home interfaces. The arg parameter is a pointer to a NetConfHdr structure, as defined in ioctl.h. A pointer to a HomeIf structure that contains a list of home interfaces is returned in the NetConfHdr pointed to by the argument. To request OSM interfaces the application must have READ authorization to the EZB.OSM.sysname.tcpname resource.

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.
```
SIOCGIFMTU
Gets the IPv4 network interface MTU (maximum transmission unit). The arg parameter is a pointer to an ifreq structure, as defined in the if.h file. The interface MTU is returned in the argument.

SIOCGPARTNERINFO
Provides an interface for an application to retrieve security information about its partner. The arg parameter is a pointer to a PartnerInfo structure, as defined by the EZBPINFC header file in the SEZANMAC dataset. For more information about using the SIOCGPARTNERINFO ioctl, see z/OS Communications Server: IP Programmer’s Guide and Reference.

Restriction: The SIOCGPARTNERINFO ioctl command is not called by the IBM listener.

Tip: If the partner end-point is the IBM Listener or a child server and partner security credentials were requested, then only the CICS address space information is returned on the SIOCGPARTNERINFO ioctl invocation.

SIOCSAPPLDATA
Enables an application to associate 40 bytes of user-specified application data with a TCP connection. Identifies socket endpoints in tools such as Netstat, SMF, or network management applications.

Requirement: When you issue the SIOCSAPPLDATA ioctl() function, ensure that the arg parameter contains a SetApplData structure as defined by the EZBYAPLC header file in the SEZANMAC dataset. See z/OS Communications Server: IP Programmer’s Guide and Reference for more information about programming the SIOCSAPPLDATA IOCTL.

SetAD_buffer
The user-defined application data comprises 40 bytes of data that is used to identify the TCP connection with the IP CICS socket API sockets application. The application data can be displayed in the following ways:

- By requesting Netstat reports. The information is displayed conditionally using the modifier APPLDATA on the ALLC/-a and COnn/-c reports and unconditionally on the ALL/-A report. See the Netstat ALL/-A report, Netstat ALLConn/-a report, and Netstat COnn/-c report in z/OS Communications Server: IP System Administrator's Commands for more information about Netstat reports.

- In the SMF 119 TCP connection termination record. See TCP connection termination record (subtype 2) in z/OS Communications Server: IP Programmer’s Guide and Reference for more information about the application data written on the SMF 119 record.

- By network management applications. See Network management interfaces in z/OS Communications Server: IP Programmer’s Guide and Reference for more information about application data.
Applications using this ioctl need to consider the following guidelines:

- The application is responsible for documenting the content, format, and meaning of the ApplData strings that it associates with sockets it owns.
- The application should uniquely identify itself with printable EBCDIC characters at the beginning of the string. Strings beginning with 3-character IBM product identifiers, such as EZA or EZB, are reserved for IBM use. IBM product identifiers begin with a letter in the range A - I.
- Printable EBCDIC characters should be used for the entire string to enable searching with Netstat filters.

**Tip:** Separate application data elements with a blank for easier reading.

**SIOCSPARTNERINFO**

The SIOCSPARTNERINFO ioctl sets an indicator to retrieve the partner security credentials during connection setup and saves the information, enabling an application to issue a SIOCSPARTNERINFO ioctl without suspending the application, or at least minimizing the time to retrieve the information. The SIOCSPARTNERINFO ioctl must be issued prior to the SIOCSPARTNERINFO ioctl. The `arg` parameter is a pointer to a constant value, PI_REQTYPE_SET_PARTNERDATA, as defined by the EZBPINFC header file in the SEZANMAC dataset. For more information about using the SIOCSPARTNERINFO ioctl, see z/OS Communications Server: IP Programmer’s Guide and Reference.

**Restriction:** The SIOCSPARTNERINFO ioctl command is not called by the IBM listener.

**SIOCTTLSCTL**

Controls Application Transparent Transport Layer Security (AT-TLS) for the connection. The `arg` parameter is a pointer to a TTLS_IOCTL structure, as defined in ezbztlsch.h. If a partner certificate is requested, the TTLS_IOCTL must include a pointer to additional buffer space and the length of that buffer. Information is returned in the TTLS_IOCTL structure. If a partner certificate is requested and one is available, it is returned in the additional buffer space. For more usage information, see z/OS Communications Server: IP Programmer’s Guide and Reference.

**ioctl() call return values**

The value 0 indicates success; the value -1 indicates an error. To determine which error occurred, check the `errno` global variable, which is set to a return code. Possible codes include:

**EBADF**

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

**EINVAL**

The request is not correct or not supported.
listen() call

The listen() call performs two tasks for a specified stream socket:
1. Completes the necessary binding if bind() has not been called for the socket.
2. Creates a connection request queue of a specified length to queue incoming connection requests.

The listen() call indicates a readiness to accept client connection requests. It transforms an active socket into a passive socket. A passive socket can never be used as an active socket to initiate connection requests.

Calling listen() is the third of four steps that a server performs to accept a connection. It is called after allocating a stream socket with socket(), and after binding a name to the socket with bind(). It must be called before calling accept() to accept a connection request from a client.

listen() call format

This call has the following format:

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

int listen(int s, int backlog)
```

listen() call parameters

- **s**  The socket descriptor.
- **backlog**  Defines the maximum length for the queue of pending connections.

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

listen() call return values

The value 0 indicates success; the value -1 indicates an error. To determine which error occurred, check the **errno** global variable, which is set to a return code.
Possible codes include:

- **EBADF**  The **s** parameter is not a valid socket descriptor.
- **EOPNOTSUPP**  The **s** parameter is not a socket descriptor that supports the listen() call.

read() call

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.

**read() call format**

This call has the following format:

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

int read(int s, char *buf, int len)
```

**read() call 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.

**read() call return values**

If successful, the number of bytes copied into the buffer is returned. The value 0 indicates that the connection is closed. The value -1 indicates an error. To determine which error occurred, check the `errno` global variable, which is set to a return code. Possible codes include:

- **EBADF**
  - `s` is not a valid socket descriptor.
- **EFAULT**
  - Using the `buf` and `len` parameters results in an attempt to access storage outside the caller’s address space.
- **EWOULDBLOCK**
  - `s` is in nonblocking mode, and data is not available to read.

**recv() call**

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.

**recv() call format**

This call has the following format:
recv() call 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, MSG_OOB, MSG_PEEK, or MSG_WAITALL.
  - **MSG_OOB**: Receive out-of-band (OOB) data (stream sockets only). Even if the OOB flag is not set, out-of-band data can be read if the SO_OOBINLINE option is set for the socket.
  - **MSG_PEEK**: Peek at the data, but do not destroy the data. If the peek flag is set, the next receive operation reads the same data.
  - **MSG_WAITALL**: Requests that the function block until the full amount of data requested can be returned (stream sockets only). The function might return a smaller amount of data if the connection is closed, an error is pending, or if the SO_RCVTIMEO value is set and the timer expired for the socket.

recv() call return values

If successful, the length of the message or datagram in bytes is returned. The value 0 indicates that the connection is closed. The value -1 indicates an error. To determine which error occurred, check the `errno` global variable, which is set to a return code. Possible codes include:

- **EBADF**: `s` is not a valid socket descriptor.
- **EFAULT**: Using the `buf` and `len` parameters results in an attempt to access storage outside the caller's address space.
- **EWOULDBLOCK**: `s` is in nonblocking mode, and data is not available to read.

recvfrom() call

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.

**recvfrom() call format**

This call has the following format:

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

int recvfrom(int s, char *buf,
int len, int flags,
struct sockaddr *name, int *namelen)
```

**recvfrom() call 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, MSG_OOB, MSG_PEEK, or MSG_WAITALL.
  - **MSG_OOB**  
    Receive out-of-band data (stream sockets only). Even if the OOB flag is not set, out-of-band data can be read if the SO-OOBINLINE option is set for the socket.
  - **MSG_PEEK**  
    Peek at the data, but do not destroy data. If the peek flag is set, the next receive operation reads the same data.
  - **MSG_WAITALL**  
    Requests that the function block until the full amount of data requested can be returned (stream sockets only). The function might return a smaller amount of data if the connection is closed, an error is pending, or if the SO_RCVTIMEO value is set and the timer expired for the socket.
- **name**  
A pointer to a socket address structure from which data is received. If `name` is a nonzero value, the source address is returned.

The following fields are used to define the IPv4 socket address structure of the socket that sent the data.

- **sin_family**  
  This field is set to AF_INET.
- **sin_port**  
  Contains the port number of the sending socket.
- **in_addr.sin_addr**  
  Contains the 32-bit IPv4 Internet address, in network byte order, of the sending socket.
- **sin_zero**  
  This field is not used and is set to all zeros.

The following fields are used to define the IPv6 socket address structure of the socket that sent the data.
**sin6_family**

This field is set to AF_INET6.

**sin6_port**

Contains the port number bound of the sending socket.

**sin6_flowinfo**

Contains the traffic class and flow label. The value of this field is undefined.

**in6_addr.sin6_addr**

Contains the 128-bit IPv6 Internet address, in network byte order, of the sending socket.

**sin6_scope_id**

Identifies a set of interfaces as appropriate for the scope of the address carried in the `in6_addr.sin6_addr` field. For a link scope `in6_addr.sin6_addr`, `sin6_scope_id` contains the link index for the `in6_addr.sin6_addr`. For all other address scopes, `sin6_scope_id` is undefined.

**namelen**

A pointer to an integer that contains the size of `name` in bytes. For an IPv4 socket address, the `namelen` parameter contains a decimal 16. For an IPv6 socket address, the `namelen` parameter contains a decimal 28.

**recvfrom() call return values**

If successful, the length of the message or datagram in bytes is returned. The value 0 indicates that the connection is closed. The value -1 indicates an error. To determine which error occurred, check the `errno` global variable, which is set to a return code. Possible codes include:

**EBADF**

`s` is not a valid socket descriptor.

**EFAULT**

Using the `buf` and `len` parameters results in an attempt to access storage outside the caller’s address space.

**EWOULDBLOCK**

`s` is in nonblocking mode, and data is not available to read.

**select() call**

The `select()` call is useful in processes where multiple operations can occur, and it is necessary for the program to be able to wait on one or several of the operations to complete.

For example, consider a program that issues a `read()` to multiple sockets whose blocking mode is set. Because the socket blocks on a `read()` call, only one socket could be read at a time. Setting the sockets nonblocking solves this problem, but requires polling each socket repeatedly until data became available. The `select()` call allows you to test several sockets and to execute a subsequent I/O call only when one of the tested sockets is ready, thereby ensuring that the I/O call does not block.
Defining which sockets to test

The select() call monitors for read operations, write operations, and exception operations:

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

Each socket is represented by a bit in a bit string. The bit strings are contained in 32-bit fullwords, numbered from right-to-left. The right-most bit represents socket 0, the leftmost bit represents socket 31, and so on. Thus, if the process uses 32 (or less) sockets, the bit string is one word long; if the process uses up to 64 sockets, the bit string is two words long, etc. You define which sockets to test by turning on the corresponding bit in the bit string.

Read operations calls:

Read operations include accept(), read(), recv(), or recvfrom() calls. A socket is ready to be read when data has been received for it, or when a connection request has occurred.

To test whether any of several sockets is ready for reading, set the appropriate bits in READFDS to ‘1’ before issuing the select() call. When the select() call returns, the corresponding bits in the READFDS indicate sockets ready for reading.

Write operations calls:

A socket is selected for writing (ready to be written) when:

- TCP/IP can accept additional outgoing data.
- A connection request is received in response to an accept() call.
- The socket is marked nonblocking, and a connect() cannot be completed immediately. In this case, ERRNO contains a value of 36 (EINPROGRESS). This is not an error condition.

A call to write(), send(), or sendto() blocks when the amount of data to be sent exceeds the amount of data TCP/IP can accept. To avoid this, you can precede the write operation with a select() call to ensure that the socket is ready for writing. After a socket is selected for write(), the program can determine the amount of TCP/IP buffer space available by issuing the getsockopt() call with the SO_SNDBUF option.

To test whether any of several sockets is ready for writing, set the WRITEFDS bits representing those sockets to 1 before issuing the select() call. When the select() call returns, the corresponding bits in the WRITEFDS indicate sockets ready for writing.

Exception operations for the select() call:
For each socket to be tested, the select() call can check for an existing exception condition. Two exception conditions are supported:

- The calling program (concurrent server) has issued a givesocket() command and the target child server has successfully issued the takesocket() call. When this condition is selected, the calling program (concurrent server) should issue close() to dissociate itself from the socket.
- A socket has received out-of-band data. On this condition, a READ returns the out-of-band data ahead of program data.

To test whether any of several sockets have an exception condition, set the EXCEPTFDS bits representing those sockets to 1. When the select() call returns, the corresponding bits in the EXCEPTFDS indicate sockets with exception conditions.

**NFDS parameter for the select() call:**

The select() call tests each bit in each string before returning results. For efficiency, the NFDS parameter can be used to specify the number of socket descriptors that need to be tested for any event type. The select() call tests only bits in the range 0 through the (NFDS-1) value.

**TIMEOUT parameter for the select() call:**

If the time specified in the TIMEOUT parameter elapses before any event is detected, the select() call returns, and RETCODE is set to 0.

**select() call format**

This call has the following format:
select() call parameters

nfds  The number of socket descriptors to check.

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

writefds  The pointer to a bit mask of descriptors to check for writing.

exceptfds  The pointer to a bit mask of descriptors to be checked for exceptional pending conditions.

timeout  The pointer to the time to wait for the select() call to complete. If timeout is a NULL pointer, a zero-valued timeval structure is substituted in the call. The zero-valued timeval structure causes TCP/IP stacks to poll the sockets and return immediately to the caller.

select() call return values

A positive value represents the total number of ready sockets in all bit masks. The value 0 indicates an expired time limit. The three bit masks indicate status (with one bit for each socket). A bit 1 indicates that the respective socket is ready; a bit 0 indicates that the respective socket is not ready. You can use the macro FD_ISSET with each socket to test its status.

The value -1 indicates an error. To determine which error occurred, check the errno global variable, which is set to a return code. Possible codes include:

EBADF  One of the bit masks specified an incorrect socket. FD_ZERO was probably not called to clear the bit mask before the sockets were set.

EFAULT  One of the bit masks pointed to a value outside the caller’s address space.

EINVAL  One of the fields in the timeval structure is not correct.

send() call

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.

#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <socket.h>
#include <bsdtypes.h>
#include <bsdtime.h>

int select(int nfds, fd_set *readfds,
fd_set *writefds, fd_set *exceptfds,
struct timeval *timeout)
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.

**send() call format**

This call has the following format:

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

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

**send() call parameters**

- **s** The socket descriptor.
- **msg** The pointer to the buffer that contains the message to transmit.
- **len** The length of the message pointed to by the `buf` parameter.
- **flags** The `flags` parameter is set by specifying one or more of the following flags. If more than one flag is specified, the logical OR operator (|) must be used to separate them.
  
  1. **MSG_OOB**
     
     Sends out-of-band data.
  
  2. **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.

**send() call return values**

A positive value represents the number of bytes sent. The value -1 indicates locally detected errors. When datagram sockets are specified, no indication of failure to deliver is implicit in a `send()` routine.

To determine which error occurred, check the `errno` global variable, which is set to a return code. Possible codes include:

1. **EBADF**
   
   `s` is not a valid socket descriptor.

2. **EFAULT**
   
   Using the `buf` and `len` parameters results in an attempt to access storage outside the caller’s address space.

3. **ENOBUSFS**
   
   Buffer space is not available to send the message.

4. **EWOULDDBLOCK**
   
   `s` is in nonblocking mode and data is not available to read.
sendto() call

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.

sendto() call format

This call has the following format:

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

int sendto(int s, char *msg, int len, int flags, struct sockaddr *to, int tolen)
```

sendto() call parameters

- `s`    The socket descriptor.
- `msg`  The pointer to the buffer that contains the message to transmit.
- `len`  The length of the message in the buffer pointed to by the `msg` parameter.
- `flags` A parameter that can be set to 0 or MSG_DONTROUTE.
- `MSG_DONTROUTE` The SO_DONTROUTE option is turned on for the duration of the operation. This is usually used only by diagnostic or routing programs.
- `to`   The address of the target socket address structure.

The following fields are used to define the IPv4 socket address structure the data is sent to.

- `sin_family` Must be set to AF_INET.
- `sin_port` Set to the port number bound to the socket.
- `in_addr.sin_addr` Set to the 32-bit IPv4 Internet address in network byte order.
- `sin_zero` This field is not used and must be set to all zeros.

The following fields are used to specify the IPv6 socket address structure the data is sent to.

- `sin6_family` Must be set to AF_INET6.
- `sin6_port` Set to the port number bound to the socket.
*sin6_flowinfo*

Used to specify the traffic class and flow label. This field must be set to zero.

*in6_addr.sin6_addr*

Set to the 128-bit IPv6 Internet address in network byte order.

*sin6_scope_id*

Used to identify a set of interfaces as appropriate for the scope of the address carried in the *in6_addr.sin6_addr* field. A value of zero indicates the *sin6_scope_id* does not identify the set of interfaces to be used, and might be specified for any address types and scopes. For a link scope *in6_addr.sin6_addr*, *sin6_scope_id* might specify a link index which identifies a set of interfaces. For all other address scopes, *sin6_scope_id* is undefined.

*tolen*

The size of the structure pointed to by *to*. For an IPv4 socket address, the *tolen* parameter contains a decimal 16. For an IPv6 socket address, the *tolen* parameter contains a decimal 28.

**sendto() call return values**

If positive, indicates the number of bytes sent. The value -1 indicates an error. No indication of failure to deliver is implied in the return value of this call when used with datagram sockets.

To determine which error occurred, check the *errno* global variable, which is set to a return code. Possible codes include:

**EBADF**

*s* is not a valid socket descriptor.

**EFAULT**

Using the *buf* and *len* parameters results in an attempt to access storage outside the caller’s address space.

**EINVAL**

tollen 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.

**ENOBUSFS**

Buffer space is not available to send the message.

**EWOULDBLOCK**

*s* is in nonblocking mode, and data is not available to read.

**setipv4sourcefilter() call**

Sets a list of the IPv4 source addresses that comprise the source filter, along with the current mode on a given interface and a multicast group for a socket. The source filter can either include or exclude the set of source addresses, depending on the filter mode (MCAST_INCLUDE or MCAST_EXCLUDE).

**setipv4sourcefilter() call format**

This call has the following format:


```
#include <manifest.h> (non-reentrant programs only)
#include <cmanfies.h> (reentrant programs only)
#include <netinet/in.h>
int setipv4sourcefilter (int s, struct in_addr interface,
                         struct in_addr group, uint32_t fmode,
                         uint32_t numsrc, struct in_addr *slist);
```

**setipv4sourcefilter() call parameters**

- **s**: The socket descriptor.
- **interface**: The local IP address of the interface.
- **group**: The IP multicast address of the group.
- **fmode**: An integer that contains the filter mode to be set. The value of the filter mode can be MCAST_INCLUDE or MCAST_EXCLUDE.
- **numsrc**: The number of source addresses in the slist array.
- **slist**: A pointer to an array of IP addresses that is either included or excluded, depending on the filter mode. If the numsrc value 0 was specified on input, you can specify a NULL pointer. A maximum of 64 IP addresses can be specified.

**setipv4sourcefilter() call return values**

When successful, the value 0 is returned. When an error occurs, the value -1 is returned and the errno value is one of the following:

- **EBADF**: The s parameter value is not a valid socket descriptor
- **EINVAL**: The interface or group parameter value is not a valid IPv4 address, the specified fmode value is not valid, or the socket s has already requested multicast setsockopt options. For more information, see "z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference"
- **EPROTOTYPE**: The socket protocol type is not correct.
- **ENOBUFS**: The number of source addresses exceeds the allowed limit.
- **ENOMEM**: Insufficient storage is available to supply the array.
- **EADDRNOTAVAIL**: The specified interface address is incorrect for this host, or the specified interface address is not multicast capable.

**setsockopt() call**

See "getsockopt(), setsockopt() calls" on page 203.
**setsourcefilter() call**

Sets a list of the IPv4 or IPv6 source addresses that comprise the source filter, along with the current mode on a given interface and a multicast group for a socket. The source filter can either include or exclude the set of source addresses, depending on the filter mode (MCAST_INCLUDE or MCAST_EXCLUDE).

**setsourcefilter() call format**

This call has the following format:

```c
#include <manifest.h> (non-reentrant programs only)
#include <cmcrafties.h> (reentrant programs only)
#include <netinet/in.h>
int setsourcefilter(int s, uint32_t interface,
        struct sockaddr *group, socklen_t grouplen,
        uint32_t fmode, uint32_t numsrc,
        struct sockaddr_storage *slist);
```

**setsourcefilter() call parameters**

- **s** The socket descriptor.
- **interface** The interface index of the interface.
- **group** A pointer to either a `sockaddr_in` structure for IPv4 addresses or a `sockaddr_in6` structure for IPv6 addresses. The pointer holds the IP multicast address of the group.
- **grouplen** The length of the `sockaddr_in` or `sockaddr_in6` structure.
- **fmode** An integer that contains the filter mode to be set. The value of the filter mode can be either MCAST_INCLUDE or MCAST_EXCLUDE.
- **numsrc** An integer that specifies the number of source addresses that are provided in the array that is pointed to by the `slist` parameter.
- **slist** A pointer to an array of IP addresses that is either included or excluded, depending on the filter mode. If the `numsrc` value 0 was specified on input, you can specify a NULL pointer.

**setsourcefilter() call return values**

When successful, the value 0 is returned. When an error occurs, the value -1 is returned and the_errno value is one of the following:

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

**EAFNOSUPPORT** The address family of the input sockaddr value is not AF_INET or AF_INET6.

**EINVAL** The socket address family of an input parameter is not correct, the specified `fmode` value is not correct, or the socket specified by the `s` parameter already requested multicast setsockopt options. See [z/OS](https://www.ibm.com)
ENOBUFS
The number of source addresses exceeds the allowed limit.

EPROTOTYPE
The socket protocol type is not correct.

ENOMEM
Insufficient storage is available to supply the array.

ENXIO
The specified interface index provided in the interface parameter does not exist.

**shutdown() call**

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

**shutdown() call format**

This call has the following format:

```c
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <socket.h>

int shutdown(int s, int how)
```

**shutdown() call 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.

**shutdown() call return values**

The value 0 indicates success; the value -1 indicates an error. To determine which error occurred, check the errno global variable, which is set to a return code. Possible codes include:

- **EBADF**
  - s is not a valid socket descriptor.

- **EINVAL**
  - The how parameter was not set to one of the valid values. Valid values are 0, 1, and 2.

**socket() call**

The socket() call creates an endpoint for communication and returns a socket descriptor representing the endpoint. Different types of sockets provide different communication services.
SOCK_STREAM sockets model duplex byte streams. They provide reliable, flow-controlled connections between peer applications. Stream sockets are either active or passive. Active sockets are used by clients that initiate connection requests with connect(). By default, socket() creates active sockets. Passive sockets are used by servers to accept connection requests with the connect() call. An active socket is transformed into a passive socket by binding a name to the socket with the bind() call and by indicating a willingness to accept connections with the listen() call. After a socket is passive, it cannot be used to initiate connection requests.

SOCK_DGRAM supports datagrams (connectionless messages) of a fixed maximum length. Transmission reliability is not guaranteed. Datagrams can be corrupted, received out of order, lost, or delivered multiple times.

Sockets are deallocated with the close() call.

socket() call format

This call has the following format:

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

int socket(int domain, int type, int protocol)
```

socket() call parameters

domain The domain parameter specifies a communication domain within which communication is to take place. This parameter selects the address family (format of addresses within a domain) that is used. The only families supported by CICS TCP/IP are AF_INET and AF_INET6, which are both the Internet domain. The AF_INET and AF_INET6 constant is defined in the socket.h header file.

type The type parameter specifies the type of socket created. These socket type constants are defined in the socket.h header file. This must be set to either SOCK_STREAM or SOCK_DGRAM.

protocol The protocol parameter specifies a particular protocol to be used with the socket. In most cases, a single protocol exists to support a particular type of socket in a particular addressing family. If the protocol parameter is set to 0, the system selects the default protocol number for the domain and socket type requested. Protocol numbers are found in the hlq.ETC.PROTO data set. The default protocol for stream sockets is TCP. The default protocol for datagram sockets is UDP.

socket() call return values

A nonnegative socket descriptor indicates success. The value -1 indicates an error. To determine which error occurred, check the errno global variable, which is set to a return code. Possible codes include:

EPROTONOSUPPORT

The protocol is not supported in this domain, or this protocol is not supported for this socket type.
**takesocket() call**

The takesocket() call acquires a socket from another program. The CICS listener passes the client ID and socket descriptor in the COMMAREA.

**takesocket() call format**

This call has the following format:

```c
#include <manifest.h> (non-reentrant programs only)
#include <manifes.h> (reentrant programs only)
#include <bsdtypes.h>
#include <socket.h>

int takesocket(struct clientid *client_id,
               int hisdesc)
```

**takesocket() call parameters**

- **clientid**  A pointer to the clientid of the application from which you are taking a socket.
- **domain**  Sets the domain of the program giving the socket. Set as either AF_INET (a decimal 2) or AF_INET6 (a decimal 19).
  
  **Rule:** An AF_INET socket can be taken only from an AF_INET givesocket(). An AF_INET6 socket can be taken only from an AF_INET6 givesocket(). EBADF is set if the domain does not match.
- **name**  Set to the address space identifier of the program that gave the socket.
- **subtaskname**  Set to the task identifier of the task that gave the socket.
- **reserved**  Binary zeros.
- **hisdesc**  The descriptor of the socket to be taken.

**takesocket() call return values**

A nonnegative socket descriptor is the descriptor of the socket to be used by this process. The value -1 indicates an error. To determine which error occurred, check the errno global variable, which is set to a return code. Possible codes include:

- **EACCES**  The other application did not give the socket to your application.
- **EBADF**  The hisdesc parameter does not specify a valid socket descriptor owned by the other application. The socket has already been taken.
- **EFAULT**  Using the clientid parameter as specified results in an attempt to access storage outside the caller’s address space.
- **EINVAL**  The clientid parameter does not specify a valid client identifier.
EMFILE
The socket descriptor table is already full.

ENOBUFS
The operation cannot be performed because of the shortage of SCB or
SKCB control blocks in the TCP/IP address space.

EPFNOSUPPORT
The domain field of the \textit{clientid} parameter is not AF_INET or AF_INET6.

\textbf{write()} call

This call writes data on a connected socket.

Stream sockets act like streams of information with no boundaries separating data.
For example, if an application wishes to send 1000 bytes, each call to this function
can send 1 byte or 10 bytes or the entire 1000 bytes. Therefore, applications using
stream sockets should place this call in a loop, calling this function until all data
has been sent.

\textbf{write()} call format

This call has the following format:

\begin{verbatim}
#include <manifest.h> (non-reentrant programs only)
#include <cmanifes.h> (reentrant programs only)
#include <socket.h>

int write(int s, char *buf, int len)
\end{verbatim}

\textbf{write()} call parameters

\begin{description}
\item \textit{s} The socket descriptor.
\item \textit{buf} The pointer to the buffer holding the data to be written.
\item \textit{len} The length in bytes of the buffer pointed to by the \textit{buf} parameter.
\end{description}

\textbf{write()} call return values

If successful, the number of bytes written is returned. The value -1 indicates an
error. To determine which error occurred, check the \textit{errno} global variable, which is
set to a return code. Possible codes include:

\begin{description}
\item [EBADF] \textit{s} is not a valid socket descriptor.
\item [EFAULT] Using the \textit{buf} and \textit{len} parameters results in an attempt to access storage
outside the caller’s address space.
\item [ENOBUFS] Buffer space is not available to send the message.
\item [EWOULDBLOCK] \textit{s} is in nonblocking mode and data is not available to write.
\end{description}

\textbf{Address Testing Macros}

This topic describes the macros that can be used to test for special IPv6 addresses.
#include <netinet/in.h>

int IN6_IS_ADDR_UNSPECIFIED (const struct in6_addr *)
int IN6_IS_ADDR_LOOPBACK (const struct in6_addr *)
int IN6_IS_ADDR_MULTICAST (const struct in6_addr *)
int IN6_IS_ADDR_LINKLOCAL (const struct in6_addr *)
int IN6_IS_ADDR_SITELOCAL (const struct in6_addr *)
int IN6_IS_ADDR_V4MAPPED (const struct in6_addr *)
int IN6_IS_ADDR_V4COMPAT (const struct in6_addr *)
int IN6_IS_ADDR_MC_NODELOCAL (const struct in6_addr *)
int IN6_IS_ADDR_MC_LINKLOCAL (const struct in6_addr *)
int IN6_IS_ADDR_MC_SITELOCAL (const struct in6_addr *)
int IN6_IS_ADDR_MC_ORGLOCAL (const struct in6_addr *)
int IN6_IS_ADDR_MC_GLOBAL (const struct in6_addr *)

IN6_IS_ADDR_UNSPECIFIED
   Returns true if the address is the unspecified IPv6 address (in6addr_any).
   Otherwise, the macro returns false.

IN6_IS_ADDR_LOOPBACK
   Returns true if the address is an IPv6 loopback address. Otherwise, the macro returns false.

IN6_IS_ADDR_MULTICAST
   Returns true if the address is an IPv6 multicast address. Otherwise, the macro returns false.

IN6_IS_ADDR_LINKLOCAL
   Returns true if the address is an IPv6 link local address. Otherwise, the macro returns false.

IN6_IS_ADDR_SITELOCAL
   Returns true for local-use IPv6 unicast addresses.
   Returns false for the IPv6 loopback address.
   Does not return true for IPv6 multicast addresses of link-local scope.

IN6_IS_ADDR_V4MAPPED
   Returns true if the address is an IPv4 mapped IPv6 address. Otherwise, the macro returns false.

IN6_IS_ADDR_V4COMPAT
   Returns true if the address is an IPv4 compatible IPv6 address. Otherwise, the macro returns false.
IN6_IS_ADDR_MC_NODELOCAL
Used to test the scope of a multicast address and returns true if the
address is a multicast address of the specified scope or false if the address
is not a multicast address or not of the specified scope.

IN6_IS_ADDR_MC_LINKLOCAL
Used to test the scope of a multicast address and returns true if the
address is a multicast address of the specified scope or false if the address
is either not a multicast address or not of the specified scope.

IN6_IS_ADDR_MC_SITELOCAL
Used to test the scope of a multicast address and returns true if the
address is a multicast address of the specified scope or false if the address
is either not a multicast address or not of the specified scope.

IN6_IS_ADDR_MC_ORGLOCAL
Used to test the scope of a multicast address and returns true if the
address is a multicast address of the specified scope or false if the address
is either not a multicast address or not of the specified scope.

IN6_IS_ADDR_MC_GLOBAL
Used to test the scope of a multicast address and returns true if the
address is a multicast address of the specified scope or false if the address
is either not a multicast address or not of the specified scope.
Chapter 8. Sockets extended API

This topic contains information about the sockets extended application programming interface (API).

Environmental restrictions and programming requirements for the Callable Socket API

The following environmental restrictions and programming requirements apply to the Callable Socket API:

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

CALL instruction API

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

Note:

- Unless your program is running in a CICS environment, reentrant code and multithread applications are not supported by this interface.
- Only one copy of an interface can exist in a single address space.
- For a PL/I program, include the following statement before your first call instruction.

  DCL EZASOKET ENTRY OPTIONS(ASM,INTER) EXT;
The entry point for the CICS Sockets Extended module (EZASOKET) is within the hlq.SEZATCP(EZACICAL) load module and should be resolved from there when processed by the binder. Therefore, EZACICAL should be included explicitly in your link-editing JCL. If not included, you could experience problems, such as the CICS region waiting for the socket calls to complete. You can use the linkage editor MAP parameter to produce the module map report to verify where EZASOKET is resolved.

See Figure 177 on page 422.

If you do not want to explicitly include EZACICAL in your link-edit JCL then you can use the EZACICSO CICS Sockets Extended module. The EZACICSO CICS Sockets Extended module is an ALIAS for EZASOKET that resides in the same entry point in EZACICAL as EZASOKET. You must also substitute any "CALL EZASOKET" invocations in your program with "CALL EZACICSO". This allows you to use the Binder’s Automatic Library Call option (AUTOCALL) to build your load modules.

SEZATCP load library data set needs to be included in the SYSLIB DD concatenation.

**Understanding COBOL, assembler, and PL/I call formats**

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

**COBOL language call format**

The following is the 'EZASOKET' call format for COBOL language programs.
The following is the 'EZACICSO' call format for the COBOL language programs.

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

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

**parm** A variable number of parameters depending on the type of call.

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

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

### Assembler language call format

The following is the ‘EZASOKET’ call format for assembler language programs. Because DATAREG is used to access the application's working storage, applications using the assembler language format should not code DATAREG but should let it default to the CICS data register.
The following is the 'EZACICSO' call format for assembler language programs.

```
CALL EZACICSO,(SOC-FUNCTION,parm1, parm2, ...—ERRNO RETCODE),VL,MF=(E, PARMLIST)
```

**PARMLIST**
A remote parameter list defined in dynamic storage DFHEISTG. This list contains addresses of 30 parameters that can be referenced by all execute forms of the CALL.

**Note:** This form of CALL is necessary to meet the CICS requirement for quasi-reentrant programming.

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

**parm n**
A variable number of parameters depending on the type call.

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

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

**PL/I language call format**

The following is the 'EZASOKET' call format for PL/I language programs.
CALL EZASOKET (SOC-FUNCTION—parm1, parm2, ...—ERRNO RETCODE);

The following is the 'EZACICSO' call format for the PL/I language programs.

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

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

parm
A variable number of parameters depending on the type call.

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

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

Converting parameter descriptions

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

Figure 117 on page 254 shows examples of storage definition statements for  
COBOL, PL/I, and assembler language programs.
**Error messages and return codes**

For information about error messages, see [z/OS Communications Server: IP Messages Volume 1 (EZA)](https://www.ibm.com/support/knowledgecenter/ST5175_2.2.0/com.ibm.zos.mvs lcmssq1/c_cmsa_intx.htm).

For information about error codes that are returned by TCP/IP, see [Appendix B, “Return codes,” on page 455.](https://www.ibm.com/support/knowledgecenter/ST5175_2.2.0/com.ibm.zos.mvssq1/c_cmsa_intx.htm)

### Code CALL instructions

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

**ACCEPT call**

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

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

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

**Note:**
- The blocking or nonblocking mode of a socket affects the operation of certain commands. The default is blocking; nonblocking mode can be established by use of the FCNTL and IOCTL calls. When a socket is in blocking mode, an I/O call...
waits for the completion of certain events. For example, a READ call blocks until the buffer contains input data. When an I/O call is issued: if the socket is blocking, program processing is suspended until the event completes; if the socket is nonblocking, program processing continues.

- If the queue has no pending connection requests, ACCEPT blocks the socket unless the socket is in nonblocking mode. The socket can be set to nonblocking by calling FCNTL or IOCTL.
- When multiple socket calls are issued, a SELECT call can be issued prior to the ACCEPT to ensure that a connection request is pending. Using this technique ensures that subsequent ACCEPT calls do not block.
- TCP/IP does not provide a function for screening clients. As a result, it is up to the application program to control which connection requests it accepts, but it can close a connection immediately after discovering the identity of the client.

The following requirements apply to this call:

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

Figure 118 on page 256 shows an example of ACCEPT call instructions.
WORKING-STORAGE SECTION.

    01 SOC-FUNCTION PIC X(16) VALUE IS 'ACCEPT'.
    01 S PIC 9(4) BINARY.

* * IPv4 Socket Address Structure. *

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


* * IPv6 Socket Address Structure. *

    01 NAME.
      03 FAMILY PIC 9(4) BINARY.
      03 PORT PIC 9(4) BINARY.
      03 FLOW-INFO PIC 9(8) BINARY.
      03 IP-ADDRESS.
        05 FILLER PIC 9(16) BINARY.
        05 FILLER PIC 9(16) BINARY.
      03 SCOPE-ID PIC 9(8) BINARY.
      01 ERRNO PIC 9(8) BINARY.
      01 RETCODE PIC S9(8) BINARY.

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

Figure 118. ACCEPT call instructions example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 253.

Parameter values set by the application for the ACCEPT call

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 for the ACCEPT call

NAME
* An IPv4 socket address structure that contains the client’s IPv4 socket address.

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

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

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

RESERVED
Specifies 8 bytes of binary zeros. This field is required, but not used.
- An IPv6 socket address structure that contains the client's IPv6 socket address.

**FAMILY**
A halfword binary field specifying the addressing family. The call returns the decimal value of 19 for AF_INET6.

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

**FLOW-INFO**
A fullword binary field specifying the traffic class and flow label. The value of this field is undefined.

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

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

**ERRNO**
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B, “Return codes,” on page 455 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 call**

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

The BIND call can either specify the required port or let the system choose the port. A listener program should always bind to the same well-known port, so that clients know what socket address to use when attempting to connect.

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

In the AF_INET or AF_INET6 domain, the BIND call for a stream socket can specify the networks from which it is willing to accept connection requests. The application can fully specify the network interface by setting the IP-ADDRESS field to the Internet address of a network interface. Alternatively, the application can use a wildcard to specify that it wants to receive connection requests from any network interface. This is done by setting the IP-ADDRESS field to the value of INADDR-ANY or IN6ADDR-ANY.

The following requirements apply to this call:
## Requirement Description

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

Figure 119 shows an example of BIND call instructions.

WORKING-STORAGE SECTION.
   01 SOC-FUNCTION PIC X(16) VALUE IS 'BIND'.
   01 S PIC 9(4) BINARY.
   *
   * IPv4 Socket Address Structure.
   *
   01 NAME.
      03 FAMILY PIC 9(4) BINARY.
      03 PORT PIC 9(4) BINARY.
      03 IP-ADDRESS PIC 9(8) BINARY.
      03 RESERVED PIC X(8).
   *
   * IPv6 Socket Address Structure.
   *
   01 NAME.
      03 FAMILY PIC 9(4) BINARY.
      03 PORT PIC 9(4) BINARY.
      03 FLOW-INFO PIC 9(8) BINARY.
      03 IP-ADDRESS.
         05 FILLER PIC 9(16) BINARY.
         05 FILLER PIC 9(16) BINARY.
      03 SCOPE-ID PIC 9(8) BINARY.
   01 ERRNO PIC 9(8) BINARY.
   01 RETCODE PIC S9(8) BINARY.

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

Figure 119. BIND call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 253.

### Parameter values set by the application for the BIND call

**SOC-FUNCTION**

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

- **S**

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

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

**FAMILY**
A halfword binary field specifying the addressing family. The value is set to a decimal 2, indicating AF_INET.

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

*Note:* If PORT is set to 0 when the call is issued, the system assigns the port number for the socket. The application can call the GETSOCKNAME call after the BIND call to discover the assigned port number.

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

**RESERVED**
Specifies an eight-byte character field that is required but not used.

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

**FAMILY**
A halfword binary field specifying the addressing family. The value is set to a decimal 19, indicating AF_INET6.

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

*Note:* If PORT is set to 0 when the call is issued, the system assigns the port number for the socket. The application can call the GETSOCKNAME call after the BIND call to discover the assigned port number.

**FLOW-INFO**
A fullword binary field specifying the traffic class and flow label. This field must be set to zero.

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

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

**Parameter values returned to the application for the BIND call**

**ERRNO**
A fullword binary field. If RETCODE is negative, this field contains an error number. See [Appendix B, “Return codes,” on page 455](#) for information about ERRNO return codes.
RETCODE
A fullword binary field that returns one of the following:

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

BIND2ADDRSEL call

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

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

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

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

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

Requirement: When your application is using stream sockets, and must prevent the stack from sending any packets whatsoever (such as SYN) to the remote host before it can verify that the local IP address meets the values specified for the IPV6_ADDR_PREFERENCES option, do not allow the CONNECT call to implicitly bind the socket to a local IP address. Instead, bind the socket with the BIND2ADDRSEL call and test the local IP address assigned with the INET6_IS_SRCADDR call. If the assigned local IP address is satisfactory, you can then use the CONNECT call to establish communication with the remote host.

- After you successfully issue the BIND2ADDRSEL call, use the GETSOCKNAME call to obtain the local IP address that is bound to the socket. When the local IP address is obtained, use the INET6_IS_SRCADDR call to verify that the local IP address meets your address selection criteria.

The following requirements apply to this call:

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

Interrupt status: Enabled for interrupts
Locks: Unlocked
Control parameters: All parameters must be addressable by the caller and in the primary address space

Figure 120 shows an example of BIND2ADDRSEL call instructions.

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

Figure 120. BIND2ADDRSEL call instructions example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 253.

Parameter values set by the application

SOC-FUNCTION
A 16-byte character field containing BIND2ADDRSEL. 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.

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

NAME
Specifies the IPv6 socket address structure of the remote host that the socket will communicate with.

The IPv6 socket structure must specify the following fields:

Field  Description

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

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

Guideline: To determine the assigned port number, issue the GETSOCKNAME call after the BIND2ADDRSEL call completes.
FLOWINFO
A fullword binary field. This field is ignored by BIND2ADDRSEL processing.

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

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

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

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

Parameter values returned to the application

ERRNO
A fullword binary field. If RETCODE is negative, this field contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.

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

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

CLOSE call

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 call” on page 391 for more information.

**Note:**
1. If a stream socket is closed while input or output data is queued, the TCP connection is reset and data transmission might be incomplete. The SETSOCKET call can be used to set a linger condition, in which TCP/IP continues to attempt to complete data transmission for a specified period of time after the CLOSE call is issued. See SO-LINGER in the description of “SETSOCKOPT call” on page 375.
2. A concurrent server differs from an iterative server. An iterative server provides services for one client at a time; a concurrent server receives connection requests from multiple clients and creates child servers that actually serve the clients. When a child server is created, the concurrent server obtains a new socket, passes the new socket to the child server, and then dissociates itself from the connection. The CICS listener is an example of a concurrent server.

3. After an unsuccessful socket call, a close should be issued and a new socket should be opened. An attempt to use the same socket with another call results in a nonzero return code.

The following requirements apply to this call:

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

Figure 121 on page 264 shows an example of CLOSE call instructions.
WORKING-STORAGE SECTION.
  01 SOC-FUNCTION PIC X(16) VALUE IS 'CLOSE'.
  01 S PIC 9(4) BINARY.
  01 ERRNO PIC 9(8) BINARY.
  01 RETCODE PIC S9(8) BINARY.

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

Figure 121. CLOSE call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 253.

Parameter values returned to the application for the CLOSE call

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 for the CLOSE call

ERRNO
A fullword binary field. If RETCODE is negative, this field contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.

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

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

CONNECT call

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

The call sequence issued by the client and server for stream sockets is:
1. The server issues BIND and LISTEN to create a passive open socket.
2. The client issues CONNECT to request the connection.
3. The server accepts the connection on the passive open socket, creating a new connected socket.

The blocking mode of the CONNECT call conditions its operation.
- If the socket is in blocking mode, the CONNECT call blocks the calling program until the connection is established, or until an error is received.
- If the socket is in nonblocking mode, the return code indicates whether the connection request was successful.
  - A RETCODE of 0 indicates that the connection was completed.
A nonzero RETCODE with an ERRNO of 36 (EINPROGRESS) indicates that the connection is not completed but because the socket is nonblocking, the CONNECT call returns normally.

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

The completion cannot be checked by issuing a second CONNECT. For more information, see “SELECT call” on page 353.

The following requirements apply to this call:

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

Figure 122 on page 266 shows an example of CONNECT call instructions.
Stream sockets and the CONNECT call

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 and the CONNECT call

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

Parameter values set by the application for the CONNECT call

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

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

NAME
- A structure that contains the IPv4 socket address of the target to which the local client socket is to be connected.
FAMILY
A halfword binary field specifying the addressing family. The value must be a decimal 2 for AF_INET.

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

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

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

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

FAMILY
A halfword binary field specifying the addressing family. The value must be a decimal 19 for AF_INET6.

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

FLOW-INFO
A fullword binary field specifying the traffic class and flow label. This field must be set to zero.

IP-ADDRESS
A 16-byte binary field that is set to the 128-bit IPv6 Internet address of the server’s host machine in network byte order. For example, if the IPv6 Internet address is 12ab:0:0:cd30:123:4567:89ab:cedf in colon-hexadecimal notation, it is set to X’12AB00000000CD300123456789ABCDEF’.

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

Parameter values returned to the application for the CONNECT call

ERRNO
A fullword binary field. If RETCODE is negative, this field contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.

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

Value    Description
Successful call
-1 Check ERRNO for an error code

**FCNTL call**

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 call” on page 322 for another way to control a socket’s blocking mode.

Values for Command which are supported by the z/OS UNIX System Services fcntl callable service is also be accepted. See the z/OS UNIX System Services Programming: Assembler Callable Services Reference for more information.

The following requirements apply to this call:

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

Figure 123 on page 269 shows an example of FCNTL call instructions.
Parameter values set by the application for the FCNTL call

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

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

**COMMAND**
A fullword binary number with the following values.

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

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

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

Parameter values returned to the application for the FCNTL call

**ERRNO**
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.

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

- If COMMAND was set to 3 (query), a bit string is returned.
  - If RETCODE contains X'00000004', the socket is nonblocking. (The FNDELAY flag is on.)
  - If RETCODE contains X'00000000', the socket is blocking. (The FNDELAY flag is off.)
If COMMAND was set to 4 (set), a successful call is indicated by 0 in this field. In both cases, a RETCODE of -1 indicates an error (Check the ERRNO field for the error number.)

FREEADDRINFO call

FREEADDRINFO frees all the address information structures returned by GETADDRINFO in the RES parameter. Figure 124 on page 271 shows an example of FREEADDRINFO call instructions.

The following requirements apply to this call:

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

Figure 124 on page 271 shows an example of FREEADDRINFO call instructions.
Parameter values set by the application for the FREEADDRINFO call

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

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

Parameter values returned to the application for the FREEADDRINFO call

ERRNO
A fullword binary field. If RETCODE is negative, ERRNO contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.

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

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

GETADDRINFO call

GETADDRINFO translates the name of a service location (for example, a host name), service name, or both and returns a set of socket addresses and associated information to be used in creating a socket with which to address the specified service or sending a datagram to the specified service. Figure 125 on page 273 shows an example of GETADDRINFO call instructions.

The following requirements apply to this call:

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

Figure 125 on page 273 shows an example of GETADDRINFO call instructions.
Figure 125. GETADDRINFO call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 253.
Parameter values set by the application for the GETADDRINFO call

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

**NODE**
Storage maximum of 255 bytes that contains the host name being queried. If the AI-NUMERICHOST flag is specified in the storage pointed to by the HINTS operand, then NODE should contain the queried hosts IP address in presentation form. This is an optional field but if specified you must also code NODELEN.

Scope information can be appended to the host name, using the format node%scope information. The combined length of the value specified must still fit within 255 bytes. For information about using scope information on GETADDRINFO processing, see z/OS Communications Server: IPv6 Network and Application Design Guide.

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

**SERVICE**
Storage maximum of 32 bytes that contains the service name being queried. If the AI-NUMERICSERV flag is specified in the storage pointed to by the HINTS operand, then SERVICE should contain the queried port number in presentation form. This is an optional field but if specified you must also code SERVLEN.

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

**HINTS**
If the HINTS argument is specified, it contains the address of an addrinfo structure containing input values that can direct the operation by providing options and by limiting the returned information to a specific socket type, address family, and protocol. If the HINTS argument is not specified, the information returned is as if it referred to a structure containing the value 0 for the FLAGS, SOCTYPE and PROTO fields, and AF_UNSPEC for the AF field. Include the EZBREHST resolver macro to enable your assembler program to contain the assembler mappings for the ADDR_INFO structure.

The EZBREHST macro is stored in SYS1.MACLIB, r hostent, addrinfo mappings, and services return codes. Copy definitions from EZACOBO sample module to your COBOL program for mapping the ADDRINFO structure. The EZACOBO sample module is stored in hlq.SEZAINST library. Copy definitions from CBLOCK sample module to your PL/I program for mapping the ADDRINFO structure. The CBLOCK sample module is stored in hlq.SEZAINST library.

This is an optional field. The address information structure has the following fields:

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

274  z/OS V2R1.0 Communications Server: IP CICS Sockets Guide
FLAGS

A fullword binary field. The value of this field must be 0 or the bitwise OR of one or more of the following flags:

AI-PASSIVE (X'00000001') or a decimal value of 1
Specifies how to fill in the name pointed to by the returned RES parameter.

If this flag is specified, the returned address information can be used to bind a socket for accepting incoming connections for the specified service (for example, using the BIND call). If you use the BIND call and if the NODE argument is not specified, the IP address portion of the socket address structure pointed to by the returned RES parameter is set to INADDR_ANY for an IPv4 address or to the IPv6 unspecified address (in6addr_any).

If this flag is not set, the returned address information can be used for the CONNECT call (for a connection-mode protocol) or on a CONNECT, SENDTO, or SENDMSG call (for a connectionless protocol). If you use a CONNECT call and if the NODE argument is not specified, the NAME pointed to by the returned RES is set to the loopback address.

This flag is ignored if the NODE argument is specified.

AI-CANONNAMEOK (X'00000002') or a decimal value of 2
If this flag is specified and the NODE argument is specified, the GETADDRINFO call attempts to determine the canonical name corresponding to the NODE argument.

AI-NUMERICHOST (X'00000004') or a decimal value of 4
If this flag is specified, the NODE argument must be a numeric host address in presentation form. Otherwise, an error of host not found [EAI_NONAME] is returned.

AI-NUMERICSERV (X'00000008') or a decimal value of 8
If this flag is specified, the SERVICE argument must be a numeric port in presentation form. Otherwise, an error [EAI_NONAME] is returned.

AI-V4MAPPED (X'00000010') or a decimal value of 16
If this flag is specified along with the AF field with the value of AF_INET6, or a value of AF_UNSPEC when IPv6 is supported on the system, the caller accepts IPv4-mapped IPv6 addresses. When the AI-ALL flag is not also specified, if no IPv6 addresses are found, a query is made for IPv4 addresses. If IPv4 addresses are found, they are returned as IPv4-mapped IPv6 addresses. If the AF field does not have the value of AF_INET6, or the AF field contains AF_UNSPEC but IPv6 is not supported on the system, then this flag is ignored.

AI-ALL (X'00000020') or a decimal value of 32
When the AF field has a value of AF_INET6 and AI-ALL is set, the AI-V4MAPPED flag must also be set to indicate that the caller accepts all addresses (IPv6 and IPv4-mapped IPv6 addresses). When the AF field has a value of AF_UNSPEC, and when the system supports IPv6 and
AI-ALL is set, the caller accepts both IPv6 and IPv4 addresses. A query is first made for IPv6 addresses and if successful, the IPv6 addresses are returned. Another query is then made for IPv4 addresses, and any IPv4 addresses found are returned as either IPv4-mapped IPv6 addresses (if AI-V4MAPPED is also specified) or as IPv4 addresses (if AI-V4MAPPED is not specified). If the AF field does not have the value of AF_INET6, or does not have the value of AF_UNSPEC when the system supports IPv6, then this flag is ignored.

**AI-ADDRCONFIG (X'00000040') or a decimal value of 64**

If this flag is specified, a query on the name in nodename occurs if the resolver determines that one of the following is true:
- If the system is IPv6 enabled and has at least one IPv6 interface, then the resolver makes a query for IPv6 (AAAA or A6 DNS records) records.
- If the system is IPv4 enabled and has at least one IPv4 interface, then the resolver makes a query for IPv4 (A DNS records) records.

**AI-EXTFLAGS (X'00000080') or a decimal value of 128.**

If this flag is specified, the address information structure contains an EFLAGS field (see the field description of EFLAGS).

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

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

**AF**

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

**SOCTYPE**

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

The following are the acceptable socket types:

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Decimal Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOCK_STREAM</td>
<td>1</td>
<td>for stream socket</td>
</tr>
<tr>
<td>SOCK_DGRAM</td>
<td>2</td>
<td>for datagram socket</td>
</tr>
<tr>
<td>SOCK_RAW</td>
<td>3</td>
<td>for raw-protocol interface</td>
</tr>
</tbody>
</table>
Anything else fails with return code EAI_SOCKTYPE. Although
SOCK_RAW is accepted, it is valid only when SERVICE is numeric (for
example, SERVICE=23). A lookup for a SERVICE name never occurs in the
appropriate services file (for example, hlq.ETC.SERVICES) using any
protocol value other than SOCK_STREAM or SOCK_DGRAM. If PROTO is
nonzero and SOCTYPE is zero, the only acceptable input values for
PROTO are IPPROTO_TCP and IPPROTO_UDP. Otherwise, the
GETADDRINFO call fails with a return code of EAI_BADFLAGS. If
SOCTYPE and PROTO are both specified as zero, GETADDRINFO
proceeds as follows:

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

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

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

PROTO

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

The following are the acceptable protocols:

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

If PROTO and SOCTYPE are both specified as zero, GETADDRINFO
proceeds as follows:

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

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

- SOCTYPE=SOCK_STREAM and PROTO=IPPROTO_TCP
- SOCTYPE=SOCK_DGRAM and PROTO=IPPROTO_UDP
- SOCTYPE=SOCK_RAW, in which case PROTO can be anything.
If the lookup for the value specified in SERVICE fails [that is, the service name does not appear in the appropriate services file (for example, hlq.ETC.SERVICES) using the input protocol], the GETADDRINFO call fails with a return code of EAI_SERVICE.

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

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

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

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

**EFLAGS**
A fullword binary field that specifies the source IPv6 address selection preferences.

This field is required if AI-EXTFLAGS is specified in the FLAGS field.

This value of this field must be 0 or the bitwise OR of one or more of the following flags:

**IPV6_PREFER_SRC_HOME**
(X'00000001') or the decimal value 1 indicates that home source IPv6 addresses are preferred over care-of source IPv6 addresses.

**IPV6_PREFER_SRC_COA**
(X'00000002') or the decimal value 2 indicates that care-of source IPv6 addresses are preferred over home source IPv6 addresses.

**IPV6_PREFER_SRC_TMP**
(X'00000004') or the decimal value 4 indicates that temporary source IPv6 addresses are preferred over public source IPv6 addresses.

**IPV6_PREFER_SRC_PUBLIC**
(X'00000008') or the decimal value 8 indicates that public source IPv6 addresses are preferred over temporary source IPv6 addresses.

**IPV6_PREFER_SRC_CGA**
(X'00000010') or the decimal value 16 indicates that cryptographically generated source IPv6 addresses are preferred over non-cryptographically generated source IPv6 addresses.

**IPV6_PREFER_SRC_NONCGA**
(X'00000020') or the decimal value 32 indicates that non-cryptographically generated source IPv6 addresses are preferred over cryptographically generated source IPv6 addresses.
If contradictory or invalid EFLAGS are specified, the
GETADDRINFO call fails with the RETCODE -1 and the ERRNO
EAI_BADEXTFLAGS (decimal value 11).

- An example of contradictory EFLAGS is
  IPV6_PREFER_SRC_TMP and IPV6_PREFER_SRC_PUBLIC
- An example of invalid EFLAGS is X'00000040' or the decimal
  value 64

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

Requirement: The structures returned by GETADDRINFO are a serially
reusable storage areas associated with the transaction. Do not use or
reference these structures from other transactions.

The address information structure contains the following fields:

Field  Description

FLAGS
A fullword binary field that is not used as output.

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

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

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

NAMELEN
A fullword binary field. The length of the NAME socket address
structure.

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

**NAME**

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

**NEXT**

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

**EFLAGS**

A fullword binary field that is not used as output.

**CANNLEN**

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

**Parameter values returned to the application for the GETADDRINFO call**

**ERRNO**

ERRNO A fullword binary field. If RETCODE is negative, ERRNO contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.

**RETCODE**

A fullword binary field that returns one of the following:

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

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

**GETCLIENTID call**

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 call” on page 314 for a discussion of the use of GIVESOCKET and TAKESOCKET calls.

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

The following requirements apply to this call:
### Requirement Description

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

[Figure 126](#) shows an example of GETCLIENTID call instructions.

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

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

*Figure 126. GETCLIENTID call instruction example*

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 253.

### Parameter values set by the application for the GETCLIENTID call

**SOC-FUNCTION**

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

### Parameter values returned to the application for the GETCLIENTID call

**CLIENT**

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

**DOMAIN**

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

**NAME**

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

**TASK**

An 8-byte character field set to the task identifier of the caller.
RESERVED
Specifies 20-byte character reserved field. This field is required, but not used.

ERRNO
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.

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

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

GETHOSTBYADDR call

The GETHOSTBYADDR call returns the domain name and alias name of a host whose Internet address is specified in the call. A given TCP/IP host can have multiple alias names and multiple host Internet addresses.

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

The following requirements apply to this call:

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

Figure 127 on page 283 shows an example of GETHOSTBYADDR call instructions.
Parameter values set by the application for the
GETHOSTBYADDR call

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

HOSTADDR
A fullword binary field set to the Internet address (specified in network
byte order) of the host whose name is being sought. See Appendix B,
“Return codes,” on page 455 for information about ERRNO return codes.

Parameter values returned to the application for the
GETHOSTBYADDR call

HOSTENT
A fullword containing the address of the HOSTENT structure.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful call</td>
</tr>
<tr>
<td>-1</td>
<td>An error occurred</td>
</tr>
</tbody>
</table>

GETHOSTBYADDR returns the HOSTENT structure shown in Figure 128 on page 284.
This structure contains:

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

The HOSTENT structure uses indirect addressing to return a variable number of alias names and Internet addresses. If you are coding in PL/I or assembler language, this structure can be processed in a relatively straightforward manner. If you are coding in COBOL, this structure might be difficult to interpret. You can use the subroutine EZACIC08 to simplify interpretation of the information returned by the GETHOSTBYADDR and GETHOSTBYNAME calls. For more information about EZACIC08, see "EZACIC08 program" on page 409. If you are coding in assembler, this structure is defined in the EZBREHST macro. The EZBREHST macro is stored in SYS1.MACLIB, and the HOSTENT structure, address information mappings, and services return codes.
GETHOSTBYNAME call

The GETHOSTBYNAME call returns the alias name and the Internet address of a host whose domain name is specified in the call. A given TCP/IP host can have multiple alias names and multiple host Internet addresses.

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

The following requirements apply to this call:

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

Figure 129 on page 286 shows an example of GETHOSTBYNAME call instructions.
WORKING-STORAGE SECTION.
   01 SOC-FUNCTION PIC X(16) VALUE IS 'GETHOSTBYNAME'.
   01 NAMELEN PIC 9(8) BINARY.
   01 NAME PIC X(255).
   01 HOSTENT PIC 9(8) BINARY.
   01 RETCODE PIC S9(8) BINARY.

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

Figure 129. GETHOSTBYNAME call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 253.

Parameter values set by the application for the GETHOSTBYNAME call

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

NAMELEN
   A value set to the length of the host name. The maximum is 255.

NAME
   A character string, up to 255 characters, set to a host name. This call returns the address of the HOSTENT structure for this name.

Parameter values returned to the application for the GETHOSTBYNAME call

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
GETHOSTBYNAME returns the HOSTENT structure shown in Figure 130. This structure contains:

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

The HOSTENT structure uses indirect addressing to return a variable number of alias names and Internet addresses. If you are coding in PL/I or assembler language, this structure can be processed in a relatively straightforward manner. If you are coding in COBOL, this structure might be difficult to interpret. You can use the subroutine EZACIC08 to simplify interpretation of the information returned by the GETHOSTBYADDR and GETHOSTBYNAME calls. For more information about EZACIC08, see “EZACIC08 program” on page 409. If you are coding in assembler, this structure is defined in the EZBREHST macro. The EZBREHST macro is stored in SYS1.MACLIB, and the HOSTENT structure, address information mappings, and services return codes.
GETHOSTID call

The GETHOSTID call returns the 32-bit IPv4 Internet address for the current host.

The following requirements apply to this call:

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

Figure 131 shows an example of GETHOSTID call instructions.

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

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

Figure 131. GETHOSTID call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 253.

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

RETCODE
   Returns a fullword binary field containing the 32-bit IPv4 Internet address of the host. There is no ERRNO parameter for this call.

GETHOSTNAME call

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

Note: The host name that is returned is the host name that the TCPIP stack learned at startup from the TCPIPDATA file that was found. For more information about hostname, see HOSTNAME statement in z/OS Communications Server: IP Configuration Reference.

The following requirements apply to this call:

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

[Figure 132] shows an example of GETHOSTNAME call instructions.

WORKING-STORAGE SECTION.

01 SOC-FUNCTION PIC X(16) VALUE IS 'GETHOSTNAME'.
01 NAMELEN PIC 9(8) BINARY.
01 NAME PIC X(24).
01 ERRNO PIC 9(8) BINARY.
01 RETCODE PIC S9(8) BINARY.

PROCEDURE DIVISION.
CALL 'EZASOKET' USING SOC-FUNCTION NAMELEN NAME ERRNO RETCODE.

Figure 132. GETHOSTNAME call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 253.

Parameter values set by the application for the GETHOSTNAME call

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

NAMELEN
A fullword binary field set to the length of the NAME field. The minimum length of the NAME field is 1 character. The maximum length of the NAME field is 255 characters.

Parameter values returned to the application for the GETHOSTNAME call

NAME
Indicates the receiving field for the host name. If the host name is shorter than the NAMELEN value, then the NAME field is filled with binary zeros after the host name. If the host name is longer than the NAMELEN value, then the name is truncated.

ERRNO
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.

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

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

GETNAMEINFO call

The GETNAMEINFO returns the node name and service location of a socket address that is specified in the call. On successful completion, GETNAMEINFO returns host name, host name length, service name, and service name length, if requested, in the buffers provided.

The following requirements apply to this call:

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

Figure 133 on page 291 shows an example of GETNAMEINFO call instructions.
PROKCEDURE DIVISION.

MOVE 28 TO NAMELEN.
MOVE 255 TO HOSTLEN.
MOVE 32 TO SERVLEN.
MOVE NI-NAMEREQD TO FLAGS.
CALL 'EZASOKET' USING SOC-FUNCTION NAME NAMELEN HOST
       HOSTLEN SERVICE SERVLEN FLAGS ERRNO RETCODE.

Figure 133. GETNAMEINFO call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 253.

Parameter values set by the application for the GETNAMEINFO call

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

NAME
A socket address structure to be translated that has the following fields:

Field   Description

FAMILY
A halfword binary number specifying the IPv4 addressing family. For TCP/IP, the value is a decimal 2, indicating AF_INET.
PORT
   A halfword binary number specifying the port number.

IP-ADDRESS
   A fullword binary number specifying the 32-bit IPv4 Internet address.

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

The IPv6 socket address structure specifies the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAMILY</td>
<td>A halfword binary field specifying the IPv6 addressing family. For TCP/IP, the value is a decimal 19, indicating AF_INET6.</td>
</tr>
<tr>
<td>PORT</td>
<td>A halfword binary number specifying the port number.</td>
</tr>
<tr>
<td>FLOW-INFO</td>
<td>A fullword binary field specifying the traffic class and flow label. This field is not implemented.</td>
</tr>
<tr>
<td>IP-ADDRESS</td>
<td>A 16-byte binary field specifying the 128-bit IPv6 Internet address, in network byte order.</td>
</tr>
<tr>
<td>SCOPE-ID</td>
<td>A fullword binary field that specifies the link scope for an IPv6 address as an interface index. The resolver ignores the SCOPE-ID field, unless the address in the IP-ADDRESS field is a link-local address and the HOST parameter is also specified.</td>
</tr>
<tr>
<td>NAMELEN</td>
<td>A fullword binary field. The length of the socket address structure pointed to by the NAME argument.</td>
</tr>
</tbody>
</table>

HOST
   On input, a storage area that is large enough to hold the returned resolved host name. The host name can be a maximum of 255 bytes, for the input socket address. If inadequate storage is specified to contain the resolved host name, then the resolver returns the host name value up to the storage amount specified and truncation can occur. If the host's name cannot be located, the numeric form of the host's address is returned instead of its name. However, if the NI_NAMEREQD option is specified and no host name is located, then an error is returned. This is an optional field, but if this field is specified, you must also code the HOSTLEN parameter. Specify both the HOST and HOSTLEN parameters or both the SERVICE and SERVLEN parameters. An error occurs if both are omitted.

If the IP-ADDRESS value represents a link-local address, and the SCOPE-ID interface index is a nonzero value, scope information is appended to the resolved host name using the format host%scope information. The scope information can be either the numeric form of the SCOPE-ID interface index, or the interface name associated with the SCOPE-ID interface index.

Use the NI_NUMERICSCOPE option to select which form of scope information should be returned. The combined host name and scope information can be a maximum of 255 characters long. For more
information about scope information and GETNAMEINFO processing, see the z/OS Communications Server: IPv6 Network and Application Design Guide for more information.

HOSTLEN
An output parameter. A fullword binary field that contains the length of the host storage (HOST parameter) used to contain the resolved host name that is returned. The HOSTLEN value must be equal to or greater than the length of the longest host name, or the host name and scope information combination, to be returned. The GETNAMEINFO call returns the host name, or hostname and scope information combination, up to the length specified by the HOSTLEN parameter. On output, the HOSTLEN value contains the length of the returned resolved host name, or the host name and scope information combination. If the HOSTLEN value 0 is specified on input, then the resolved host name is not returned. This is an optional field, but if it is specified, you must also code the HOST parameter. Specify both the HOST and HOSTLEN parameters or both the SERVICE and SERVLEN parameters. An error occurs if both are omitted.

SERVICE
On input, storage capable of holding the returned resolved service name, which can be a maximum of 32 bytes, for the input socket address. If inadequate storage is specified to contain the resolved service name, then the resolver returns the service name up to the storage specified and truncation can occur. If the service name cannot be located, or if NI_NUMERICSERV was specified in the FLAGS operand, then the numeric form of the service address is returned instead of its name. This is an optional field, but if specified, you must also code SERVLEN. Specify both the HOST and HOSTLEN parameters or both the SERVICE and SERVLEN parameters. An error occurs if both are omitted.

SERVLEN
An output parameter. A fullword binary field. The length of the SERVICE storage used to contain the returned resolved service name. SERVLEN must be equal to or greater than the length of the longest service name to be returned. GETNAMEINFO returns the service name up to the length specified by SERVLEN. On output, SERVLEN contains the length of the returned resolved service name. If SERVLEN is 0 on input, then the service name information is not returned. This is an optional field but if specified you must also code SERVICE. Specify both the HOST and HOSTLEN parameters or both the SERVICE and SERVLEN parameters. An error occurs if both are omitted.

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

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

Parameter values returned to the application for the GETNAMEINFO call

**ERRNO**
A fullword binary field. If RETCODE is negative, ERRNO contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.

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

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

**GETPEERNAME call**

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

The following requirements apply to this call:

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

Figure 134 on page 295 shows an example of GETPEERNAME call instructions.
For equivalent PL/I and assembler language declarations, see
"Converting parameter descriptions" on page 253.

Parameter values set by the application for the GETPEERNAME call

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

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

Parameter values returned to the application for the GETPEERNAME call

NAME
An IPv4 socket address structure to contain the peer name. The structure
that is returned is the socket address structure for the remote socket that is
connected to the local socket specified in field S.

FAMILY
A halfword binary field containing the connection peer’s IPv4
addressing family. The call always returns the decimal value 2,
indicating AF_INET.

PORT
A halfword binary field set to the connection peer’s port number.

IP-ADDRESS
A fullword binary field set to the 32-bit IPv4 Internet address of
the connection peer’s host machine.
RESERVED
Specifies an eight-byte reserved field. This field is required, but not
d used.

An IPv6 socket address structure to contain the peer name. The structure
that is returned is the socket address structure for the remote socket that is
connected to the local socket specified in field S.

FAMILY
A halfword binary field containing the connection peer’s IPv6
addressing family. The call always returns the decimal value 19,
indicating AF_INET6.

PORT
A halfword binary field set to the connection peer’s port number.

FLOW-INFO
A fullword binary field specifying the traffic class and flow label.
The value of this field is undefined.

IP-ADDRESS
A 16-byte binary field set to the 128-bit IPv6 Internet address of the
connection peer’s host machine.

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

ERRNO
A fullword binary field. If RETCODE is negative, the field contains an
error number. See [Appendix B, “Return codes,” on page 455] for
information about ERRNO return codes.

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

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

GETSOCKNAME call

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

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

The following requirements apply to this call:

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

Figure 135 shows an example of GETSOCKNAME call instructions.

```plaintext
WORKING-STORAGE SECTION.
   01 SOC-FUNCTION PIC X(16) VALUE IS 'GETSOCKNAME'.
   01 S PIC 9(4) BINARY.
   *
   * IPv4 Socket Address Structure.
   *
   01 NAME.
      03 FAMILY PIC 9(4) BINARY.
      03 PORT PIC 9(4) BINARY.
      03 IP-ADDRESS PIC 9(8) BINARY.
      03 RESERVED PIC X(8).
   *
   * IPv6 Socket Address Structure.
   *
   01 NAME.
      03 FAMILY PIC 9(4) BINARY.
      03 PORT PIC 9(4) BINARY.
      03 FLOW-INFO PIC 9(8) BINARY.
      03 IP-ADDRESS.
      05 FILLER PIC 9(16) BINARY.
      05 FILLER PIC 9(16) BINARY.
      03 SCOPE-ID PIC 9(8) BINARY.
   01 ERRNO PIC 9(8) BINARY.
   01 RETCODE PIC S9(8) BINARY.

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

Figure 135. GETSOCKNAME call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 253.

Parameter values set by the application for the GETSOCKNAME call

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

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

Parameter values returned to the application for the GETSOCKNAME call

**NAME**
Specifies the IPv4 socket address structure returned by the call.
FAMILY
A halfword binary field containing the addressing family. The call always returns the decimal value 2, indicating AF_INET.

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

IP-ADDRESS
A fullword binary field set to the 32-bit IPv4 Internet address of the local host machine. If the socket is not bound, the address is the IPv6 unspecified address (in6addr_any).

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

Specifies the IPv6 socket address structure returned by the call.

FAMILY
A halfword binary field containing the addressing family. The call always returns the decimal value of 19, indicating AF_INET6.

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

FLOW-INFO
A fullword binary field specifying the traffic class and flow label. The value of this field is undefined.

IP-ADDRESS
A 16-byte binary field set to the 128-bit IPv6 Internet address of the local host machine. If the socket is not bound, the address is IN6ADDR_ANY.

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

ERRNO
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.

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

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

GETSOCKOPT call

The GETSOCKOPT call queries the options that are set by the SETSOCKOPT call.

Several options are associated with each socket. These options are described in this topic. You must specify the option to be queried when you issue the GETSOCKOPT call.
The following requirements apply to this call:

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

Figure 136 shows an example of GETSOCKOPT call instructions.

WORKING-STORAGE SECTION.

```plaintext
01 SOC-FUNCTION PIC X(16) VALUE IS 'GETSOCKOPT'.
01 S PIC 9(4) BINARY.
01 OPTNAME PIC 9(8) BINARY.
01 OPTVAL PIC 9(8) BINARY.
01 OPTLEN PIC 9(8) BINARY.
01 ERRNO PIC 9(8) BINARY.
01 RETCODE PIC S9(8) BINARY.
```

PROCEDURE DIVISION.

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

Figure 136. GETSOCKOPT call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 253.

Parameter values set by the application for the GETSOCKOPT call

SOC-FUNCTION

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

S

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

OPTNAME

Input parameter. Set OPTNAME to the required option before you issue GETSOCKOPT. See “Parameter values returned to the application for the GETSOCKOPT call” on page 300 for a list of the options and their unique requirements. See Appendix C, “GETSOCKOPT/SETSOCKOPT command values,” on page 471 for the numeric values of OPTNAME.

Note: COBOL programs cannot contain field names with the underscore character. Fields representing the option name should contain dashes instead.
Parameter values returned to the application for the GETSOCKOPT call

**OPTVAL**
Output parameter. Contains the status of the specified option. See the table in this topic for a list of the options and their unique requirements.

**OPTLEN**
Output parameter. A fullword binary field containing the length of the data returned in OPTVAL. See the table in this topic for how to determine the value of OPTLEN.

**ERRNO**
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.

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

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

Table 20. OPTNAME options for GETSOCKOPT and SETSOCKOPT

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IP_ADD_MEMBERSHIP</strong></td>
<td>Contains the IP_MREQ structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 interface address.</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ.</td>
<td></td>
</tr>
<tr>
<td><strong>IP_ADD_SOURCE_MEMBERSHIP</strong></td>
<td>Contains the IP_MREQ_SOURCE structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ_SOURCE structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 source address and a 4-byte IPv4 interface address.</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ-SOURCE.</td>
<td></td>
</tr>
<tr>
<td>OPTNAME options (input)</td>
<td>SETSOCKOPT, OPTVAL (input)</td>
<td>GETSOCKOPT, OPTVAL (output)</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td><strong>IP_BLOCK_SOURCE</strong></td>
<td>Contains the IP_MREQ_SOURCE structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ_SOURCE structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 source address and a 4-byte IPv4 interface address.</td>
<td>N/A</td>
</tr>
<tr>
<td>Use this option to enable an application to block multicast packets that have a source address that matches the given IPv4 source address. You must specify an interface and a source address with this option. The specified multicast group must have been joined previously.</td>
<td>See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE.</td>
<td></td>
</tr>
<tr>
<td>This is an IPv4-only socket option.</td>
<td>See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ-SOURCE.</td>
<td></td>
</tr>
<tr>
<td><strong>IP_DROP_MEMBERSHIP</strong></td>
<td>Contains the IP_MREQ structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 interface address.</td>
<td>N/A</td>
</tr>
<tr>
<td>Use this option to enable an application to exit a multicast group or to exit all sources for a multicast group.</td>
<td>See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ.</td>
<td></td>
</tr>
<tr>
<td>This is an IPv4-only socket option.</td>
<td>See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ.</td>
<td></td>
</tr>
<tr>
<td><strong>IP_DROP_SOURCE_MEMBERSHIP</strong></td>
<td>Contains the IP_MREQ_SOURCE structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ_SOURCE structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 source address and a 4-byte IPv4 interface address.</td>
<td>N/A</td>
</tr>
<tr>
<td>Use this option to enable an application to exit a source multicast group.</td>
<td>See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE.</td>
<td></td>
</tr>
<tr>
<td>This is an IPv4-only socket option.</td>
<td>See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ-SOURCE.</td>
<td></td>
</tr>
<tr>
<td><strong>IP_MULTICAST_IF</strong></td>
<td>A 4-byte binary field containing an IPv4 interface address.</td>
<td>A 4-byte binary field containing an IPv4 interface address.</td>
</tr>
<tr>
<td>Use this option to set or obtain the IPv4 interface address used for sending outbound multicast datagrams from the socket application.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>This is an IPv4-only socket option.</td>
<td>Note: Multicast datagrams can be transmitted only on one interface at a time.</td>
<td></td>
</tr>
<tr>
<td>OPTNAME options (input)</td>
<td>SETSOCKOPT, OPTVAL (input)</td>
<td>GETSOCKOPT, OPTVAL (output)</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>----------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td><strong>IP_MULTICAST_LOOP</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use this option to control or determine</td>
<td>A 1-byte binary field.</td>
<td></td>
</tr>
<tr>
<td>whether a copy of multicast datagrams are</td>
<td>To enable, set to 1.</td>
<td></td>
</tr>
<tr>
<td>looped back for multicast datagrams sent to</td>
<td>To disable, set to 0.</td>
<td></td>
</tr>
<tr>
<td>a group to which the sending host itself</td>
<td></td>
<td></td>
</tr>
<tr>
<td>belongs. The default is to loop the</td>
<td></td>
<td></td>
</tr>
<tr>
<td>datagrams back.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>This is an IPv4-only socket option.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>IP_MULTICAST_TTL</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use this option to set or obtain the IP</td>
<td>A 1-byte binary field</td>
<td>A 1-byte binary field</td>
</tr>
<tr>
<td>time-to-live of outgoing multicast</td>
<td>containing the value of '00'x to 'FF'x.</td>
<td>containing the value of '00'x to 'FF'x.</td>
</tr>
<tr>
<td>datagrams. The default value is '01'x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>meaning that multicast is available only to</td>
<td></td>
<td></td>
</tr>
<tr>
<td>the local subnet.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>This is an IPv4-only socket option.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>IP_UNBLOCK_SOURCE</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use this option to enable an application to</td>
<td>Contains the IP_MREQ_SOURCE</td>
<td></td>
</tr>
<tr>
<td>unblock a previously blocked source for a</td>
<td>structure as defined in</td>
<td></td>
</tr>
<tr>
<td>given IPv4 multicast group. You must</td>
<td>SYS1.MACLIB(BPXYSOCK). The</td>
<td></td>
</tr>
<tr>
<td>specify an interface and a source address</td>
<td>IP_MREQ_SOURCE structure</td>
<td></td>
</tr>
<tr>
<td>with this option.</td>
<td>contains a 4-byte IPv4</td>
<td></td>
</tr>
<tr>
<td>This is an IPv4-only socket option.</td>
<td>multicast address followed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>by a 4-byte IPv4 source</td>
<td></td>
</tr>
<tr>
<td></td>
<td>address and a 4-byte IPv4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>interface address.</td>
<td></td>
</tr>
<tr>
<td>See SEZAINST(CBLOCK) for the PL/I example</td>
<td>See SEZAINST(EZACOBOL) for</td>
<td></td>
</tr>
<tr>
<td>of IP_MREQ_SOURCE.</td>
<td>the COBOL example of IP-MREQ-SOURCE.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 20. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPV6_ADDR_PREFERENCES</td>
<td>Contains the 4-byte flags field IPV6_ADDR_PREFERENCES_FLAGS that is defined in SYS1.MACLIB(BPXYSOCK) with the following flags:</td>
<td>Contains the 4-byte flags field IPV6_ADDR_PREFERENCES_FLAGS that is defined in SYS1.MACLIB(BPXYSOCK) with the following flags:</td>
</tr>
<tr>
<td></td>
<td>IPV6_PREFSRC_HOME</td>
<td>IPV6_PREFSRC_HOME</td>
</tr>
<tr>
<td></td>
<td>(X'00000001')</td>
<td>(X'00000001')</td>
</tr>
<tr>
<td></td>
<td>Prefer home address</td>
<td>Prefer home address</td>
</tr>
<tr>
<td></td>
<td>IPV6_PREFSRC_COA</td>
<td>IPV6_PREFSRC_COA</td>
</tr>
<tr>
<td></td>
<td>(X'00000002')</td>
<td>(X'00000002')</td>
</tr>
<tr>
<td></td>
<td>Prefer care-of address</td>
<td>Prefer care-of address</td>
</tr>
<tr>
<td></td>
<td>IPV6_PREFSRC_TMP</td>
<td>IPV6_PREFSRC_TMP</td>
</tr>
<tr>
<td></td>
<td>(X'00000004')</td>
<td>(X'00000004')</td>
</tr>
<tr>
<td></td>
<td>Prefer temporary address</td>
<td>Prefer temporary address</td>
</tr>
<tr>
<td></td>
<td>IPV6_PREFSRC_PUBLIC</td>
<td>IPV6_PREFSRC_PUBLIC</td>
</tr>
<tr>
<td></td>
<td>(X'00000008')</td>
<td>(X'00000008')</td>
</tr>
<tr>
<td></td>
<td>Prefer public address</td>
<td>Prefer public address</td>
</tr>
<tr>
<td></td>
<td>IPV6_PREFSRC_CGA</td>
<td>IPV6_PREFSRC_CGA</td>
</tr>
<tr>
<td></td>
<td>(X'00000010')</td>
<td>(X'00000010')</td>
</tr>
<tr>
<td></td>
<td>Prefer cryptographically generated address</td>
<td>Prefer cryptographically generated address</td>
</tr>
<tr>
<td></td>
<td>IPV6_PREFSRC_NONCGA</td>
<td>IPV6_PREFSRC_NONCGA</td>
</tr>
<tr>
<td></td>
<td>(X'00000020')</td>
<td>(X'00000020')</td>
</tr>
<tr>
<td></td>
<td>Prefer non-cryptographically generated address</td>
<td>Prefer non-cryptographically generated address</td>
</tr>
<tr>
<td></td>
<td>Some of these flags are contradictory. Combining contradictory flags, such as IPV6_PREFSRC_CGA and IPV6_PREFSRC_NONCGA, results in error code EINVAL.</td>
<td>See IPV6_ADDR_PREFERENCES and Mapping of GAI_HINTS/GAI_ADDRINFO EFLAGS in SEZAINST(CBLOCK) for the PL/I example of the OPTNAME and flag definitions.</td>
</tr>
<tr>
<td></td>
<td>See IPV6_ADDR_PREFERENCES and Mapping of GAI_HINTS/GAI_ADDRINFO EFLAGS in SEZAINST(CBLOCK) for the PL/I example of the OPTNAME and flag definitions.</td>
<td>See IPV6_ADDR_PREFERENCES and AI_EFLAGS mappings in SEZAINST(EZACOBOL) for the COBOL example of the OPTNAME and flag definitions.</td>
</tr>
</tbody>
</table>

Guideline: Use the INET6_IS_SRCADDR function to test whether the source IP address matches one or more IPV6_ADDR_PREFERENCES flags.

Result: These flags are only preferences. The stack could assign a source IP address that does not conform to the IPV6_ADDR_PREFERENCES flags that you specify.

This is an AF_INET6-only socket option.
<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IPV6_JOIN_GROUP</strong></td>
<td>Contains the IPV6_MREQ structure as defined in SYS1.MACLIB(BPXYSOCK). The IPV6_MREQ structure contains a 16-byte IPv6 multicast address followed by a 4-byte IPv6 interface index number. If the interface index number is 0, then the stack chooses the local interface. See the SEZAINST(CBLOCK) for the PL/I example of IPV6_MREQ. See SEZAINST(EZACOBOL) for the COBOL example of IPV6-MREQ.</td>
<td>N/A</td>
</tr>
<tr>
<td>Use this option to control the reception of multicast packets and specify that the socket join a multicast group. This is an IPv6-only socket option.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>IPV6_LEAVE_GROUP</strong></td>
<td>Contains the IPV6_MREQ structure as defined in SYS1.MACLIB(BPXYSOCK). The IPV6_MREQ structure contains a 16-byte IPv6 multicast address followed by a 4-byte IPv6 interface index number. If the interface index number is 0, then the stack chooses the local interface. See the SEZAINST(CBLOCK) for the PL/I example of IPV6_MREQ. See SEZAINST(EZACOBOL) for the COBOL example of IPV6-MREQ.</td>
<td>N/A</td>
</tr>
<tr>
<td>Use this option to control the reception of multicast packets and specify that the socket leave a multicast group. This is an IPv6-only socket option.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>IPV6_MULTICAST_HOPS</strong></td>
<td>Contains a 4-byte binary value specifying the multicast hops. If not specified, then the default is 1 hop. -1 indicates use stack default. 0 – 255 is the valid hop limit range. <strong>Note:</strong> An application must be APF authorized to enable it to set the hop limit value above the system defined hop limit value. CICS applications cannot execute as APF authorized.</td>
<td>Contains a 4-byte binary value in the range 0 – 255 indicating the number of multicast hops.</td>
</tr>
<tr>
<td>Use to set or obtain the hop limit used for outgoing multicast packets. This is an IPv6-only socket option.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPTNAME options (input)</td>
<td>SETSOCKOPT, OPTVAL (input)</td>
<td>GETSOCKOPT, OPTVAL (output)</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>IPV6_MULTICAST_IF</td>
<td>Contains a 4-byte binary field containing an IPv6 interface index number.</td>
<td>Contains a 4-byte binary field containing an IPv6 interface index number.</td>
</tr>
<tr>
<td></td>
<td>This is an IPv6-only socket option.</td>
<td></td>
</tr>
<tr>
<td>IPV6_MULTICAST_LOOP</td>
<td>A 4-byte binary field.</td>
<td>A 4-byte binary field.</td>
</tr>
<tr>
<td></td>
<td>To enable, set to 1.</td>
<td>If enabled, contains a 1.</td>
</tr>
<tr>
<td></td>
<td>To disable, set to 0.</td>
<td>If disabled, contains a 0.</td>
</tr>
<tr>
<td></td>
<td>This is an IPv6-only socket option.</td>
<td></td>
</tr>
<tr>
<td>IPV6_UNICAST_HOPS</td>
<td>Contains a 4-byte binary value specifying the unicast hops. If not specified, then the default is 1 hop.</td>
<td>Contains a 4-byte binary value in the range 0 – 255 indicating the number of unicast hops.</td>
</tr>
<tr>
<td></td>
<td>-1 indicates use stack default.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 – 255 is the valid hop limit range.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> APF authorized applications are permitted to set a hop limit that exceeds the system configured default. CICS applications cannot execute as APF authorized.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This is an IPv6-only socket option.</td>
<td></td>
</tr>
<tr>
<td>IPV6_V6ONLY</td>
<td>A 4-byte binary field.</td>
<td>A 4-byte binary field.</td>
</tr>
<tr>
<td></td>
<td>To enable, set to 1.</td>
<td>If enabled, contains a 1.</td>
</tr>
<tr>
<td></td>
<td>To disable, set to 0.</td>
<td>If disabled, contains a 0.</td>
</tr>
<tr>
<td></td>
<td>This is an IPv6-only socket option.</td>
<td></td>
</tr>
<tr>
<td>OPTNAME options (input)</td>
<td>SETSOCKOPT, OPTVAL (input)</td>
<td>GETSOCKOPT, OPTVAL (output)</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td><strong>MCAST_BLOCK_SOURCE</strong></td>
<td>Contains the GROUP_SOURCE_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_SOURCE_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address and a socket address structure of the source address.</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>See SEZAINST(CBLOCK) for the PL/I example of GROUP_SOURCE_REQ.</td>
<td>See SEZAINST(EZACOBOL) for the COBOL example of GROUP-SOURCE-REQ.</td>
</tr>
<tr>
<td><strong>MCAST_JOIN_GROUP</strong></td>
<td>Contains the GROUP_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address.</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>See SEZAINST(CBLOCK) for the PL/I example of GROUP_REQ.</td>
<td>See SEZAINST(EZACOBOL) for the COBOL example of GROUP-REQ.</td>
</tr>
<tr>
<td><strong>MCAST_JOIN_SOURCE_GROUP</strong></td>
<td>Contains the GROUP_SOURCE_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_SOURCE_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address and a socket address structure of the source address.</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>See SEZAINST(CBLOCK) for the PL/I example of GROUP_SOURCE_REQ.</td>
<td>See SEZAINST(EZACOBOL) for the COBOL example of GROUP-SOURCE-REQ.</td>
</tr>
<tr>
<td>OPTNAME options (input)</td>
<td>SETSOCKOPT, OPTVAL (input)</td>
<td>GETSOCKOPT, OPTVAL (output)</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td><strong>MCAST_LEAVE_GROUP</strong></td>
<td>Contains the GROUP_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address. See SEZAINST(CBLOCK) for the PL/I example of GROUP_REQ. See SEZAINST(EZACOBOL) for the COBOL example of GROUP-REQ.</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>MCAST_LEAVE_SOURCE_GROUP</strong></td>
<td>Contains the GROUP_SOURCE_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_SOURCE_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address and a socket address structure of the source address. See SEZAINST(CBLOCK) for the PL/I example of GROUP_SOURCE_REQ. See SEZAINST(EZACOBOL) for the COBOL example of GROUP-SOURCE-REQ.</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>MCAST_UNBLOCK_SOURCE</strong></td>
<td>Contains the GROUP_SOURCE_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_SOURCE_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address and a socket address structure of the source address. See SEZAINST(CBLOCK) for the PL/I example of GROUP_SOURCE_REQ. See SEZAINST(EZACOBOL) for the COBOL example of GROUP-SOURCE-REQ.</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Table 20. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SO_ASCII</strong></td>
<td>To enable, set to ON.</td>
<td>If enabled, contains ON.</td>
</tr>
<tr>
<td></td>
<td>To disable, set to OFF.</td>
<td>If disabled, contains OFF.</td>
</tr>
<tr>
<td><strong>Note:</strong> This is a REXX-only socket option.</td>
<td><strong>Note:</strong> The optvalue is returned and is optionally followed by the name of the translation table that is used if translation is applied to the data.</td>
<td><strong>Note:</strong> The optvalue is returned and is optionally followed by the name of the translation table that is used if translation is applied to the data.</td>
</tr>
<tr>
<td><strong>SO_BROADCAST</strong></td>
<td>A 4-byte binary field.</td>
<td>A 4-byte field.</td>
</tr>
<tr>
<td></td>
<td>To enable, set to 1 or a positive value.</td>
<td>If enabled, contains a 1.</td>
</tr>
<tr>
<td></td>
<td>To disable, set to 0.</td>
<td>If disabled, contains 0.</td>
</tr>
<tr>
<td><strong>Note:</strong> This option has no meaning for stream sockets.</td>
<td><strong>Note:</strong> This is a REXX-only socket option.</td>
<td><strong>Note:</strong> This is a REXX-only socket option.</td>
</tr>
<tr>
<td><strong>SO_DEBUG</strong></td>
<td>To enable, set to ON.</td>
<td>If enabled, contains ON.</td>
</tr>
<tr>
<td></td>
<td>To disable, set to OFF.</td>
<td>If disabled, contains OFF.</td>
</tr>
<tr>
<td><strong>Note:</strong> 1. This is a REXX-only socket option. 2. This option has meaning only for stream sockets.</td>
<td><strong>Note:</strong> This is a REXX-only socket option.</td>
<td><strong>Note:</strong> This is a REXX-only socket option.</td>
</tr>
<tr>
<td><strong>SO_EBCDIC</strong></td>
<td>To enable, set to ON.</td>
<td>If enabled, contains ON.</td>
</tr>
<tr>
<td></td>
<td>To disable, set to OFF.</td>
<td>If disabled, contains OFF.</td>
</tr>
<tr>
<td><strong>Note:</strong> This is a REXX-only socket option.</td>
<td><strong>Note:</strong> This is a REXX-only socket option.</td>
<td><strong>Note:</strong> This is a REXX-only socket option.</td>
</tr>
<tr>
<td><strong>SO_ERROR</strong></td>
<td>N/A</td>
<td>A 4-byte binary field containing the most recent ERRNO for the socket.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> This is a REXX-only socket option.</td>
</tr>
<tr>
<td>OPTNAME options (input)</td>
<td>SETSOCKOPT, OPTVAL (input)</td>
<td>GETSOCKOPT, OPTVAL (output)</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td><strong>SO_KEEPALIVE</strong></td>
<td>A 4-byte binary field.</td>
<td>A 4-byte binary field.</td>
</tr>
<tr>
<td>Use this option to set or determine whether the keep alive mechanism periodically sends a packet on an otherwise idle connection for a stream socket.</td>
<td>To enable, set to 1 or a positive value.</td>
<td>If enabled, contains a 1.</td>
</tr>
<tr>
<td>The default is disabled.</td>
<td>To disable, set to 0.</td>
<td>If disabled, contains a 0.</td>
</tr>
<tr>
<td>When activated, the keep alive mechanism periodically sends a packet on an otherwise idle connection. If the remote TCP does not respond to the packet or to retransmissions of the packet, the connection is terminated with the error ETIMEDOUT.</td>
<td>A 4-byte binary field.</td>
<td>Contains an 8-byte field containing two 4-byte binary fields.</td>
</tr>
<tr>
<td><strong>SO_LINGER</strong></td>
<td>Contains an 8-byte field containing two 4-byte binary fields.</td>
<td>A nonzero value returned in ONOFF indicates enabled, a 0 indicates disabled. LINGER indicates the number of seconds that TCP/IP will try to send data after the CLOSE is issued.</td>
</tr>
<tr>
<td>Use this option to control or determine how TCP/IP processes data that has not been transmitted when a CLOSE is issued for the socket. The default is disabled.</td>
<td>Assembler coding: ONOFF DS F LINGER DS F</td>
<td>COBOL coding: ONOFF PIC 9(8) BINARY. LINGER PIC 9(8) BINARY.</td>
</tr>
<tr>
<td>Note:</td>
<td>COBOL coding: ONOFF PIC 9(8) BINARY. LINGER PIC 9(8) BINARY.</td>
<td>A nonzero value returned in ONOFF indicates enabled, a 0 indicates disabled. LINGER indicates the number of seconds that TCP/IP will try to send data after the CLOSE is issued.</td>
</tr>
<tr>
<td>1. This option has meaning only for stream sockets.</td>
<td>Set ONOFF to a nonzero value to enable and set to 0 to disable this option. Set LINGER to the number of seconds that TCP/IP lingers after the CLOSE is issued.</td>
<td></td>
</tr>
<tr>
<td>2. If you set a zero linger time, the connection cannot close in an orderly manner, but stops, resulting in a RESET segment being sent to the connection partner. Also, if the aborting socket is in nonblocking mode, the close call is treated as though no linger option had been set.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>When SO_LINGER is set and CLOSE is called, the calling program is blocked until the data is successfully transmitted or the connection has timed out.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>When SO_LINGER is not set, the CLOSE returns without blocking the caller, and TCP/IP continues to attempt to send data for a specified time. This usually allows sufficient time to complete the data transfer.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use of the SO_LINGER option does not guarantee successful completion because TCP/IP waits only the amount of time specified in OPTVAL for SO_LINGER.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 20. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SO_OOBINLINE</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use this option to control or determine whether out-of-band data is received. <strong>Note:</strong> This option has meaning only for stream sockets.</td>
<td>A 4-byte binary field.</td>
<td>A 4-byte binary field.</td>
</tr>
<tr>
<td></td>
<td>To enable, set to 1 or a positive value.</td>
<td>If enabled, contains a 1.</td>
</tr>
<tr>
<td></td>
<td>To disable, set to 0.</td>
<td>If disabled, contains a 0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SO_RCVBUF</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use this option to control or determine the size of the data portion of the TCP/IP receive buffer.</td>
<td>A 4-byte binary field.</td>
<td>A 4-byte binary field.</td>
</tr>
<tr>
<td>The size of the data portion of the receive buffer is protocol-specific, based on the following values prior to any SETSOCKOPT call:</td>
<td>To enable, set to a positive value specifying the size of the data portion of the TCP/IP receive buffer.</td>
<td>If enabled, contains a positive value indicating the size of the data portion of the TCP/IP receive buffer.</td>
</tr>
<tr>
<td>• TCPRCVBufsize keyword on the TCPCONFIG statement in the PROFILE.TCPIP data set for a TCP Socket</td>
<td>To disable, set to a 0.</td>
<td>If disabled, contains a 0.</td>
</tr>
<tr>
<td>• UDPRCVBufsize keyword on the UDPCONFIG statement in the PROFILE.TCPIP data set for a UDP Socket</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• The default of 65 535 for a raw socket</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 20. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SO_RCVTIMEO</td>
<td>This option requires a TIMEVAL structure, which is defined in SYS1.MACLIB (BPXYRLIM) macro. The TIMEVAL structure contains the number of seconds and microseconds specified as fullword binary numbers. The seconds can be a value in the range 0 - 2678400 (equal to 31 days), and the microseconds can be a value in the range 0 - 1000000 (equal to 1 second). Although TIMEVAL value can be specified using microsecond granularity, the internal TCP/IP timers that are used to implement this function have a granularity of approximately 100 milliseconds.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This option stores a TIMEVAL structure that is defined in the SYS1.MACLIB(BPXYRLIM) macro. The TIMEVAL structure contains the number of seconds and microseconds, which are specified as fullword binary numbers. The number of seconds value that is returned is in the range 0 - 2678400 (equal to 31 days). The number of microseconds value that is returned is in the range 0 - 1000000.</td>
<td></td>
</tr>
</tbody>
</table>

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

If a receive-type function has blocked for the maximum length of time that was specified without receiving data, control is returned with an errno set to EWOULDBLOCK. The default value for this option is 0, which indicates that a receive-type function does not time out.

When the MSG_WAITALL flag (stream sockets only) is specified, the timeout takes precedence. The receive-type function can return the partial count. See the explanation of that operation's MSG_WAITALL flag parameter.

The following receive-type functions are supported:
- READ
- READV
- RECV
- RECVFROM
- RECVMSG
Table 20. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

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

Use this option to control or determine whether local addresses are reused. The default is disabled. This alters the normal algorithm used with BIND. The normal BIND algorithm allows each Internet address and port combination to be bound only once. If the address and port have been already bound, then a subsequent BIND will fail and result error will be EADDRINUSE.

When this option is enabled, the following situations are supported:

- A server can BIND the same port multiple times as long as every invocation uses a different local IP address and the wildcard address INADDR_ANY is used only one time per port.
- A server with active client connections can be restarted and can bind to its port without having to close all of the client connections.
- For datagram sockets, multicasting is supported so multiple bind() calls can be made to the same class D address and port number.
- If you require multiple servers to BIND to the same port and listen on INADDR_ANY, see the SHAREPORT option on the PORT statement in TCPIP.PROFILE.

SO_SNDBUF

Use this option to control or determine the size of the data portion of the TCP/IP send buffer. The size is of the TCP/IP send buffer is protocol specific and is based on the following:

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

A 4-byte binary field.
To enable, set to a positive value indicating the size of the data portion of the TCP/IP send buffer.
To disable, set to a 0.

A 4-byte binary field.
If enabled, contains a positive value.
If disabled, contains a 0.
<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SO_SNDTIMEO</strong></td>
<td>This option requires a TIMEVAL structure, which is defined in the SYS1.MACLIB(BPXYRLIM) macro. The TIMEVAL structure contains the number of seconds and microseconds specified as fullword binary numbers. The seconds value is in the range 0 - 2678400 (equal to 31 days), and the microseconds value is in the range 0 - 1000000 (equal to 1 second). Although the TIMEVAL value can be specified using microsecond granularity, the internal TCP/IP timers that are used to implement this function have a granularity of approximately 100 milliseconds.</td>
<td>This option stores a TIMEVAL structure that is defined in SYS1.MACLIB(BPXYRLIM). The TIMEVAL structure contains the number of seconds and microseconds, which are specified as fullword binary numbers. The number of seconds value that is returned is in the range 0 - 2678400 (equal to 31 days). The microseconds value that is returned is in the range 0 - 100000.</td>
</tr>
<tr>
<td><strong>SO_TYPE</strong></td>
<td>N/A</td>
<td>This option stores a TIMEVAL structure that is defined in SYS1.MACLIB(BPXYRLIM). The TIMEVAL structure contains the number of seconds and microseconds, which are specified as fullword binary numbers. The number of seconds value that is returned is in the range 0 - 2678400 (equal to 31 days). The microseconds value that is returned is in the range 0 - 100000.</td>
</tr>
<tr>
<td><strong>TCP_KEEPALIVE</strong></td>
<td>A 4-byte binary field.</td>
<td>A 4-byte binary field.</td>
</tr>
<tr>
<td></td>
<td>To enable, set to a value in the range of 1 - 2147460.</td>
<td>If enabled, contains the specific timer value (in seconds) that is in effect for the given socket.</td>
</tr>
<tr>
<td></td>
<td>To disable, set to a value of 0.</td>
<td>If disabled, contains a 0 indicating keep alive timing is not active.</td>
</tr>
</tbody>
</table>
Table 20. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

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

Note: Use the following to set TCP_NODELAY OPTNAME value for COBOL programs:

```
01 TCP-NODELAY-VAL PIC 9(10) COMP VALUE 2147483649.
01 TCP-NODELAY-REDEF REDEFINES TCP-NODELAY-VAL.
05 FILLER PIC 9(6) BINARY.
05 TCP-NODELAY PIC 9(8) BINARY.
```

GIVESOCKET call

The GIVESOCKET call is used to pass a socket from one process to another.

UNIX-based platforms use a command called FORK to create a new child process that has the same descriptors as the parent process. You can use this new child process in the same way that you used the parent process.

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

1. A process issues a GETCLIENTID call to get the job name of its region and its MVS subtask identifier. This information is used in a GIVESOCKET call.
2. The process issues a GIVESOCKET call to prepare a socket for use by a child process.
3. The child process issues a TAKESOCKET call to get the socket. The socket now belongs to the child process, and can be used by TCP/IP to communicate with another process.

Note: The TAKESOCKET call returns a new socket descriptor in RETCODE. The child process must use this new socket descriptor for all calls that use this socket. The socket descriptor that was passed to the TAKESOCKET call must not be used.

4. After issuing the GIVESOCKET command, the parent process issues a SELECT command that waits for the child to get the socket.
5. When the child gets the socket, the parent receives an exception condition that releases the SELECT command.
6. The parent process closes the socket.
The original socket descriptor can now be reused by the parent.

Sockets which have been given, but not taken for a period of four days, are closed and are no longer be available for taking. If a select for the socket is outstanding, it is posted.

The following requirements apply to this call:

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

Figure 137 shows an example of GIVESOCKET call instructions.

WORKING-STORAGE SECTION.
01 SOC-FUNCTION PIC X(16) VALUE IS 'GIVESOCKET'.
01 S PIC 9(4) BINARY.
01 CLIENT.
   03 DOMAIN PIC 9(8) BINARY.
   03 NAME PIC X(8).
   03 TASK PIC X(8).
   03 RESERVED PIC X(20).
01 ERRNO PIC 9(8) BINARY.
01 RETCODE PIC S9(8) BINARY.

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

Figure 137. GIVESOCKET call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 253.

Parameter values set by the application for the GIVESOCKET call

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

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

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

DOMAIN A fullword binary number that must be set to a decimal 2, indicating AF_INET, or a decimal 19, indicating AF_INET6.
Rule: A socket given by GIVESOCKET can be taken only by a TAKESOCKET with the same DOMAIN, address family (such as, AF_INET or AF_INET6).

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

- If the socket-taking application is in the same address space as the socket-giving application (as in CICS), NAME can be specified. The socket-giving application can determine its own address space name by issuing the GETCLIENTID call.
- If the socket-taking application is in a different MVS address space (as in IMS), this field should be set to blanks. When this is done, any MVS address space that requests the socket can have it.

TASK
Specifies an 8-character field that can be set to blanks, or to the identifier of the socket-taking MVS subtask. If this field is set to blanks, any subtask in the address space specified in the NAME field can take the socket.

- If used by CICS IP sockets, the field should be set to blanks.
- If TASK identifier is nonblank, the socket-receiving task should already be in execution when the GIVESOCKET is issued.

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

Parameter values returned to the application for the GIVESOCKET call

ERRNO
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.

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

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

INET6_IS_SRCADDR call

The INET6_IS_SRCADDR call tests whether the input IP address matches an IP address in the node that conforms to all IPV6_ADDR_PREFERENCES flags specified in the call. You can use this call with IPv6 addresses or with IPv4-mapped IPv6 addresses.

You can use this call to test local IP addresses to verify that these addresses have the characteristics required by your application.

The following requirements apply to this call:

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

Figure 138 shows an example of INET6_IS_SRCADDR call instructions.

```
WORKING-STORAGE SECTION.
  01 SOC-FUNCTION PIC X(16) VALUE IS 'INET6_IS_SRCADDR'.
    * IPv6 socket address structure.
  01 NAME.
    03 FAMILY PIC 9(4) BINARY.
    03 PORT PIC 9(4) BINARY.
    03 FLOWINFO PIC 9(8) BINARY.
    03 IP-ADDRESS.
      10 FILLER PIC 9(16) BINARY.
      10 FILLER PIC 9(16) BINARY.
    03 SCOPE-ID PIC 9(8) BINARY.
  01 FLAGS PIC 9(8) BINARY
    88 IPV6-PREFER-SRC-HOME PIC 9(8) BINARY VALUE 1.
    88 IPV6-PREFER-SRC-COA PIC 9(8) BINARY VALUE 2.
    88 IPV6-PREFER-SRC-TMP PIC 9(8) BINARY VALUE 4.
    88 IPV6-PREFER-SRC-PUBLIC PIC 9(8) BINARY VALUE 8.
    88 IPV6-PREFER-SRC-CGA PIC 9(8) BINARY VALUE 16.
    88 IPV6-PREFER-SRC-NONCGA PIC 9(8) BINARY VALUE 32.
  01 ERRNO PIC 9(8) BINARY.
  01 RETCODE PIC S9(8) BINARY.

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

Figure 138. INET6_IS_SRCADDR call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 253.

Parameter values set by the application

SOC-FUNCTION

A 16-byte character field containing INET6_IS_SRCADDR.

NAME

Specifies the AF_INET6 socket address structure for the address that is to be tested.

**Requirement:** You must specify an AF_INET6 address. You can specify an IPv6 address or an IPv4-mapped IPv6 address.

The IPv6 socket address structure specifies the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
</table>
FAMILY
A halfword binary field specifying the IPv6 addressing family. For TCP/IP, the value is decimal 19, indicating AF_INET6.

PORT
A halfword binary field. This field is ignored by INET6_IS_SRCADDR processing.

FLOWINFO
A fullword binary field specifying the traffic class and flow label. This field is ignored by INET6_IS_SRCADDR processing.

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

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

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

Requirements:
- If the IP address is a link-local address, this field must be set to a nonzero value.
- If the IP address is not a link-local address, this field must be set to 0.

FLAGS
A fullword binary field containing one or more IPV6_ADDR_PREFERENCES flags. The following table defines the valid IPV6_ADDR_PREFERENCES flags.

<table>
<thead>
<tr>
<th>Flag name</th>
<th>Binary value</th>
<th>Decimal value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPV6-PREFER-SRC-HOME</td>
<td>x'00000001'</td>
<td>1</td>
<td>Test whether the input IP address is a home address.</td>
</tr>
<tr>
<td>IPV6-PREFER-SRC-COA</td>
<td>x'00000002'</td>
<td>2</td>
<td>Test whether the input IP address is a care-of address.</td>
</tr>
<tr>
<td>IPV6-PREFER-SRC-TMP</td>
<td>x'00000004'</td>
<td>4</td>
<td>Test whether the input IP address is a temporary address.</td>
</tr>
<tr>
<td>IPV6-PREFER-SRC-PUBLIC</td>
<td>x'00000008'</td>
<td>8</td>
<td>Test whether the input IP address is a public address.</td>
</tr>
<tr>
<td>IPV6-PREFER-SRC-CGA</td>
<td>x'00000010'</td>
<td>16</td>
<td>Test whether the input IP address is cryptographically generated.</td>
</tr>
<tr>
<td>Flag name</td>
<td>Binary value</td>
<td>Decimal value</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------</td>
<td>---------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IPV6-PREFER-SRC-NONCGA</td>
<td>x'00000020'</td>
<td>32</td>
<td>Test whether the input IP address is not cryptographically generated.¹</td>
</tr>
</tbody>
</table>

**Note:**
1. Any valid IP address that is known to the stack satisfies this flag.
2. z/OS Communications Server does not support this type of address. The call always returns FALSE when this flag is specified with a valid IP address that is known to the stack.

**Tips:**
- The samples SEZAINST(EZACOBOL) and SEZAINST(CBLOCK) contain mappings for these flags.
- Some of these flags are contradictory. For example:
  - The flag IPV6_PREFER_SRC_HOME contradicts the flag IPV6_PREFER_SRC_COA.
  - The flag IPV6_PREFER_SRC_CGA contradicts the flag IPV6_PREFER_SRC_NONCGA.
  - The flag IPV6_PREFER_SRC_TMP contradicts the flags IPV6_PREFER_SRC_PUBLIC.

**Result:** If you specify contradictory flags in the call, the result is FALSE.

**Parameter values returned to the application**

**ERRNO**
A fullword binary field. If RETCODE is negative, this field contains an error number. See Appendix F, “Related protocol specifications,” on page 649 for information about ERRNO return codes.

**RETCODE**
A fullword binary field that is set to one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>FALSE</td>
</tr>
<tr>
<td></td>
<td>The call was successful and the result is FALSE. The input AF_INET6 address</td>
</tr>
<tr>
<td></td>
<td>corresponds to an IP address on the node, but does not conform to one or</td>
</tr>
<tr>
<td></td>
<td>more of the IPV6_ADDR_PREFERENCES flags specified in the call.</td>
</tr>
<tr>
<td>1</td>
<td>TRUE</td>
</tr>
<tr>
<td></td>
<td>The call was successful, and the result is TRUE. The input AF_INET6 address</td>
</tr>
<tr>
<td></td>
<td>corresponds to an IP address on the node, and conforms to all the IPV6_</td>
</tr>
<tr>
<td></td>
<td>ADDR_PREFERENCES flags specified in the call.</td>
</tr>
<tr>
<td>-1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

**INITAPI and INITAPIX calls**

The INITAPI and INITAPIX calls connect an application to the TCP/IP interface. The sole difference between INITAPI and INITAPIX is explained in the description of the IDENT parameter. INITAPI is preferred over INITAPIX unless there is a
specific need to connect applications to alternate TCP/IP stacks. CICS sockets programs that are written in COBOL, PL/I, or assembler language should issue the INITAPI or INITAPIX macro before they issue other calls to the CICS sockets interface.

If a CICS task's first call to the CICS socket interface is not an INITAPI or INITAPIX, then the CICS socket interface generates a default INITAPI call.

The following requirements apply to this call:

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

Figure 139 on page 321 shows an example of INITAPI call instructions. The same example can be used for the INITAPIX call by simply changing the SOC-FUNCTION value to 'INITAPIX'.

320  z/OS V2R1.0 Communications Server: IP CICS Sockets Guide
WORKING-STORAGE SECTION.
 01 SOC-FUNCTION PIC X(16) VALUE IS 'INITAPI'.
 01 MAXSOC-FWD PIC 9(8) BINARY.
 01 MAXSOC-RDF REDEFINES MAXSOC-FWD.
    02 FILLER PIC X(2).
    02 MAXSOC PIC 9(4) BINARY.
 01 IDENT.
    02 TCPNAME PIC X(8).
    02 ADSNAME PIC X(8).
 01 SUBTASK PIC X(8).
 01 MAXSNO PIC 9(8) BINARY.
 01 ERRNO PIC 9(8) BINARY.
 01 RETCODE PIC 9(8) BINARY.

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

Figure 139. INITAPI call instruction example

For equivalent PL/I and assembler language declarations, see [Converting parameter descriptions" on page 253](#).

Parameter values set by the application for the INITAPI and INITAPIX calls

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

MAXSOC
A halfword binary field set to the maximum number of sockets this application ever has open at one time. The maximum number is 65535 and the minimum number is 50. This value is used to determine the amount of memory that is allocated for socket control blocks and buffers. If less than 50 are requested, MAXSOC defaults to 50.

IDENT
A 16-byte structure containing the name of the TCP/IP address space (TCPNAME) and the name of calling program’s address space (ADSNAME).

The way that the CICS socket interface handles the TCPNAME part of the structure differs between INITAPI and INITAPIX (as explained in the following description of TCPNAME).

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

If the function is INITAPI, then the CICS socket interface always overrides this with the value in the TCPADDR configuration parameter. In OS/390® V2R8 and earlier, the INITAPIX functions the same way. In z/OS V1R1 and higher, the TCPNAME passed by the application program on an INITAPIX call overrides the TCPADDR value.

ADSNAME
An 8-byte character field set to the identity of the calling program’s address space. It is the name of the CICS startup job. The CICS socket interface always overrides this value with VTAM APPLID of the CICS address space. For explicit-mode IMS server programs,
use the TIMSrvAddrSpc field passed in the TIM. If ADSNAME is not specified, the system derives a value from the MVS control block structure.

**SUBTASK**
Indicates an 8-byte field containing a unique subtask identifier that is used to distinguish between multiple subtasks within a single address space. For your subtask name, use the zoned decimal value of the CICS task ID (EIBTASKN), plus a unique displayable character. In CICS, if no value is specified, the zoned-decimal value of the CICS task ID appended with the letter C is used.

**Result:** Using the letter L as the last character in the subtask parameter causes the tasking mechanism to assume the CICS transaction is a listener and schedule it using a non-reusable subtask by way of MVS attach processing when OTE=NO. This has no effect when OTE=YES.

**Parameter values returned to the application for the INITAPI and INITAPIX calls**

**MAXSNO**
A fullword binary field that contains the highest socket number assigned to this application. The lowest socket number is zero. If you have 50 sockets, they are numbered from 0 to 49. If MAXSNO is not specified, the value for MAXSNO is 49.

**ERRNO**
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.

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

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

**IOCTL call**
The IOCTL call is used to control certain operating characteristics for a socket.

Before you issue an IOCTL call, you must load a value representing the characteristic that you want to control into the COMMAND field.

The variable length parameters REQARG and RETARG are arguments that are passed to and returned from IOCTL. The length of REQARG and RETARG is determined by the value that you specify in COMMAND. See Table 21 on page 331 for information about REQARG and RETARG.

The following requirements apply to this call:

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

Note: See “Addressability mode (Amode) considerations” under “Environmental restrictions and programming requirements for the Callable Socket API” on page 249.

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

Figure 140 shows an example of IOCTL call instructions.

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

01 IFREQ.
  05 NAME PIC X(16).
  05 FAMILY PIC 9(4) BINARY.
  05 PORT PIC 9(4) BINARY.
  05 ADDRESS PIC 9(8) BINARY.
  05 FILLER PIC X(8).

01 IFREQOUT.
  05 NAME PIC X(16).
  05 FAMILY PIC 9(4) BINARY.
  05 PORT PIC 9(4) BINARY.
  05 ADDRESS PIC 9(8) BINARY.
  05 FILLER PIC X(8).

01 GRP-IOCTL-TABLE.
  05 IOCTL-ENTRY OCCURS 1 TO max TIMES DEPENDING ON count.
    10 NAME PIC X(16).
    10 FAMILY PIC 9(4) BINARY.
    10 PORT PIC 9(4) BINARY.
    10 ADDRESS PIC 9(8) BINARY.
    10 FILLER PIC X(8).

01 IOCTL-REQARG USAGE IS POINTER.
01 IOCTL-RETARG USAGE IS POINTER.
01 ERRNO PIC 9(8) BINARY.
01 RETCODE PIC 9(8) BINARY.

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

Figure 140. IOCTL call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 253.
Parameter values set by the application for the IOCTL call

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

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

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

FIONBIO
Sets or clears blocking status.

FIONREAD
Returns the number of immediately readable bytes for the socket.

SIOCGHOMEIF6
Requests all IPv6 home interfaces. When the SIOCGHOMEIF6 IOCTL is issued, the REQARG must contain a Network Configuration Header. The NETCONFHDR is defined in SYS1.MACLIB(BPXYIOC6) for Assembler programs.

To request OSM interfaces the application must have READ authorization to the EZB.OM.sysname.tcpname resource.

Requirement: The following input fields must be filled out:

NchEyeCatcher
Contains eye catcher '6NCH'.

NchIoctl
Contains the command code.

NchBufferLength
Buffer length large enough to contain all the IPv6 interface records. Each interface record is length of HOME-IF-ADDRESS. If buffer is not large enough, then errno is set to ERANGE and the NchNumEntryRet is set to number of interfaces. Based on NchNumEntryRet and size of HOME-IF-ADDRESS, calculate the necessary storage to contain the entire list.

NchBufferPtr
This is a pointer to an array of HOME-IF structures returned on a successful call. The size depends on the number of qualifying interfaces returned.

NchNumEntryRet
If return code is zero, this is set to number of HOME-IF-ADDRESS returned. If errno is ERANGE, then this is set to number of qualifying interfaces. No interfaces are returned. Recalculate the NchBufferLength based on this value times the size of HOME-IF-ADDRESS.
REQARG and RETARG
Point to the arguments that are passed between the calling program and IOCTL. The length of the argument is determined by the COMMAND request. REQARG is an input parameter and is used to pass arguments to IOCTL. RETARG is an output parameter and is used for arguments returned by IOCTL. For the lengths and meanings of REQARG and RETARG for each COMMAND type, see Table 21 on page 331.

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

SIOCGIFADDR
Requests the network interface address for a given interface name. See the following source members for a description of the REQARG value of this IOCTL command:
- For assembler, see the IOCN_IFNAME field in SYS1.MACLIB(BPXYIOCC).
- For COBOL, see the IFR-NAME field in SEZAINST(EZACOBOL).
- For PL/I, see the IFR_NAME field in SEZAINST(CBLOCK).

SIOCGIFBRDADDR
Requests the network interface broadcast address for a given
interface name. See the following source members for a description of the REQARG value of this IOCTL command:

- For assembler, see the IOCN_IFNAME field in SYS1.MACLIB(BPXYIOCC).
- For COBOL, see the IFR-NAME field in SEZAINST(EZACOBOL).
- For PL/I, see the IFR_NAME field in SEZAINST(CBLOCK).

**SIOCGIFCONF**

Requests the network interface configuration. The configuration consists of a variable number of 32-byte structures. The SIOCGIFCONF structure is specified the REQARG value for this IOCTL command. For assembler, see the IOCN_IFREQ field in SYS1.MACLIB(BPXYIOCC) for the structure format. For COBOL, see IFREQ in SEZAINST(EZACOBOL) for the structure format. For PL/I, see IFREQ in SEZAINST(CBLOCK) for the structure format.

When IOCTL is issued, the REQARG field 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 that is being requested. The maximum number of array elements that TCP/IP can return is 100.

When IOCTL is issued, the RETARG field must be set to the beginning portion of the storage area that you have defined in your program for the array to be returned.

**SIOCGIFDSTADDR**

Requests the network interface destination address for a given interface name. See the following source members for a description of this IOCTL commands REQARG value:

- For assembler, see the IOCN_IFNAME field in SYS1.MACLIB(BPXYIOCC).
- For COBOL, see the IFR-NAME field in SEZAINST(EZACOBOL).
- For PL/I, see the IFR_NAME field in SEZAINST(CBLOCK).

**SIOCGIFMTU**

Requests the IPv4 network interface MTU (maximum transmission unit) for a given interface name. See the following source members for a description the REQARG value of this IOCTL command:

- For assembler, see the IOCN_IFNAME field in SYS1.MACLIB(BPXYIOCC).
- For COBOL, see the IFR-NAME field in SEZAINST(EZACOBOL).
- For PL/I, see the IFR_NAME field in SEZAINST(CBLOCK).

**SIOCGIFNAMEINDEX**

Requests all interface names and indexes including local loopback but excluding VIPAs. Information is returned for both IPv4 and IPv6 interfaces whether they are active or inactive. For IPv6 interfaces, information is returned for an interface only if it has at least one available IP address. The configuration consists of the IF_NAMEINDEX structure [defined in SYS1.MACLIB(BPX1IOCC) for assembler programs].
When the SIOCGIFNAMEINDEX IOCTL is issued, the first word in REQARG must contain the length (in bytes) to contain an IF-NAME-INDEX structure to return the interfaces. The following steps describe how to compute this length as follows:

1. Determine the number of interfaces expected to be returned upon successful completion of this command.
2. Multiply the number of interfaces by the array element (size of IF-NINDEX, IF-NINAME, and IF-NIEXT) to get the size of the array element.
3. To the size of the array, add the size of IF-NITOTALIF and IF-NIENTRIES to get the total number of bytes needed to accommodate the name and index information returned.

When IOCTL is issued, RETARG must be set to the address of the beginning of the area in your program's storage that is reserved for the IF-NAMEINDEX structure that IOCTL returns.

The 'SIOCGIFNAMEINDEX' command returns a variable number of all the qualifying network interfaces.

To request OSM interfaces the application must have READ authorization to the EZB.OM.sysname.tcpname resource.

WORKING-STORAGE SECTION.
01 SIOCGIFNAMEINDEX PIC X(4) VALUE '4000F603'.
01 reqarg pic 9(8) binary.
01 reqarg-header-only pic 9(8) binary.

01 IF-NIHEADER.
  05 IF-NITOTALIF PIC 9(8) BINARY.
  05 IF-NIENTRIES PIC 9(8) BINARY.

01 IF-NAME-INDEX-ENTRY.
  05 IF-NINDEX PIC 9(8) BINARY.
  05 IF-NINAME PIC X(16).
  05 IF-NINAMETERM PIC X(1).
  05 IF-NIRESV1 PIC X(3).

01 OUTPUT-STORAGE PIC X(500).

Procedure Division.
move 8 to reqarg-header-only.
Call 'EZASOKET' using soket-ioctl socket-descriptor SIOCGIFNAMEINDEX
  REQARG-HEADER-ONLY IF-NIHEADER
  errno retcode.

move 500 to reqarg.
Call 'EZASOKET' using soket-ioctl socket-descriptor SIOCGIFNAMEINDEX
  REQARG OUTPUT-STORAGE
  errno retcode.

Figure 142. COBOL language example for SIOCGIFNAMEINDEX

SIOCGIPMSFILTER
Requests a list of the IPv4 source addresses that comprise the source filter along with the current mode on a given interface and a multicast group for a socket. The source filter can include or exclude the set of source addresses, depending on the filter mode (MCAST_INCLUDE or MCAST_EXCLUDE).
When the SIOCGIPMSFILTER IOCTL is issued, the REQARG parameter must contain a IP_MSFILTER structure; this structure is defined in SYS1.MACLIB(BPXYIOCC) for assembler, in SEZAINST(CBLOCK) for PL/I, and in SEZAINST(EZACOBOL) for COBOL. The IP_MSFILTER structure must include an interface address (input), a multicast address (input), filter mode (output), the number of source addresses in the following array (input and output), and an array of source addresses (output). On input, the number of source addresses contains the number of source addresses that fit in the input array. On output, the number of source addresses contains the total number of source filters in the output array. If the application does not know the size of the source list prior to processing, it can make a reasonable guess (for example, 0). When the process completes, if the number of source addresses contains a larger value, the IOCTL can be repeated with a larger buffer. That is, on output, the number of source addresses is always updated to be the total number of sources in the filter; the array holds as many source addresses as fit, up to the minimum of the array size passed in as the input number.

The size of the IP_MSFILTER value is calculated as follows:
1. Determine the number of source addresses that is expected.
2. Multiply the number of source addresses by the array element (size of IMSF_SrcEntry) to get the size of all array elements.
3. Add the size of all array elements with the size of the IMSF_Header structure to get the total number of bytes needed to accommodate the source address information that is returned.

**SIOCGMSFILTER**

Requests a list of the IPv4 or IPv6 source addresses that comprise the source filter, along with the current mode on a given interface index and a multicast group for a socket. The source filter can include or exclude the set of source address, depending on the filter mode (MCAST_INCLUDE or MCAST_EXCLUDE).

When the SIOCGMSFILTER IOCTL is issued, the REQARG parameter must contain a GROUP_FILTER structure; this structure is defined in SYS1.MACLIB(BPXYIOCC) for assembler, in SEZAINST(CBLOCK) for PL/I, and in SEZAINST(EZACOBOL) for COBOL. The GROUP_FILTER structure must include an interface index (input), a socket address structure of the multicast address (input), filter mode (output), the number of source addresses in the following array (output), and an array of the socket address structure of source addresses (input and output). On input, the number of source addresses contains the number of source addresses that fit in the input array. On output, the number of source addresses contains the total number of source filters in the output array.

If the application does not know the size of the source list before processing, it can make a reasonable guess (for example, 0). When the process completes, if the number of source addresses holds a larger value, the IOCTL can be repeated with a larger buffer. That is, on output, the number of source addresses is always updated to be the total number of sources in the filter, and the array holds as many source addresses as fit, up to the minimum of the array size that is passed in as the input number.

The application calculates the size of the GROUP_FILTER value as follows:
1. Determine the number of source addresses expected.
2. Multiply the number of source addresses by the array element (size of GF_SrcEntry) to get the size of all array elements.
3. Add the size of all array elements to the size of the GF_Header structure to get the total number of bytes needed to accommodate the source addresses information returned.

**SIOCGPARTNERINFO**

Provides an interface for an application to retrieve security information about its partner. When you issue the SIOCGPARTNERINFO IOCTL, the REQARG parameter must contain a PartnerInfo structure. The PartnerInfo structure is defined in members within SEZANMAC; EZBPINF1 defines the PL/I layout, EZBPINFA defines the assembler layout, and EZBPINF3 defines the COBOL layout. For more information about using the SIOCGPARTNERINFO IOCTL, see [z/OS Communications Server: IP Programmer's Guide and Reference](https://publib.boulder.ibm.com/infocenter/iseries/v2r1/topic/com.ibm.zos.doc/zos_comms_ip.html).

**Restriction:** The SIOCGPARTNERINFO IOCTL is not called by the IBM listener.

**Tip:** If the partner end-point is the IBM Listener or a child server and partner security credentials were requested, then only the CICS address space information is returned on the SIOCGPARTNERINFO ioctl invocation.

**SIOCSAPPLDATA**

Enables an application to associate 40 bytes of user-specified application data with a socket endpoint. This application data can be used to identify TCP connections in interfaces such as Netstat, SMF, or network management applications.

**Requirement:** When you issue the SIOCSAPPLDATA IOCTL, ensure that the REQARG parameter contains a SetApplData structure as defined by the EZBYAPPL macro in the SEZANMAC dataset. See the CBLOCK and the EZACOBOL samples for the equivalent SetApplData and SetADcontainer structure definitions for PL/I and COBOL programming environments. See [z/OS Communications Server: IP Programmer’s Guide and Reference](https://publib.boulder.ibm.com/infocenter/iseries/v2r1/topic/com.ibm.zos.doc/zos_comms_ip.html) for more information about programming the SIOCSAPPLDATA IOCTL.

**SetAD_buffer**

User-defined application data that comprises 40 bytes of data that is used to identify the TCP connection with the IP CICS socket API sockets application. The application data can be displayed in the following ways:

- By requesting Netstat reports. The information is displayed conditionally by using the modifier APPLDATA on the ALLC/-a and CONn/-c reports, and unconditionally on the ALL/-A report. See the Netstat ALL/-A report, the Netstat ALLConn/-a report, and the Netstat CONn/-c report information in [z/OS Communications Server: IP System Administrator’s Commands](https://publib.boulder.ibm.com/infocenter/iseries/v2r1/topic/com.ibm.zos.doc/zos_comms_ip.html).
- In the SMF 119 TCP connection termination record. See [TCP connection termination record (subtype 2)](https://publib.boulder.ibm.com/infocenter/iseries/v2r1/topic/com.ibm.zos.doc/zos_comms_ip.html) for more information about the application data written on the SMF 119 record.
- By network management applications. See [Network management interfaces](https://publib.boulder.ibm.com/infocenter/iseries/v2r1/topic/com.ibm.zos.doc/zos_comms_ip.html) for more information about application data.
Applications using this ioctl need to consider the following guidelines:

- The application is responsible for documenting the content, format, and meaning of the ApplData strings it associates with sockets that it owns.
- The application should uniquely identify itself with printable EBCDIC characters at the beginning of the string. Strings beginning with 3-character IBM product identifiers, such as EZA or EZB, are reserved for IBM use. IBM product identifiers begin with a letter in the range A - I.
- Printable EBCDIC characters should be used for the entire string to enable searching with Netstat filters.

Tip: Separate application data elements with a blank for easier reading.

**SIOCSIPMSFILTER**

Sets a list of the IPv4 source addresses that comprise the source filter along with the current mode on a given interface and a multicast group for a socket. The source filter can either include or exclude the set of source address, depending on the filter mode (MCAST_INCLUDE or MCAST_EXCLUDE). A maximum of 64 source addresses can be specified. When the SIOCSIPMSFILTER IOCTL is issued, the REQARG parameter must contain a IP_MSFILTER structure; this structure is defined in SYS1.MACLIB(BPXYIOCC) for assembler, in SEZAINST(CBLOCK) for PL/I and in SEZAINST(EZACOBOL) for COBOL. The IP_MSFILTER structure must include an interface address, a multicast address, filter mode, the number of source addresses in the following array, and an array of source addresses.

The application program calculates the size of the IP_MSFILTER value as follows:

1. Determine the number of source addresses expected.
2. Multiply the number of source addresses by the array element (size of the IMSF_SrcEntry structure) to get the size of all array elements.
3. Add the size of all array elements to the size of IMSF_Header to get the total number of bytes needed to accommodate the source addresses information returned.

**SIOCSMSFILTER**

Sets a list of the IPv4 or IPv6 source addresses that comprise the source filter, along with the current mode on a given interface index and a multicast group for a socket. The source filter can include or exclude the set of source address, depending on the filter mode (INCLUDE or EXCLUDE). A maximum of 64 source addresses can be specified. When the SIOCSMSFILTER IOCTL is issued, the REQARG parameter must contain a GROUP_FILTER structure; this structure is defined in SYS1.MACLIB(BPXYIOCC) for assembler, in SEZAINST(CBLOCK) for PL/I and in SEZAINST(EZACOBOL) for COBOL. The GROUP_FILTER structure must include an interface address, a socket address structure of the multicast address, filter mode, the number of source addresses in the following array, an array of the socket address structure of source addresses.

Calculate the size of the GROUP_FILTER value as follows:

1. Determine the number of source addresses expected.
2. Multiply the number of source addresses by the array element (size of GF_SrcEntry) to get the size of all array elements.
3. Add the size of all array elements to the size of GF_Header to get the total number of bytes needed to accommodate the source addresses information returned.

**SIOSCSPARTNERINFO**

The SIOSCSPARTNERINFO ioctl sets an indicator to retrieve the partner security credentials during connection setup and saves the information, enabling an application to issue a SIOSCSPARTNERINFO ioctl without suspending the application, or at least minimizing the time to retrieve the information. The SIOSCSPARTNERINFO IOCTL must be issued prior to the SIOSCSPARTNERINFO IOCTL. When you issue the SIOSCSPARTNERINFO IOCTL, the REQARG parameter must contain a constant value, PL_REQTYPE_SET_PARTNERDATA. This constant is defined in members within SEZANMAC; EZBPINF1 defines the PL/I layout, EZBPINF1A defines the assembler layout, and EZBPINF1B defines the COBOL layout. For more information about using the SIOSCSPARTNERINFO IOCTL, see z/OS Communications Server: IP Programmer’s Guide and Reference.

**Restriction:** The SIOSCSPARTNERINFO IOCTL is not called by the IBM listener.

**SIOCTTLSCTL**

Controls Application Transparent Transport Layer Security (AT-TLS) for the connection. REQARG and RETARG must contain a TTLS-IOCTL structure. If a partner certificate is requested, the TTLS-IOCTL must include a pointer to additional buffer space and the length of that buffer. Information is returned in the TTLS-IOCTL structure. If a partner certificate is requested and one is available, it is returned in the additional buffer space. The TTLS-IOCTL structure is defined in the control block structures in SEZANMAC. EZBZTL5 defines the PL/I layout, EZBZTLP defines the assembler layout, and EZBZTLSB defines the COBOL layout. For more usage information and samples, see z/OS Communications Server: IP Programmer’s Guide and Reference.

**REQARG and RETARG**

REQARG is used to pass and receive arguments to and from IOCTL, and RETARG receives arguments from IOCTL. The REQARG and RETARG parameters are described in Table 21.

**Table 21. IOCTL call arguments**

<table>
<thead>
<tr>
<th>COMMAND/CODE</th>
<th>SIZE</th>
<th>REQARG</th>
<th>SIZE</th>
<th>RETARG</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIONBIO X’8004A77E’</td>
<td>4</td>
<td>Set socket mode to one of the following: X’00’=blocking; X’01’=nonblocking</td>
<td>0</td>
<td>Not used</td>
</tr>
<tr>
<td>FIONREAD X’4004A77F’</td>
<td>0</td>
<td>Not used</td>
<td>4</td>
<td>Number of characters available for read</td>
</tr>
<tr>
<td>SIOCATMARK X’4004A707’</td>
<td>0</td>
<td>Not used</td>
<td>4</td>
<td>X’00’ = at OOB data; X’01’ = not at OOB data</td>
</tr>
<tr>
<td>SIOCGHOMEIF6 X’C014F608’</td>
<td>20</td>
<td>NetConfHdr</td>
<td>See Figure 141 on page 325</td>
<td></td>
</tr>
</tbody>
</table>
### Table 21. IOCTL call arguments (continued)

<table>
<thead>
<tr>
<th>COMMAND/CODE</th>
<th>SIZE</th>
<th>REQARG</th>
<th>SIZE</th>
<th>RETARG</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIOCGIFADDR X'C020A70D'</td>
<td>32</td>
<td>First 16 bytes is the interface name. Last 16 bytes—not used</td>
<td>32</td>
<td>Network interface address. For assembler, see the IOCN_SADDRIF field in SYS1.MACLIB(BPXYIOCC). For COBOL, see the IFR_ADDR field in SEZAINST(EZACOBOL). For PL/I, see the IFR_ADDR field in SEZAINST(CBLOCK).</td>
</tr>
<tr>
<td>SIOCGIFBRDADDR X'C020A712'</td>
<td>32</td>
<td>First 16 bytes is the interface name. Last 16 bytes—not used</td>
<td>32</td>
<td>Network interface address. For assembler, see the IOCN_SADDRIFBROADCAST field in SYS1.MACLIB(BPXYIOCC). For COBOL, see the IFR-BROADCAST field in SEZAINST(EZACOBOL). For PL/I, see the IFR-BROADCAST field in SEZAINST(CBLOCK).</td>
</tr>
<tr>
<td>SIOCGIFCONF X'C008A714'</td>
<td>8</td>
<td>Size of RETARG</td>
<td></td>
<td>When you call the IOCTL with the SIOCGIFCONF command set, the REQARG parameter should contain the length in bytes of RETARG. Each interface is assigned a 32-byte array element; the REQARG parameter should be set to the number of interfaces multiplied by 32. TCP/IP for z/OS can return up to 100 array elements.</td>
</tr>
<tr>
<td>SIOCGIFDSTADDR X'C020A70F'</td>
<td>32</td>
<td>First 16 bytes is the interface name. Last 16 bytes not used.</td>
<td>32</td>
<td>Destination interface address. For assembler, see the IOCN_SADDRIFDEST field in SYS1.MACLIB(BPXYIOCC). For COBOL, see the IFR-DSTADDR field in SEZAINST(EZACOBOL). For PL/I, see the IFR-DSTADDR field in SEZAINST(CBLOCK).</td>
</tr>
<tr>
<td>SIOCGIFMTU X'C020A726'</td>
<td>32</td>
<td>First 16 bytes is the interface name. Last 16 bytes not used.</td>
<td>32</td>
<td>IPv4 interface MTU (maximum transmission unit). For assembler, see the IOCN_MTUSIZE field in SYS1.MACLIB(BPXYIOCC). For COBOL, see the IFR_MTU field in SEZAINST(EZACOBOL). For PL/I, see the IFR_MTU field in SEZAINST(CBLOCK).</td>
</tr>
<tr>
<td>SIOCGIFNAMEINDEX X'4000F603'</td>
<td>4</td>
<td>First 4 bytes of return the buffer</td>
<td></td>
<td>See <a href="#">Figure 142 on page 327</a></td>
</tr>
</tbody>
</table>
### Table 21. IOCTL call arguments (continued)

<table>
<thead>
<tr>
<th>COMMAND/ CODE</th>
<th>SIZE</th>
<th>REQARG</th>
<th>SIZE</th>
<th>RETARG</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIOCGIPMSFILTER</td>
<td>–</td>
<td>See the IP_MSFILTER structure in macro BPXYIOCC. See note 1.</td>
<td>0</td>
<td>Not used.</td>
</tr>
<tr>
<td>X’C000A724’</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIOCGMSFILTER</td>
<td>–</td>
<td>See the GROUP_FILTER structure in macro BPXYIOCC. See note 2.</td>
<td>0</td>
<td>Not used.</td>
</tr>
<tr>
<td>X’C000F610’</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIOCGPARTNERINFO</td>
<td>–</td>
<td>For the PartnerInfo structure layout, see SEZANMAC(EZBPINFA) for assembler, SEZANMAC(EZBPINF1) for PL/I, and SEZANMAC(EZBPINF2) for COBOL. See note 3.</td>
<td>Not used.</td>
<td></td>
</tr>
<tr>
<td>X’C000F612’</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIOCSAPPLDATA</td>
<td>–</td>
<td>See the SETAPPLDATA structure in macro EZBYAPPL</td>
<td>0</td>
<td>Not used.</td>
</tr>
<tr>
<td>X’8018D90C’</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIOCSIIPMSFILTER</td>
<td>–</td>
<td>See the IP_MSFILTER structure in macro BPXYIOCC. See note 1.</td>
<td>0</td>
<td>Not used.</td>
</tr>
<tr>
<td>X’8000A725’</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIOCSMSFILTER</td>
<td>–</td>
<td>See the GROUP_FILTER structure in macro BPXYIOCC. See note 2.</td>
<td>0</td>
<td>Not used.</td>
</tr>
<tr>
<td>X’8000F611’</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIOCSPARTNERINFO</td>
<td>4</td>
<td>See PL_REQTYPE_SET_PARTNERDATA in SEZANMAC(EZBPINFA) for assembler, SEZANMAC(EZBPINF1) for PL/I, and SEZANMAC(EZBPINF2) for COBOL.</td>
<td>Not used.</td>
<td></td>
</tr>
<tr>
<td>X’8004F613’</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIOCTTLSCTLX’C038D90B’</td>
<td>56</td>
<td>For the IOCTL structure layout, see SEZANMAC(EZBZTLS1) for PL/I, SEZANMAC(EZBZTLSP) for assembler, and SEZANMAC(EZBZTLSB) for COBOL.</td>
<td>56</td>
<td>For the IOCTL structure layout, see SEZANMAC(EZBZTLS1) for PL/I, SEZANMAC(EZBZTLSP) for assembler, and SEZANMAC(EZBZTLSB) for COBOL.</td>
</tr>
</tbody>
</table>

**Note:**
- The size of IP_MSFILTER structure must be equal to or greater than the size of the IMSF_Header structure.
- The size of GROUP_FILTER structure must be equal to or greater than the size of the GF_Header structure.
- The size of the PartnerInfo structure must be equal to or greater than the PI_FIXED_SIZE value.

---

**Parameter values returned to the application for the IOCTL call**

**RETARG**

Returns an array whose size is based on the value in COMMAND. See Table 21 on page 331 for information about REQARG and RETARG.

**ERRNO**

A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.
RETCODE
A fullword binary field that returns one of the following:

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

The COMMAND SIOGIFCONF returns a variable number of network interface configurations. Figure 143 contains an example of a COBOL II routine that can be used to work with such a structure.

Note: This call can be programmed only in languages that support address pointers. Figure 143 shows a COBOL II example for SIOCGIFCONF.

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

Figure 143. COBOL II example for SIOCGIFCONF

LISTEN call

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

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

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

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

The following requirements apply to this call:
### Requirement Description

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

[Figure 144](#) shows an example of LISTEN call instructions.

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

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

*Figure 144. LISTEN call instruction example*

For equivalent PL/I and assembler language declarations, see [Converting parameter descriptions](#) on page 253.

### Parameter values set by the application for the LISTEN call

**SOC-FUNCTION**

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

**S**

A halfword binary number set to the socket descriptor.

**BACKLOG**

A fullword binary number set to the number of communication requests to be queued.

### Parameter values returned to the application for the LISTEN call

**ERRNO**

A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.

**RETCODE**

A fullword binary field that returns one of the following:

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

NTOP converts an IP address from its numeric binary form into a standard text presentation form. On successful completion, NTOP returns the converted IP address in the buffer provided.

The following requirements apply to this call:

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

Figure 145 shows an example of NTOP call instructions.

```
WORKING-STORAGE SECTION.
  01 SOC-NTOP-FUNCTION PIC X(16) VALUE IS 'NTOP'.
  01 S PIC 9(4) BINARY.

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

* IPv6 socket structure.
  01 NAME.
    03 FAMILY PIC 9(4) BINARY.
    03 PORT PIC 9(4) BINARY.
    03 FLOWINFO PIC 9(8) BINARY.
    03 IP-ADDRESS.
      10 FILLER PIC 9(16) BINARY.
      10 FILLER PIC 9(16) BINARY.
    03 SCOPE-ID PIC 9(8) BINARY.
  01 NTOP-FAMILY PIC 9(8) BINARY.
  01 ERRNO PIC 9(8) BINARY.
  01 RETCODE PIC 9(8) BINARY.
  01 PRESENTABLE-ADDRESS PIC X(45).
  01 PRESENTABLE-ADDRESS-LEN PIC 9(4) BINARY.

PROCEDURE DIVISION.
CALL 'EZASOKET' USING SOC-NTOP-FUNCTION NTOP-FAMILY
  IP-ADDRESS
  PRESENTABLE-ADDRESS
  PRESENTABLE-ADDRESS-LEN
  ERRNO RETURN-CODE.
```

Figure 145. NTOP call instruction example
For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 253.

Parameter values set by the application for the NTOP call

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

FAMILY
The addressing family for the IP address being converted. The value of decimal 2 must be specified for AF_INET and 19 for AF_INET6.

IP-ADDRESS
A field containing the numeric binary form of the IPv4 or IPv6 address being converted. For an IPv4 address this field must be a fullword and for an IPv6 address this field must be 16 bytes. The address must be in network byte order.

Parameter values returned to the application for the NTOP call

PRESENTABLE-ADDRESS
A field used to receive the standard text presentation form of the IPv4 or IPv6 address being converted. For IPv4, the address is in dotted-decimal format and for IPv6 the address is in colon-hexadecimal format. The size of the IPv4 address is a maximum of 15 bytes and the size of the converted IPv6 address is a maximum of 45 bytes. Consult the value returned in PRESENTABLE-ADDRESS-LEN for the actual length of the value in PRESENTABLE-ADDRESS.

PRESENTABLE-ADDRESS-LEN
Initially, an input parameter. The address of a halfword binary field (that is used to specify the length of DSTADDR field on input and on a successful return) contains the length of converted IP address.

ERRNO
A fullword binary field. If RETCODE is negative, ERRNO contains an error number. See Appendix B, "Return codes," on page 455 for information about ERRNO return codes.

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

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

PTON call

PTON converts an IP address in its standard text presentation form to its numeric binary form. On successful completion, PTON returns the converted IP address in the buffer provided.

The following requirements apply to this call:

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

Figure 146 on page 339 shows an example of PTON call instructions.
WORKING-STORAGE SECTION.
  01 SOC-NTOP-FUNCTION PIC X(16) VALUE IS 'PTON'.
  01 S PIC 9(4) BINARY.

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

* IPv6 socket structure.
  01 NAME.
    03 FAMILY PIC 9(4) BINARY.
    03 PORT PIC 9(4) BINARY.
    03 FLOWINFO PIC 9(8) BINARY.
    03 IP-ADDRESS.
      10 FILLER PIC 9(16) BINARY.
      10 FILLER PIC 9(16) BINARY.
    03 SCOPE-ID PIC 9(8) BINARY.

  01 AF-INET PIC 9(8) BINARY VALUE 2.
  01 AF-INET6 PIC 9(8) BINARY VALUE 19.

* IPv4 address.
  01 PRESENTABLE-ADDRESS PIC X(45).
  01 PRESENTABLE-ADDRESS-IPV4 REDEFINES PRESENTABLE-ADDRESS.
    05 PRESENTABLE-IPV4-ADDRESS PIC X(15)
      VALUE '192.26.5.19'.
    05 FILLER PIC X(30).
  01 PRESENTABLE-ADDRESS-LEN PIC 9(4) BINARY VALUE 11.

* IPv6 address.
  01 PRESENTABLE-ADDRESS PIC X(45)
    VALUE '12f9:0:0:c30:123:457:9cb:1112'.
  01 PRESENTABLE-ADDRESS-LEN PIC 9(4) BINARY VALUE 29.

* IPv4-mapped IPv6 address.
  01 PRESENTABLE-ADDRESS PIC X(45)
  01 PRESENTABLE-ADDRESS-LEN PIC 9(4) BINARY VALUE 32.

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

  01 PRESENTABLE-ADDRESS PIC X(45).
  01 PRESENTABLE-ADDRESS-LEN PIC 9(4) BINARY.

PROCEDURE DIVISION.

* IPv4 address.
  CALL 'EZASOKET' USING SOC-PTON-FUNCTION AF-INET
    PRESENTABLE-ADDRESS
    PRESENTABLE-ADDRESS-LEN
    IP-ADDRESS
    ERRNO RETURN-CODE.

* IPv6 address.
  CALL 'EZASOKET' USING SOC-PTON-FUNCTION AF-INET6
    PRESENTABLE-ADDRESS
    PRESENTABLE-ADDRESS-LEN
    IP-ADDRESS
    ERRNO RETURN-CODE.

Figure 146. PTON call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 253.
Parameter values set by the application for the PTON call

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

FAMILY
The addressing family for the IP address being converted. The value of decimal 2 must be specified for AF_INET and 19 for AF_INET6.

PRESENTABLE-ADDRESS
A field containing the standard text presentation form of the IPv4 or IPv6 address being converted. For IPv4, the address is in dotted-decimal format and for IPv6 the address is in colon-hexadecimal format.

PRESENTABLE-ADDRESS-LEN
An input parameter. The address of a halfword binary field that must contain the length of IP address to be converted.

Parameter values returned to the application for the PTON call

IP-ADDRESS
A field containing the numeric binary form of the IPv4 or IPv6 address being converted. For an IPv4 address this field must be a fullword and for an IPv6 address this field must be 16 bytes. The address in network byte order.

ERRNO
A fullword binary field. If RETCODE is negative, ERRNO contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.

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

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

READ call

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

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

**Note:** See “EZACIC05 program” on page 405 for a subroutine that translates ASCII input data to EBCDIC.

The following requirements apply to this call:
<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization:</td>
<td>Supervisor state or problem state, any PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN</td>
</tr>
<tr>
<td>Amode:</td>
<td>31-bit or 24-bit</td>
</tr>
<tr>
<td>Note:</td>
<td>See “Addressability mode (Amode) considerations” under “Environmental restrictions and programming requirements for the Callable Socket API” on page 249</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary address space control (ASC) mode</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>Unlocked</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>All parameters must be addressable by the caller and in the primary address space</td>
</tr>
</tbody>
</table>

Figure 147 shows an example of READ call instructions.

WORKING-STORAGE SECTION.
01 SOC-FUNCTION PIC X(16) VALUE IS 'READ'.
01 S PIC 9(4) BINARY.
01 NBYTE PIC 9(8) BINARY.
01 BUF PIC X(length of buffer).
01 ERRNO PIC 9(8) BINARY.
01 RETCODE PIC S9(8) BINARY.

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

Figure 147. READ call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 253.

Parameter values set by the application for the READ call

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

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

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

Parameter values returned to the application for the READ call

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

ERRNO
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.
RETCODE

A fullword binary field that returns one of the following:

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

READV call

The READV function reads data on a socket and stores it in a set of buffers. If a datagram packet is too long to fit in the supplied buffers, datagram sockets discard extra bytes.

The following requirements apply to this call:

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

Figure 148 on page 343 shows an example of READV call instructions.
Parameter values set by the application for the READV call

S  A value or the address of a halfword binary number specifying the descriptor of the socket into which the data is to be read.

IOV  An array of tripleword structures with the number of structures equal to the value in IOVCNT and the format of the structures as follows:

Fullword 1  Pointer to the address of a data buffer, which is filled in on completion of the call.

Fullword 2  Reserved.

Fullword 3  The length of the data buffer referenced in fullword one.

IOVCNT  A fullword binary field specifying the number of data buffers provided for this call.

Parameter values returned to the application for the READV call

ERRNO  A fullword binary field. If RETCODE is negative, this contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A 0 return code indicates that the connection is closed and no data is available.</td>
</tr>
</tbody>
</table>
>0 A positive value indicates the number of bytes copied into the buffer.

-1 Check ERRNO for an error code.

**RECV call**

The RECV call, like READ, receives data on a socket with descriptor S. RECV applies only to connected sockets. If a datagram packet is too long to fit in the supplied buffers, datagram sockets discard extra bytes.

For additional control of the incoming data, RECV can:

- Peek at the incoming message without having it removed from the buffer.
- Read out-of-band data.

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

If data is not available for the socket, and the socket is in blocking mode, RECV blocks the caller until data arrives. If data is not available and the socket is in nonblocking mode, RECV returns a -1 and sets ERRNO to 35 (EWOULDBLOCK). See “FCNTL call” on page 268 or “IOCTL call” on page 322 for a description of how to set nonblocking mode.

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

**Note:** See “EZACIC05 program” on page 405 for a subroutine that translates ASCII input data to EBCDIC.

The following requirements apply to this call:

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

**Figure 149 on page 345** shows an example of RECV call instructions.
Parameter values set by the application for the RECV call

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

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

**FLAGS**
A fullword binary field that should be 4 bytes in length.

<table>
<thead>
<tr>
<th>Literal value</th>
<th>Binary value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO-FLAG</td>
<td>x'00000000'</td>
<td>Read data.</td>
</tr>
<tr>
<td>MSG-OOB</td>
<td>x'00000001'</td>
<td>Receive out-of-band data (stream sockets only). Even if the OOB flag is not set, out-of-band data can be read if the SO-OOBINLINE option is set for the socket.</td>
</tr>
<tr>
<td>MSG-PEEK</td>
<td>x'00000002'</td>
<td>Peek at the data, but do not destroy data. If the peek flag is set, the next receive operation reads the same data.</td>
</tr>
<tr>
<td>MSG-WAITALL</td>
<td>x'00000040'</td>
<td>Requests that the function block until the full amount of data requested can be returned (stream sockets only). The function might return a smaller amount of data if the connection is terminated, an error is pending, or if the SO_RCVTIMEO value is set and the timer expired for the socket.</td>
</tr>
</tbody>
</table>

**NBYTE**
A value or the address of a fullword binary number set to the size of BUF. RECV does not receive more than the number of bytes of data in NBYTE even if more data is available.

Parameter values returned to the application for the RECV call

**BUF**
The input buffer to receive the data.
ERRNO
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.

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

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

RECVFROM call

The RECVFROM call receives data on a socket with descriptor S and stores it in a buffer. The RECVFROM call applies to both connected and unconnected sockets. The IPv4 or IPv6 socket address is returned in the NAME structure. If a datagram packet is too long to fit in the supplied buffers, datagram sockets discard extra bytes.

For datagram protocols, the RECVFROM call returns the source address associated with each incoming datagram. For connection-oriented protocols like TCP, the GETPEERNAME call returns the address associated with the other end of the connection.

On return, NBYTE contains the number of data bytes received.

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

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

If data is not available for the socket, and the socket is in blocking mode, RECVFROM blocks the caller until data arrives. If data is not available and the socket is in nonblocking mode, RECVFROM returns a -1 and sets ERRNO to 35 (EWOULDBLOCK). See “FCNTL call” on page 268 or “IOCTL call” on page 322 for a description of how to set nonblocking mode.

**Note:** See “EZACIC05 program” on page 405 for a subroutine that translates ASCII input data to EBCDIC.

The following requirements apply to this call:

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

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

Figure 150 shows an example of RECVFROM call instructions.

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

* IPv4 Socket Address Structure.
  *
  01 NAME.
    03 FAMILY PIC 9(4) BINARY.
    03 PORT PIC 9(4) BINARY.
    03 IP-ADDRESS PIC 9(8) BINARY.
    03 RESERVED PIC X(8).

* IPv6 Socket Address Structure.
  *
  01 NAME.
    03 FAMILY PIC 9(4) BINARY.
    03 PORT PIC 9(4) BINARY.
    03 FLOW-INFO PIC 9(8) BINARY.
    03 IP-ADDRESS.
    05 FILLER PIC 9(16) BINARY.
    05 FILLER PIC 9(16) BINARY.
    03 SCOPE-ID PIC 9(8) BINARY.

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

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

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 253.

**Parameter values set by the application for the RECVFROM call**

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

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

**FLAGS**
A fullword binary field that should be 4 bytes in length.
<table>
<thead>
<tr>
<th>Literal value</th>
<th>Binary value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO-FLAG</td>
<td>x'00000000'</td>
<td>Read data.</td>
</tr>
<tr>
<td>MSG-OOB</td>
<td>x'00000001'</td>
<td>Receive out-of-band data (stream sockets only). Even if the OOB flag is not set, out-of-band data can be read if the SO-OOBINLINE option is set for the socket.</td>
</tr>
<tr>
<td>MSG-PEEK</td>
<td>x'00000002'</td>
<td>Peek at the data, but do not destroy data. If the peek flag is set, the next RECVFROM call reads the same data.</td>
</tr>
<tr>
<td>MSG-WAITALL</td>
<td>x'00000040'</td>
<td>Requests that the function block until the full amount of data requested can be returned (stream sockets only). The function might return a smaller amount of data if the connection is terminated, an error is pending, or the SO_RCVTIMEO value is set and the timer expired for the socket.</td>
</tr>
</tbody>
</table>

**NBYTE**

A fullword binary number specifying the length of the input buffer.

**Parameter values returned to the application for the RECVFROM call**

**BUF** Defines an input buffer to receive the input data.

**NAME**

An IPv4 socket structure containing the address of the socket that sent the data. The structure is:

**FAMILY**
A halfword binary number specifying the addressing family. The value is a decimal 2, indicating AF_INET.

**PORT**
A halfword binary number specifying the port number of the sending socket.

**IP-ADDRESS**
A fullword binary number specifying the 32-bit IPv4 Internet address of the sending socket.

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

An IPv6 socket structure containing the address of the socket that sent the data. The structure is:

**FAMILY**
A halfword binary number specifying the addressing family. The value is a decimal 19, indicating AF_INET6.

**PORT**
A halfword binary number specifying the port number of the sending socket.

**FLOW-INFO**
A fullword binary field specifying the traffic class and flow label. The value of this field is undefined.

**IP-ADDRESS**
A 16-byte binary number specifying the 128-bit IPv6 Internet address of the sending socket.
SCOPE-ID
A fullword binary field that identifies a set of interfaces as appropriate for the scope of the address carried in the IP-ADDRESS field. For a link scope IP-ADDRESS, SCOPE-ID contains the link index for the IP-ADDRESS. For all other address scopes, SCOPE-ID is undefined.

ERRNO
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.

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

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

RECVMSG call
The RECVMSG call receives messages on a socket with descriptor S and stores them in an array of message headers. If a datagram packet is too long to fit in the supplied buffers, datagram sockets discard extra bytes.

For datagram protocols, the RECVMSG call returns the source address associated with each incoming datagram. For connection-oriented protocols like TCP, the GETPEERNAME call returns the address associated with the other end of the connection.

The following requirements apply to this call:

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

Figure 151 on page 350 shows an example of RECVMSG call instructions.
Figure 151. RECVMSG call instruction example (Part 1 of 2)
Parameter values set by the application for the RECVMSG call

**S**  
A value or the address of a halfword binary number specifying the socket descriptor.

**MSG**  
On input, a pointer to a message header into which the message is received upon completion of the call.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>On input, a pointer to a buffer where the sender address is stored upon completion of the call. The storage being pointed to should be for an IPv4 socket address or an IPv6 socket address. The IPv4 socket address structure contains the following fields:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAMILY</td>
<td>Output parameter. A halfword binary number specifying the IPv4 addressing family. The value for IPv4 socket descriptor (for example, S parameter) is a decimal 2, indicating AF_INET.</td>
</tr>
</tbody>
</table>

| PORT    | Output parameter. A halfword binary number specifying the port number of the sending socket. |

| IP-ADDRESS | Output parameter. A fullword binary number specifying the 32-bit IPv4 Internet address of the sending socket. |

| RESERVED | Output parameter. An 8-byte reserved field. This field is required, but is not used. |

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 253.
The IPv6 socket address structure contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAMILY</td>
<td>Output parameter. A halfword binary field specifying the IPv6 addressing family. The value for IPv6 socket descriptor (for example, S parameter) is a decimal 19, indicating AF_INET6.</td>
</tr>
<tr>
<td>PORT</td>
<td>Output parameter. A halfword binary number specifying the port number of the sending socket.</td>
</tr>
<tr>
<td>FLOW-INFO</td>
<td>Output parameter. A fullword binary field specifying the traffic class and flow label. The value of this field is undefined.</td>
</tr>
<tr>
<td>IP-ADDRESS</td>
<td>Output parameter. A two doubleword, 16-byte binary field specifying the 128-bit IPv6 Internet address, in network byte order, of the sending socket.</td>
</tr>
<tr>
<td>SCOPE-ID</td>
<td>A fullword binary field which identifies a set of interfaces as appropriate for the scope of the address carried in the IP-ADDRESS field. For a link scope IP-ADDRESS, SCOPE-ID contains the link index for the IP-ADDRESS. For all other address scopes, SCOPE-ID is undefined.</td>
</tr>
</tbody>
</table>

NAME-LEN

On input, a pointer to the size of the NAME buffer that is filled in on completion of the call.

IOV

On input, a pointer to an array of tripleword structures with the number of structures equal to the value in IOVCNT and the format of the structures as follows:

**Fullword 1**

A pointer to the address of a data buffer. The data buffer must be in the home address space.

**Fullword 2**

Reserved. This storage is cleared.

**Fullword 3**

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

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

IOVCNT

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

ACCRIGHTS

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

FLAGS
A fullword binary field that should be 4 bytes in length.

<table>
<thead>
<tr>
<th>Literal value</th>
<th>Binary value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO-FLAG</td>
<td>x'00000000'</td>
<td>Read data.</td>
</tr>
<tr>
<td>MSG-OOB</td>
<td>x'00000001'</td>
<td>Receive out-of-band data (stream sockets only). Even if the OOB flag is not set, out-of-band data can be read if the SO-OOBINLINE option is set for the socket.</td>
</tr>
<tr>
<td>MSG-PEEK</td>
<td>x'00000004'</td>
<td>Peek at the data, but do not destroy data. If the peek flag is set, the next receive operation reads the same data.</td>
</tr>
<tr>
<td>MSG-WAITALL</td>
<td>x'00000040'</td>
<td>Requests that the function block until the full amount of data requested can be returned (stream sockets only). The function might return a smaller amount of data if the connection is terminated, an error is pending, or the SO_RCVTIMEO value is set and the timer expired for the socket.</td>
</tr>
</tbody>
</table>

Parameter values returned by the application for the RECVMSG call

ERRNO
A fullword binary field. If RETCODE is negative, this contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.

RETCODE
A fullword binary field with the following values:

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

SELECT call

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

To use the SELECT call as a timer in your program, do one of the following:
- Set the read, write, and exception arrays to zeros.
- Specify MAXSOC <= 0.
The following requirements apply to this call:

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

**Defining which sockets to test for the SELECT call**

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

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

Each socket descriptor is represented by a bit in a bit string. The bit strings are contained in 32-bit fullwords, numbered from right to left. The rightmost bit of the first fullword represents socket descriptor 0 and the leftmost bit of the first fullword represents socket descriptor 31. If your process uses 32 or fewer sockets, the bit string is one fullword. If your process uses 33 sockets, the bit string is two fullwords. The rightmost bit of the second fullword represents socket descriptor 32, and the leftmost bit of the second fullword represents socket descriptor 63. This pattern repeats itself for each subsequent fullword. That is, the leftmost bit of fullword n represents socket 32n-1 and the rightmost bit represents socket 32(n-1).

You define the sockets that you want to test by turning on bits in the string. Although the bits in the fullwords are numbered from right to left, the fullwords are numbered from left to right with the leftmost fullword representing socket descriptor 0–31. For example:

```
First fullword   Second fullword   Third fullword
socket descriptor 31...0  socket descriptor 63...32  socket descriptor 95...64
```

**Note:** To simplify string processing in COBOL, you can use the program EZACIC06 to convert each bit in the string to a character. For more information, see “EZACIC06 program” on page 407.
Calls included for read operations

Read operations include ACCEPT, READ, READV, RECV, RECVFROM, or RECVMSG calls. A socket is ready to be read when data has been received for it, or when a connection request has occurred.

To test whether any of several sockets is ready for reading, set the appropriate bits in RSNDMSK to one before issuing the SELECT call. When the SELECT call returns, the corresponding bits in the RRETMSK indicate sockets ready for reading.

Write operations

A socket is selected for writing (ready to be written) when:
- TCP/IP stacks can accept additional outgoing data.
- The socket is marked nonblocking and a previous CONNECT did not complete immediately. In this case, CONNECT returned an ERRNO with a value of 36 (EINPROGRESS). This socket is selected for write when the CONNECT completes.

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

To test whether any of several sockets is ready for writing, set the WSNDMSK bits representing those sockets to one before issuing the SELECT call. When the SELECT call returns, the corresponding bits in the WRETMSK indicate sockets ready for writing.

Exception operations for the SELECT call

For each socket to be tested, the SELECT call can check for an existing exception condition. Two exception conditions are supported:
- The calling program (concurrent server) has issued a GIVESOCKET command and the target child server has successfully issued the TAKESOCKET call. When this condition is selected, the calling program (concurrent server) should issue CLOSE to dissociate itself from the socket.
- A socket has received out-of-band data. On this condition, a READ returns the out-of-band data ahead of program data.

To test whether any of several sockets have an exception condition, set the ESNDMSK bits representing those sockets to one. When the SELECT call returns, the corresponding bits in the ERETMSK indicate sockets with exception conditions.

MAXSOC parameter for the SELECT call

The SELECT call must test each bit in each string before the call returns any results. For efficiency, the MAXSOC parameter can be used to specify the largest socket descriptor number that needs to be tested for any event type. The SELECT call tests only bits in the range 0 up to the MAXSOC value minus 1. For example, if the MAXSOC parameter is set to 50, the range is 0 - 49.
TIMEOUT parameter for the SELECT call

If the time specified in the TIMEOUT parameter elapses before any event is detected, the SELECT call returns and RETCODE is set to 0.

Figure 153 shows an example of SELECT call instructions.

WORKING-STORAGE SECTION.
  01 SOC-FUNCTION PIC X(16) VALUE IS 'SELECT'.
  01 MAXSOC PIC 9(8) BINARY.
  01 TIMEOUT.
    03 TIMEOUT-SECONDS PIC 9(8) BINARY.
    03 TIMEOUT-MICROSEC PIC 9(8) BINARY.
  01 RSNDMSK PIC X(*)..
  01 WSNDMSK PIC X(*)..
  01 ESNDMSK PIC X(*)..
  01 RRTEMK PIC X(*)..
  01 WRTEMK PIC X(*)..
  01 ERTEMK PIC X(*)..
  01 ERRNO PIC 9(8) BINARY.
  01 RETCODE PIC S9(8) BINARY.

PROCEDURE DIVISION.
  CALL 'EZASOKET' USING SOC-FUNCTION MAXSOC TIMEOUT
    RSNDMSK WSNDMSK ESNDMSK
    RRTEMK WRTEMK ERTEMK
    ERRNO RETCODE.

* The bit mask lengths can be determined from the expression:
  ((maximum socket number +32)/32 (drop the remainder))*4

Figure 153. SELECT call instruction example

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

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 253.

Parameter values set by the application for the SELECT call

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

MAXSOC
  A fullword binary field that specifies the largest socket descriptor number being checked. The SELECT call tests only bits in the range 0 through the MAXSOC value minus 1. For example, if the MAXSOC value is set to 50, the bits that are tested are in the range 0 - 49.

  Guideline: For the INITAPI call, the MAXSOC field is a halfword binary field. Therefore, do not reuse this field for the SELECT and INITAPI calls.

TIMEOUT
  If TIMEOUT is a positive value, it specifies the maximum interval to wait for the selection to complete. If TIMEOUT-SECONDS is a negative value, the SELECT call blocks until a socket becomes ready or an ECB in a list is posted. To poll the sockets and return immediately, specify the TIMEOUT value to be 0.
TIMEOUT is specified in the two-word TIMEOUT as follows:

- TIMEOUT-SECONDS, word one of the TIMEOUT field, is the seconds component of the timeout value.
- TIMEOUT-MICROSEC, word two of the TIMEOUT field, is the microseconds component of the timeout value (0—999999).

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

RSNDMSK
A bit string sent to request read event status.

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

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

WSNDMSK
A bit string sent to request write event status.

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

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

ESNDMSK
A bit string sent to request exception event status.

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

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

Parameter values returned to the application for the SELECT call

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

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

ERETMSK
A bit string returned with the status of exception events. The length of the string should be equal to the maximum number of sockets to be checked. For each socket that has an exception status, the corresponding bit is set to 1; bits that represent sockets that do not have exception status are set to 0.
ERRNO
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;0</td>
<td>Indicates the sum of all ready sockets in the three masks</td>
</tr>
<tr>
<td>0</td>
<td>Indicates that the SELECT time limit has expired</td>
</tr>
<tr>
<td>-1</td>
<td>Check ERRNO for an error code</td>
</tr>
</tbody>
</table>

SELECTEX call

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 exception arrays to zeros
- Specify MAXSOC ≤ 0

The following requirements apply to this call:

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

Figure 154 on page 360 shows an example of SELECTEX call instructions.
If an application intends to pass a single ECB on the SELECTEX call, then the corresponding working storage definitions and CALL instruction should be coded as follows:

```
WORKING-STORAGE SECTION.
  01 SOC-FUNCTION PIC X(16) VALUE IS 'SELECTEX'.
  01 MAXSOC PIC 9(B) BINARY.
  01 TIMEOUT.
    03 TIMEOUT-SECONDS PIC 9(8) BINARY.
    03 TIMEOUT-MINUTES PIC 9(8) BINARY.
  01 RSNDMSK PIC X(*).
  01 WSNDMSK PIC X(*).
  01 ESNDMSK PIC X(*).
  01 RRETMSK PIC X(*).
  01 WRETMSK PIC X(*).
  01 ERETMMSK PIC X(*).
  01 SELECB PIC X(4).
  01 ERRNO PIC 9(B) BINARY.
  01 RETCODE PIC S9(B) BINARY.
Where * is the size of the select mask

PROCEDURE DIVISION.
  CALL 'EZASOKET' USING SOC-FUNCTION MAXSOC TIMEOUT
    RSNDMSK WSNDMSK ESNDMSK
    RRETMSK WRETMSK ERETMMSK
    SELECB ERRNO RETCODE.

Where * is the size of the select mask.

PROCEDURE DIVISION.
  CALL 'EZASOKET' USING SOC-FUNCTION MAXSOC TIMEOUT
    RSNDMSK WSNDMSK ESNDMSK
    RRETMSK WRETMSK ERETMMSK
    SELECB ERRNO RETCODE.
```

However, if the application intends to pass the address of an ECB list on the SELECTEX call, then the application must set the high-order bit in the ECB list address and pass that address using the BY VALUE option as in the following example. The remaining parameters must be reset to the default value by specifying BY REFERENCE before the ERRNO value:

```
WORKING-STORAGE SECTION.
  01 SOC-FUNCTION PIC X(16) VALUE IS 'SELECTEX'.
  01 MAXSOC PIC 9(B) BINARY.
  01 TIMEOUT.
    03 TIMEOUT-SECONDS PIC 9(8) BINARY.
    03 TIMEOUT-MINUTES PIC 9(8) BINARY.
  01 RSNDMSK PIC X(*).
  01 WSNDMSK PIC X(*).
  01 ESNDMSK PIC X(*).
  01 RRETMSK PIC X(*).
  01 WRETMSK PIC X(*).
  01 ERETMMSK PIC X(*).
  01 ECBLIST-PTR USAGE IS POINTER.
  01 ERRNO PIC 9(B) BINARY.
  01 RETCODE PIC S9(B) BINARY.
An asterisk (*) represents the size of the select mask.

PROCEDURE DIVISION.
  CALL 'EZASOKET' USING SOC-FUNCTION MAXSOC TIMEOUT
    RSNDMSK WSNDMSK ESNDMSK
    RRETMSK WRETMSK ERETMMSK
    BY VALUE ECBLIST-PTR
    BY REFERENCE ERRNO RETCODE.
```

Figure 154. SELECTEX call instruction example
Defining which sockets to test for the SELECTEX call

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

- When a socket is ready to read, one of the following has occurred:
  - A buffer for the specified sockets contains input data. If input data is available for a given socket, a read operation on that socket does not block.
  - A connection has been requested on that socket.

- When a socket is ready to write, TCP/IP stacks can accommodate additional output data. If TCP/IP stacks can accept additional output for a given socket, a write operation on that socket does not block.
- When an exception condition has occurred on a specified socket it is an indication that a TAKESOCKET has occurred for that socket.
- A timeout occurs on the SELECTEX call. The timeout period can be specified when the SELECTEX call is issued.
- The ECB (or one of the ECBS in the ECB list) passed on the SELECTEX call has been posted.

Each socket descriptor is represented by a bit in a bit string. The bit strings are contained in 32-bit fullwords, numbered from right to left. The rightmost bit of the first fullword represents socket descriptor 0 and the leftmost bit of the first fullword represents socket descriptor 31. If your process uses 32 or fewer sockets, the bit string is one fullword. If your process uses 33 sockets, the bit string is two fullwords. The rightmost bit of the second fullword represents socket descriptor 32, and the leftmost bit of the second fullword represents socket descriptor 63. This pattern repeats itself for each subsequent fullword. That is, the leftmost bit of fullword n represents socket 32n-1 and the rightmost bit represents socket 32(n-1).

You define the sockets that you want to test by turning on bits in the string. Although the bits in the fullwords are numbered from right to left, the fullwords are numbered from left to right with the leftmost fullword representing socket descriptor 0-31. For example:

<table>
<thead>
<tr>
<th>First fullword</th>
<th>Second fullword</th>
<th>Third fullword</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket descriptor 31...0</td>
<td>socket descriptor 63...32</td>
<td>socket descriptor 95...64</td>
</tr>
</tbody>
</table>

Note: To simplify string processing in COBOL, you can use the program EZACIC06 to convert each bit in the string to a character. For more information, see the EZACIC06 topic.

Read operations for the SELECTEX call

Read operations include ACCEPT, READ, READV, RECV, RECVFROM, or RECVMSG calls. A socket is ready to be read when data has been received for it, or when a connection request has occurred.

To test whether any of several sockets is ready for reading, set the appropriate bits in RSNDMSK to one before issuing the SELECTEX call. When the SELECTEX call returns, the corresponding bits in the RRETMSK indicate sockets ready for reading.

Write operations for the SELECTEX call

A socket is selected for writing (ready to be written) when:
- TCP/IP stacks can accept additional outgoing data.
• The socket is marked nonblocking and a previous CONNECT did not complete immediately. In this case, CONNECT returned an ERRNO with a value of 36 (EINPROGRESS). This socket is selected for write when the CONNECT completes.

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

To test whether any of several sockets is ready for writing, set the WSNDMSK bits representing those sockets to one before issuing the SELECTEX call. When the SELECTEX call returns, the corresponding bits in the WRETMSK indicate sockets ready for writing.

**Exception operations for the SELECTEX call**

For each socket to be tested, the SELECTEX call can check for an existing exception condition. Two exception conditions are supported:

- The calling program (concurrent server) has issued a GIVESOCKET command and the target child server has successfully issued the TAKESOCKET call. When this condition is selected, the calling program (concurrent server) should issue CLOSE to dissociate itself from the socket.
- A socket has received out-of-band data. On this condition, a READ returns the out-of-band data ahead of program data.

To test whether any of several sockets have an exception condition, set the ESNDMSK bits representing those sockets to one. When the SELECTEX call returns, the corresponding bits in the ERETMSK indicate sockets with exception conditions.

**MAXSOC parameter for the SELECTEX call**

The SELECTEX call must test each bit in each string before the returns any results. For efficiency, the MAXSOC parameter can be used to specify the largest socket descriptor number that needs to be tested for any event type. The SELECTEX call tests only bits in the range 0 up to the MAXSOC value minus 1. For example, if MAXSOC is set to 50, the range is 0 - 49.

**TIMEOUT parameter for the SELECTEX call**

If the time specified in the TIMEOUT parameter elapses before any event is detected, the SELECTEX call returns and RETCODE is set to 0.

**Parameter values set by the application for the SELECTEX call**

**MAXSOC**

A fullword binary field that specifies the largest socket descriptor number being checked. The SELECT call tests bits in the range 0 through the MAXSOC value minus 1. For example, if the MAXSOC value is set to 50, the bits that would be tested are in the range 0 - 49.

**TIMEOUT**

If TIMEOUT is a positive value, it specifies a maximum interval to wait for the selection to complete. If TIMEOUT-SECONDS is a negative value, the
SELECT call blocks until a socket becomes ready. To poll the sockets and return immediately, set TIMEOUT to be zeros.

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

- TIMEOUT-SECONDS, word one of the TIMEOUT field, is the seconds component of the timeout value.
- TIMEOUT-MICROSEC, word two of the TIMEOUT field, is the microseconds component of the timeout value (0—999999).

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

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

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

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

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

If an ECB list is specified, you must set the high-order bit of the last entry in the ECB list to one to signify it is the last entry, and you must add the LIST keyword. The ECBs must reside in the caller primary address space.

If the application intends to pass the address of an ECB list on the SELECTEX call, then the application must set the high-order bit in the ECB list address and pass that address using the “BY VALUE” option as documented in the following example. The remaining parameters must be set back to the default by specifying "BY REFERENCE" before ERRNO:

```
WORKING-STORAGE SECTION.
  01 SOC-FUNCTION PIC X(16) VALUE IS 'SELECTEX'.
  01 MAXSOC PIC 9(8) BINARY.
  01 TIMEOUT.
  03 TIMEOUT-SECONDS PIC 9(8) BINARY.
  03 TIMEOUT-MINUTES PIC 9(8) BINARY.
  01 RSNDMSK PIC X(*)
  01 WSNDMSK PIC X(*)
  01 ESNDMSK PIC X(*)
  01 RRETMSK PIC X(*)
  01 WRETMSK PIC X(*)
  01 ERETMSK PIC X(*)
  01 ECBLIST-PTR USAGE IS POINTER.
  01 ERRNO PIC 9(8) BINARY.
  01 RETCODE PIC S9(8) BINARY.
```

Where * is the size of the select mask

PROCEDURE DIVISION.
  CALL 'EZASOKET' USING SOC-FUNCTION MAXSOC TIMEOUT
Note:
- The maximum number of ECBs that can be specified in a list is 63
- Perform an MVS POST (not a CICS POST) to post the ECB.

**Parameter values returned by the application for the SELECTEX call**

**ERRNO**
A fullword binary field; if RETCODE is negative, this contains an error number. See [Appendix B, “Return codes,” on page 455](#) for information about ERRNO return codes.

**RETCODE**
A fullword binary field

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;0</td>
<td>The number of ready sockets.</td>
</tr>
<tr>
<td>0</td>
<td>Either the SELECTEX time limit has expired (ECB value is 0) or one of the caller’s ECBs has been posted (ECB value is nonzero and the caller’s descriptor sets are set to 0). The caller must initialize the ECB values to 0 before issuing the SELECTEX call.</td>
</tr>
<tr>
<td>-1</td>
<td>Error; check ERRNO.</td>
</tr>
</tbody>
</table>

**RRETMSK**
The bit-mask array returned by the SELECT if RSNDMSK is specified. The length of this bit-mask array is dependent on the value in MAXSOC.

**WRETMSK**
The bit-mask array returned by the SELECT if WSNDMSK is specified. The length of this bit-mask array is dependent on the value in MAXSOC.

**ERETMSK**
The bit-mask array returned by the SELECT if ESNDMSK is specified. The length of this bit-mask array is dependent on the value in MAXSOC.

**Note:** See EZACIC06 for information about bits mask conversion.

**Note:** See [Appendix E, “Sample programs,” on page 571](#) for sample programs.

**SEND call**
The SEND call sends data on a specified connected socket.

The FLAGS field allows you to:
- Send out-of-band data, for example, interrupts, aborts, and data marked urgent. Only stream sockets created in the AF_INET or AF_INET6 address family support out-of-band data.
- Suppress use of local routing tables. This implies that the caller takes control of routing and writing network software.

For datagram sockets, SEND transmits the entire datagram if it fits into the receiving buffer. Extra data is discarded.
For stream sockets, data is processed as streams of information with no boundaries separating the data. For example, if a program is required to send 1000 bytes, each call to this function can send any number of bytes, up to the entire 1000 bytes, with the number of bytes sent returned in RETCODE. Therefore, programs using stream sockets should place this call in a loop, reissuing the call until all data has been sent.

**Note:** See “EZACIC04 program” on page 404 for a subroutine that translates EBCDIC input data to ASCII.

The following requirements apply to this call:

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

[Figure 155 on page 366](#) shows an example of SEND call instructions.
Parameter values set by the application for the SEND call

SOC-FUNCTION

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

S

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

FLAGS

The binary field should be 4 bytes hexadecimal bytes in length.

<table>
<thead>
<tr>
<th>Literal value</th>
<th>Binary value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO-FLAG</td>
<td>x'00000000'</td>
<td>No flag is set. The command behaves like a WRITE call.</td>
</tr>
<tr>
<td>MSG-OOB</td>
<td>x'00000001'</td>
<td>Send out-of-band data (stream sockets only). Even if the OOB flag is not set, out-of-band data can be read if the SO-OOBINLINE option is set for the socket.</td>
</tr>
<tr>
<td>MSG-DONTROUTE</td>
<td>x'00000004'</td>
<td>Do not route. Routing is provided by the calling program.</td>
</tr>
</tbody>
</table>

NBYTE

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

BUF

The buffer containing the data to be transmitted. BUF should be the size specified in NBYTE.

Parameter values returned to the application for the SEND call

ERRNO

A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.

RETCODE

A fullword binary field that returns one of the following:

| Value | Description |
|-------|-------------|-------------|

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 253.
A successful call. The value is set to the number of bytes transmitted.

-1 Check ERRNO for an error code

**SENDMSG call**

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

The following requirements apply to this call:

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

[Figure 156 on page 368](#) shows an example of SENDMSG call instructions.
WORKING-STORAGE SECTION.
01 SOC-FUNCTION PIC X(16) VALUE IS 'SENDMSG'.
01 S PIC 9(4) BINARY.
01 MSG.
  03 NAME USAGE IS POINTER.
  03 NAME-LEN USAGE IS POINTER.
  03 IOV USAGE IS POINTER.
  03 IOVCNT USAGE IS POINTER.
  03 ACCRIGTHS USAGE IS POINTER.
  03 ACCRLEN USAGE IS POINTER.
01 FLAGS PIC 9(8) BINARY.
  01 NO-FLAG PIC 9(8) BINARY VALUE IS 0.
  01 OOB PIC 9(8) BINARY VALUE IS 1.
  01 DONTROUTE PIC 9(8) BINARY VALUE IS 4.
01 ERRNO PIC 9(8) BINARY.
01 RETCODE PIC S9(8) BINARY.
01 SENDMSG-IPV4ADDR PIC 9(8) BINARY.
01 SENDMSG-IPV6ADDR.
  03 FILLER PIC 9(16) BINARY.
  03 FILLER PIC 9(16) BINARY.

LINKAGE SECTION.
01 L1
  03 SENDMSG-IOVECTOR.
    05 IOV1A USAGE IS POINTER.
    05 IOV1AL PIC 9(8) COMP.
    05 IOV1L PIC 9(8) COMP.
    05 IOV2A USAGE IS POINTER.
    05 IOV2AL PIC 9(8) COMP.
    05 IOV2L PIC 9(8) COMP.
    05 IOV3A USAGE IS POINTER.
    05 IOV3AL PIC 9(8) COMP.
    05 IOV3L PIC 9(8) COMP.

* IPv4 Socket Address Structure.

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

* IPv6 Socket Address Structure.

  03 SENDMSG-NAME.
    05 FAMILY PIC 9(4) BINARY.
    05 PORT PIC 9(4) BINARY.
    05 FLOW-INFO PIC 9(8) BINARY.
    05 IP-ADDRESS.
      10 FILLER PIC 9(16) BINARY.
      10 FILLER PIC 9(16) BINARY.
    05 SCOPE-ID PIC 9(8) BINARY.
    03 SENDMSG-BUFFER1 PIC X(16).
    03 SENDMSG-BUFFER2 PIC X(16).
    03 SENDMSG-BUFFER3 PIC X(16).
    03 SENDMSG-BUFNO PIC 9(8) COMP.

PROCEDURE DIVISION USING L1.

* For IPv6
MOVE 19 TO FAMILY.
MOVE 1234 TO PORT.
MOVE 0 TO FLOW-INFO.
MOVE SENDMSG-IPV6ADDR TO IP-ADDRESS.
MOVE 0 TO SCOPE-ID.

* For IPv4

MOVE 2 TO FAMILY.
MOVE 1234 TO PORT.
MOVE SENDMSG-IPV4ADDR TO IP-ADDRESS.

SET NAME TO ADDRESS OF SENDMSG-NAME.
SET IOV TO ADDRESS OF SENDMSG-IOVECTOR.
MOVE LENGTH OF SENDMSG-NAME TO NAME-LEN.
SET IOV1A TO ADDRESS OF SENDMSG-BUFFER1.
MOVE 0 TO IOV1AL.
MOVE LENGTH OF SENDMSG-BUFFER1 TO IOV1L.
SET IOV2A TO ADDRESS OF SENDMSG-BUFFER2.
MOVE 0 TO IOV2AL.
MOVE LENGTH OF SENDMSG-BUFFER2 TO IOV2L.
SET IOV3A TO ADDRESS OF SENDMSG-BUFFER3.
MOVE 0 TO IOV3AL.
MOVE LENGTH OF SENDMSG-BUFFER3 TO IOV3L.
SET ACCRIGHTS TO NULLS.
SET ACCRLEN TO NULLS.
MOVE 0 TO FLAGS.
MOVE "MESSAGE TEXT 1" TO SENDMSG-BUFFER1.
MOVE "MESSAGE TEXT 2" TO SENDMSG-BUFFER2.
MOVE "MESSAGE TEXT 3" TO SENDMSG-BUFFER3.

CALL 'EZASOKET' USING SOC-FUNCTION MSG FLAGS ERRNO RETCODE.

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 253.

Parameter values set by the application for the SENDMSG call

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>A value or the address of a halfword binary number specifying the socket descriptor.</td>
</tr>
<tr>
<td>MSG</td>
<td>A pointer to an array of message headers from which messages are sent.</td>
</tr>
</tbody>
</table>

**Field Description**

**NAME**

On input, a pointer to a buffer where the sender's address is stored upon completion of the call. The storage being pointed to should be for an IPv4 socket address or an IPv6 socket address.

The IPv4 socket address structure contains the following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAMILY</td>
<td>A halfword binary number specifying the IPv4 addressing family. The value for IPv4 socket descriptor (that is, S parameter) is a decimal 2, indicating AF_INET.</td>
</tr>
<tr>
<td>PORT</td>
<td>A halfword binary number specifying the port number of the sending socket.</td>
</tr>
<tr>
<td>IP-ADDRESS</td>
<td>A fullword binary number specifying the 32-bit IPv4 Internet address of the sending socket.</td>
</tr>
</tbody>
</table>
RESERVED
An 8-byte reserved field. This field is required, but is not used.

The IPv6 socket address structure contains the following fields:

**Field** | **Description**
---|---
FAMILY | A halfword binary field specifying the IPv6 addressing family. The value for IPv6 socket descriptor (for example, S parameter) is a decimal 19, indicating AF_INET6.
PORT | A halfword binary number specifying the port number of the sending socket.
FLOW-INFO | A fullword binary field specifying the traffic class and flow label. This field must be set to zero.
IP-ADDRESS | A two doubleword, 16-byte binary field specifying the 128-bit IPv6 Internet address, in network byte order, of the sending socket.
SCOPE-ID | A fullword binary field which identifies a set of interfaces as appropriate for the scope of the address carried in the IP-ADDRESS field. A value of zero indicates the SCOPE-ID field does not identify the set of interfaces to be used, and can be specified for any address types and scopes. For a link scope IP-ADDRESS, SCOPE-ID can specify a link index which identifies a set of interfaces. For all other address scopes, SCOPE-ID must be set to zero.
NAME-LEN | On input, a pointer to the size of the address buffer that is filled in on completion of the call.
IOV | On input, a pointer to an array of three fullword structures with the number of structures equal to the value in IOVCNT and the format of the structures as follows:
Fullword 1 | A pointer to the address of a data buffer
Fullword 2 | Reserved
Fullword 3 | A pointer to the length of the data buffer referenced in Fullword 1.
In COBOL, the IOV structure must be defined separately in the Linkage portion, as shown in the example.
IOVCNT | On input, a pointer to a fullword binary field specifying the number of data buffers provided for this call.
ACCRIGHTS
On input, a pointer to the access rights received. This field is ignored.

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

FLAGS
The binary field should be 4 bytes hexadecimal bytes in length.

<table>
<thead>
<tr>
<th>Literal value</th>
<th>Binary value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO-FLAG</td>
<td>x'00000000'</td>
<td>No flag is set. The command behaves like a WRITE call.</td>
</tr>
<tr>
<td>MSG-OOB</td>
<td>x'00000001'</td>
<td>Send out-of-band data (stream sockets only). Even if the OOB flag is not set, out-of-band data can be read if the SO-OOBINLINE option is set for the socket.</td>
</tr>
<tr>
<td>MSG-DONTROUTE</td>
<td>x'00000004'</td>
<td>Do not route. Routing is provided by the calling program.</td>
</tr>
</tbody>
</table>

Parameter values returned by the application for the SENDMSG call

ERRNO
A fullword binary field. If RETCODE is negative, this contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.

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

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

SENDTO call

SENDTO is similar to SEND, except that it includes the destination address parameter. The destination address allows you to use the SENDTO call to send datagrams on a UDP socket, regardless of whether the socket is connected.

The FLAGS parameter allows you to:
- Send out-of-band data such as interrupts, aborts, and data marked as urgent.
- Suppress use of local routing tables. This implies that the caller takes control of routing, which requires writing network software.

For datagram sockets SENDTO transmits the entire datagram if it fits into the receiving buffer. Extra data is discarded.

For stream sockets, data is processed as streams of information with no boundaries separating the data. For example, if a program is required to send 1000 bytes, each call to this function can send any number of bytes, up to the entire 1000 bytes,
with the number of bytes sent returned in RETCODE. Therefore, programs using
stream sockets should place SENDTO in a loop that repeats the call until all data
has been sent.

**Note:** See “EZACIC04 program” on page 404 for a subroutine that translates
EBCDIC input data to ASCII.

The following requirements apply to this call:

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

Figure 157 on page 373 shows an example of SENDTO call instructions.
WORKING-STORAGE SECTION.
   01 SOC-FUNCTION PIC X(16) VALUE IS 'SENDTO'.
   01 S PIC 9(4) BINARY.
   01 FLAGS. PIC 9(8) BINARY.
   01 NO-FLAG PIC 9(8) BINARY VALUE IS 0.
   01 OOB PIC 9(8) BINARY VALUE IS 1.
   01 DONT-ROUTE PIC 9(8) BINARY VALUE IS 4.
   01 NBYTE PIC 9(8) BINARY.
   01 BUF PIC X(length of buffer).
*
* IPv4 Socket Address Structure.
*
   01 NAME.
      03 FAMILY PIC 9(4) BINARY.
      03 PORT PIC 9(4) BINARY.
      03 IP-ADDRESS PIC 9(8) BINARY.
      03 RESERVED PIC X(8).
*
* IPv6 Socket Address Structure.
*
   01 NAME.
      03 FAMILY PIC 9(4) BINARY.
      03 PORT PIC 9(4) BINARY.
      03 FLOW-INFO PIC 9(8) BINARY.
      03 IP-ADDRESS.
         05 FILLER PIC 9(16) BINARY.
         05 FILLER PIC 9(16) BINARY.
      03 SCOPE-ID PIC 9(8) BINARY.
 01 ERRNO PIC 9(8) BINARY.
 01 RETCODE PIC S9(8) BINARY.
PROCEDURE DIVISION.
   CALL 'EZASOKET' USING SOC-FUNCTION S FLAGS NBYTE
                     BUF NAME ERRNO RETCODE.

Figure 157. SENDTO call instruction example

For equivalent PL/I and assembler language declarations, see "Converting
parameter descriptions" on page 253.

Parameter values set by the application for the SENDTO call

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

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

FLAGS
   A fullword binary field that should be 4 bytes in length.

<table>
<thead>
<tr>
<th>Literal value</th>
<th>Binary value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO-FLAG</td>
<td>x'00000000'</td>
<td>No flag is set. The command behaves like a WRITE call.</td>
</tr>
<tr>
<td>MSG-OOB</td>
<td>x'00000001'</td>
<td>Send out-of-band data (stream sockets only). Even if the OOB flag is not set, out-of-band data can be read if the SO-OOBINLINE option is set for the socket.</td>
</tr>
<tr>
<td>MSG-DONTROUTE</td>
<td>x'00000004'</td>
<td>Do not route. Routing is provided by the calling program.</td>
</tr>
</tbody>
</table>
NBYTE
A fullword binary number set to the number of bytes to transmit.

BUF
Specifies the buffer containing the data to be transmitted. BUF should be the size specified in NBYTE.

NAME
Specifies the IPv4 socket address structure as follows:

FAMILY
A halfword binary field containing the addressing family. For TCP/IP the value must be a decimal 2, indicating AF_INET.

PORT
A halfword binary field containing the port number bound to the socket.

IP-ADDRESS
A fullword binary field containing the socket’s 32-bit IPv4 Internet address.

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

Specifies the IPv6 socket address structure as follows:

FAMILY
A halfword binary field containing the addressing family. For TCP/IP stacks the value must be a decimal 19, indicating AF_INET6.

PORT
A halfword binary field containing the port number bound to the socket.

FLOW-INFO
A fullword binary field specifying the traffic class and flow label. This field must be set to zero.

IP-ADDRESS
A 16-byte binary field containing the socket’s 128-bit IPv6 Internet address.

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

Parameter values returned to the application for the SENDTO call

ERRNO
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.
RETCODE
A fullword binary field that returns one of the following:

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

SETSOCKOPT call

The SETSOCKOPT call sets the options associated with a socket.

The OPTVAL and OPTLEN parameters are used to pass data used by the particular set command. The OPTVAL parameter points to a buffer containing the data needed by the set command. The OPTLEN parameter must be set to the size of the data pointed to by OPTVAL.

The following requirements apply to this call:

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

Figure 158 on page 376 shows an example of SETSOCKOPT call instructions.
Parameter values set by the application for the SETSOCKOPT call

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

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

OPTNAME
Input parameter. See "Parameter values returned to the application for the GETSOCKOPT call" on page 300 for a list of the options and their unique requirements. See Appendix C, "GETSOCKOPT/SETSOCKOPT command values," on page 471 for the numeric values of OPTNAME.

Note: COBOL programs cannot contain field names with the underscore character. Fields representing the option name should contain dashes instead.

OPTVAL
Input parameter. Contains data that further defines the option specified in OPTNAME. See "Parameter values returned to the application for the GETSOCKOPT call" on page 300 for a list of the options and their unique requirements.

OPTLEN
Input parameter. A fullword binary field specifying the length of the data specified in OPTVAL. See "Parameter values returned to the application for the GETSOCKOPT call" on page 300 for how to determine the value of OPTLEN.

Parameter values returned to the application for the SETSOCKOPT call

ERRNO
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B, "Return codes," on page 455 for information about ERRNO return codes.

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>&lt;0</td>
<td>Error</td>
</tr>
</tbody>
</table>
Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IP_ADD_MEMBERSHIP</strong></td>
<td>Contains the IP_MREQ structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 interface address. See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ. See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ.</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>IP_ADD_SOURCE_MEMBERSHIP</strong></td>
<td>Contains the IP_MREQ_SOURCE structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ_SOURCE structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 source address and a 4-byte IPv4 interface address. See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE. See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ-SOURCE.</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>IP_BLOCK_SOURCE</strong></td>
<td>Contains the IP_MREQ_SOURCE structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ_SOURCE structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 source address and a 4-byte IPv4 interface address. See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE. See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ-SOURCE.</td>
<td>N/A</td>
</tr>
</tbody>
</table>

0 Successful call.

-1 Check ERRNO for an error code.
<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP_DROP_MEMBERSHIP</td>
<td>Contains the IP_MREQ structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 interface address. See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ. See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ.</td>
<td>N/A</td>
</tr>
<tr>
<td>IP_DROP_SOURCE_MEMBERSHIP</td>
<td>Contains the IP_MREQ_SOURCE structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ_SOURCE structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 source address and a 4-byte IPv4 interface address. See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE. See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ-SOURCE.</td>
<td>N/A</td>
</tr>
<tr>
<td>IP_MULTICAST_IF</td>
<td>A 4-byte binary field containing an IPv4 interface address.</td>
<td>A 4-byte binary field containing an IPv4 interface address.</td>
</tr>
<tr>
<td></td>
<td>Note: Multicast datagrams can be transmitted only on one interface at a time.</td>
<td></td>
</tr>
<tr>
<td>IP_MULTICAST_LOOP</td>
<td>A 1-byte binary field. To enable, set to 1. To disable, set to 0.</td>
<td>A 1-byte binary field. If enabled, will contain a 1. If disabled, will contain a 0.</td>
</tr>
<tr>
<td></td>
<td>This is an IPv4-only socket option.</td>
<td></td>
</tr>
</tbody>
</table>

This is an IPv4-only socket option.
<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IP_MULTICAST_TTL</strong></td>
<td>Use this option to set or obtain the IP time-to-live of outgoing multicast datagrams. The default value is ‘01’x meaning that multicast is available only to the local subnet.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This is an IPv4-only socket option.</td>
<td></td>
</tr>
<tr>
<td><strong>IP_UNBLOCK_SOURCE</strong></td>
<td>Use this option to enable an application to unblock a previously blocked source for a given IPv4 multicast group. You must specify an interface and a source address with this option.</td>
<td>Contains the IP_MREQ_SOURCE structure as defined in SYS1.MACLIB(BPXYSOCK). The IP_MREQ_SOURCE structure contains a 4-byte IPv4 multicast address followed by a 4-byte IPv4 source address and a 4-byte IPv4 interface address. See SEZAINST(CBLOCK) for the PL/I example of IP_MREQ_SOURCE. See SEZAINST(EZACOBOL) for the COBOL example of IP-MREQ-SOURCE.</td>
</tr>
</tbody>
</table>
Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPV6_ADDR_PREFERENCES</td>
<td>Contains the 4-byte flags field IPV6_ADDR_PREFERENCES_FLAGS that is defined in SYS1.MACLIB(BPXYSOCK) with the following flags: IPV6_PREFER_SRC_HOME (X'00000001') Prefer home address IPV6_PREFER_SRC_COA (X'00000002') Prefer care-of address IPV6_PREFER_SRC_TMP (X'00000004') Prefer temporary address IPV6_PREFER_SRC_PUBLIC (X'00000008') Prefer public address IPV6_PREFER_SRC_CGA (X'00000010') Prefer cryptographically generated address IPV6_PREFER_SRC_NONCGA (X'00000020') Prefer non-cryptographically generated address</td>
<td>Contains the 4-byte flags field IPV6_ADDR_PREFERENCES_FLAGS that is defined in SYS1.MACLIB(BPXYSOCK) with the following flags: IPV6_PREFER_SRC_HOME (X'00000001') Prefer home address IPV6_PREFER_SRC_COA (X'00000002') Prefer care-of address IPV6_PREFER_SRC_TMP (X'00000004') Prefer temporary address IPV6_PREFER_SRC_PUBLIC (X'00000008') Prefer public address IPV6_PREFER_SRC_CGA (X'00000010') Prefer cryptographically generated address IPV6_PREFER_SRC_NONCGA (X'00000020') Prefer non-cryptographically generated address</td>
</tr>
</tbody>
</table>

**Use this option to query or set IPv6 address preferences of a socket. The default source address selection algorithm considers these preferences when it selects an IP address that is appropriate to communicate with a given destination address.**

This is an AF_INET6-only socket option.

**Result:** These flags are only preferences. The stack could assign a source IP address that does not conform to the IPV6_ADDR_PREFERENCES flags that you specify.

**Guideline:** Use the INET6_IS_SRCADDADDR function to test whether the source IP address matches one or more IPV6_ADDR_PREFERENCES flags.
<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IPV6_JOIN_GROUP</strong></td>
<td>Contains the IPv6_MREQ structure as defined in SYS1.MACLIB(BPXYSOCK). The IPv6_MREQ structure contains a 16-byte IPv6 multicast address followed by a 4-byte IPv6 interface index number.</td>
<td>N/A</td>
</tr>
<tr>
<td>Use this option to control the reception of multicast packets and specify that the socket join a multicast group.</td>
<td>If the interface index number is 0, then the stack chooses the local interface.</td>
<td></td>
</tr>
<tr>
<td>This is an IPv6-only socket option.</td>
<td>See the SEZAINST(CBLOCK) for the PL/I example of IPV6_MREQ.</td>
<td>See SEZAINST(EZACOBOL) for the COBOL example of IPV6-MREQ.</td>
</tr>
<tr>
<td><strong>IPV6_LEAVE_GROUP</strong></td>
<td>Contains the IPv6_MREQ structure as defined in SYS1.MACLIB(BPXYSOCK). The IPv6_MREQ structure contains a 16-byte IPv6 multicast address followed by a 4-byte IPv6 interface index number.</td>
<td>N/A</td>
</tr>
<tr>
<td>Use this option to control the reception of multicast packets and specify that the socket leave a multicast group.</td>
<td>If the interface index number is 0, then the stack chooses the local interface.</td>
<td></td>
</tr>
<tr>
<td>This is an IPv6-only socket option.</td>
<td>See the SEZAINST(CBLOCK) for the PL/I example of IPV6_MREQ.</td>
<td>See SEZAINST(EZACOBOL) for the COBOL example of IPV6-MREQ.</td>
</tr>
<tr>
<td><strong>IPV6_MULTICAST_HOPS</strong></td>
<td>Contains a 4-byte binary value specifying the multicast hops. If not specified, then the default is 1 hop.</td>
<td>N/A</td>
</tr>
<tr>
<td>Use to set or obtain the hop limit used for outgoing multicast packets.</td>
<td>-1 indicates use stack default.</td>
<td>Contains a 4-byte binary value in the range 0 – 255 indicating the number of multicast hops.</td>
</tr>
<tr>
<td>This is an IPv6-only socket option.</td>
<td>0 – 255 is the valid hop limit range.</td>
<td></td>
</tr>
</tbody>
</table>

*Note:* An application must be APF authorized to enable it to set the hop limit value above the system defined hop limit value. CICS applications cannot execute as APF authorized.
Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IPV6_MULTICAST_IF</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use this option to set or obtain the index of the IPv6 interface used for sending outbound multicast datagrams from the socket application.</td>
<td>Contains a 4-byte binary field containing an IPv6 interface index number.</td>
<td>Contains a 4-byte binary field containing an IPv6 interface index number.</td>
</tr>
<tr>
<td>This is an IPv6-only socket option.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>IPV6_MULTICAST_LOOP</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use this option to control or determine whether a multicast datagram is looped back on the outgoing interface by the IP layer for local delivery when datagrams are sent to a group to which the sending host itself belongs. The default is to loop multicast datagrams back.</td>
<td>A 4-byte binary field. To enable, set to 1. To disable, set to 0.</td>
<td>A 4-byte binary field. If enabled, contains a 1. If disabled, contains a 0.</td>
</tr>
<tr>
<td>This is an IPv6-only socket option.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>IPV6_UNICAST_HOPS</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use this option to set or obtain the hop limit used for outgoing unicast IPv6 packets.</td>
<td>Contains a 4-byte binary value specifying the unicast hops. If not specified, then the default is 1 hop. -1 indicates use stack default. 0 – 255 is the valid hop limit range. <strong>Note:</strong> APF authorized applications are permitted to set a hop limit that exceeds the system configured default. CICS applications cannot execute as APF authorized.</td>
<td>Contains a 4-byte binary value in the range 0 – 255 indicating the number of unicast hops.</td>
</tr>
<tr>
<td>This is an IPv6-only socket option.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>IPV6_V6ONLY</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use this option to set or determine whether the socket is restricted to send and receive only IPv6 packets. The default is to not restrict the sending and receiving of only IPv6 packets.</td>
<td>A 4-byte binary field. To enable, set to 1. To disable, set to 0.</td>
<td>A 4-byte binary field. If enabled, contains a 1. If disabled, contains a 0.</td>
</tr>
<tr>
<td>This is an IPv6-only socket option.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPTNAME options (input)</td>
<td>SETSOCKOPT, OPTVAL (input)</td>
<td>GETSOCKOPT, OPTVAL (output)</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>MCAST_BLOCK_SOURCE</td>
<td>Contains the GROUP_SOURCE_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_SOURCE_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address and a socket address structure of the source address.</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>See SEZAINST(CBLOCP) for the PL/I example of GROUP_SOURCE_REQ.</td>
<td>See SEZAINST(EZACOBOL) for the COBOL example of GROUP-SOURCE-REQ.</td>
</tr>
<tr>
<td>MCAST_JOIN_GROUP</td>
<td>Contains the GROUP_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address.</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>See SEZAINST(CBLOCP) for the PL/I example of GROUP_REQ.</td>
<td>See SEZAINST(EZACOBOL) for the COBOL example of GROUP-REQ.</td>
</tr>
<tr>
<td>MCAST_JOIN_SOURCE_GROUP</td>
<td>Contains the GROUP_SOURCE_REQ structure as defined in SYS1.MACLIB(BPXYSOCK). The GROUP_SOURCE_REQ structure contains a 4-byte interface index number followed by a socket address structure of the multicast address and a socket address structure of the source address.</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>See SEZAINST(CBLOCP) for the PL/I example of GROUP_SOURCE_REQ.</td>
<td>See SEZAINST(EZACOBOL) for the COBOL example of GROUP-SOURCE-REQ.</td>
</tr>
<tr>
<td>OPTNAME options (input)</td>
<td>SETSOCKOPT, OPTVAL (input)</td>
<td>GETSOCKOPT, OPTVAL (output)</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td><strong>MCAST_LEAVE_GROUP</strong></td>
<td>Contains the GROUP_REQ</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>structure as defined in</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SYS1.MACLIB(BPXYSOCK). The</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GROUP_REQ structure contains</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a 4-byte interface index</td>
<td></td>
</tr>
<tr>
<td></td>
<td>number followed by a socket</td>
<td></td>
</tr>
<tr>
<td></td>
<td>address structure of the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>multicast address.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>See SEZAINST(CBLOCK) for the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PL/I example of GROUP_REQ.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>See SEZAINST(EZACOBOL) for</td>
<td></td>
</tr>
<tr>
<td></td>
<td>the COBOL example of GROUP-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REQ.</td>
<td></td>
</tr>
<tr>
<td><strong>MCAST_LEAVE_SOURCE_GROUP</strong></td>
<td>Contains the GROUP_SOURCE_REQ</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>structure as defined in</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SYS1.MACLIB(BPXYSOCK). The</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GROUP_SOURCE_REQ structure</td>
<td></td>
</tr>
<tr>
<td></td>
<td>contains a 4-byte interface</td>
<td></td>
</tr>
<tr>
<td></td>
<td>index number followed by a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>socket address structure of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>the multicast address and a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>socket address structure of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>the source address.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>See SEZAINST(CBLOCK) for the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PL/I example of GROUP_SOURCE_REQ.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>See SEZAINST(EZACOBOL) for</td>
<td></td>
</tr>
<tr>
<td></td>
<td>the COBOL example of GROUP-SOURCE-REQ.</td>
<td></td>
</tr>
<tr>
<td><strong>MCAST_UNBLOCK_SOURCE</strong></td>
<td>Contains the GROUP_SOURCE_REQ</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>structure as defined in</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SYS1.MACLIB(BPXYSOCK). The</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GROUP_SOURCE_REQ structure</td>
<td></td>
</tr>
<tr>
<td></td>
<td>contains a 4-byte interface</td>
<td></td>
</tr>
<tr>
<td></td>
<td>index number followed by a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>socket address structure of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>the multicast address and a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>socket address structure of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>the source address.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>See SEZAINST(CBLOCK) for the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PL/I example of GROUP_SOURCE_REQ.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>See SEZAINST(EZACOBOL) for</td>
<td></td>
</tr>
<tr>
<td></td>
<td>the COBOL example of GROUP-SOURCE-REQ.</td>
<td></td>
</tr>
</tbody>
</table>
Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SO_ASCII</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use this option to set or determine the translation to ASCII data option. When SO_ASCII is set, data is translated to ASCII. When SO_ASCII is not set, data is not translated to or from ASCII. <strong>Note:</strong> This is a REXX-only socket option.</td>
<td>To enable, set to ON.</td>
<td>If enabled, contains ON.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> The optvalue is returned and is optionally followed by the name of the translation table that is used if translation is applied to the data.</td>
</tr>
<tr>
<td><strong>SO_BROADCAST</strong></td>
<td>A 4-byte binary field.</td>
<td>A 4-byte binary field.</td>
</tr>
<tr>
<td>Use this option to set or determine whether a program can send broadcast messages over the socket to destinations that can receive datagram messages. The default is disabled. <strong>Note:</strong> This option has no meaning for stream sockets.</td>
<td>To enable, set to 1 or a positive value.</td>
<td>If enabled, contains a 1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SO_DEBUG</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use SO_DEBUG to set or determine the status of the debug option. The default is disabled. The debug option controls the recording of debug information. <strong>Note:</strong> 1. This is a REXX-only socket option. 2. This option has meaning only for stream sockets.</td>
<td>To enable, set to ON.</td>
<td>If enabled, contains ON.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SO_EBCDIC</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use this option to set or determine the translation to EBCDIC data option. When SO_EBCDIC is set, data is translated to EBCDIC. When SO_EBCDIC is not set, data is not translated to or from EBCDIC. This option is ignored by EBCDIC hosts. <strong>Note:</strong> This is a REXX-only socket option.</td>
<td>To enable, set to ON.</td>
<td>If enabled, contains ON.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SO_ERROR</strong></td>
<td>N/A</td>
<td>A 4-byte binary field containing the most recent ERRNO for the socket.</td>
</tr>
<tr>
<td>Use this option to request pending errors on the socket or to check for asynchronous errors on connected datagram sockets or for other errors that are not explicitly returned by one of the socket calls. The error status is clear afterwards.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPTNAME options (input)</td>
<td>SETSOCKOPT, OPTVAL (input)</td>
<td>GETSOCKOPT, OPTVAL (output)</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td><strong>SO_KEEPALIVE</strong></td>
<td>A 4-byte binary field.</td>
<td>A 4-byte binary field.</td>
</tr>
<tr>
<td>Use this option to set or determine whether the keep alive mechanism periodically sends a packet on an otherwise idle connection for a stream socket. The default is disabled. When activated, the keep alive mechanism periodically sends a packet on an otherwise idle connection. If the remote TCP does not respond to the packet or to retransmissions of the packet, the connection is terminated with the error ETIMEDOUT.</td>
<td>To enable, set to 1 or a positive value. To disable, set to 0.</td>
<td></td>
</tr>
<tr>
<td><strong>SO_LINGER</strong></td>
<td>Contains an 8-byte field containing two 4-byte binary fields.</td>
<td>Contains an 8-byte field containing two 4-byte binary fields.</td>
</tr>
<tr>
<td>Use this option to control or determine how TCP/IP processes data that has not been transmitted when a CLOSE is issued for the socket. The default is disabled.</td>
<td>Assembler coding: ONOFF DS F LINGER DS F</td>
<td>Assembler coding: ONOFF DS F LINGER DS F</td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td>COBOL coding: ONOFF PIC 9(8) BINARY. LINGER PIC 9(8) BINARY.</td>
<td>COBOL coding: ONOFF PIC 9(8) BINARY. LINGER PIC 9(8) BINARY.</td>
</tr>
<tr>
<td>1. This option has meaning only for stream sockets.</td>
<td>Set ONOFF to a nonzero value to enable and set to 0 to disable this option. Set LINGER to the number of seconds that TCP/IP lingers after the CLOSE is issued.</td>
<td>A nonzero value returned in ONOFF indicates enabled, a 0 indicates disabled. LINGER indicates the number of seconds that TCP/IP will try to send data after the CLOSE is issued.</td>
</tr>
<tr>
<td>2. If you set a zero linger time, the connection cannot close in an orderly manner, but stops, resulting in a RESET segment being sent to the connection partner. Also, if the aborting socket is in nonblocking mode, the close call is treated as though no linger option had been set. When SO_LINGER is set and CLOSE is called, the calling program is blocked until the data is successfully transmitted or the connection has timed out. When SO_LINGER is not set, the CLOSE returns without blocking the caller, and TCP/IP continues to attempt to send data for a specified time. This usually allows sufficient time to complete the data transfer. Use of the SO_LINGER option does not guarantee successful completion because TCP/IP waits only the amount of time specified in OPTVAL for SO_LINGER.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPTNAME options (input)</td>
<td>SETSOCKOPT, OPTVAL (input)</td>
<td>GETSOCKOPT, OPTVAL (output)</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td><strong>SO_OOBINLINE</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Use this option to control or determine whether out-of-band data is received. **Note:** This option has meaning only for stream sockets.

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

When this option is disabled, out-of-band data is placed in the priority data input queue as it is received and is available to a RECV or a RECVFROM only when the OOB flag is set in the RECV or the RECVFROM.

A 4-byte binary field.

To enable, set to 1 or a positive value.

To disable, set to 0.

A 4-byte binary field.

If enabled, contains a 1.

If disabled, contains a 0.

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

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

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

A 4-byte binary field.

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

To disable, set to a 0.

A 4-byte binary field.

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

If disabled, contains a 0.
### Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SO_RCVTIMEO</td>
<td>This option requires a TIMEVAL structure, which is defined in SYS1.MACLIB( BPXYRLIM) macro. The TIMEVAL structure contains the number of seconds and microseconds specified as fullword binary numbers. The seconds can be a value in the range 0 - 2678400 (equal to 31 days), and the microseconds can be a value in the range 0 - 1000000 (equal to 1 second). Although TIMEVAL value can be specified using microsecond granularity, the internal TCP/IP timers that are used to implement this function have a granularity of approximately 100 milliseconds.</td>
<td>This option stores a TIMEVAL structure that is defined in the SYS1.MACLIB(BPXYRLIM) macro. The TIMEVAL structure contains the number of seconds and microseconds, which are specified as fullword binary numbers. The number of seconds value that is returned is in the range 0 - 2678400 (equal to 31 days). The number of microseconds value that is returned is in the range 0 - 1000000.</td>
</tr>
<tr>
<td>OPTNAME options (input)</td>
<td>SETSOCKOPT, OPTVAL (input)</td>
<td>GETSOCKOPT, OPTVAL (output)</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td><strong>SO_REUSEADDR</strong></td>
<td>A 4-byte binary field.</td>
<td>A 4-byte binary field.</td>
</tr>
<tr>
<td></td>
<td>To enable, set to 1 or a positive value.</td>
<td>If enabled, contains a 1.</td>
</tr>
<tr>
<td></td>
<td>To disable, set to 0.</td>
<td>If disabled, contains a 0.</td>
</tr>
</tbody>
</table>

Use this option to control or determine whether local addresses are reused. The default is disabled. This alters the normal algorithm used with BIND. The normal BIND algorithm allows each Internet address and port combination to be bound only once. If the address and port have been already bound, then a subsequent BIND will fail and result error will be EADDRINUSE.

When this option is enabled, the following situations are supported:

- A server can BIND the same port multiple times as long as every invocation uses a different local IP address and the wildcard address INADDR_ANY is used only one time per port.
- A server with active client connections can be restarted and can bind to its port without having to close all of the client connections.
- For datagram sockets, multicasting is supported so multiple bind() calls can be made to the same class D address and port number.
- If you require multiple servers to BIND to the same port and listen on INADDR_ANY, see the SHAREPORT option on the PORT statement in TCPIP.PROFILE.

| **SO_SNDBUF**          | A 4-byte binary field.     | A 4-byte binary field.      |
|                       | To enable, set to a positive value specifying the size of the data portion of the TCP/IP send buffer. | If enabled, contains a positive value indicating the size of the data portion of the TCP/IP send buffer. |
|                       | To disable, set to a 0.    | If disabled, contains a 0.  |

Use this option to control or determine the size of the data portion of the TCP/IP send buffer. The size is of the TCP/IP send buffer is protocol specific and is based on the following:

- The TCPSendBufsize keyword on the TCPCONFIG statement in the PROFILE.TCPIP data set for a TCP socket
- The UDPSendBufsize keyword on the UDPCONFIG statement in the PROFILE.TCPIP data set for a UDP socket
- The default of 65 535 for a raw socket
Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

<table>
<thead>
<tr>
<th>OPTNAME options (input)</th>
<th>SETSOCKOPT, OPTVAL (input)</th>
<th>GETSOCKOPT, OPTVAL (output)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SO_SNDTIMEO</strong></td>
<td>This option requires a TIMEVAL structure, which is defined in the SYS1.MACLIB(BPXYRLIM) macro. The TIMEVAL structure contains the number of seconds and microseconds specified as fullword binary numbers. The seconds value is in the range 0 - 2678400 (equal to 31 days), and the microseconds value is in the range 0 - 1000000 (equal to 1 second). Although the TIMEVAL value can be specified using microsecond granularity, the internal TCP/IP timers that are used to implement this function have a granularity of approximately 100 milliseconds.</td>
<td>This option stores a TIMEVAL structure that is defined in SYS1.MACLIB(BPXYRLIM). The TIMEVAL structure contains the number of seconds and microseconds, which are specified as fullword binary numbers. The number of seconds value that is returned is in the range 0 - 2678400 (equal to 31 days). The microseconds value that is returned is in the range 0 - 1000000.</td>
</tr>
<tr>
<td><strong>SO_TYPE</strong></td>
<td>N/A</td>
<td>A 4-byte binary field indicating the socket type: X’1’ indicates SOCK_STREAM. X’2’ indicates SOCK_DGRAM. X’3’ indicates SOCK_RAW.</td>
</tr>
<tr>
<td><strong>TCP_KEEPALIVE</strong></td>
<td>A 4-byte binary field.</td>
<td>A 4-byte binary field. If enabled, contains the specific timer value (in seconds) that is in effect for the given socket. If disabled, contains a 0 indicating keep alive timing is not active.</td>
</tr>
</tbody>
</table>

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

If a send-type function has blocked for this length of time, it returns with a partial count or, if no data is sent, with an errno set to EWOULDBLOCK. The default value for this is 0, which indicates that a send-type function does not time out.

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

Use this option to return the socket type.

See the [z/OS Communications Server: IP Programmer’s Guide and Reference](https://www.ibm.com/support/docview.wss?uid=comsvrsipp) for more information about the socket option parameters.

A socket-specific timeout value (in seconds) is to be used in place of a configuration-specific value whenever keep alive timing is active for that socket.

When activated, the socket-specified timer value remains in effect until respecified by SETSOCKOPT or until the socket is closed.
### Table 22. OPTNAME options for GETSOCKOPT and SETSOCKOPT (continued)

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

**Note:** Use the following to set TCP_NODELAY OPTNAME value for COBOL programs:

```
01 TCP-NODELAY-VAL PIC 9(10) COMP VALUE 2147483649.
01 TCP-NODELAY-REDEF REDEFINES TCP-NODELAY-VAL.
05 FILLER PIC 9(6) BINARY.
05 TCP-NODELAY PIC 9(8) BINARY.
```

---

### SHUTDOWN call

One way to terminate a network connection is to issue the CLOSE call which attempts to complete all outstanding data transmission requests prior to breaking the connection. The SHUTDOWN call can be used to close one-way traffic while completing data transfer in the other direction. The HOW parameter determines the direction of traffic to shutdown.

When the CLOSE call is used, the SETSOCKOPT OPTVAL LINGER parameter determines the amount of time the system waits before releasing the connection. For example, with a LINGER value of 30 seconds, system resources (including the IMS or CICS transaction) remain in the system for up to 30 seconds after the CLOSE call is issued. In high volume, transaction-based systems like CICS and IMS, this can impact performance severely.

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

If you issue SHUTDOWN for a socket that currently has outstanding socket calls pending, see Table 23 to determine the effects of this operation on the outstanding socket calls.

#### Table 23. Effect of SHUTDOWN socket call

<table>
<thead>
<tr>
<th>Socket calls in local program</th>
<th>Local program</th>
<th>Remote program</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SHUTDOWN SEND</td>
<td>SHUTDOWN RECEIVE</td>
</tr>
<tr>
<td>Write calls</td>
<td>Error number EPipe on first call</td>
<td>Error number EPipe on second call*</td>
</tr>
</tbody>
</table>
Table 23. Effect of SHUTDOWN socket call (continued)

<table>
<thead>
<tr>
<th>Socket calls in local program</th>
<th>Local program</th>
<th>Remote program</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SHUTDOWN SEND</td>
<td>SHUTDOWN RECEIVE</td>
</tr>
<tr>
<td>Read calls</td>
<td>Zero length return code</td>
<td>Zero length return code</td>
</tr>
</tbody>
</table>

* If you issue two write calls immediately, both might be successful, and an EPIPE error number might not be returned until a third write call is issued.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization:</td>
<td>Supervisor state or problem state, any PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN</td>
</tr>
<tr>
<td>Amode:</td>
<td>31-bit or 24-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary address space control (ASC) mode</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>Unlocked</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>All parameters must be addressable by the caller and in the primary address space</td>
</tr>
</tbody>
</table>

Figure 159 on page 393 shows an example of SHUTDOWN call instructions.
The SOCKET call creates an endpoint for communication and returns a socket descriptor representing the endpoint.

The following requirements apply to this call:

WORKING-STORAGE SECTION.
   01 SOC-FUNCTION PIC X(16) VALUE IS 'SHUTDOWN'.
   01 S PIC 9(4) BINARY.
   01 HOW PIC 9(8) BINARY.
   01 END-FROM PIC 9(8) BINARY VALUE 0.
   01 END-TO PIC 9(8) BINARY VALUE 1.
   01 END-BOTH PIC 9(8) BINARY VALUE 2.
   01 ERRNO PIC 9(8) BINARY.
   01 RETCODE PIC S9(8) BINARY.

PROCEDURE DIVISION.
   CALL 'EZASOKET' USING SOC-FUNCTION S HOW ERRNO RETCODE.

Figure 159. SHUTDOWN call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 253.

Parameter values set by the application for the SHUTDOWN call

SOC-FUNCTION
   A 16-byte character field containing SHUTDOWN. The field is left-aligned and padded on the right with blanks.

S
   A halfword binary number set to the socket descriptor of the socket to be shutdown.

HOW
   A fullword binary field. Set to specify whether all or part of a connection is to be shut down. The following values can be set:

   Value   Description
   0 (END-FROM)  Ends further receive operations.
   1 (END-TO)    Ends further send operations.
   2 (END-BOTH)  Ends further send and receive operations.

Parameter values returned to the application for the SHUTDOWN call

ERRNO
   A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B, "Return codes," on page 453 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 call

The SOCKET call creates an endpoint for communication and returns a socket descriptor representing the endpoint.

The following requirements apply to this call:
Requirement | Description
---|---
Authorization: | Supervisor state or problem state, any PSW key
Dispatchable unit mode: | Task
Cross memory mode: | PASN = HASN
Amode: | 31-bit or 24-bit
ASC mode: | Primary address space control (ASC) mode
Interrupt status: | Enabled for interrupts
Locks: | Unlocked
Control parameters: | All parameters must be addressable by the caller and in the primary address space

Figure 160 shows an example of SOCKET call instructions.

**WORKING-STORAGE SECTION.**

```
01 SOC-FUNCTION PIC X(16) VALUE IS 'SOCKET'.
* For AF_INET
  01 AF PIC 9(8) COMP VALUE 2.
* For AF_INET6
  01 AF PIC 9(8) BINARY VALUE 19.
  01 SOCTYPE PIC 9(8) BINARY.
  01 STREAM PIC 9(8) BINARY VALUE 1.
  01 DATAGRAM PIC 9(8) BINARY VALUE 2.
  01 PROTO PIC 9(8) BINARY.
  01 ERRNO PIC 9(8) BINARY.
  01 RETCODE PIC S9(8) BINARY.
```

**PROCEDURE DIVISION.**

```
CALL 'EZASOKET' USING SOC-FUNCTION AF SOCTYPE PROTO ERRNO RETCODE.
```

*Figure 160. SOCKET call instruction example*

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 253.

**Parameter values set by the application for the SOCKET call**

**SOC-FUNCTION**

A 16-byte character field containing 'SOCKET'. The field is left-aligned and padded on the right with blanks.

**AF**

A fullword binary field set to the addressing family. For TCP/IP the value is set to a decimal 2 for AF_INET, or a decimal 19, indicating AF_INET6.

**SOCTYPE**

A fullword binary field set to the type of socket required. The types are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Stream sockets provide sequenced, two-way byte streams that are reliable and connection-oriented. They support a mechanism for out-of-band data.</td>
</tr>
<tr>
<td>2</td>
<td>Datagram sockets provide datagrams, which are connectionless messages of a fixed maximum length whose reliability is not guaranteed. Datagrams can be corrupted, received out of order, lost, or delivered multiple times.</td>
</tr>
</tbody>
</table>
PROTO
A fullword binary field set to the protocol to be used for the socket. If this field is set to 0, the default protocol is used. For streams, the default is TCP; for datagrams, the default is UDP.

PROTO numbers are found in the hlq.etc.proto data set.

Parameter values returned to the application for the SOCKET call

ERRNO
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.

RETCODE
A fullword binary field that returns one of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 0 or = 0</td>
<td>Contains the new socket descriptor</td>
</tr>
<tr>
<td>-1</td>
<td>Check ERRNO for an error code</td>
</tr>
</tbody>
</table>

TAKESOCKET call

The TAKESOCKET call acquires a socket from another program and creates a new socket. Typically, a child server issues this call using client ID and socket descriptor data that it obtained from the concurrent server. See “GIVESOCKET call” on page 314 for a discussion of the use of GETSOCKOPT and TAKESOCKET calls.

Note: When TAKESOCKET is issued, a new socket descriptor is returned in RETCODE. You should use this new socket descriptor in subsequent calls such as GETSOCKOPT, which require the S (socket descriptor) parameter.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization:</td>
<td>Supervisor state or problem state, any PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN</td>
</tr>
<tr>
<td>Amode:</td>
<td>31-bit or 24-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary address space control (ASC) mode</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>Unlocked</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>All parameters must be addressable by the caller and in the primary address space</td>
</tr>
</tbody>
</table>

Figure 161 on page 396 shows an example of TAKESOCKET call instructions.
WORKING-STORAGE SECTION.
   01 SOC-FUNCTION PIC X(16) VALUE IS 'TAKESOCKET'.
   01 SOCRECV PIC 9(4) BINARY.
   01 CLIENT.
      03 DOMAIN PIC 9(8) BINARY.
      03 NAME PIC X(8).
      03 TASK PIC X(8).
      03 RESERVED PIC X(20).
   01 ERRNO PIC 9(8) BINARY.
   01 RETCODE PIC S9(8) BINARY.

PROCEDURE DIVISION.
   CALL 'EZASOKET' USING SOC-FUNCTION SOCRECV CLIENT ERRNO RETCODE.

Figure 161. TAKESOCKET call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 253.

Parameter values set by the application for the TAKESOCKET call

SOC-FUNCTION
   A 16-byte character field containing TAKESOCKET. The field is left-aligned and padded to the right with blanks.

SOCRECV
   A halfword binary field set to the descriptor of the socket to be taken. The socket to be taken is passed by the concurrent server.

CLIENT
   Specifies the client ID of the program that is giving the socket. In CICS, these parameters are passed by the listener program to the program that issues the TAKESOCKET call. The information is obtained using EXEC CICS RETRIEVE.

DOMAIN
   A fullword binary field set to the domain of the program giving the socket. It is always a decimal 2, indicating AF_INET, or a decimal 19, indicating AF_INET6.

   Rule: The TAKESOCKET can acquire only a socket of the same address family from a GIVESOCKET.

NAME
   Specifies an 8-byte character field set to the MVS address space identifier of the program that gave the socket.

TASK
   Specifies an 8-byte 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 for the TAKESOCKET call

ERRNO
   A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B, "Return codes," on page 455 for information about ERRNO return codes.
RETCODE
A fullword binary field that returns one of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; or = 0</td>
<td>Contains the new socket descriptor</td>
</tr>
<tr>
<td>-1</td>
<td>Check ERRNO for an error code</td>
</tr>
</tbody>
</table>

**TERMAPI call**

This call terminates the session created by INITAPI. All TCP/IP stacks resources allocated to the task are cleaned up. This includes any outstanding open sockets or sockets that have been given away with the GIVESOCKET call but have not been taken with a TAKESOCKET call.

In the CICS environment, the use of TERMAPI is not recommended. CICS task termination processing automatically performs the functions of TERMAPI. A CICS application program should issue TERMAPI only if there is a particular need to terminate the session before task termination.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization:</td>
<td>Supervisor state or problem state, any PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN</td>
</tr>
<tr>
<td>Amode:</td>
<td>31-bit or 24-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary address space control (ASC) mode</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>Unlocked</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>All parameters must be addressable by the caller and in the primary address space</td>
</tr>
</tbody>
</table>

Figure 162 on page 398 shows an example of TERMAPI call instructions.
For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 253.

Parameter values set by the application for the TERMAPI call

SOC-FUNCTION
A 16-byte character field containing TERMAPI. The field is left-aligned and padded to the right with blanks.

WRITE call

The WRITE call writes data on a connected socket. This call is similar to SEND, except that it lacks the control flags available with SEND.

For datagram sockets the WRITE call writes the entire datagram if it fits into the receiving buffer.

Stream sockets act like streams of information with no boundaries separating data. For example, if a program wishes to send 1000 bytes, each call to this function can send any number of bytes, up to the entire 1000 bytes. The number of bytes sent are returned in RETCODE. Therefore, programs using stream sockets should place this call in a loop, calling this function until all data has been sent.

See "EZACIC04 program" on page 404 for a subroutine that translates EBCDIC output data to ASCII.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization:</td>
<td>Supervisor state or problem state, any PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN</td>
</tr>
<tr>
<td>Amode:</td>
<td>31-bit or 24-bit</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> See “Addressability mode (Amode) considerations” under &quot;Environmental restrictions and programming requirements for the Callable Socket API&quot; on page 249.</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary address space control (ASC) mode</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>Unlocked</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>All parameters must be addressable by the caller and in the primary address space</td>
</tr>
</tbody>
</table>

Figure 163 on page 399 shows an example of WRITE call instructions.
Parameter values set by the application for the WRITE call

SOC-FUNCTION
A 16-byte character field containing WRITE. The field is left-aligned and padded on the right with blanks.

S
A halfword binary field set to the socket descriptor.

NBYTE
A fullword binary field set to the number of bytes of data to be transmitted.

BUF
Specifies the buffer containing the data to be transmitted.

Parameter values returned to the application for the WRITE call

ERRNO
A fullword binary field. If RETCODE is negative, the field contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.

RETCODE
A fullword binary field that returns one of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>≥0</td>
<td>A successful call. A return code greater than zero indicates the number of bytes of data written.</td>
</tr>
<tr>
<td>-1</td>
<td>Check ERRNO for an error code.</td>
</tr>
</tbody>
</table>

WRITEV call

The WRITEV function writes data on a socket from a set of buffers.

The following requirements apply to this call:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization:</td>
<td>Supervisor state or problem state, any PSW key</td>
</tr>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN</td>
</tr>
<tr>
<td>Amode:</td>
<td>31-bit or 24-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary address space control (ASC mode)</td>
</tr>
<tr>
<td>Requirement</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>Unlocked</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>All parameters must be addressable by the caller and in the primary address space</td>
</tr>
</tbody>
</table>

Figure 164 shows an example of WRITEV call instructions.

```plaintext
WORKING-STORAGE SECTION.
01 SOCKET-FUNCTION PIC X(16) VALUE 'WRITEV'.
01 S PIC 9(4) BINARY.
01 IOVCNT PIC 9(8) BINARY.
01 IOV.
   03 BUFFER-ENTRY OCCURS N TIMES.
   05 BUFFER-POINTER USAGE IS POINTER.
   05 RESERVED PIC X(4).
   05 BUFFER-LENGTH PIC 9(8) BINARY.
01 ERRNO PIC 9(8) BINARY.
01 RETCODE PIC 9(8) BINARY.

PROCEDURE DIVISION.
SET BUFFER-POINTER(1) TO ADDRESS OF BUFFER1.
SET BUFFER-LENGTH(1) TO LENGTH OF BUFFER1.
SET BUFFER-POINTER(2) TO ADDRESS OF BUFFER2.
SET BUFFER-LENGTH(2) TO LENGTH OF BUFFER2.
   = " " " 
   = " " " 
SET BUFFER-POINTER(n) TO ADDRESS OF BUFFERn.
SET BUFFER-LENGTH(n) TO LENGTH OF BUFFERn.
CALL 'EZASOKET' USING SOC-FUNCTION S IOV IOVCNT ERRNO RETCODE.
```

Figure 164. WRITEV call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 253.

Parameter values set by the application for the WRITEV call

**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:

1. **Fullword 1**
   The address of a data buffer.

2. **Fullword 2**
   Reserved.

3. **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 for the WRITEV call

ERRNO
A fullword binary field. If RETCODE is negative, this contains an error number. See Appendix B, “Return codes,” on page 455 for information about ERRNO return codes.

RETCODE
A fullword binary field.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0</td>
<td>Error. Check ERRNO.</td>
</tr>
<tr>
<td>0</td>
<td>Connection partner has closed connection.</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Number of bytes sent.</td>
</tr>
</tbody>
</table>

Using data translation programs for socket call interface

In addition to the socket calls, you can use the following utility programs to translate data.

Data translation from ASCII and EBCDIC data notation

TCP/IP hosts and networks use ASCII data notation; MVS TCP/IP and its subsystems use EBCDIC data notation. In situations where data must be translated from one notation to the other, you can use the following utility programs:

EZACIC04
Translates EBCDIC data to ASCII data using an EBCDIC-to-ASCII translation table as described in z/OS Communications Server: IP Configuration Reference.

EZACIC05
Translates ASCII data to EBCDIC data using an ASCII-to-EBCDIC translation table as described in z/OS Communications Server: IP Configuration Reference.

EZACIC14
An alternative to EZACIC04 that translates EBCDIC data to ASCII data using the translation table listed in “EZACIC14 program” on page 416.

EZACIC15
An alternative to EZACIC05 that translates ASCII data to EBCDIC data using the translation table listed in “EZACIC15 program” on page 418.

A sample program that performs these translations is also available; you can modify them to perform any translations not provided by these routines. See the EZACICTR member in the SEZAINST data set for more information.

It is not necessary to define these programs to CICS. If your application dynamically links these programs, then you must define them to CICS as follows:

```
DEFINE PROGRAM(EZACIC04)
DESCRIPTION(TRANSLATE EBCDIC-8 BIT TO ASCII-8 BIT)
GROUP(SOCKETS)
CEDF(YES) DATALLOCATION(ANY) EXECKEY(USER)
RELOAD(NO) RESIDENT(NO) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(NORMAL)
CONCURRENCY(THREADSAFE)
```
DEFINE PROGRAM(EZACIC05)
DESCRIPTION(TRANSLATE ASCII-8 BIT TO EBCDIC-8 BIT)
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(USER)
RELOAD(NO) RESIDENT(NO) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(NORMAL)
CONCURRENCY(THREADSAFE)

DEFINE PROGRAM(EZACIC14)
DESCRIPTION(TRANSLATE EBCDIC-8 BIT TO ASCII-8 BIT)
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(USER)
RELOAD(NO) RESIDENT(NO) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(NORMAL)
CONCURRENCY(THREADSAFE)

DEFINE PROGRAM(EZACIC15)
DESCRIPTION(TRANSLATE ASCII-8 BIT TO EBCDIC-8 BIT)
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(USER)
RELOAD(NO) RESIDENT(NO) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(NORMAL)
CONCURRENCY(THREADSAFE)

For more information about specifying the key that CICS uses to give control to the program and details about RDO resource types and their attributes, Program Definition Attributes, and the EXECKEY attribute, see the CICS Transaction Server information on this website: http://www-01.ibm.com/software/htp/cics/library/

**Bit string processing**

In C-language, bit strings are often used to convey flags, switch settings, and so on; TCP/IP stacks makes frequent uses of bit strings. However, because bit strings are difficult to decode in COBOL, TCP/IP includes:

**EZACIC06**

Translates bit-masks into character arrays and character arrays into bit-masks.

**EZACIC08**

Interprets the variable length address list in the HOSTENT structure returned by GETHOSTBYNAME or GETHOSTBYADDR.

**EZACIC09**

Interprets the ADDRINFO structure returned by GETADDRINFO.

It is not necessary to define these programs to CICS. If your application dynamically links these programs, then you must define them to CICS as follows:

DEFINE PROGRAM(EZACIC06)
DESCRIPTION(TRANSLATE EBCDIC-8 BIT TO ASCII-8 BIT)
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(USER)
RELOAD(NO) RESIDENT(NO) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(NORMAL)
CONCURRENCY(THREADSAFE)

DEFINE PROGRAM(EZACIC08)
DESCRIPTION(INTERPRET HOSTENT)
GROUP(SOCKETS)
CEDF(YES) DATALOCATION(ANY) EXECKEY(USER)
RELOAD(NO) RESIDENT(NO) USELPACOPY(NO)
LANGUAGE(ASSEMBLER) STATUS(ENABLED) USAGE(NORMAL)
CONCURRENCY(THREADSAFE)
CALL instruction utility programs

This topic describes the CALL instruction API for TCP/IP application programs written in the COBOL, PL/I, or High Level Assembler language. The format and parameters are described for each utility call.

Note: For a PL/I program, include the following statement before your first call instruction:
DCL EZASOKET ENTRY OPTIONS(RETCODE,ASM,INTER) EXT;

Understanding COBOL, assembler, and PL/I call formats:

These utility programs are invoked by calling the EZACICnn program. The parameters look differently due to the differences in the programming languages.

COBOL language call format sample:

The following sample illustrates the utility program call format for COBOL language programs:

```assembler
>>-- CALL 'EZACICnn' USING parm1, parm2, ... . --><
```

`parm n`
A variable number of parameters that depends on the type call.

The utility programs in this topic contain an explanation of the call parameters.

Assembler language call format sample:

The following sample illustrates the utility program call format for assembler language programs. Because DATAREG is used to access the application's working storage, applications using the assembler language format should not code DATAREG but should let it default to the CICS data register.

```assembler
>>-- CALL EZACICnn,(parm1, parm2, ... ),VL,MF=(E, PARMLIST) --><
```

PARMLIST is a remote parameter list defined in dynamic storage DFHEISTG. This list contains addresses of 30 parameters that can be referenced by all execute forms of the CALL.

Note: This form of CALL is necessary to meet the CICS requirement for quasi-reentrant programming

`parm n`
A variable number of parameters that depends on the type call.

The utility programs in this topic contain an explanation of the call parameters.
PL/I language call format sample:

The following sample illustrates the utility program call format for PL/I language programs:

```plaintext
>>>- CALL EZACICnn (parm1, parm2, ... ); --<<
parm n
parm n
A variable number of parameters that depends on the type call.
```

See the utility programs in this topic for an explanation of the parameters.

### EZACIC04 program

The EZACIC04 program is used to translate EBCDIC data to ASCII data.

Figure 165 shows an example of how EZACIC04 translates a byte of EBCDIC data to ASCII data.

<table>
<thead>
<tr>
<th>ASCII output by EZACIC04</th>
<th>second hex digit of byte of EBCDIC data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>1</td>
<td>0A</td>
</tr>
<tr>
<td>2</td>
<td>1A</td>
</tr>
</tbody>
</table>

Figure 165. EZACIC04 EBCDIC-to-ASCII table

Figure 166 on page 405 shows an example of EZACIC04 call instructions.
WORKING-Storage SECTION.
   01 OUT-BUFFER PIC X(length of output).
   01 LENGTH PIC 9(8) BINARY.

PROCEDURE DIVISION.
   CALL 'EZACIC04' USING OUT-BUFFER LENGTH. IF RETURN-CODE > 0 THEN
      DISPLAY 'TRANSLATION FAILED ' RETURN-CODE.

Figure 166. EZACIC04 call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 253.

OUT-BUFFER
   A buffer that contains the following:
   • When called – EBCDIC data
   • Upon return – ASCII data

LENGTH
   Specifies the length of the data to be translated.

EZACIC05 program

The EZACIC05 program is used to translate ASCII data to EBCDIC data. EBCDIC data is required by COBOL, PL/I, and assembler language programs.

Figure 167 on page 406 shows an example of how EZACIC05 translates a byte of ASCII data to EBCDIC data.
Figure 168 on page 407 shows an example of EZACIC05 call instructions.

<table>
<thead>
<tr>
<th>EBCDIC output by EZACIC05</th>
<th>second hex digit of byte of ASCII data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>F0</td>
</tr>
<tr>
<td>4</td>
<td>7C</td>
</tr>
<tr>
<td>5</td>
<td>D7</td>
</tr>
<tr>
<td>6</td>
<td>79</td>
</tr>
<tr>
<td>7</td>
<td>97</td>
</tr>
<tr>
<td>8</td>
<td>00</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

Figure 167. EZACIC05 ASCII-to-EBCDIC
WORKING-STORAGE SECTION.
  01 IN-BUFFER PIC X(length of output)
  01 LENGTH PIC 9(8) BINARY VALUE

PROCEDURE DIVISION.
  CALL 'EZACIC05' USING IN-BUFFER LENGTH. IF RETURN-CODE > 0 THEN
  DISPLAY 'TRANSLATION FAILED ' RETURN-CODE.

Figure 168. EZACIC05 call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 253.

IN-BUFFER
  A buffer that contains the following:
  • When called – ASCII data
  • Upon return – EBCDIC data

LENGTH
  Specifies the length of the data to be translated.

EZACIC06 program

The SELECT call uses bit strings to specify the sockets to test and to return the results of the test. Because bit strings are difficult to manage in COBOL, use the EZACIC06 utility program to translate bit strings to character strings to be used with the SELECT or SELECTEX call.

Figure 169 on page 408 shows an example of EZACIC06 call instructions.
For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 253.

CHAR-MASK
Specifies the character array where nn is the maximum number of sockets in the array. The first character in the array represents socket 0, the second represents socket 1, and so on. Keep in mind that the index is 1 greater than the socket number. That is, CHAR-ENTRY(1) represents socket 0, CHAR-ENTRY(2) represents socket 1, and so on.

BIT-MASK
Specifies the bit string to be translated for the SELECT call. Within each fullword of the bit string, the bits are ordered right to left. The rightmost bit in the first fullword represents socket 0 and the leftmost bit represents socket 31. The rightmost bit in the second fullword represents socket 32 and the leftmost bit represents socket 63. The number of fullwords in the bit string should be calculated by dividing the sum of 31 and the character array length by 32 (truncate the remainder).

COMMAND
BTOC—Specifies bit string to character array translation.
CTOB—Specifies character array to bit string translation.

CHAR-MASK-LENGTH
Specifies the length of the character array. This field should be no greater than 1 plus the MAXSNO value returned on the INITAPI (which is usually the same as the MAXSOC value specified on the INITAPI).

RETCODE
A binary field that returns one of the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>

Figure 169. EZACIC06 call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 253.
Successful call

Check ERRNO for an error code

Examples

If you want to use the SELECT call to test sockets 0, 5, and 32, and you are using a character array to represent the sockets, you must set the appropriate characters in the character array to 1. In the following example, index position 1, 6, and 33 in the character array are set to 1. Then you can call EZACIC06 with the COMMAND parameter set to CTOB.

When EZACIC06 returns, the first fullword of BIT-MASK contains B'00000000000000000000000000100001' to indicate that sockets 0 and 5 are checked. The second word of BIT-MASK contains B'00000000000000000000000000000001' to indicate that socket 32 is checked. These instructions process the bit string shown in the following example:

MOVE ZEROS TO CHAR-STRING.
MOVE '1' TO CHAR-ENTRY(1), CHAR-ENTRY(6), CHAR-ENTRY(33).
CALL 'EZACIC06' USING TOKEN CTOB BIT-MASK CH-MASK
CHAR-MASK-LENGTH RETCODE.
MOVE BIT-MASK TO ....

When the select call returns and you want to check the bit-mask string for socket activity, enter the following instructions.

MOVE ..... TO BIT-MASK.
CALL 'EZACIC06' USING TOKEN BTOC BIT-MASK CH-MASK
CHAR-MASK-LENGTH RETCODE.
PERFORM TEST-SOCKET THRU TEST-SOCKET-EXIT VARYING IDX FROM 1 BY 1 UNTIL IDX EQUAL CHAR-MASK-LENGTH.
TEST-SOCKET.
IF CHAR-ENTRY(IDX) EQUAL '1'
THEN PERFORM SOCKET-RESPONSE THRU
SOCKET-RESPONSE-EXIT
ELSE NEXT SENTENCE.
TEST-SOCKET-EXIT.
EXIT.

EZACIC08 program

The GETHOSTBYNAME and GETHOSTBYADDR calls were derived from C socket calls that return a structure known as HOSTENT. A given TCP/IP stacks host can have multiple alias names and host Internet addresses.

TCP/IP stacks uses indirect addressing to connect the variable number of alias names and Internet addresses in the HOSTENT structure that is returned by the GETHOSTBYADDR AND GETHOSTBYNAME calls.

If you are coding in PL/I or Assembler language, the HOSTENT structure can be processed in a relatively straightforward manner. However, if you are coding in COBOL, HOSTENT can be more difficult to process and you should use the EZACIC08 subroutine to process it for you.

It works as follows:

- GETHOSTBYADDR or GETHOSTBYNAME returns a HOSTENT structure that indirectly addresses the lists of alias names and Internet addresses.
Upon return from GETHOSTBYADDR or GETHOSTBYNAME your program calls EZACIC08 and passes it the address of the HOSTENT structure. EZACIC08 processes the structure and returns the following:

1. The length of host name, if present
2. The host name
3. The number of alias names for the host
4. The alias name sequence number
5. The length of the alias name
6. The alias name
7. The host Internet address type, always 2 for AF_INET
8. The host Internet address length, always 4 for AF_INET
9. The number of host Internet addresses for this host
10. The host Internet address sequence number
11. The host Internet address

If the GETHOSTBYADDR or GETHOSTBYNAME call returns more than one alias name or host Internet address (steps 3 and 9 in this topic), the application program should repeat the call to EZACIC08 until all alias names and host Internet addresses have been retrieved.

Figure 170 shows an example of EZACIC08 call instructions.

WORKING-STORAGE SECTION.

01 HOSTENT-ADDR PIC 9(8) BINARY.
01 HOSTNAME-LENGTH PIC 9(4) BINARY.
01 HOSTNAME-VALUE PIC X(255).
01 HOSTALIAS-COUNT PIC 9(4) BINARY.
01 HOSTALIAS-SEQ PIC 9(4) BINARY.
01 HOSTALIAS-LENGTH PIC 9(4) BINARY.
01 HOSTALIAS-VALUE PIC X(255).
01 HOSTADDR-TYPE PIC 9(4) BINARY.
01 HOSTADDR-LENGTH PIC 9(4) BINARY.
01 HOSTADDR-COUNT PIC 9(4) BINARY.
01 HOSTADDR-SEQ PIC 9(4) BINARY.
01 HOSTADDR-VALUE PIC 9(8) BINARY.
01 RETURN-CODE PIC 9(8) BINARY.

PROCEDURE DIVISION.

CALL 'EZASOKET' USING 'GETHOSTBYADDR' HOSTADDR HOSTENT-ADDR RETCODE.

CALL 'EZASOKET' USING 'GETHOSTBYNAME' NAMELEN NAME HOSTENT-ADDR RETCODE.

CALL 'EZACIC08' USING HOSTENT-ADDR HOSTNAME-LENGTH HOSTNAME-VALUE HOSTALIAS-COUNT HOSTALIAS-SEQ HOSTALIAS-LENGTH HOSTALIAS-VALUE HOSTADDR-TYPE HOSTADDR-LENGTH HOSTADDR-COUNT HOSTADDR-SEQ HOSTADDR-VALUE RETURN-CODE

Figure 170. EZACIC08 call instruction example

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 253.

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 one to the current value of HOSTALIAS-SEQ and uses the resulting value to index into the table of alias names. Therefore, for a given instance of GETHOSTBYxxxx, this field should be set to 0 for the initial call to EZACIC08. For all subsequent calls to EZACIC08, this field should contain the HOSTALIAS-SEQ number returned by the previous invocation.

HOSTADDR-SEQ
This halfword field is used by EZACIC08 to index the list of IP addresses. When EZACIC08 is called, it adds one to the current value of HOSTADDR-SEQ and uses the resulting value to index into the table of IP addresses. Therefore, for a given instance of GETHOSTBYxxxx, this field should be set to 0 for the initial call to EZACIC08. For all subsequent calls to EZACIC08, this field should contain the HOSTADDR-SEQ number returned by the previous call.

Parameter values returned to the application

HOSTNAME-LENGTH
This halfword binary field contains the length of the host name (if host name was returned).

HOSTNAME-VALUE
This 255-byte character string contains the host name (if host name was returned).

HOSTALIAS-COUNT
This halfword binary field contains the number of alias names returned.

HOSTALIAS-SEQ
This halfword binary field is the sequence number of the alias name currently found in HOSTALIAS-VALUE.

HOSTALIAS-LENGTH
This halfword binary field contains the length of the alias name currently found in HOSTALIAS-VALUE.

HOSTALIAS-VALUE
This 255-byte character string contains the alias name returned by this instance of the call. The length of the alias name is contained in HOSTALIAS-LENGTH.

HOSTADDR-TYPE
This halfword binary field contains the type of host address. For FAMILY type AF_INET, HOSTADDR-TYPE is always 2.

HOSTADDR-LENGTH
This halfword binary field contains the length of the host Internet address currently found in HOSTADDR-VALUE. For FAMILY type AF_INET, HOSTADDR-LENGTH is always set to 4.
HOSTADDR-COUNT
This halfword binary field contains the number of host Internet addresses returned by this instance of the call.

HOSTADDR-SEQ
This halfword binary field contains the sequence number of the host Internet address currently found in HOSTADDR-VALUE.

HOSTADDR-VALUE
This fullword binary field contains a host Internet address.

RETURN-CODE
This fullword binary field contains the EZACIC08 return code:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion</td>
</tr>
<tr>
<td>-1</td>
<td>Invalid HOSTENT address</td>
</tr>
<tr>
<td>-2</td>
<td>Invalid HOSTALIAS-SEQ value</td>
</tr>
<tr>
<td>-3</td>
<td>Invalid HOSTADDR-SEQ value</td>
</tr>
</tbody>
</table>

EZACIC09 program
The GETADDRINFO call was derived from the C socket call that returns a structure known as RES. A given TCP/IP stacks host can have multiple sets of NAMES. TCP/IP stacks uses indirect addressing to connect the variable number of NAMES in the RES structure that the GETADDRINFO call returns. If you are coding in PL/1 or Assembler language, the RES structure can be processed in a relatively straightforward manner. However, if you are coding in COBOL, RES can be more difficult to process and you should use the EZACIC09 subroutine to process it for you. It works as follows:

- GETADDRINFO returns a RES structure that indirectly addresses the lists of socket address structures.
- Upon return from GETADDRINFO, your program calls EZACIC09 and passes it the address of the next address information structure as referenced by the NEXT argument. EZACIC09 processes the structure and returns the following:
  1. The socket address structure
  2. The next address information structure
- If the GETADDRINFO call returns more than one socket address structure, the application program should repeat the call to EZACIC09 until all socket address structures have been retrieved.

Figure 171 on page 414 shows an example of EZACIC09 call instructions.
WORKING-STORAGE SECTION.
*
* Variables used for the GETADDRINFO call
*
01 getaddrinfo-parms.
  02 node-name pic x(255).
  02 node-name-len pic 9(8) binary.
  02 service-name pic x(32).
  02 service-name-len pic 9(8) binary.
  02 canonical-name-len pic 9(8) binary.
  02 ai-passive pic 9(8) binary value 1.
  02 ai-canonnameok pic 9(8) binary value 2.
  02 ai-numerichost pic 9(8) binary value 4.
  02 ai-numericerv pic 9(8) binary value 8.
  02 ai-v4mapped pic 9(8) binary value 16.
  02 ai-all pic 9(8) binary value 32.
  02 ai-addrcconfig pic 9(8) binary value 64.
*
* Variables used for the EZACIC09 call
*
01 ezacic09-parms.
  02 res usage is pointer.
  02 res-name-len pic 9(8) binary.
  02 res-canonical-name pic x(256).
  02 res-name usage is pointer.
  02 res-next-addrinfo usage is pointer.
*
* Socket address structure
*
01 server-socket-address.
  05 server-family pic 9(4) Binary Value 19.
  05 server-port pic 9(4) Binary Value 9997.
  05 server-flowinfo pic 9(8) Binary Value 0.
  05 server-ipaddr.
    10 filler pic 9(16) binary value 0.
    10 filler pic 9(16) binary value 0.
  05 server-scopeid pic 9(8) Binary Value 0.
LINKAGE SECTION.

01 L1.
  03 HINTS-ADDRINFO.
    05 HINTS-AI-FLAGS PIC 9(8) BINARY.
    05 HINTS-AI-FAMILY PIC 9(8) BINARY.
    05 HINTS-AI-SOCKTYPE PIC 9(8) BINARY.
    05 HINTS-AI-PROTOCOL PIC 9(8) BINARY.
    05 FILLER PIC 9(8) BINARY.
    05 FILLER PIC 9(8) BINARY.
    05 FILLER PIC 9(8) BINARY.
    05 FILLER PIC 9(8) BINARY.
  03 HINTS-ADDRINFO-PTR USAGE IS POINTER.
  03 RES-ADDRINFO-PTR USAGE IS POINTER.
*
* RESULTS ADDRESS INFO
*
01 RESULTS-ADDRINFO.
  05 RESULTS-AI-FLAGS PIC 9(8) BINARY.
  05 RESULTS-AI-FAMILY PIC 9(8) BINARY.
  05 RESULTS-AI-SOCKTYPE PIC 9(8) BINARY.
  05 RESULTS-AI-PROTOCOL PIC 9(8) BINARY.
  05 RESULTS-AI-ADDR-LEN PIC 9(8) BINARY.
  05 RESULTS-AI-CANONICAL-NAME USAGE IS POINTER.
  05 RESULTS-AI-ADDR-PTR USAGE IS POINTER.
  05 RESULTS-AI-NEXT-PTR USAGE IS POINTER.

Figure 171. EZACIC09 call instruction example (Part 1 of 2)
* SOCKET ADDRESS STRUCTURE FROM EZACIC09.
*
01 OUTPUT-NAME-PTR  USAGE IS POINTER.
01 OUTPUT-IP-NAME.
   03 OUTPUT-IP-FAMILY PIC 9(4) BINARY.
   03 OUTPUT-IP-PORT PIC 9(4) BINARY.
   03 OUTPUT-IP-SOCK-DATA PIC X(24).
   03 OUTPUT-IPV4-SOCK-DATA REDEFINES OUTPUT-IP-SOCK-DATA.
      05 OUTPUT-IPV4-IPADDR PIC 9(8) BINARY.
      05 OUTPUT-IPV4-IPADDR.
      10 FILLER PIC 9(16) BINARY.
      10 FILLER PIC 9(16) BINARY.
   05 OUTPUT-IPV6-SOCK-DATA REDEFINES OUTPUT-IP-SOCK-DATA.
      05 OUTPUT-IPV6-SCOPEID PIC 9(8) BINARY.
      05 OUTPUT-IPV6-SCOPEID.
PROCEDURE DIVISION USING L1.
*
* Get an address from the resolver.
* move 'yournodename' to node-name.
  move 12 to node-name-len.
  move spaces to service-name.
  move 0 to service-name-len.
  move af-inet6 to hints-ai-family.
  move 49 to hints-ai-flags
  move 0 to hints-ai-socktype.
  move 0 to hints-ai-protocol.
  set address of results-addrinfo to res-addrinfo-ptr.
  set hints-addrinfo-ptr to address of hints-addrinfo.
  call 'EZASOKET' using soket-getaddrinfo
     node-name node-name-len
     service-name service-name-len
     hints-addrinfo-ptr
     res-addrinfo-ptr
     canonical-name-len
     errno retcode.
*
* Use EZACIC09 to extract the IP address
*
  set address of results-addrinfo to res-addrinfo-ptr.
  set res to address of results-addrinfo.
  move zeros to res-name-len.
  move spaces to res-canonical-name.
  set res-name to nulls.
  set res-next-addrinfo to nulls.
  call 'EZACIC09' using res
     res-name-len
     res-canonical-name
     res-name
     res-next-addrinfo
     retcode.
  set address of output-ip-name to res-name.
  move output-ipv6-ipaddr to server-ipv6.

Figure 172. EZACIC09 call instruction example (Part 2 of 2)

For equivalent PL/I and assembler language declarations, see "Converting
parameter descriptions" on page 253.

Parameter values set by the application
RES  This fullword binary field must contain the address of the ADDRINFO structure (as returned by the GETADDRINFO call). This variable is the same as the RES variable in the GETADDRINFO socket call.

RES-NAME-LEN
A fullword binary field that contains the length of the socket address structure as returned by the GETADDRINFO call.

Parameter values returned to the application

RES-CANONICAL-NAME
A field large enough to hold the canonical name. The maximum field size is 256 bytes. The canonical name length field indicates the length of the canonical name as returned by the GETADDRINFO call.

RES-NAME
The address of the subsequent socket address structure.

RES-NEXT
The address of the next address information structure.

RETURN-CODE
This fullword binary field contains the EZACIC09 return code:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Successful completion</td>
</tr>
<tr>
<td>-1</td>
<td>Invalid HOSTENT address</td>
</tr>
</tbody>
</table>

EZACIC14 program

The EZACIC14 program is an alternative to EZACIC04, which is used to translate EBCDIC data to ASCII data.

Figure 173 on page 417 shows an example of how EZACIC14 translates a byte of EBCDIC data.
<table>
<thead>
<tr>
<th>ASCII output by EZACIC14</th>
<th>second hex digit of byte of EBCDIC data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>90</td>
</tr>
<tr>
<td>4</td>
<td>00</td>
</tr>
<tr>
<td>5</td>
<td>0B</td>
</tr>
<tr>
<td>A</td>
<td>B0</td>
</tr>
<tr>
<td>B</td>
<td>C0</td>
</tr>
<tr>
<td>C</td>
<td>D0</td>
</tr>
<tr>
<td>D</td>
<td>E0</td>
</tr>
<tr>
<td>E</td>
<td>F0</td>
</tr>
<tr>
<td>F</td>
<td>00</td>
</tr>
</tbody>
</table>

Figure 173. EZACIC14 EBCDIC-to-ASCII table

Figure 174 on page 418 shows an example of EZACIC14 call instructions.
WORKING-_STORAGE SECTION.
  01 OUT-BUFFER PIC X(length of output).
  01 LENGTH PIC 9(8) BINARY.

PROCEDURE DIVISION.
  CALL 'EZACIC14' USING OUT-BUFFER LENGTH.
  IF RETURN-CODE > 0
   THEN
      DISPLAY 'TRANSLATION FAILED ' RETURN-CODE.

Figure 174. EZACIC14 call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 253.

OUT-BUFFER
   A buffer that contains the following:
   • When called – EBCDIC data
   • Upon return – ASCII data

LENGTH
   Specifies the length of the data to be translated.

EZACIC15 program

The EZACIC15 program is an alternative to EZACIC05 which is used to translate ASCII data to EBCDIC data.

Figure 175 on page 419 shows an example of how EZACIC15 translates a byte of ASCII data.
<table>
<thead>
<tr>
<th>EBCDIC output by EZACIC15</th>
<th>second hex digit of byte of ASCII data</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>F0</td>
</tr>
<tr>
<td>4</td>
<td>7C</td>
</tr>
<tr>
<td>5</td>
<td>7D</td>
</tr>
<tr>
<td>6</td>
<td>79</td>
</tr>
<tr>
<td>7</td>
<td>97</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
</tr>
<tr>
<td>9</td>
<td>30</td>
</tr>
<tr>
<td>A</td>
<td>41</td>
</tr>
<tr>
<td>B</td>
<td>90</td>
</tr>
<tr>
<td>C</td>
<td>64</td>
</tr>
<tr>
<td>D</td>
<td>AC</td>
</tr>
<tr>
<td>E</td>
<td>44</td>
</tr>
<tr>
<td>F</td>
<td>8C</td>
</tr>
</tbody>
</table>

Figure 175. EZACIC15 ASCII-to-EBCDIC table

Figure 176 on page 420 shows an example of EZACIC15 call instructions.
WORKING-STORAGE SECTION.
  01 OUT-BUFFER PIC X(length of output).
  01 LENGTH PIC 9(8) BINARY.

PROCEDURE DIVISION.
  CALL 'EZACIC15' USING OUT-BUFFER LENGTH.
  IF RETURN-CODE > 0
    THEN
      DISPLAY 'TRANSLATION FAILED ' RETURN-CODE.

Figure 176. EZACIC15 call instruction example

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 253.

OUT-BUFFER
  A buffer that contains the following:
  • When called – ASCII data
  • Upon return – EBCDIC data

LENGTH
  Specifies the length of the data to be translated.
Appendix A. Original COBOL application programming interface (EZACICAL)

The EZACICAL does not formally support IPv6 and it is not a recommended API.

This topic describes the first COBOL API provided with TCP/IP Version 2.2.1 for MVS. It is referred to as the EZACICAL API to distinguish it from the Sockets Extended API. (EZACICAL is the routine that is called for this API.)

It gives the format of each socket call and describes the call parameters. It starts with guidance on compiling COBOL programs.

Using the EZACICAL or Sockets Extended API

The EZACICAL API (described in this topic) and the Sockets Extended API (described in Chapter 8, “Sockets extended API,” on page 249) 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 177 on page 422. 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
Figure 177. Modified JCL for COBOL compilation
The EZACICAL API

The EZACICAL API can be used by assembler language, COBOL, or PL/I programs and is invoked by calling the EZACICAL routine. Although the calls to this routine perform the same function as the C language calls described in Chapter 7, “C language application programming,” on page 165, 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.

**EZACICAL call format for COBOL**

The following is the EZACICAL call format for COBOL:

```plaintext
CALL 'EZACICAL' USING TOKEN COMMAND parm1, parm2, ... ERRNO RETCODE.
```

**TOKEN**

A 16-character field with the value 'TCPIIUCVSTREAMS'

**COMMAND**

A halfword binary value from 1 to 32, identifying the socket call.

**parm**

The parameters particular to each socket call. For example, BIND, described in “COBOL call for BIND” on page 426, has two such parameters: S (socket), which is a halfword binary value, 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.

**EZACICAL call format for PL/I**

The following is the EZACICAL call format for PL/I:

```plaintext
CALL 'EZACICAL' USING TOKEN COMMAND parm1, parm2, ... ERRNO RETCODE.
```

**TOKEN**

A 16-character field with the value 'TCPIIUCVSTREAMS'

**COMMAND**

A halfword binary value from 1 to 32, identifying the socket call.

**parm**

The parameters particular to each socket call. For example, BIND, described in “COBOL call for BIND” on page 426, has two such parameters: S (socket), which is a halfword binary value, 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.
CALL EZACICAL (TOKEN COMMAND parm1, parm2, ... ERRNO RETCODE);

TOKEN
A 16-character field with the value 'TCPIPIUCVSTREAMS'

COMMAND
A halfword binary value from 1 to 32, identifying the socket call.

parm
The parameters particular to each socket call. For example, BIND, described in "COBOL call for BIND" on page 426, has two such parameters: S (socket), which is a halfword binary value, 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.

EZACICAL call format for assembler language

The following is the EZACICAL call format for assembler language:

CALL EZACICAL (TOKEN,CMD, parm1, parm2, ... ERRNO RETCODE),VL

The parameter descriptions in this topic are written using the COBOL language syntax and conventions. For assembler language, use the following conversions:

COBOL PIC

<table>
<thead>
<tr>
<th>PIC</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S9(4) COMP</td>
<td>HALFWORD BINARY VALUE</td>
</tr>
<tr>
<td>S9(8) COMP</td>
<td>FULLWORD BINARY VALUE</td>
</tr>
<tr>
<td>X(n)</td>
<td>CHARACTER FIELD OF n BYTES</td>
</tr>
</tbody>
</table>

ASSEMBLER DECLARATION

<table>
<thead>
<tr>
<th>DS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>HALFWORD BINARY VALUE</td>
</tr>
<tr>
<td>F</td>
<td>FULLWORD BINARY VALUE</td>
</tr>
<tr>
<td>CLn</td>
<td>CHARACTER FIELD OF n BYTES</td>
</tr>
</tbody>
</table>

COBOL and assembler language socket calls

The remainder of this topic describes the EZACICAL API call formats.

The descriptions assume you are using VS COBOL II. If you are using an earlier version, the picture clauses should read COMP rather than BINARY.

The following abbreviations are used:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>Halfword</td>
</tr>
<tr>
<td>F</td>
<td>Fullword</td>
</tr>
<tr>
<td>D</td>
<td>Doubleword</td>
</tr>
<tr>
<td>CLn</td>
<td>Character format, length n bytes</td>
</tr>
</tbody>
</table>
COBOL call for ACCEPT

This call functions in the same way as the equivalent call described on page 254. 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 "EZACICAL call format for assembler language" on page 424).

Parameter lengths in assembler language and COBOL for ACCEPT

<table>
<thead>
<tr>
<th>Assembler language</th>
<th>Assembler language</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN</td>
<td>CL16</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>S</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>ZERO-FWRD</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>NEW-S</td>
<td>F</td>
<td>PIC S9(8) BINARY</td>
</tr>
</tbody>
</table>

NAME STRUCTURE:
- **Internet Family**: H PIC 9(4) BINARY
- **Port**: H PIC 9(4) BINARY
- **Internet Address**: F PIC 9(8) BINARY
- **Zeros** : XL8 PIC X(8)
- **ERRNO**: F PIC 9(8) BINARY
- **RETCODE**: F PIC S9(8) BINARY

Parameter values to be set by the application for ACCEPT

**TOKEN**
- Must be set to 'TCPIPIUCVSTREAMS'

**COMMAND**
- Must be set to 1 for the ACCEPT command

**S**
- The descriptor of the local socket on which the connection is accepted

**ZERO-FWRD**
- Set to zeros

**NEW-S**
- Set to -1. The system returns the socket number in the RETCODE field.

**Note**: Be sure to use only the socket number returned by the system.

Parameter values returned to the application for ACCEPT

**NAME**
- Structure giving the name of the port to which the new socket is connected
  - **Internet Family**: AF-INET is always returned
  - **Port**: The port address of the new socket
  - **Internet Address**: The IP address of the new socket
**Zeros**  
Set to binary zeros or LOW VALUES

**ERRNO**  
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 455.

**RETCODE**  
The socket number for new socket is returned. A RETCODE of -1 indicates an error.

**COBOL call for BIND**

This call functions in the same way as the equivalent call described in “BIND call” on page 257. 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 “EZACICAL call format for assembler language” on page 424).

**Parameter lengths in assembler language and COBOL for BIND**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Assembler language</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN</td>
<td>CL16</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>S</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>NAME STRUCTURE:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internet Family</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>Port</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>Internet Address</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>Zeros</td>
<td>XL8</td>
<td>PIC X(8)</td>
</tr>
<tr>
<td>ERRNO</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>RETCODE</td>
<td>F</td>
<td>PIC S9(8) BINARY</td>
</tr>
</tbody>
</table>

**Parameter values to be set by the application for BIND**

**TOKEN**  
Must be set to 'TCPIIUCVSTREAMS'

**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 for BIND

NAME (Port)
If Port was set to 0, the system returns an available port.

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 455.

RETCODE
A return of 0 indicates a successful call. A return of -1 indicates an error.

COBOL call for CLOSE

This call functions in the same way as the equivalent call described in “CLOSE call” on page 262. 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 “EZACICAL call format for assembler language” on page 424).

Parameter lengths in assembler language and COBOL for CLOSE

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Assembler language</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN</td>
<td>CL16</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>S</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>DZERO</td>
<td>D</td>
<td>PIC X(8)</td>
</tr>
<tr>
<td>ERRNO</td>
<td>F</td>
<td>PIC S9(8) BINARY</td>
</tr>
<tr>
<td>RETCODE</td>
<td>F</td>
<td>PIC S9(8) BINARY</td>
</tr>
</tbody>
</table>

Parameter values to be set by the application for CLOSE

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
Must be set to 3 for the CLOSE command

S
The descriptor of the socket to be closed

DZERO
Set to binary zeros or low values

Parameter values returned to the application for CLOSE

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 455.

RETCODE
A return of 0 indicates a successful call. A return of -1 indicates an error.
COBOL call for CONNECT

This call functions in the same way as the equivalent call described in "CONNECT call" on page 264. The format of the COBOL call for the CONNECT function is:

```cobol
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 "EZACICAL call format for assembler language" on page 424).

### Parameter lengths in assembler language and COBOL for CONNECT

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Assembler language</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN</td>
<td>CL16</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>S</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>NAME STRUCTURE:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internet Family</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>Port</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>Internet Address</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>Zeros</td>
<td>XL8</td>
<td>PIC X(8)</td>
</tr>
<tr>
<td>ERRNO</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>RETCODE</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
</tbody>
</table>

### Parameter values to be set by the application for CONNECT

**TOKEN**  
Must be set to 'TCPIPIUCVSTREAMS'

**COMMAND**  
Must be set to 4 for the CONNECT command

**S**  
The descriptor of the local socket to be used to establish a connection

**NAME**  
Structure giving the name of the port to which the socket is to be connected, consisting of:

- **Internet Family**  
  Must be set to 2 (AF-INET)

- **Port**  
The remote port number to which the socket is to be connected

- **Internet Address**  
The remote IP address to which the socket is to be connected

- **Zeros**  
Set to binary zeros or low values

### Parameter values returned to the application for CONNECT

**ERRNO**  
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 455.

**RETCODE**  
A return of 0 indicates a successful call. A return of -1 indicates an error.
COBOL call for FCNTL

This call functions in the same way as the equivalent call described in "FCNTL call" on page 268. 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 "EZACICAL call format for assembler language" on page 424).

Parameter lengths in assembler language and COBOL for FCNTL

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Assembler language</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN</td>
<td>CL16</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>S</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>CMD</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>ARG</td>
<td>F</td>
<td>PIC 9(8)</td>
</tr>
<tr>
<td>ERRNO</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>RETCODE</td>
<td>F</td>
<td>PIC S9(8) BINARY</td>
</tr>
</tbody>
</table>

Parameter values to be set by the application for FCNTL

TOKEN
Must be set to 'TCPIIUCVSTREAMS'

COMMAND
Must be set to 5 for the FCNTL command

S
The socket descriptor whose FNDELAY flag is to be set or queried

CMD
Set a value of 3 to query the FNDELAY flag of socket s. This is equivalent to setting the cmd parameter to F-GETFL in the fcntl() C call.

Set a value of 4 to set the FNDELAY flag of socket s. This is equivalent to setting the cmd parameter to F-SETFL in the fcntl() C call.

ARG
If CMD is set to 4, setting ARG to 4 sets the FNDELAY flag; setting ARG to 3 resets the FNDELAY flag.

Parameter values returned to the application for FCNTL

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 455.

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.

COBOL call for GETCLIENTID

This call functions in the same way as the equivalent call described in "GETCLIENTID call" on page 280. 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 “EZACICAL call format for assembler language” on page 424).

Parameter lengths in assembler language and COBOL for GETCLIENTID

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Assembler language</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN</td>
<td>CL16</td>
<td>TOKEN CL16 PIC X(16)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>H</td>
<td>COMMAND H PIC 9(4) BINARY</td>
</tr>
<tr>
<td>HZERO</td>
<td>H</td>
<td>HZERO H PIC 9(4) BINARY</td>
</tr>
<tr>
<td>DZERO</td>
<td>D</td>
<td>DZERO D PIC X(8)</td>
</tr>
<tr>
<td>CLIENTID STRUCTURE:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Domain</td>
<td>F</td>
<td>Domain F PIC 9(8) BINARY</td>
</tr>
<tr>
<td>Name</td>
<td>CL8</td>
<td>Name CL8 PIC X(8)</td>
</tr>
<tr>
<td>Task</td>
<td>CL8</td>
<td>Task CL8 PIC X(8)</td>
</tr>
<tr>
<td>Reserved</td>
<td>XL20</td>
<td>Reserved XL20 PIC X(20)</td>
</tr>
<tr>
<td>ERRNO</td>
<td>F</td>
<td>ERRNO F PIC 9(8) BINARY</td>
</tr>
<tr>
<td>RETCODE</td>
<td>F</td>
<td>RETCODE F PIC S9(8) BINARY</td>
</tr>
</tbody>
</table>

Parameter values to be set by the application for GETCLIENTID

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
Must be set to 30 for the GETCLIENTID command

HZERO
Set to binary zeros or LOW VALUES

DZERO
Set to binary zeros or LOW VALUES

CLIENTID
Domain
Must be set to 2 (AF-INET)

Parameter values returned to the application for GETCLIENTID

CLIENTID
Structure identifying the client as follows:

Name
Address space identification is returned

Task
Task identification is returned

Reserved
Zeros or LOW VALUES are returned

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 455.

RETCODE
A return of 0 indicates a successful call. A return of -1 indicates an error.
COBOL call for GETHOSTID

This call functions in the same way as the equivalent call described in "GETHOSTBYADDR call" on page 282. 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 "EZACICAL call format for assembler language" on page 424).

Parameter lengths in assembler language and COBOL for GETHOSTID

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Assembler language</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN</td>
<td>CL16</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>HZERO</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>DZERO</td>
<td>D</td>
<td>PIC X(8)</td>
</tr>
<tr>
<td>ERRNO</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>RETCODE</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
</tbody>
</table>

Parameter values to be set by the application for GETHOSTID

TOKEN

Must be set to 'TCPIPIUCVSTREAMS'

COMMAND

Must be set to 7 for the GETHOSTID command

HZERO

Set to binary zeros or low values

DZERO

Set to binary zeros or low values

Parameter values returned to the application for GETHOSTID

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.

COBOL call for GETHOSTNAME

This call functions in the same way as the equivalent call described in "GETHOSTBYNAME call" on page 285.

Note: The host name returned is the host name the TCPIP stack learned at startup from the TCPIP.DATA file that was found. For more information about hostname, see HOSTNAME statement in z/OS Communications Server: IP Configuration Reference.
The format of the COBOL call for the GETHOSTNAME function is:

```
CALL 'EZACICAL' USING TOKEN COMMAND HZERO DZERO NAMELEN NAME ERRNO RETCODE.
```

In assembler language, issue the macro call CALL EZACICAL, using standard assembler call syntax (for the call format, see “EZACICAL call format for assembler language” on page 424).

### Parameter lengths in assembler language and COBOL for GETHOSTNAME

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Assembler language</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN</td>
<td>CL16</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>HZERO</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>DZERO</td>
<td>D</td>
<td>PIC X(8)</td>
</tr>
<tr>
<td>NAMELEN</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>NAME</td>
<td>NAMELEN or larger</td>
<td>NAMELEN or larger</td>
</tr>
<tr>
<td>ERRNO</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>RETCODE</td>
<td>F</td>
<td>PIC 99(8) BINARY</td>
</tr>
</tbody>
</table>

### Parameter values to be set by the application for GETHOSTNAME

**TOKEN**

Must be set to 'TCPIPIUCVSTREAMS'

**COMMAND**

Must be set to 8 for the GETHOSTNAME command

**HZERO**

Set to 0

**DZERO**

Set to binary zeros or low values

**NAMELEN**

The length of the NAME field. The minimum length of the NAME field is 1 character. The maximum length of the NAME field is 255 characters.

### Parameter values returned to the application for GETHOSTNAME

**NAME**

The host name returned from the call. If the host name is shorter than the NAMELEN value, then the NAME field is filled with binary zeros after the host name. If the host name is longer than the NAMELEN value, then the name is truncated.

**ERRNO**

If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 455.

**RETCODE**

A return of 0 indicates a successful call. A return of -1 indicates an error.
COBOL call for GETPEERNAME

This call functions in the same way as the equivalent call described in “GETPEERNAME call” on page 294. 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 “EZACICAL call format for assembler language” on page 424).

Parameter lengths in assembler language and COBOL for GETPEERNAME

<table>
<thead>
<tr>
<th>Parameter</th>
<th>COBOL</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN</td>
<td>CL16 PIC X(16)</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>H PIC 9(4) BINARY</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>S</td>
<td>H PIC 9(4) BINARY</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>DZERO</td>
<td>D PIC X(8)</td>
<td>PIC X(8)</td>
</tr>
<tr>
<td>NAME</td>
<td>CL16 PIC X(16)</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>ERRNO</td>
<td>F PIC 9(8) BINARY</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>RETCODE</td>
<td>F PIC S9(8) BINARY</td>
<td>PIC S9(8) BINARY</td>
</tr>
</tbody>
</table>

Parameter values to be set by the application for GETPEERNAME

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
Must be set to 9 for the GETPEERNAME command

S
The descriptor of the local socket connected to the requested peer

DZERO
Set to binary zeros or low values

Parameter values returned to the application for GETPEERNAME

NAME
The peer name returned from the call

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 455.

RETCODE
A return of 0 indicates a successful call. A return of -1 indicates an error.

COBOL call for GETSOCKNAME

This call functions in the same way as the equivalent call described in “GETSOCKNAME call” on page 296. 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 "EZACICAL call format for assembler language" on page 424).

Parameter lengths in assembler language and COBOL for GETSOCKNAME

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Assembler language</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN</td>
<td>CL16</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>S</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>DZERO</td>
<td>D</td>
<td>PIC X(8)</td>
</tr>
</tbody>
</table>

NAME STRUCTURE:
- Internet Family: H PIC 9(4) BINARY
- Port: H PIC 9(4) BINARY
- Internet Address: F PIC 9(8) BINARY
- Zeros: XL8 PIC X(8)
- ERRNO: F PIC 9(8) BINARY
- RETCODE: F PIC S9(8) BINARY

Parameter values to be set by the application for GETSOCKNAME

TOKEN
Must be set to ’TCPIPIUCVSTREAMS’

COMMAND
Must be set to 10 for the GETSOCKNAME command

S
The descriptor of the local socket whose address is required

DZERO
Set to binary zeros or low values

NAME
Structure giving the name of the port to which the socket is bound, consisting of:
- Internet Family
  Must be set to 2 (AF-INET).
- Port
  The local port address to which the socket is bound
- Internet Address
  The local IP address to which the socket is bound
- Zeros
  Set to binary zeros or low values

Parameter values returned to the application for GETSOCKNAME

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 455.

RETCODE
A return of 0 indicates a successful call. A return of -1 indicates an error.
COBOL call for GETSOCKOPT

This call functions in the same way as the equivalent call described in “GETSOCKOPT call” on page 298. 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 “EZACICAL call format for assembler language” on page 424).

Parameter lengths in assembler language and COBOL for GETSOCKOPT

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Assembler language</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN</td>
<td>CL16</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>S</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>LEVEL</td>
<td>F</td>
<td>PIC X(4)</td>
</tr>
<tr>
<td>OPTNAME</td>
<td>F</td>
<td>PIC X(4)</td>
</tr>
<tr>
<td>OPTLEN</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>OPTVAL</td>
<td>CL4</td>
<td>PIC X(4)</td>
</tr>
<tr>
<td>ERRNO</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>RETCODE</td>
<td>F</td>
<td>PIC S9(8) BINARY</td>
</tr>
</tbody>
</table>

Parameter values to be set by the application for GETSOCKOPT

**TOKEN**

Must be set to 'TCPIPIUCVSTREAMS'

**COMMAND**

Must be set to 11 for the GETSOCKOPT command

**S**

The descriptor of the socket whose option settings are required

**LEVEL**

This must be set to X'0000FFFF'.

**OPTNAME**

Set this field to specify the option to be queried, as shown here. For a description of these options, see “GETSOCKOPT call” on page 298

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'0000004'</td>
<td>SO-REUSEADDR</td>
</tr>
<tr>
<td>X'00000020'</td>
<td>SO-BROADCAST</td>
</tr>
<tr>
<td>X'00001007'</td>
<td>SO-ERROR</td>
</tr>
<tr>
<td>X'00000080'</td>
<td>SO-LINGER</td>
</tr>
</tbody>
</table>
X'00000100'
    SO-OOBINLINE

X'00000101'
    SO-SNDBUF

X'00000108'
    SO-TYPE

X'80000008'
    TCP_KEEPALIVE

X'80000001'
    TCP_NODELAY

Parameter values returned to the application for GETSOCKOPT

OPTLEN
The length of the option data

OPTVAL
The value of the option. For all options except SO-LINGER, an integer
indicates that the option is enabled, while a 0 indicates it is disabled. For
SO-LINGER, the following structure is returned:

ONOFF F PIC X(4)
LINGER F PIC 9(4)

A nonzero value of ONOFF indicates that the option is enabled, and 0, that
it is disabled. The LINGER value indicates the amount of time to linger
after close.

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are
described in Appendix B, "Return codes," on page 455.

RETCODE
A return of 0 indicates a successful call. A return of -1 indicates an error.

COBOL call for GIVESOCKET

This call functions in the same way as the equivalent call described in
"GIVESOCKET call" on page 314. The format of the COBOL call for the
GIVESOCKET function is:
In assembler language, issue the macro call `CALL EZACICAL`, using standard assembler call syntax (for the call format, see "EZACICAL call format for assembler language" on page 424).

Parameter lengths in assembler language and COBOL for `GIVESOCKET`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Assembler language</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN</td>
<td>CL16</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>S</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>CLIENTID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Domain</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>Name</td>
<td>CL8</td>
<td>PIC X(8)</td>
</tr>
<tr>
<td>Task</td>
<td>CL8</td>
<td>PIC X(8)</td>
</tr>
<tr>
<td>Reserved</td>
<td>XL20</td>
<td>PIC X(20)</td>
</tr>
<tr>
<td>ERRNO</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>RETCODE</td>
<td>F</td>
<td>PIC 99(8) BINARY</td>
</tr>
</tbody>
</table>

Parameter values to be set by the application for `GIVESOCKET`

**TOKEN**
- Must be set to 'TCPIPIUCVSTREAMS'

**COMMAND**
- Must be set to 31 for the `GIVESOCKET` command

**S**
- The socket descriptor of the socket to be given

**CLIENTID**
- Structure identifying the client ID of this application, as follows:
  - **Domain**
    - Must be set to 2 (AF-INET)
  - **Name**
    - Set to the address space identifier obtained from `GETCLIENTID`
  - **Task**
    - Set to blanks
  - **Reserved**
    - Set to binary zeros or low values

Parameter values returned to the application for `GIVESOCKET`

**ERRNO**
- If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 455.

**RETCODE**
- A return of 0 indicates a successful call. A return of -1 indicates an error.

COBOL call for `INITAPI`

The format of the COBOL call for the `INITAPI` 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 “EZACICAL call format for assembler language” on page 424).

### Parameter lengths in assembler language and COBOL for INITAPI

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Assembler language</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN</td>
<td>CL16</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>MAX-SOCK</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>API</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>SUBTASK</td>
<td>XL8</td>
<td>PIC X(8)</td>
</tr>
<tr>
<td>FZERO</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>ERRNO</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>RETCODE</td>
<td>F</td>
<td>PIC S9(8) BINARY</td>
</tr>
</tbody>
</table>

### Parameter values to be set by the application for INITAPI

- **TOKEN**
  Must be set to ‘TCPIPIUCVSTREAMS’

- **COMMAND**
  Must be set to 0 for the INITAPI command

- **MAX-SOCK**
  The maximum number of sockets to be supported in this application. This value cannot exceed 65535. The minimum value is 50.

- **API**
  Must be set to 2, indicating use of the sockets API

- **SUBTASK**
  A unique subtask identifier. It should consist of the 7-character CICS task number and any printable character.

  **Note:** Using the letter L as the last character in the subtask parameter causes the tasking mechanism to assume the CICS transaction is a Listener and schedule it using a non-reusable subtask by way of MVS attach processing when OTE=NO. This has no effect when OTE=YES.

- **FZERO**
  Zeros

### Parameter values returned to the application for INITAPI

- **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.
COBOL call for IOCTL

This call functions in the same way as the equivalent call described in “IOCTL call” on page 322. 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 “EZACICAL call format for assembler language” on page 424).

Parameter lengths in assembler language and COBOL for IOCTL

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Assembler language</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN</td>
<td>CL16   PIC X(16)</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>H       PIC 9(4) BINARY</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>S</td>
<td>H       PIC 9(4) BINARY</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>IOCTLCMD</td>
<td>F       PIC 9(8)</td>
<td>PIC 9(8)</td>
</tr>
<tr>
<td>REQARG</td>
<td>var     var</td>
<td>var</td>
</tr>
<tr>
<td>RETARG</td>
<td>var     var</td>
<td>var</td>
</tr>
<tr>
<td>ERRNO</td>
<td>F       PIC S9(8) BINARY</td>
<td>PIC S9(8) BINARY</td>
</tr>
<tr>
<td>RETCODE</td>
<td>F       PIC S9(8) BINARY</td>
<td>PIC S9(8) BINARY</td>
</tr>
</tbody>
</table>

Parameter values to be set by the application for IOCTL

TOKEN
Must be set to 'TCPIIUCVSTREAMS'

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 call” on page 322 for values and descriptions.

REQARG
The request argument associated with the command. See “IOCTL call” on page 322 for a list and description of possible argument values.

Parameter values returned to the application for IOCTL

RETARG
The return argument. See “IOCTL call” on page 322 for a description of the return argument for each command.

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 455.

RETCODE
A return value of 0 indicates a successful call. A return value of -1 indicates an error.
COBOL call for LISTEN

This call functions in the same way as the equivalent call described in "LISTEN call" on page 334. 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 "EZACICAL call format for assembler language" on page 424).

Parameter lengths in assembler language and COBOL for LISTEN

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Assembler language</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN</td>
<td>CL16</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>S</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>FZERO</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>BACKLOG</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>ERRNO</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>RETCODE</td>
<td>F</td>
<td>PIC S9(8) BINARY</td>
</tr>
</tbody>
</table>

Parameter values to be set by the application for LISTEN

**TOKEN**

Must be set to 'TCPIPIUCVSTREAMS'

**COMMAND**

Must be set to 13 for the LISTEN command

**S**

The descriptor of the socket that is going to listen for incoming connection requests

**FZERO**

Set to binary zeros or low values

**BACKLOG**

Set to the number of connection requests to be queued.

*Note:* The BACKLOG value specified on the LISTEN command cannot be greater than the value configured by the SOMAXCONN statement in the stack's TCPIP PROFILE (default=10); no error is returned if a larger backlog is requested. If you want a larger backlog, update the SOMAXCONN statement. See z/OS Communications Server: IP Configuration Reference for details.

Parameter values returned to the application for LISTEN

**ERRNO**

If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, "Return codes," on page 455.

**RETCODE**

A return value of 0 indicates a successful call. A return value of -1 indicates an error.
COBOL call for READ

This call functions in the same way as the equivalent call described in "READ call" on page 340. 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 "EZACICAL call format for assembler language" on page 424).

Parameter lengths in assembler language and COBOL for READ

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Assembler language</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN</td>
<td>CL16</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>S</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>DZERO</td>
<td>D</td>
<td>PIC X(8)</td>
</tr>
<tr>
<td>NBYTE</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>FILLER</td>
<td>CL16</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>BUF</td>
<td>NBYTE or larger</td>
<td>NBYTE or larger</td>
</tr>
<tr>
<td>ERRNO</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>RETCODE</td>
<td>F</td>
<td>PIC S9(8) BINARY</td>
</tr>
</tbody>
</table>

Parameter values to be set by the application for READ

TOKEN
    Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
    Must be set to 14 for the READ command

S
    The descriptor of the socket that is going to read data

DZERO
    Set to binary zeros or low values

NBYTE
    Set to the length of the buffer (maximum 32 767 bytes)

Parameter values returned to the application for READ

FILLER
    Your program should ignore this field.

BUF
    The input buffer.

ERRNO
    If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, "Return codes," on page 455.

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 program” on page 405 for a subroutine that translates ASCII data to EBCDIC.

**COBOL call for RECVFROM**

This call functions in the same way as the equivalent call described in “RECV call” on page 344. The format of the COBOL call for the RECVFROM function is:

```cobol
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 “EZACICAL call format for assembler language” on page 424).

**Parameter lengths in assembler language and COBOL for RECVFROM**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Assembler language</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN</td>
<td>CL16</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>S</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>FZERO</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>FLAGS</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>NBYTE</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>FROM</td>
<td>CL16</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>BUF</td>
<td>NBYTE or larger</td>
<td>NBYTE or larger</td>
</tr>
<tr>
<td>ERRNO</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>RETCODE</td>
<td>F</td>
<td>PIC S9(8) BINARY</td>
</tr>
</tbody>
</table>

**Parameter values to be set by the application for RECVFROM**

**TOKEN**

Must be set to 'TCPIPIUCVSTREAMS'

**COMMAND**

Must be set to 16 for the RECVFROM command

**S**

The descriptor of the socket receiving data

**FZERO**

Set to binary zeros or low values

**FLAGS**

Set to 2 to peek at (read) data, but not destroy it, so that any subsequent RECVFROM calls reads the same data. CICS TCP/IP does not support out-of-band data.

**NBYTE**

Set to the length of the input buffer. This length cannot exceed 32768 bytes.

**Parameter values returned to the application for RECVFROM**

**FROM**

The socket address structure identifying the from address of the data.

**BUF**

The input buffer.
ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 455.

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 program” on page 405 for a subroutine that translates ASCII data to EBCDIC.

COBOL call for SELECT

This call functions in the same way as the equivalent call described in “SELECT call” on page 353. 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 “EZACICAL call format for assembler language” on page 424).

Parameter lengths in assembler language and COBOL for SELECT

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Assembler language</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN</td>
<td>CL16</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>LOM</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>NUM-FDS</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>TIME-SW</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>RD-SW</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>WR-SW</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>EX-SW</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>TIMEOUT STRUCTURE:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Seconds</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>Milliseconds</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>RD-MASK</td>
<td>Length Of Mask*</td>
<td>Length Of Mask*</td>
</tr>
<tr>
<td>WR-MASK</td>
<td>Length of Mask*</td>
<td>Length of Mask*</td>
</tr>
<tr>
<td>EX-MASK</td>
<td>Length of Mask*</td>
<td>Length of Mask*</td>
</tr>
<tr>
<td>DZERO</td>
<td>D</td>
<td>PIC X(8)</td>
</tr>
<tr>
<td>R-R-MASK</td>
<td>Length of Mask*</td>
<td>Length of Mask*</td>
</tr>
<tr>
<td>R-W-MASK</td>
<td>Length of Mask*</td>
<td>Length of Mask*</td>
</tr>
<tr>
<td>R-E-MASK</td>
<td>Length of Mask*</td>
<td>Length of Mask*</td>
</tr>
<tr>
<td>ERRNO</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>RETCODE</td>
<td>F</td>
<td>PIC S9(8) BINARY</td>
</tr>
</tbody>
</table>

How to calculate Length of Mask (LOM)
1. LOM = ((NUM-FDS + 31)/32) * 4, using integer arithmetic.
2. So, for NUM-FDS ≤ 32, LOM = 4 bytes.
3. For \(33 \leq \text{NUM-FDS} \leq 64\), \(\text{LOM} = 8\) bytes, and so on.

**Parameter values to be set by the application for SELECT**

**TOKEN**
- Must be set to 'TCPIIUCVSTREAMS'

**COMMAND**
- Must be set to 19 for the SELECT command

**LOM**
- Set to the length of mask. The calculation method is given in "How to calculate Length of Mask (LOM)" on page 443.

**NUM-FDS**
- The number of socket descriptors to check. For efficiency, it should be set to the largest number of socket descriptors plus 1.

**TIME-SW**
- Set to 0 to specify a wait forever on socket descriptor activity. Set to 1 to specify a timeout value; this blocks the call until the timeout value is exceeded or until there is socket activity.

**RD-SW**
- Set either 0 (do not check for read interrupts) or 1 (check for read interrupts).

**WR-SW**
- Set either 0 (do not check for write interrupts) or 1 (check for write interrupts).

**EX-SW**
- Set either 0 (do not check for exception interrupts) or 1 (check for exception interrupts).

**TIMEOUT**
- Use this structure to set the timeout value if no activity is detected. Setting this structure to (0,0) indicates that SELECT should act as a polling function; that is, as nonblocking.

- **Seconds**
  - Set to the seconds component of the timeout value.

- **Milliseconds**
  - Set to the milliseconds component of the timeout value (in the range 0 - 999).

**RD-MASK**
- Set the bit mask array for reads. See z/OS Communications Server: IP Programmer's Guide and Reference for more information.

**WR-MASK**
- Set the bit mask array for writes. See z/OS Communications Server: IP Programmer's Guide and Reference for more information.

**EX-MASK**
- Set the bit mask array for exceptions. See z/OS Communications Server: IP Programmer's Guide and Reference for more information.

**DZERO**
- Set to binary zeros or low values.
Parameter values returned to the application for SELECT

R-R-MASK

R-W-MASK

R-E-MASK

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 455.

RETCODE
A positive value indicates the total number of ready sockets in all bit masks. A value of 0 indicates an expired time limit. A value of -1 indicates an error.

COBOL call for SEND

This call functions in the same way as the equivalent call described in “SEND call” on page 364. 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 “EZACICAL call format for assembler language” on page 424).

Parameter lengths in assembler language and COBOL for SEND

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Assembler language</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN</td>
<td>CL16</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>S</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>NBYTE</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>FLAGS</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>DZERO</td>
<td>D</td>
<td>PIC X(8)</td>
</tr>
<tr>
<td>BUF</td>
<td>NBYTE or larger</td>
<td>NBYTE or larger</td>
</tr>
<tr>
<td>ERRNO</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>RETCODE</td>
<td>F</td>
<td>PIC 99(8) BINARY</td>
</tr>
</tbody>
</table>

Parameter values to be set by the application for SEND

TOKEN
Must be set to 'TCPIIUCVSTREAMS'

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 32768 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 for SEND

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 455.

RETCODE
A value of -1 indicates an error. Other values have no meaning.

See “EZACIC04 program” on page 404 for a subroutine that translates EBCDIC data to ASCII.

COBOL call for SENDTO

This call functions in the same way as the equivalent call described in “SENDTO call” on page 371. 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 “EZACICAL call format for assembler language” on page 424).

Parameter lengths in assembler language and COBOL for SENDTO

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Assembler language</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN</td>
<td>CL16</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>S</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>LEN</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>FLAGS</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>NAME STRUCTURE:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>in-family</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>in-port</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>in-address</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>dzero</td>
<td>D</td>
<td>PIC X(8)</td>
</tr>
<tr>
<td>BUF</td>
<td>LEN or larger</td>
<td>LEN or larger</td>
</tr>
<tr>
<td>ERRNO</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>RETCODE</td>
<td>F</td>
<td>PIC S9(8) BINARY</td>
</tr>
</tbody>
</table>

Parameter values to be set by the application for SENDTO

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
Must be set to 22 for the SENDTO command
S  The descriptor of the socket sending the data
LEN  The number of bytes to be transmitted (maximum 32768 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 for SENDTO
ERRNO  If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 455.
RETCODE  A value of -1 indicates an error. Other values have no meaning.

See “EZACIC04 program” on page 404 for a subroutine that translates EBCDIC data to ASCII.

COBOL call for SETSOCKOPT

This call functions in the same way as the equivalent call described “GETSOCKOPT call” on page 298. 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 "EZACICAL call format for assembler language" on page 424).

Parameter lengths in assembler language and COBOL for SETSOCKOPT

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Assembler language</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEn</td>
<td>CL16</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>COMMAND S</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>LEN</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>LEVEL</td>
<td>F</td>
<td>PIC X(4)</td>
</tr>
<tr>
<td>OPTNAME</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>OPTVAL</td>
<td>CL4</td>
<td>PIC X(4)</td>
</tr>
<tr>
<td>ERRNO</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>RETCODE</td>
<td>F</td>
<td>PIC S9(8) BINARY</td>
</tr>
</tbody>
</table>

Parameter values to be set by the application for SETSOCKOPT

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'.

OPTNAME
Set this field to specify the option to be set, as shown here. See “SETSOCKOPT call” on page 375 for a description of these settings.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'000000020'</td>
<td>SO-BROADCAST</td>
</tr>
<tr>
<td>X'000000080'</td>
<td>SO-LINGER</td>
</tr>
<tr>
<td>X'000001000'</td>
<td>SO-OOBINLINE</td>
</tr>
<tr>
<td>X'000000004'</td>
<td>SO-REUSEADDR</td>
</tr>
<tr>
<td>X'800000008'</td>
<td>TCP_KEEPALIVE</td>
</tr>
<tr>
<td>X'800000001'</td>
<td>TCP_NODELAY</td>
</tr>
</tbody>
</table>
OPTVAL
For SO-BROADCAST, SO-OOBINLINE, and SO-REUSEADDR, set to a nonzero integer to enable the option specified in OPTNAME, and set to 0 to disable the option. For SO-LINGER, see the equivalent OPTVAL parameter in “SETSOCKOPT call” on page 375.

Parameter values returned to the application for SETSOCKOPT

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 455.

RETCODE
A return value of 0 indicates a successful call. A return value of -1 indicates an error.

COBOL call for SHUTDOWN

This call functions in the same way as the equivalent call described in “SHUTDOWN call” on page 391. The format of the COBOL call for the SHUTDOWN function is:

CALL 'EZACICAL' USING TOKEN COMMAND S FZERO HOW ERRNO RETCODE.

In assembler language, issue the macro call CALL EZACICAL, using standard assembler call syntax (for the call format, see “EZACICAL call format for assembler language” on page 424).

Parameter lengths in assembler language and COBOL for SHUTDOWN

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Assembler language</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN</td>
<td>CL16</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>S</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>FZERO</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>HOW</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>ERRNO</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>RETCODE</td>
<td>F</td>
<td>PIC S9(8) BINARY</td>
</tr>
</tbody>
</table>

Parameter values to be set by the application for SHUTDOWN

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
Must be set to 24 for the SHUTDOWN command

S The descriptor of the socket to be shut down

FZERO Set to zeros

HOW Set this to specify whether all or part of a connection is to be shut down, as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Ends communication from the socket</td>
</tr>
</tbody>
</table>
1. Ends communication to the socket
2. Ends communication both to and from the socket

### Parameter values returned to the application for SHUTDOWN

**ERRNO**
- If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 455.

**RETCODE**
- A return value of 0 indicates a successful call. A return value of -1 indicates an error.

### COBOL call for SOCKET

This call functions in the same way as the equivalent call described in “SOCKET call” on page 393. The format of the COBOL call for the SOCKET function is:

```cobol
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 “EZACICAL call format for assembler language” on page 424).

### Parameter lengths in assembler language and COBOL for SOCKET

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Assembler language</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN</td>
<td>CL16</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>HZERO</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>AF</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>TYPE</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>PROTOCOL</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>SOCKNO</td>
<td>F</td>
<td>PIC S9(8) BINARY</td>
</tr>
<tr>
<td>ERRNO</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>RETCODE</td>
<td>F</td>
<td>PIC S9(8) BINARY</td>
</tr>
</tbody>
</table>

### Parameter values to be set by the application for SOCKET

**TOKEN**
- Must be set to ’TCPIPIUCVSTREAMS’

**COMMAND**
- Must be set to 25 for the SOCKET command

**HZERO**
- Set to binary zeros or low values

**AF**
- Must be set to 2 (AF-INET)

**TYPE**
- Set to 1 for TCP sockets; 2 for UDP sockets.

**PROTOCOL**
- Set to 0. (The system selects the appropriate protocol for the TYPE specified in [450])

**SOCKNO**
- Set to -1. The system returns the socket number in the RETCODE field.
Note: Use only the socket number returned by the system.

Parameter values returned to the application for SOCKET

ERRNO
If RETCODE is negative, this contains an error number. Error numbers are described in [Appendix B, “Return codes,” on page 455].

RETCODE
The socket number for the new socket is returned. A RETCODE of -1 indicates an error.

COBOL call for TAKESOCKET

This call functions in the same way as the equivalent call described in “TAKESOCKET call” on page 395. 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 “EZACICAL call format for assembler language” on page 424).

Parameter lengths in assembler language and COBOL for TAKESOCKET

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Assembler language</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN</td>
<td>CL16</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>HZERO</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
</tbody>
</table>
| CLIENTID STRUCTURE:
| Domain          | F                  | PIC 9(8) BINARY |
| Name            | CL8                | PIC X(8)  |
| Task            | CL8                | PIC X(8)  |
| Reserved        | CL20               | PIC 9(20) |
| L-DESC          | F                  | PIC 9(8) BINARY |
| SOCKNO          | F                  | PIC 59(8) BINARY |
| ERRNO           | F                  | PIC 9(8) BINARY |
| RETCODE         | F                  | PIC 9(8) BINARY |

Parameter values to be set by the application for TAKESOCKET

TOKEN
Must be set to 'TCPIPIUCVSTREAMS'

COMMAND
Must be set to 32 for the TAKESOCKET command

HZERO
Set to zeros

CLIENTID
Structure specifying the client ID of this program:

Domain
Must be set to 2 (AF-INET)
Name  Set to address space identifier, obtained from GETCLIENTID

Task  Set to CICS task number with L at the right end

Reserved  
  Set to binary zeros or LOW VALUES

L-DESC  
  Set to the descriptor (as used by the socket-giving program) of the socket being passed.

SOCKNO  
  Set to -1. The system returns the socket number in the RETCODE field.

Note: Be sure to use only the socket number returned by the system.

Parameter values returned to the application for TAKESOCKET

ERRNO  
  If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, “Return codes,” on page 455.

RETCODE  
  The socket number for the new socket is returned. A RETCODE of -1 indicates an error.

COBOL call for WRITE

This call functions in the same way as the equivalent call described in “WRITE call” on page 398. 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 "EZACICAL call format for assembler language" on page 424).

### Parameter lengths in assembler language and COBOL for WRITE

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Assembler language</th>
<th>COBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOKEN</td>
<td>CL16</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>COMMAND</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>S</td>
<td>H</td>
<td>PIC 9(4) BINARY</td>
</tr>
<tr>
<td>NBYTE</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>FZERO</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>SZERO</td>
<td>XL16</td>
<td>PIC X(16)</td>
</tr>
<tr>
<td>BUF</td>
<td>NBYTE or larger</td>
<td>NBYTE or larger</td>
</tr>
<tr>
<td>ERRNO</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
<tr>
<td>RETCODE</td>
<td>F</td>
<td>PIC 9(8) BINARY</td>
</tr>
</tbody>
</table>

### Parameter values to be set by the application for WRITE

**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 32768 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 for WRITE

**ERRNO**
- If RETCODE is negative, this contains an error number. Error numbers are described in Appendix B, "Return codes," on page 455.

**RETCODE**
- The number of bytes written is returned. A RETCODE of -1 indicates an error.

See "EZACIC04 program" on page 404 for a subroutine that translates EBCDIC data to ASCII.
Appendix B. Return codes

This topic covers the following return codes and error messages:
- Error numbers from z/OS TCP/IP.
- Error codes from the Sockets Extended interface.

Sockets return codes (ERRNOs)

This section provides the system-wide message numbers and codes set by the system calls. These message numbers and codes are in the TCPERRNO.H include file supplied with TCP/IP Services.

Table 24. Sockets ERRNOs

<table>
<thead>
<tr>
<th>Error number</th>
<th>Message name</th>
<th>Socket API type</th>
<th>Error description</th>
<th>Programmer's response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EAI_NONAME</td>
<td>GETADDRINFO GETNAMEINFO</td>
<td>NODE or HOST cannot be found.</td>
<td>Ensure the NODE or HOST name can be resolved.</td>
</tr>
<tr>
<td>1</td>
<td>EDOM</td>
<td>All</td>
<td>Argument too large.</td>
<td>Check parameter values of the function call.</td>
</tr>
<tr>
<td>1</td>
<td>EPERM</td>
<td>All</td>
<td>Permission is denied. No owner exists.</td>
<td>Check that TPC/IP is still active; check protocol value of socket() call.</td>
</tr>
<tr>
<td>1</td>
<td>EPERM</td>
<td>IOCTL (SIOCSPARTNERINFO)</td>
<td>Both endpoints do not reside in the same security domain.</td>
<td>Check and modify the security domain name for the endpoints. After you correct the security domain name, the application might need to close the connection if the IOCTL is needed.</td>
</tr>
<tr>
<td>1</td>
<td>EPERM</td>
<td>IOCTL (SIOCSPARTNERINFO, SIOCSPARTNERINFO)</td>
<td>The security domain name is not defined.</td>
<td>Define the security domain name on both endpoints. After you define the security domain name, the application might need to close the connection if the IOCTL is needed.</td>
</tr>
<tr>
<td>1</td>
<td>EPERM</td>
<td>IOCTL (SIOCTTLSCTL requesting both TTLS_INIT_CONNECTION and TTLS_RESET_SESSION or both TTLS_INIT_CONNECTION and TTLS_RESET_CIPHER)</td>
<td>The combination of requests specified is not permitted.</td>
<td>Request TTLS_RESET_SESSION and TTLS_RESET_CIPHER only when TTLS_INIT_CONNECTION has been previously requested for the connection.</td>
</tr>
<tr>
<td>1</td>
<td>EPERM</td>
<td>IOCTL (SIOCTTLSCTL)</td>
<td>Denotes one of the following error conditions:</td>
<td>Request TTLS_RESET_SESSION and TTLS_RESET_CIPHER only when TTLS_INIT_CONNECTION and TTLS_STOP_CONNECTION are not requested. Always request TTLS_INIT_CONNECTION when TTLS_ALLOW_HSTIMEOUT is requested. Use separate SIOCTTLSCTL ioctl to request TTLS_INIT_CONNECTION and TTLS_STOP_CONNECTION.</td>
</tr>
<tr>
<td>2</td>
<td>EAI_AGAIN</td>
<td>FREEADDRINFO GETADDRINFO GETNAMEINFO</td>
<td>For GETADDRINFO, NODE could not be resolved within the configured time interval. For GETNAMEINFO, HOST could not be resolved within the configured time interval. The Resolver address space has not been started. The request can be retried later.</td>
<td>Ensure the Resolver is active, then retry the request.</td>
</tr>
<tr>
<td>2</td>
<td>ENOENT</td>
<td>All</td>
<td>The data set or directory was not found.</td>
<td>Check files used by the function call.</td>
</tr>
<tr>
<td>2</td>
<td>ERANGE</td>
<td>All</td>
<td>The result is too large.</td>
<td>Check parameter values of the function call.</td>
</tr>
<tr>
<td>3</td>
<td>EAI_FAIL</td>
<td>FREEADDRINFO GETADDRINFO GETNAMEINFO</td>
<td>This is an unrecoverable error. NODELEN, HOSTLEN, or SERVLEN is incorrect. For FREEADDRINFO, the resolver storage does not exist.</td>
<td>Correct the NODELEN, HOSTLEN, or SERVLEN. Otherwise, call your system administrator.</td>
</tr>
<tr>
<td>Error number</td>
<td>Message name</td>
<td>Socket API type</td>
<td>Error description</td>
<td>Programmer's response</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------</td>
<td>----------------</td>
<td>------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>3</td>
<td>ESRCH</td>
<td>All</td>
<td>The process was not found. A table entry was not located.</td>
<td>Check parameter values and structures pointed to by the function parameters.</td>
</tr>
<tr>
<td>4</td>
<td>EINVAL</td>
<td>GETNAMEINFO</td>
<td>The output buffer for the host name or service name was too small.</td>
<td>Increase the size of the buffer to 255 characters, which is the maximum size permitted.</td>
</tr>
<tr>
<td>4</td>
<td>EINVAL</td>
<td>All</td>
<td>A system call was interrupted.</td>
<td>Check that the socket connection and TCP/IP are still active.</td>
</tr>
<tr>
<td>5</td>
<td>EINVAL</td>
<td>GETADDRINFO</td>
<td>The AF or the FAMILY is incorrect.</td>
<td>Correct the AF or the FAMILY.</td>
</tr>
<tr>
<td>5</td>
<td>EINVAL</td>
<td>All</td>
<td>An I/O error occurred.</td>
<td>Check status and contents of source database if this occurred during a file access.</td>
</tr>
<tr>
<td>6</td>
<td>EINVAL</td>
<td>GETADDRINFO</td>
<td>The resolver cannot obtain storage to process the host name.</td>
<td>Contact your system administrator.</td>
</tr>
<tr>
<td>6</td>
<td>EINVAL</td>
<td>All</td>
<td>The device or driver was not found.</td>
<td>Check status of the device attempting to access.</td>
</tr>
<tr>
<td>7</td>
<td>EINVAL</td>
<td>All</td>
<td>The argument list is too long.</td>
<td>Check the number of function parameters.</td>
</tr>
<tr>
<td>7</td>
<td>EINVAL</td>
<td>GETADDRINFO</td>
<td>FLAGS has an incorrect value.</td>
<td>Correct the FLAGS.</td>
</tr>
<tr>
<td>8</td>
<td>EINVAL</td>
<td>GETADDRINFO</td>
<td>The SERVICE was not recognized for the specified socket type.</td>
<td>Correct the SERVICE.</td>
</tr>
<tr>
<td>8</td>
<td>EINVAL</td>
<td>All</td>
<td>An EXEC format error occurred.</td>
<td>Check that the target module on an exec call is a valid executable module.</td>
</tr>
<tr>
<td>9</td>
<td>EINVAL</td>
<td>GETADDRINFO</td>
<td>The SOCTYPE was not recognized.</td>
<td>Correct the SOCTYPE.</td>
</tr>
<tr>
<td>9</td>
<td>EINVAL</td>
<td>All</td>
<td>An incorrect socket descriptor was specified.</td>
<td>Check socket descriptor value. It might be currently not in use or incorrect.</td>
</tr>
<tr>
<td>9</td>
<td>EINVAL</td>
<td>givesocket</td>
<td>The socket has already been given. The socket domain is not AF_INET or AF_INET6.</td>
<td>Check the validity of function parameters.</td>
</tr>
<tr>
<td>9</td>
<td>EINVAL</td>
<td>Select</td>
<td>One of the specified descriptor sets is an incorrect socket descriptor.</td>
<td>Check the validity of function parameters.</td>
</tr>
<tr>
<td>9</td>
<td>EINVAL</td>
<td>Takesocket</td>
<td>The socket has already been taken.</td>
<td>Check the validity of function parameters.</td>
</tr>
<tr>
<td>9</td>
<td>EINVAL</td>
<td>GETADDRINFO</td>
<td>The SOCTYPE was not recognized.</td>
<td>Correct the SOCTYPE.</td>
</tr>
<tr>
<td>10</td>
<td>EINVAL</td>
<td>All</td>
<td>There are no children.</td>
<td>Check if created subtasks still exist.</td>
</tr>
<tr>
<td>11</td>
<td>EINVAL</td>
<td>All</td>
<td>There are no more processes.</td>
<td>Retry the operation. Data or condition might not be available at this time.</td>
</tr>
<tr>
<td>11</td>
<td>EINVAL</td>
<td>All</td>
<td>TCP/IP is not active at the time of the request.</td>
<td>Start TCP/IP, and retry the request.</td>
</tr>
<tr>
<td>11</td>
<td>EINVAL</td>
<td>IOCTL (SIOCGPARTNERINFO)</td>
<td>The IOCTL was issued in no-suspended mode and the SIOCGPARTNERINFO IOCTL has not been issued.</td>
<td>Reissue the IOCTL with a timeout value to set the amount of time to wait while the partner security credentials are being retrieved.</td>
</tr>
<tr>
<td>12</td>
<td>EINVAL</td>
<td>All</td>
<td>There is not enough storage.</td>
<td>Check the validity of function parameters.</td>
</tr>
<tr>
<td>13</td>
<td>EINVAL</td>
<td>All</td>
<td>Permission denied, caller not authorized.</td>
<td>Check access authority of file.</td>
</tr>
<tr>
<td>13</td>
<td>EINVAL</td>
<td>IOCTL (SIOCGPARTNERINFO)</td>
<td>The application is not running in supervisor state, is not AFP authorized, or is not permitted to the appropriate SERVAUTH profile.</td>
<td>Allow the application to issue this IOCTL, or provide the user ID with the proper SERVAUTH permission.</td>
</tr>
<tr>
<td>13</td>
<td>EINVAL</td>
<td>IOCTL (SIOCTTLSCTL)</td>
<td>The IOCTL is requesting a function that requires that the socket be mapped to policy that specifies Application Controlled On.</td>
<td>Check policy and add Application Controlled On if the application should be permitted to issue the controlled SIOCTTLSCTL functions.</td>
</tr>
<tr>
<td>13</td>
<td>EINVAL</td>
<td>Takesocket</td>
<td>The other application (listener) did not give the socket to your application. Permission denied, caller not authorized.</td>
<td>Check access authority of file.</td>
</tr>
<tr>
<td>Error number</td>
<td>Message name</td>
<td>Socket API type</td>
<td>Error description</td>
<td>Programmer's response</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------</td>
<td>----------------</td>
<td>------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>14</td>
<td>EFAULT</td>
<td>All</td>
<td>An incorrect storage address or length was specified.</td>
<td>Check the validity of function parameters.</td>
</tr>
<tr>
<td>14</td>
<td>EFAULT</td>
<td>All</td>
<td>The exit routine has abnormally ended (ABEND condition).</td>
<td>Correct the error in the routine's code. Add an ESTAE routine to the exit.</td>
</tr>
<tr>
<td>14</td>
<td>EFAULT</td>
<td>IOCTL (SIOCSAPPLDATA)</td>
<td>An abend occurred while attempting to copy the SetADContainer structure from the address provided in the SetAD_ptr field.</td>
<td>Check the validity of function parameters.</td>
</tr>
<tr>
<td>15</td>
<td>EFAULT</td>
<td>All</td>
<td>A block device is required.</td>
<td>Check device status and characteristics.</td>
</tr>
<tr>
<td>16</td>
<td>EFAULT</td>
<td>All</td>
<td>A block device is required.</td>
<td>Check the validity of function parameters.</td>
</tr>
<tr>
<td>17</td>
<td>EFAULT</td>
<td>All</td>
<td>The data set exists.</td>
<td>Remove or rename existing file.</td>
</tr>
<tr>
<td>18</td>
<td>EFAULT</td>
<td>All</td>
<td>This is a cross-device link. A link to a file on another file system was attempted.</td>
<td>Check file permissions.</td>
</tr>
<tr>
<td>19</td>
<td>EFAULT</td>
<td>All</td>
<td>The specified device does not exist.</td>
<td>Check file name and if it exists.</td>
</tr>
<tr>
<td>20</td>
<td>EFAULT</td>
<td>All</td>
<td>The specified directory is not a directory.</td>
<td>Use a valid file that is a directory.</td>
</tr>
<tr>
<td>21</td>
<td>EFAULT</td>
<td>All</td>
<td>The specified directory is a directory.</td>
<td>Use a valid file that is not a directory.</td>
</tr>
<tr>
<td>22</td>
<td>EFAULT</td>
<td>All</td>
<td>An incorrect argument was specified.</td>
<td>Check the validity of function parameters.</td>
</tr>
<tr>
<td>22</td>
<td>EFAULT</td>
<td>Multicast Source filter APIs</td>
<td>Mix of any-source, source-specific or full-state APIs</td>
<td>Specify the correct type of APIs.</td>
</tr>
<tr>
<td>22</td>
<td>EFAULT</td>
<td>MCAST_JOIN_GROUP, MCAST_LEAVE_GROUP, MCAST_BLOCK_SOURCE, SIOCGMSFILTER, SIOCSMSFILTER</td>
<td>The socket address family or the socket length of the input multicast group or the source IP address is not correct.</td>
<td>Specify the correct value.</td>
</tr>
<tr>
<td>22</td>
<td>EFAULT</td>
<td>SIOCSAPPLDATA</td>
<td>The specified filter mode is not correct.</td>
<td>Specify the correct value.</td>
</tr>
<tr>
<td>23</td>
<td>EFAULT</td>
<td>All</td>
<td>Data set table overflow occurred.</td>
<td>Reduce the number of open files.</td>
</tr>
<tr>
<td>24</td>
<td>EFAULT</td>
<td>All</td>
<td>The socket descriptor table is full.</td>
<td>Check the maximum sockets specified in MAXDESC().</td>
</tr>
<tr>
<td>25</td>
<td>EFAULT</td>
<td>All</td>
<td>An incorrect device call was specified.</td>
<td>Check specified IOCTL() values.</td>
</tr>
<tr>
<td>26</td>
<td>EFAULT</td>
<td>All</td>
<td>A text data set is busy.</td>
<td>Check the current use of the file.</td>
</tr>
<tr>
<td>27</td>
<td>EFAULT</td>
<td>All</td>
<td>The specified data set is too large.</td>
<td>Check size of accessed dataset.</td>
</tr>
<tr>
<td>28</td>
<td>EFAULT</td>
<td>All</td>
<td>There is no space left on the device.</td>
<td>Increase the size of accessed file.</td>
</tr>
<tr>
<td>29</td>
<td>EFAULT</td>
<td>All</td>
<td>An incorrect seek was attempted.</td>
<td>Check the offset parameter for seek operation.</td>
</tr>
<tr>
<td>30</td>
<td>EFAULT</td>
<td>All</td>
<td>The data set system is Read only.</td>
<td>Access data set for read only operation.</td>
</tr>
<tr>
<td>31</td>
<td>EFAULT</td>
<td>All</td>
<td>There are too many links.</td>
<td>Reduce the number of links to the accessed file.</td>
</tr>
<tr>
<td>32</td>
<td>EFAULT</td>
<td>All</td>
<td>The connection is broken. For socket write/send, peer has shut down one or both directions.</td>
<td>Reconnect with the peer.</td>
</tr>
<tr>
<td>32</td>
<td>EFAULT</td>
<td>IOCTL (SIOCTTLSCTL requesting TTLS_INIT_CONNECTION, TTLS_RESET_CIPHER, or TTLS_STOP_CONNECTION)</td>
<td>The TCP connection is not in the established state.</td>
<td>Issue the SIOCTTLSCTL IOCTL when the socket is connected.</td>
</tr>
<tr>
<td>33</td>
<td>EFAULT</td>
<td>All</td>
<td>The specified argument is too large.</td>
<td>Check and correct function parameters.</td>
</tr>
<tr>
<td>34</td>
<td>EFAULT</td>
<td>All</td>
<td>The result is too large.</td>
<td>Check function parameter values.</td>
</tr>
<tr>
<td>35</td>
<td>EFAULT</td>
<td>Accept</td>
<td>The socket is in nonblocking mode and connections are not queued. This is not an error condition.</td>
<td>Reissue Accept().</td>
</tr>
<tr>
<td>35</td>
<td>EFAULT</td>
<td>IOCTL (SIOCTTLSCTL)</td>
<td>The handshake is in progress and the socket is a nonblocking socket.</td>
<td>For a nonblocking socket, you can wait for the handshake to complete by issuing Select or Poll for Socket Writable.</td>
</tr>
<tr>
<td>35</td>
<td>EFAULT</td>
<td>Read Recvfrom</td>
<td>The socket is in nonblocking mode and read data is not available. This is not an error condition.</td>
<td>Issue a select on the socket to determine when data is available to be read or reissue the Read()/Recvfrom().</td>
</tr>
<tr>
<td>Error number</td>
<td>Message name</td>
<td>Socket API type</td>
<td>Error description</td>
<td>Programmer’s response</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
<td>-----------------</td>
<td>-------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>35</td>
<td>EWOULDBLOCK</td>
<td>All receive calls (RECV, RECVMSG, RECVFROM, READY, READ), when the socket is set with the SO_RCVTIMEO socket option</td>
<td>The socket is in blocking mode and the receive call has blocked for the time period that was specified in the SO_RCVTIMEO option. No data was received.</td>
<td>The application should reissue the receive call.</td>
</tr>
<tr>
<td>35</td>
<td>EWOULDBLOCK</td>
<td>Send Sendto Write</td>
<td>The socket is in nonblocking mode and buffers are not available.</td>
<td>Issue a select on the socket to determine when data is available to be written or reissue the Send(), Sendto(), or Write().</td>
</tr>
<tr>
<td>35</td>
<td>EWOULDBLOCK</td>
<td>All send calls (SEND, SENDMSG, SENDTO, WRITEV, WRITE), when the socket is set with the SO_SNDRTIMEO socket option</td>
<td>The socket is in blocking mode and the send call has blocked for the time period that was specified in the SO_SNDRTIMEO option. No data was sent.</td>
<td>The application should reissue the send call.</td>
</tr>
<tr>
<td>36</td>
<td>EINPROGRESS</td>
<td>Connect</td>
<td>The socket is marked nonblocking and the connection cannot be completed immediately. This is not an error condition.</td>
<td>See the Connect() description for possible responses.</td>
</tr>
<tr>
<td>36</td>
<td>EINPROGRESS</td>
<td>IOCTL (SIOCGPARTNERINFO)</td>
<td>The IOCTL was issued in no-suspend mode after the SIOCGPARTNERINFO IOCTL was issued, but the partner security credentials are not currently available.</td>
<td>Retry the IOCTL, or issue the IOCTL with a timeout value to set the amount of time to wait while the partner security credentials are being retrieved.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Restriction: You cannot use a select mask to determine when an IOCTL is complete, because an IOCTL is not affected by whether the socket is running in blocking or nonblocking mode. If the IOCTL times out, reissue the IOCTL to retrieve the partner security credentials.</td>
</tr>
<tr>
<td>36</td>
<td>EINPROGRESS</td>
<td>IOCTL (SIOCTTLSCTL requesting TTLS_INIT_CONNECTION or TTLS_STOP_CONNECTION)</td>
<td>The handshake is already in progress and the socket is a nonblocking socket.</td>
<td>For a nonblocking socket, you can wait for the handshake to complete by issuing Select or Poll for Socket Writable.</td>
</tr>
<tr>
<td>37</td>
<td>EALREADY</td>
<td>Connect</td>
<td>The socket is marked nonblocking and the previous connection has not been completed.</td>
<td>Reissue Connect().</td>
</tr>
<tr>
<td>37</td>
<td>EALREADY</td>
<td>IOCTL (SIOCGPARTNERINFO)</td>
<td>The request is already in progress. Only one IOCTL can be outstanding.</td>
<td>Check and modify the socket descriptor, if specified; otherwise, no action is needed.</td>
</tr>
<tr>
<td>37</td>
<td>EALREADY</td>
<td>IOCTL (SIOCTTLSCTL requesting TTLS_INIT_CONNECTION or TTLS_STOP_CONNECTION)</td>
<td>For TTLS_INIT_CONNECTION, the socket is already secure. For TTLS_STOP_CONNECTION, the socket is not secure.</td>
<td>Modify the application so that it issues the SIOCTTLSCTL IOCTL that requests TTLS_INIT_CONNECTION only when the socket is not already secure and that requests TTLS_STOP_CONNECTION only when the socket is secure.</td>
</tr>
<tr>
<td>37</td>
<td>EALREADY</td>
<td>Maxdesc</td>
<td>A socket has already been created calling Maxdesc() or multiple calls to Maxdesc().</td>
<td>Issue Getablesize() to query it.</td>
</tr>
<tr>
<td>37</td>
<td>EALREADY</td>
<td>Setibmopt</td>
<td>A connection already exists to a TCP/IP image. A call to SETIBMOPT (BMTCP_IMAGE), has already been made.</td>
<td>Call Setibmopt() only once.</td>
</tr>
<tr>
<td>38</td>
<td>ENOTSOCK</td>
<td>All</td>
<td>A socket operation was requested on a nonsocket connection. The value for socket descriptor was not valid.</td>
<td>Correct the socket descriptor value and reissue the function call.</td>
</tr>
<tr>
<td>39</td>
<td>EDESTADDRREQ</td>
<td>All</td>
<td>A destination address is required.</td>
<td>Fill in the destination field in the correct socket and reissue the function call.</td>
</tr>
<tr>
<td>40</td>
<td>EMSGSIZE</td>
<td>Sendto Sendmsg Send Write for Datagram (UDP) or RAW sockets</td>
<td>The message is too long. It exceeds the IP limit of 64K or the limit set by the setsockopt() call.</td>
<td>Either correct the length parameter, or send the message in smaller pieces.</td>
</tr>
<tr>
<td>41</td>
<td>EPROTOTYPE</td>
<td>All</td>
<td>The specified protocol type is incorrect for this socket.</td>
<td>Correct the protocol type parameter.</td>
</tr>
<tr>
<td>41</td>
<td>EPROTOTYPE</td>
<td>bind2addrsel</td>
<td>The referenced socket is not a stream (TCP) or datagram (UDP) socket.</td>
<td>Issue bind2addrsel() on TCP or UDP sockets only.</td>
</tr>
<tr>
<td>41</td>
<td>EPROTOTYPE</td>
<td>IOCTL (SIOCGPARTNERINFO, SIOCGAPPDATA, SIOCGPARTNERINFO, SIOCTTLSCTL)</td>
<td>Socket is not a TCP socket.</td>
<td>Issue the IOCTL on TCP sockets only.</td>
</tr>
<tr>
<td>42</td>
<td>ENOPROTOOPT</td>
<td>Getsockopt Setsockopt</td>
<td>The socket option specified is incorrect or the level is not SOL_SOCKET. Either the level or the specified optname is not supported.</td>
<td>Correct the level or optname.</td>
</tr>
</tbody>
</table>
### Table 24. Sockets ERRNOs (continued)

<table>
<thead>
<tr>
<th>Error number</th>
<th>Message name</th>
<th>Socket API type</th>
<th>Error description</th>
<th>Programmer's response</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>ENOPROTOOPT</td>
<td>Getibmsockopt Setibmsockopt</td>
<td>Either the level or the specified optname is not supported.</td>
<td>Correct the level or optname.</td>
</tr>
<tr>
<td>43</td>
<td>EPROTONOSUPPORT</td>
<td>Socket</td>
<td>The specified protocol is not supported.</td>
<td>Correct the protocol parameter.</td>
</tr>
<tr>
<td>44</td>
<td>ESOCKTNOSUPPORT</td>
<td>All</td>
<td>The specified socket type is not supported.</td>
<td>Correct the socket type parameter.</td>
</tr>
<tr>
<td>45</td>
<td>EOPNOTSUPP</td>
<td>Accept Givesocket</td>
<td>The selected socket is not a stream socket.</td>
<td>Use a valid socket.</td>
</tr>
<tr>
<td>45</td>
<td>EOPNOTSUPP</td>
<td>bind2addrsel</td>
<td>The referenced socket is not a type that supports the requested function</td>
<td>Use a socket of the correct type.</td>
</tr>
<tr>
<td>45</td>
<td>EOPNOTSUPP</td>
<td>Getibmsockopt</td>
<td>The socket does not support this function call. This command is not supported for this function.</td>
<td>Correct the command parameter. See Getibmsockopt() for valid commands. Correct by ensuring a Listen() was not issued before the Connect().</td>
</tr>
<tr>
<td>45</td>
<td>EOPNOTSUPP</td>
<td>GETSOCKOPT</td>
<td>The specified GETSOCKOPT OPTNAME option is not supported by this socket API.</td>
<td>Correct the GETSOCKOPT OPTNAME option.</td>
</tr>
<tr>
<td>45</td>
<td>EOPNOTSUPP</td>
<td>IOCTL</td>
<td>The request must be issued before the listen call or the connect call.</td>
<td>Check and modify the socket descriptor, or close the connection and reissue the call.</td>
</tr>
<tr>
<td>45</td>
<td>EOPNOTSUPP</td>
<td>IOCTL (SIOCSFRUPTERINFO)</td>
<td>Mapped policy indicates that AT-TLS is not enabled for the connection.</td>
<td>Modify the policy to enable AT-TLS for the connection.</td>
</tr>
<tr>
<td>45</td>
<td>EOPNOTSUPP</td>
<td>Listen</td>
<td>The socket does not support the Listen call.</td>
<td>Change the type on the Socket() call when the socket was created. Listen() supports only a socket type of SOCK_STREAM.</td>
</tr>
<tr>
<td>45</td>
<td>EOPNOTSUPP</td>
<td>RECV, RECVFROM, RECVMSG, SEND, SENDTO, SENDMSG</td>
<td>The specified flags are not supported on this socket type or protocol.</td>
<td>Correct the FLAG.</td>
</tr>
<tr>
<td>45</td>
<td>EOPNOTSUPP</td>
<td>All</td>
<td>The specified protocol family is not supported or the specified domain for the client identifier is not AF_INET=2.</td>
<td>Correct the protocol family.</td>
</tr>
<tr>
<td>46</td>
<td>EPFNOSUPPORT</td>
<td>bind2addr sel inet6_is_srcaddr</td>
<td>You specified an IP address that is not an AF_INET6 IP address</td>
<td>Correct the IP address. If the IP address is an IPv4 address, you must specify it as an IPv4-mapped IPv6 address.</td>
</tr>
<tr>
<td>47</td>
<td>EAFNOSUPPORT</td>
<td>bind2addr sel inet6_is_srcaddr</td>
<td>You attempted an IPv6-only API for a stack that does not support the AF_INET6 domain.</td>
<td>Activate the AF_INET6 stack, and retry the request.</td>
</tr>
<tr>
<td>47</td>
<td>EAFNOSUPPORT</td>
<td>Bind Connect Socket</td>
<td>The specified address family is not supported by this protocol family.</td>
<td>For Socket(), set the domain parameter to AF_INET. For Bind() and Connect(), set Sin_Family in the socket address structure to AF_INET.</td>
</tr>
<tr>
<td>47</td>
<td>EAFNOSUPPORT</td>
<td>Getclient Givesocket</td>
<td>The socket specified by the socket descriptor parameter was not created in the AF_INET domain.</td>
<td>The Socket() call used to create the socket should be changed to use AF_INET for the domain parameter.</td>
</tr>
<tr>
<td>47</td>
<td>EAFNOSUPPORT</td>
<td>IOCTL</td>
<td>You attempted to use an IPv4-only ioctl on an AF_INET6 socket.</td>
<td>Use the correct socket type for the ioctl or use an ioctl that supports AF_INET6 sockets.</td>
</tr>
<tr>
<td>48</td>
<td>EADDRINUSE</td>
<td>Bind</td>
<td>The address is in a timed wait because a LINGER delay from a previous close or another process is using the address. This error can also occur if the port specified in the bind call has been configured as RESERVED on a port reservation statement in the TCP/IP profile.</td>
<td>If you want to reuse the same address, use Setsockopt() with SO_REUSEADDR. Refer to the section about Setsockopt() in z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference for more information. Otherwise, use a different address or port in the socket address structure.</td>
</tr>
<tr>
<td>48</td>
<td>EADDRINUSE</td>
<td>IP_ADD_MEMBERSHIP, IP_ADD_SOURCE_ MEMBERSHIP, IPv6_JOIN_GROUP, MCASr JOIN_ GROUP, MCASr JOIN_ SOURCE_ GROUP</td>
<td>The specified multicast address and interface address (or interface index) pair is already in use.</td>
<td>Correct the specified multicast address, interface address, or interface index.</td>
</tr>
<tr>
<td>49</td>
<td>EADDRNOTAVAIL</td>
<td>Bind</td>
<td>The specified address is incorrect for this host.</td>
<td>Correct the function address parameter.</td>
</tr>
<tr>
<td>49</td>
<td>EADDRNOTAVAIL</td>
<td>Connect</td>
<td>The calling host cannot reach the specified destination.</td>
<td>Correct the function address parameter.</td>
</tr>
<tr>
<td>Error number</td>
<td>Message name</td>
<td>Socket API type</td>
<td>Error description</td>
<td>Programmer's response</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------</td>
<td>-----------------</td>
<td>-------------------</td>
<td>-----------------------</td>
</tr>
</tbody>
</table>
| 49          | EADDRNOTAVAIL | bind2addresel   | For the specified destination address, there is no source address that the application can bind to. Possible reasons can be one of the following situations:  
  - The socket is a stream socket, but the specified destination address is a multicast address.  
  - No ephemeral ports are available to assign to the socket. | Correct the function address parameter or issue the request when ephemeral ports are available. |
| 49          | EADDRNOTAVAIL | inet6_is_srcaddr | The address specified is not correct for one of these reasons:  
  - The address is not an address on this node.  
  - The address was not active at the time of the request.  
  - The scope ID specified for a link-local IPv6 address is incorrect. | Correct or activate the address |
| 49          | EADDRNOTAVAIL | IP_BLOCK_SOURCE, IP_ADD_SOURCE_MEMBERSHIP, MCAST_BLOCK_SOURCE, MCAST_JOIN_SOURCE_GROUP | A duplicate source IP address is specified on the multicast group and interface pair. | Correct the specified source IP address. |
| 49          | EADDRNOTAVAIL | IP_UNBLOCK_SOURCE, IP_DROP_SOURCE_MEMBERSHIP, MCAST_UNBLOCK_SOURCE, MCAST_LEAVE_SOURCE_GROUP | A previously blocked source multicast group cannot be found. | Correct the specified address. |
| 49          | EADDRNOTAVAIL | Multicast APIs | The specified multicast address, interface address, or interface index is not correct. | Correct the specified address. |
| 50          | ENETDOWN      | All             | The network is down. | Retry when the connection path is up. |
| 51          | ENETUNREACH   | Connect         | The network cannot be reached. | Ensure that the target application is active. |
| 52          | ENETRESET     | All             | The network dropped a connection on a reset. | Reestablish the connection between the applications. |
| 53          | ECONNABORTED  | All             | The software caused a connection abend. | Reestablish the connection between the applications. |
| 54          | ECONNRESET    | All             | The connection to the destination host is not available. | N/A |
| 54          | ECONNRESET    | Send Write      | The connection to the destination host is not available. | The socket is closing. Issue Send() or Write() before closing the socket. |
| 55          | ENOBUFS       | All             | No buffer space is available. | Check the application for massive storage allocation call. |
| 55          | ENOBUFS       | Accept          | Not enough buffer space is available to create the new socket. | Call your system administrator. |
| 55          | ENOBUFS       | IOCTL (SIOCGBPARTNERINFO) | The buffer size provided is too small. | Create a larger input buffer based on the value returned in the PL Bufflen field. |
| 55          | ENOBUFS       | IOCTL (SIOCSAPPLDATA) | There is no storage available to store the associated data. | Call your system administrator. |
| 55          | ENOBUFS       | IOCTL (SIOCCTL,TLS_Version=1 requesting,TLS_RETURN_ CERTIFICATE or TLS_Version=2 query) | The buffer size provided is too small. | For TLS_Version=1 use the returned certificate length to allocate a larger buffer and reissue IOCTL with the larger buffer. |
| 55          | ENOBUFS       | IP_BLOCK_SOURCE, IP_ADD_SOURCE.Membership, MCAST_BLOCK_SOURCE, MCAST_JOIN_SOURCE_GROUP, SIOCIPMSFILTER, SIOCMSFILTER, setipv4sourcefilter, setsourcefilter | A maximum of 64 source filters can be specified per multicast address, interface address pair. | Remove unneeded source IP addresses and reissue the command. |
| 55          | ENOBUFS       | Send Sendto Write | Not enough buffer space is available to send the new message. | Call your system administrator. |
| 55          | ENOBUFS       | Takesocket      | Not enough buffer space is available to create the new socket. | Call your system administrator. |
| 56          | EISCONN       | Connect         | The socket is already connected. | Correct the socket descriptor on Connect() or do not issue a Connect() twice for the socket. |
| 57          | ENOTCONN      | All             | The socket is not connected. | Connect the socket before communicating. |
| 57          | ENOTCONN      | IOCTL (SIOCGBPARTNERINFO) | The requested socket is not connected. | Check and modify the socket descriptor, or reissue the IOCTL after the connect call from the client side or after the accept call from the server side. |

460 z/OS V2R1.0 Communications Server: IP CICS Sockets Guide
<table>
<thead>
<tr>
<th>Error number</th>
<th>Message name</th>
<th>Socket API type</th>
<th>Error description</th>
<th>Programmer's response</th>
</tr>
</thead>
<tbody>
<tr>
<td>57</td>
<td>ENOTCONN</td>
<td>IOCTL (SIOCTTLSCTL)</td>
<td>The socket is not connected.</td>
<td>Issue the SIOCTTLSCTL IOCTL only after the socket is connected.</td>
</tr>
<tr>
<td>58</td>
<td>ESHUTDOWN</td>
<td>All</td>
<td>A Send cannot be processed after socket shutdown.</td>
<td>Issue read/receive before shutting down the read side of the socket.</td>
</tr>
<tr>
<td>59</td>
<td>ETOOMANYREFS</td>
<td>All</td>
<td>There are too many references. A splice cannot be completed.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>59</td>
<td>ETOOMANYREFS</td>
<td>IP_ADD_MEMBERSHIP, IP_ADD_SOURCE_ MEMBERSHIP, MCAST_JOIN_GROUP, MCAST_JOIN_SOURCE_GROUP, IPv6_JOIN_GROUP</td>
<td>A maximum of 20 multicast groups per single UDP socket or a maximum of 256 multicast groups per single RAW socket can be specified.</td>
<td>Remove unneeded multicast groups and reissue the command.</td>
</tr>
<tr>
<td>60</td>
<td>ETIMEDOUT</td>
<td>Connect</td>
<td>The connection timed out before it was completed.</td>
<td>Ensure the server application is available.</td>
</tr>
<tr>
<td>61</td>
<td>ECONNREFUSED</td>
<td>Connect</td>
<td>The requested connection was refused.</td>
<td>Ensure server application is available and at specified port.</td>
</tr>
<tr>
<td>62</td>
<td>ELOOP</td>
<td>All</td>
<td>There are too many symbolic loop levels.</td>
<td>Reduce symbolic links to specified file.</td>
</tr>
<tr>
<td>63</td>
<td>ENAMETOOLONG</td>
<td>All</td>
<td>The file name is too long.</td>
<td>Reduce size of specified file name.</td>
</tr>
<tr>
<td>64</td>
<td>EHOSTDOWN</td>
<td>All</td>
<td>The host is down.</td>
<td>Restart specified host.</td>
</tr>
<tr>
<td>65</td>
<td>EHOSTUNREACH</td>
<td>All</td>
<td>There is no route to the host.</td>
<td>Set up network path to specified host and verify that host name is valid.</td>
</tr>
<tr>
<td>66</td>
<td>ENOTEMPTY</td>
<td>All</td>
<td>The directory is not empty.</td>
<td>Clear out specified directory and reissue call.</td>
</tr>
<tr>
<td>67</td>
<td>EPROCLIM</td>
<td>All</td>
<td>There are too many processes in the system.</td>
<td>Decrease the number of processes or increase the process limit.</td>
</tr>
<tr>
<td>68</td>
<td>EUSERS</td>
<td>All</td>
<td>There are too many users on the system.</td>
<td>Decrease the number of users or increase the user limit.</td>
</tr>
<tr>
<td>69</td>
<td>EDQUOT</td>
<td>All</td>
<td>The disk quota has been exceeded.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>70</td>
<td>ESTALE</td>
<td>All</td>
<td>An old NPSV data set handle was found.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>71</td>
<td>EREMOTE</td>
<td>All</td>
<td>There are too many levels of remote in the path.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>72</td>
<td>ENOSTR</td>
<td>All</td>
<td>The device is not a stream device.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>73</td>
<td>ETIME</td>
<td>All</td>
<td>The timer has expired.</td>
<td>Increase timer values or reissue function.</td>
</tr>
<tr>
<td>73</td>
<td>ETIME</td>
<td>IOCTL (SIOCGPARTNERINFO)</td>
<td>The wait time for the request has expired, possibly as the result of network problems.</td>
<td>Retry the request. <strong>Restriction:</strong> You cannot use a select mask to determine when an IOCTL is complete, because an IOCTL is not affected by whether the socket is running in blocking or nonblocking mode. If the IOCTL times out, reissue the IOCTL to retrieve the partner security credentials.</td>
</tr>
<tr>
<td>74</td>
<td>ENOSR</td>
<td>All</td>
<td>There are no more stream resources.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>75</td>
<td>ENOMSG</td>
<td>All</td>
<td>There is no message of the desired type.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>76</td>
<td>ERADMSG</td>
<td>All</td>
<td>The system cannot read the message.</td>
<td>Verify that z/OS Communications Server installation was successful and that message files were properly loaded.</td>
</tr>
<tr>
<td>77</td>
<td>EIDRM</td>
<td>All</td>
<td>The identifier has been removed.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>78</td>
<td>EDEADLK</td>
<td>All</td>
<td>A deadlock condition has occurred.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>78</td>
<td>EDEADLK</td>
<td>Select Selectex</td>
<td>None of the sockets in the socket descriptor sets are either AF_INET or AF_JUCV sockets and there is no timeout value or no ECB specified. The select/selectex would never complete.</td>
<td>Correct the socket descriptor sets so that an AF_INET or AF_JUCV socket is specified. A timeout or ECB value can also be added to avoid the select/selectex from waiting indefinitely.</td>
</tr>
<tr>
<td>79</td>
<td>ENOLCK</td>
<td>All</td>
<td>No record locks are available.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>80</td>
<td>ENONET</td>
<td>All</td>
<td>The requested machine is not on the network.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>81</td>
<td>EREREMOTE</td>
<td>All</td>
<td>The object is remote.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>82</td>
<td>ENOLINK</td>
<td>All</td>
<td>The link has been severed.</td>
<td>Release the sockets and reinitialize the client-server connection.</td>
</tr>
<tr>
<td>83</td>
<td>EADV</td>
<td>All</td>
<td>An ADVERTISE error has occurred.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>84</td>
<td>ESRMNT</td>
<td>All</td>
<td>An SRMOUNT error has occurred.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>85</td>
<td>ECOMM</td>
<td>All</td>
<td>A communication error has occurred on a Send call.</td>
<td>Call your system administrator.</td>
</tr>
</tbody>
</table>
## Table 24. Sockets ERRNOs (continued)

<table>
<thead>
<tr>
<th>Error number</th>
<th>Message name</th>
<th>Socket API type</th>
<th>Error description</th>
<th>Programmer’s response</th>
</tr>
</thead>
<tbody>
<tr>
<td>86</td>
<td>EPROTO</td>
<td>All</td>
<td>A protocol error has occurred.</td>
<td>Call your system administrator.</td>
</tr>
</tbody>
</table>
|              | EPROTO       | IOCTL (SIOCTTLSCTL request in TTLS_RESET_SESSION, TTLS_RESET_CIPHER, TTLS_STOP_CONNECTION, or TTLS.Allow_HSTIMEOUT) | One of the following errors occurred:  
  - A TTLS_INIT_CONNECTION request was not received for the connection.  
  - TTLS_STOP_CONNECTION was requested on a connection that has outstanding application data. For unread application data, the errno junior is JrTTLSStopReadDataPending. For unwritten application data, the errno junior is JrTTLSStopWriteDataPending.  
  - TTLS_RESET_CIPHER or TTLS_STOP_CIPHER was requested on a connection that is secured using SSL version 2.  
  - TTLS.Allow_HSTIMEOUT was requested but the policy has the HandshakeRole value client or the HandshakeTimeout value is 0. | • Request TTLS_INIT_CONNECTION before requesting TTLS_RESET_SESSION or TTLS_RESET_CIPHER.  
• Request TTLS_STOP_CONNECTION after all application data is cleared from the connection. For JrTTLSStopReadDataPending, read all available application data. For JrTTLSStopWriteDataPending, wait for all the outstanding application data to be written.  
• Request TTLS_RESET_CIPHER or TTLS_STOP_CONNECTION only on connections secured using SSL version 3 or TLS version 1.0 or higher.  
• Request TTLS.Allow_HSTIMEOUT only when the security type is TTLS_SEC_SERVER or higher and the HandshakeTimeout value is not 0. |
| 87           | EMULTIHOP    | All             | A multi-hop address link was attempted. | Call your system administrator. |
| 88           | EDOTDOT      | All             | A cross-mount point was detected. This is not an error. | Call your system administrator. |
| 89           | EREMCHG      | All             | The remote address has changed. | Call your system administrator. |
| 90           | ECONNCLOSED  | All             | The connection was closed by a peer. | Check that the peer is running. |
| 113          | EBADF        | All             | Socket descriptor is not in correct range. The maximum number of socket descriptors is set by MAXDESC(). The default range is 0–49. | Reissue function with corrected socket descriptor. |
| 113          | EBADF        | Bind socket     | The socket descriptor is already being used. | Correct the socket descriptor. |
| 113          | EBADF        | Givensocket     | The socket has already been given. The socket domain is not AF_INET. | Correct the socket descriptor. |
| 113          | EBADF        | Select          | One of the specified descriptor sets is an incorrect socket descriptor. | Correct the socket descriptor. Set on Select() or Selectext(). |
| 113          | EBADF        | Takesocket      | The socket has already been taken. | Correct the socket descriptor. |
| 113          | EBADF        | Accept          | A Listen() has not been issued before the Accept() | Issue Listen() before Accept(). |
| 121          | EINVAL       | All             | An incorrect argument was specified. | Check and correct all function parameters. |
| 121          | EINVAL       | IOCTL (SIOCSAPPLDATA) | The input parameter is not a correctly formatted SetApplData structure.  
  - The SetAD.eye1 value is not valid.  
  - The SetAD.ver value is not valid.  
  - The storage pointed to by SetAD.ptr does not contain a correctly formatted SetADContainer structure.  
  - The SetAD.eye2 value is not valid.  
  - The SetAD.len value contains an incorrect length for the SetAD.ver version of the SetADContainer structure. | Check and correct all function parameters. |
<table>
<thead>
<tr>
<th>Error number</th>
<th>Message name</th>
<th>Socket API type</th>
<th>Error description</th>
<th>Programmer's response</th>
</tr>
</thead>
</table>
| 121          | EINVAL       | inet6_is_srcaddr | • One or more invalid IPV6_ADDR_\_PREFERENCES flags were specified  
• A scope ID was omitted for a link local IP address  
• A scope ID was specified for an IP address that is not link-local  
• The socket address length was not valid | Correct the function parameters |
| 122          | ECLOSED      |                 |                   |                        |
| 126          | ENMELONG     |                 |                   |                        |
| 134          | ENOSYS       | IOCTL           | The function is not implemented | Either configure the system to support the ioctl command or remove the ioctl command from your program. |
| 134          | ENOSYS       | IOCTL - siogifnameindex | The TCP/IP stack processing the siogifnameindex IOCTL is configured as a pure IPv4 TCP/IP stack. Additionally, UNIX System Services is configured to process as INET. | Either configure the system to support the ioctl command or remove the ioctl command from your program. |
| 136          | ENOTEMPT     |                 |                   |                        |
| 145          | E2BIG        | All             | The argument list is too long. | Eliminate excessive number of arguments. |
| 156          | EMVSINITIAL  | All             | Process initialization error. This indicates an z/OS UNIX process initialization failure. This is usually an indication that a proper OMVS RACF segment is not defined for the user ID associated with application. The RACF OMVS segment might not be defined or might contain errors such as an improper HOME() directory specification. | Attempt to initialize again. After ensuring that an OMVS Segment is defined, if the errno is still returned, call your MVS system programmer to have IBM service contacted. |
| 157          | EMISSSED     |                 |                   |                        |
| 157          | EMYSERR      |                 | An MVS environmental or internal error occurred. | |
| 1002         | EIBMsockoutofrange | Socket, Accept, Takesocket | A new socket cannot be created because the MAXSOC value, which is specified on the INITAPI call, has been reached. | Take either one of the following actions:  
• Verify whether all open sockets are intended to be in use:  
• Increase the MAXSOC value to a value that is appropriate for the current workload. If the default value is currently being used, you might be required to add the INITAPI call. |
<p>| 1003         | EIBMsockinuse | Socket          | A socket number assigned by the client interface code is already in use. | Use a different socket descriptor. |
| 1004         | EIBMrcverr   | All             | The request failed because of an IUCV error. This error is generated by the client stub code. | Ensure IUCV/VMCF is functional. |
| 1008         | EIBMconflict | All             | This request conflicts with a request already queued on the same socket. | Cancel the existing call or wait for its completion before reissuing this call. |
| 1009         | EIBMcancelled | All             | The request was canceled by the CANCEL call. | Informational, no action needed. |
| 1011         | EIBMbadtcpname | All             | A TCP/IP name that is not valid was detected. | Correct the name specified in the IBM_TCPIMAGE structure. |
| 1011         | EIBMbadtcpname | Setbmopt        | A TCP/IP name that is not valid was detected. | Correct the name specified in the IBM_TCPIMAGE structure. |
| 1011         | EIBMbadtcpname | INITAPI         | A TCP/IP name that is not valid was detected. | Correct the name specified on the IDENT option TCPNAME field. |
| 1012         | EIBMbadrequestcode | All             | A request code that is not valid was detected. | Contact your system administrator. |
| 1013         | EIBMbadconnectionstate | All             | A connection token that is not valid was detected; bad state. | Verify TCP/IP is active. |
| 1014         | EIBMunauthorizedcaller | All             | An unauthorized caller specified an authorized keyword. | Ensure user ID has authority for the specified operation. |</p>
<table>
<thead>
<tr>
<th>Error number</th>
<th>Message name</th>
<th>Socket API type</th>
<th>Error description</th>
<th>Programmer's response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1015</td>
<td>EIBMBA_READCOMPLETION</td>
<td>All</td>
<td>A connection token that is not valid was detected. There is no such connection.</td>
<td>Verify TCP/IP is active.</td>
</tr>
<tr>
<td>1016</td>
<td>EIBMTCPABEND</td>
<td>All</td>
<td>An abend occurred when TCP/IP was processing this request.</td>
<td>Reissue after previous call has completed.</td>
</tr>
<tr>
<td>1023</td>
<td>EIBMTERMERROR</td>
<td>All</td>
<td>Encountered a terminating error while processing.</td>
<td>Call your system administrator.</td>
</tr>
<tr>
<td>1026</td>
<td>EIBMINVDELETE</td>
<td>All</td>
<td>Delete requestor did not create the connection.</td>
<td>Delete the request from the process that created it.</td>
</tr>
<tr>
<td>1027</td>
<td>EIBMINVSOCKET</td>
<td>All</td>
<td>A connection token that is not valid was detected. No such socket exists.</td>
<td>Call your system programmer.</td>
</tr>
<tr>
<td>1028</td>
<td>EIBMINVTCPCONNECTION</td>
<td>All</td>
<td>Connection terminated by TCP/IP. The token was invalidated by TCP/IP.</td>
<td>Reestablish the connection to TCP/IP.</td>
</tr>
<tr>
<td>1032</td>
<td>EIBMSCALLINPROGRESS</td>
<td>All</td>
<td>Another call was already in progress.</td>
<td>Reissue after previous call has completed.</td>
</tr>
<tr>
<td>1036</td>
<td>EIBMNOACTIVE_TCP</td>
<td>All</td>
<td>TCP/IP is not installed or not active.</td>
<td>Correct TCP/IP name used.</td>
</tr>
<tr>
<td>1036</td>
<td>EIBMNOACTIVE_TCP</td>
<td>Select</td>
<td>EIBMNOACTIVE_TCP</td>
<td>Ensure TCP/IP is active.</td>
</tr>
<tr>
<td>1036</td>
<td>EIBMNOACTIVE_TCP</td>
<td>Gethsopts</td>
<td>No TCP/IP image was found.</td>
<td>Ensure TCP/IP is active.</td>
</tr>
<tr>
<td>1037</td>
<td>EIBMINVTSRBRUSERDATA</td>
<td>All</td>
<td>The request control block contained data that is not valid.</td>
<td>Call your system programmer.</td>
</tr>
<tr>
<td>1038</td>
<td>EIBMINVUSERDATA</td>
<td>All</td>
<td>The request control block contained user data that is not valid.</td>
<td>Check your function parameters and call your system programmer.</td>
</tr>
<tr>
<td>1040</td>
<td>EIBMSELECTEXPOST</td>
<td>SELECTEX</td>
<td>SELECTEX passed an ECB that was already posted.</td>
<td>Check whether the user's ECB was already posted.</td>
</tr>
<tr>
<td>1112</td>
<td>ECANCEL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1162</td>
<td>ENOPARTNERINFO</td>
<td>IOCTL (SIOCGPARTNERINFO)</td>
<td>The partner resides in a TCP/IP stack running a release that is earlier than V1R12, or the partner is not in the same sysplex.</td>
<td>Ensure that both endpoints reside in TCP/IP stacks that are running V1R12 or any later release, or check and modify the socket descriptor. If the partner is not in the same sysplex, security credentials will not be returned.</td>
</tr>
<tr>
<td>2001</td>
<td>EINVALDRsoCKETCALL</td>
<td>REXX</td>
<td>A syntax error occurred in the RXSOCKET parameter list.</td>
<td>Correct the parameter list passed to the REXX socket call.</td>
</tr>
<tr>
<td>2002</td>
<td>ECONSOLEINTERRUPT</td>
<td>REXX</td>
<td>A console interrupt occurred.</td>
<td>Retry the task.</td>
</tr>
<tr>
<td>2003</td>
<td>ESUBTASKINVALID</td>
<td>REXX</td>
<td>The subtask ID is incorrect.</td>
<td>Correct the subtask ID on the INITIALIZE call.</td>
</tr>
<tr>
<td>2004</td>
<td>ESUBTASKALREADYACTIVE</td>
<td>REXX</td>
<td>The subtask is already active.</td>
<td>Issue the INITIALIZE call only once in your program.</td>
</tr>
<tr>
<td>2005</td>
<td>ESUBTASKNOTACTIVE</td>
<td>REXX</td>
<td>The subtask is not active.</td>
<td>Issue the INITIALIZE call before any other socket call.</td>
</tr>
<tr>
<td>2006</td>
<td>ESOCKETNOTALLOCATED</td>
<td>REXX</td>
<td>The specified socket or needed control block could not be allocated.</td>
<td>Increase the user storage allocation for this job.</td>
</tr>
<tr>
<td>2007</td>
<td>EMAXSOCKETSREACHED</td>
<td>REXX</td>
<td>The maximum number of sockets has been reached.</td>
<td>Increase the number of allocate sockets, or decrease the number of sockets used by your program.</td>
</tr>
<tr>
<td>2009</td>
<td>ESOCKETNOTDEFINED</td>
<td>REXX</td>
<td>The socket is not defined.</td>
<td>Issue the SOCKET call before the call that fails.</td>
</tr>
<tr>
<td>2011</td>
<td>EDOMAINSERVERFAILURE</td>
<td>REXX</td>
<td>A Domain Name Server failure occurred.</td>
<td>Call your MVS system programmer.</td>
</tr>
<tr>
<td>2012</td>
<td>EINVALIDNAME</td>
<td>REXX</td>
<td>An incorrect name was received from the TCP/IP server.</td>
<td>Call your MVS system programmer.</td>
</tr>
<tr>
<td>2013</td>
<td>EINVALIDCLIENTID</td>
<td>REXX</td>
<td>An incorrect clientid was received from the TCP/IP server.</td>
<td>Call your MVS system programmer.</td>
</tr>
<tr>
<td>2014</td>
<td>EINVALIDFILENAME</td>
<td>REXX</td>
<td>An error occurred during NUCEXT processing.</td>
<td>Specify the correct translation table file name, or verify that the translation table is valid.</td>
</tr>
<tr>
<td>2016</td>
<td>EHOSTNOTFOUND</td>
<td>REXX</td>
<td>The host is not found.</td>
<td>Call your MVS system programmer.</td>
</tr>
<tr>
<td>2017</td>
<td>EPADDNOTFOUND</td>
<td>REXX</td>
<td>Address not found.</td>
<td>Call your MVS system programmer.</td>
</tr>
<tr>
<td>2019</td>
<td>ENORECOVERY</td>
<td>REXX</td>
<td>A non-recoverable failure occurred during the Resolver's processing of the GETHOSTBYADDR or GETHOSTBYDNAME call.</td>
<td>Contact the IBM support center.</td>
</tr>
<tr>
<td>2020</td>
<td>EINVALCOMBINATION</td>
<td>REXX</td>
<td>An invalid combination of IPV6_ADDR_{PREFERENCES} flags was received from the caller.</td>
<td>Correct the specified flags.</td>
</tr>
</tbody>
</table>
### Table 24. Sockets ERRNOs (continued)

<table>
<thead>
<tr>
<th>Error number</th>
<th>Message name</th>
<th>Socket API type</th>
<th>Error description</th>
<th>Programmer's response</th>
</tr>
</thead>
<tbody>
<tr>
<td>2021</td>
<td>EOPTNAMEMISMATCH</td>
<td>REXX</td>
<td>The caller specified an OPTNAME that is invalid for the LEVEL that it specified.</td>
<td>Correct either the OPTNAME or the LEVEL.</td>
</tr>
<tr>
<td>2022</td>
<td>EFLAGSMISMATCH</td>
<td>REXX</td>
<td>The caller issued a GETADDRINFO with conflicting FLAGS and EFLAGS parameters: either AL_EXT_FLAGS was specified with a null EFLAGS, or AL_EXT_FLAGS was not specified but EFLAGS was not null.</td>
<td>Correct either the FLAGS parameter or the EFLAGS parameter. A non-null EFLAGS should be specified if and only if AL_EXT_FLAGS is specified in the FLAGS.</td>
</tr>
<tr>
<td>2051</td>
<td>EFORMATTERROR</td>
<td>REXX</td>
<td>The name server was unable to interpret the query.</td>
<td>Contact the IBM support center.</td>
</tr>
<tr>
<td>3412</td>
<td>ENODATA</td>
<td></td>
<td>Message does not exist.</td>
<td></td>
</tr>
<tr>
<td>3416</td>
<td>ELINKED</td>
<td></td>
<td>Stream is linked.</td>
<td></td>
</tr>
<tr>
<td>3419</td>
<td>EERECURSE</td>
<td></td>
<td>Recursive attempt rejected.</td>
<td></td>
</tr>
<tr>
<td>3420</td>
<td>EASYNC</td>
<td></td>
<td>Asynchronous I/O scheduled. This is a normal, internal event that is NOT returned to the user.</td>
<td></td>
</tr>
<tr>
<td>3448</td>
<td>EUNATCH</td>
<td></td>
<td>The protocol required to support the specified address family is not available.</td>
<td></td>
</tr>
<tr>
<td>3464</td>
<td>ETERM</td>
<td></td>
<td>Operation terminated.</td>
<td></td>
</tr>
<tr>
<td>3474</td>
<td>EUNKNOWN</td>
<td></td>
<td>Unknown system state.</td>
<td></td>
</tr>
<tr>
<td>3495</td>
<td>EBADOBJ</td>
<td></td>
<td>You attempted to reference an object that does not exist.</td>
<td></td>
</tr>
<tr>
<td>3513</td>
<td>EOUTOFSTATE</td>
<td></td>
<td>Protocol engine has received a command that is not acceptable in its current state.</td>
<td></td>
</tr>
</tbody>
</table>

### Sockets extended ERRNOs

#### Table 25. Sockets extended ERRNOs

<table>
<thead>
<tr>
<th>Error code</th>
<th>Problem description</th>
<th>System action</th>
<th>Programmer’s response</th>
</tr>
</thead>
<tbody>
<tr>
<td>10100</td>
<td>An ESTAE macro did not complete normally.</td>
<td>End the call.</td>
<td>Call your MVS system programmer.</td>
</tr>
<tr>
<td>10101</td>
<td>A STORAGE OBTAIN failed.</td>
<td>End the call.</td>
<td>Increase MVS storage in the application's address space.</td>
</tr>
<tr>
<td>10108</td>
<td>The first call issued was not a valid first call.</td>
<td>End the call.</td>
<td>Almost all sockets programs that are written in COBOL, PL/I, or assembler language must issue the INITAPI call before they issue other sockets calls.</td>
</tr>
<tr>
<td>10110</td>
<td>LOAD of EZBSOH03 (alias EZASOH03) failed.</td>
<td>End the call.</td>
<td>Call the IBM Software Support Center.</td>
</tr>
<tr>
<td>10154</td>
<td>Errors were found in the parameter list for an IOCTL call.</td>
<td>Disable the subtask for interrupts. Return an error code to the caller.</td>
<td>Correct the IOCTL call. You might have incorrect sequencing of socket calls.</td>
</tr>
<tr>
<td>10155</td>
<td>The length parameter for an IOCTL call is less than or equal to 0.</td>
<td>Disable the subtask for interrupts. Return an error code to the caller.</td>
<td>Correct the IOCTL call. You might have incorrect sequencing of socket calls.</td>
</tr>
<tr>
<td>10156</td>
<td>The length parameter for an IOCTL call is 3200 (32 x 100).</td>
<td>Disable the subtask for interrupts. Return an error code to the caller.</td>
<td>Correct the IOCTL call. You might have incorrect sequencing of socket calls.</td>
</tr>
<tr>
<td>Error code</td>
<td>Problem description</td>
<td>System action</td>
<td>Programmer's response</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------</td>
<td>---------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>10159</td>
<td>A 0 or negative data length was specified for a READ or READV call.</td>
<td>Disable the subtask for interrupts. Return an error code to the caller.</td>
<td>Correct the length in the READ call.</td>
</tr>
<tr>
<td>10161</td>
<td>The REQARG parameter in the IOCTL parameter list is 0.</td>
<td>End the call.</td>
<td>Correct the program.</td>
</tr>
<tr>
<td>10163</td>
<td>A 0 or negative data length was found for a RECV, RECVFROM, or RECVMSG call.</td>
<td>Disable the subtask for interrupts. Sever the DLC path. Return an error code to the caller.</td>
<td>Correct the data length.</td>
</tr>
<tr>
<td>10167</td>
<td>The descriptor set size for a SELECT or SELECTEX call is less than or equal to 0.</td>
<td>Disable the subtask for interrupts. Return an error code to the caller.</td>
<td>Correct the SELECT or SELECTEX call. You might have incorrect sequencing of socket calls.</td>
</tr>
<tr>
<td>10168</td>
<td>The descriptor set size in bytes for a SELECT or SELECTEX call is greater than 8192. A number greater than the maximum number of allowed sockets (65534 is the maximum) has been specified.</td>
<td>Disable the subtask for interrupts. Return an error code to the caller.</td>
<td>Correct the descriptor set size.</td>
</tr>
<tr>
<td>10170</td>
<td>A 0 or negative data length was found for a SEND or SENDMSG call.</td>
<td>Disable the subtask for interrupts. Return an error code to the caller.</td>
<td>Correct the data length in the SEND call.</td>
</tr>
<tr>
<td>10174</td>
<td>A 0 or negative data length was found for a SENDTO call.</td>
<td>Disable the subtask for interrupts. Return an error code to the caller.</td>
<td>Correct the data length in the SENDTO call.</td>
</tr>
<tr>
<td>10178</td>
<td>The SETSOCKOPT option length is less than the minimum length.</td>
<td>Disable the subtask for interrupts. Return an error code to the caller.</td>
<td>Correct the OPTLEN parameter.</td>
</tr>
<tr>
<td>10179</td>
<td>The SETSOCKOPT option length is greater than the maximum length.</td>
<td>Disable the subtask for interrupts. Return an error code to the caller.</td>
<td>Correct the OPTLEN parameter.</td>
</tr>
<tr>
<td>10184</td>
<td>A data length of 0 was specified for a WRITE call.</td>
<td>Disable the subtask for interrupts. Return an error code to the caller.</td>
<td>Correct the data length in the WRITE call.</td>
</tr>
<tr>
<td>10186</td>
<td>A negative data length was specified for a WRITE or WRITEV call.</td>
<td>Disable the subtask for interrupts. Return an error code to the caller.</td>
<td>Correct the data length in the WRITE call.</td>
</tr>
<tr>
<td>10190</td>
<td>The GETHOSTNAME option length is not in the range 1 - 255.</td>
<td>Disable the subtask for interrupts. Return an error code to the caller.</td>
<td>Correct the length parameter.</td>
</tr>
<tr>
<td>Error code</td>
<td>Problem description</td>
<td>System action</td>
<td>Programmer’s response</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------</td>
</tr>
<tr>
<td>10193</td>
<td>The SETSOCKOPT or GETSOCKOPT option length is shorter than the minimum length or longer than the maximum length.</td>
<td>End the call.</td>
<td>Correct the length parameter.</td>
</tr>
<tr>
<td>10197</td>
<td>The application issued an INITAPI call after the connection was already established.</td>
<td>Bypass the call.</td>
<td>Correct the logic that produces the INITAPI call that is not valid.</td>
</tr>
<tr>
<td>10198</td>
<td>The maximum number of sockets specified for an INITAPI exceeds 65535.</td>
<td>Return to the user.</td>
<td>Correct the INITAPI call.</td>
</tr>
<tr>
<td>10200</td>
<td>The first call issued was not a valid first call.</td>
<td>End the call.</td>
<td>Almost all sockets programs that are written in COBOL, PL/I, or assembler language must issue the INITAPI call before they issue other sockets calls.</td>
</tr>
<tr>
<td>10202</td>
<td>The RETARG parameter in the IOCTL call is 0.</td>
<td>End the call.</td>
<td>Correct the parameter list. You might have incorrect sequencing of socket calls.</td>
</tr>
<tr>
<td>10203</td>
<td>The requested socket number is a negative value.</td>
<td>End the call.</td>
<td>Correct the requested socket number.</td>
</tr>
<tr>
<td>10205</td>
<td>The requested socket number is a duplicate.</td>
<td>End the call.</td>
<td>Correct the requested socket number.</td>
</tr>
<tr>
<td>10208</td>
<td>The NAMELEN parameter for a GETHOSTBYNAME call was not specified.</td>
<td>End the call.</td>
<td>Correct the NAMELEN parameter. You might have incorrect sequencing of socket calls.</td>
</tr>
<tr>
<td>10209</td>
<td>The NAME parameter on a GETHOSTBYNAME call was not specified.</td>
<td>End the call.</td>
<td>Correct the NAME parameter. You might have incorrect sequencing of socket calls.</td>
</tr>
<tr>
<td>10210</td>
<td>The HOSTENT parameter on a GETHOSTBYNAME or GETHOSTBYADDR call was not specified.</td>
<td>End the call.</td>
<td>Correct the HOSTENT parameter. You might have incorrect sequencing of socket calls.</td>
</tr>
<tr>
<td>10211</td>
<td>The HOSTADDR parameter on a GETHOSTBYNAME or GETHOSTBYADDR call is incorrect.</td>
<td>End the call.</td>
<td>Correct the HOSTADDR parameter. You might have incorrect sequencing of socket calls.</td>
</tr>
<tr>
<td>10212</td>
<td>The resolver program failed to load correctly for a GETHOSTBYNAME or GETHOSTBYADDR call.</td>
<td>End the call.</td>
<td>Check the JOBLIB, STEPLIB, and linklib datasets and rerun the program.</td>
</tr>
<tr>
<td>10213</td>
<td>Not enough storage is available to allocate the HOSTENT structure.</td>
<td>End the call.</td>
<td>Increase the user storage allocation for this job.</td>
</tr>
<tr>
<td>10214</td>
<td>The HOSTENT structure was not returned by the resolver program.</td>
<td>End the call.</td>
<td>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.</td>
</tr>
<tr>
<td>Error code</td>
<td>Problem description</td>
<td>System action</td>
<td>Programmer’s response</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------</td>
<td>---------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>10215</td>
<td>The APITYPE parameter on an INITAPI call instruction was not 2 or 3.</td>
<td>End the call.</td>
<td>Correct the APITYPE parameter.</td>
</tr>
<tr>
<td>10218</td>
<td>The application programming interface (API) cannot locate the specified TCP/IP.</td>
<td>End the call.</td>
<td>Ensure that an API that supports the performance improvements related to CPU conservation is installed on the system and verify that a valid TCP/IP name was specified on the INITAPI call. This error call might also mean that EZASOKIN could not be loaded.</td>
</tr>
<tr>
<td>10219</td>
<td>The NS parameter is greater than the maximum socket for this connection.</td>
<td>End the call.</td>
<td>Correct the NS parameter on the ACCEPT, SOCKET or TAKESOCKET call.</td>
</tr>
<tr>
<td>10221</td>
<td>The AF parameter of a SOCKET call is not AF_INET.</td>
<td>End the call.</td>
<td>Set the AF parameter equal to AF_INET.</td>
</tr>
<tr>
<td>10222</td>
<td>The SOCTYPE parameter of a SOCKET call must be stream, datagram, or raw (1, 2, or 3).</td>
<td>End the call.</td>
<td>Correct the SOCTYPE parameter.</td>
</tr>
<tr>
<td>10223</td>
<td>No ASYNC parameter specified for INITAPI with APITYPE=3 call.</td>
<td>End the call.</td>
<td>Add the ASYNC parameter to the INITAPI call.</td>
</tr>
<tr>
<td>10224</td>
<td>The IOVCNT parameter is less than or equal to 0, for a READV, RECVMSG, SENDMSG, or WRITEV call.</td>
<td>End the call.</td>
<td>Correct the IOVCNT parameter.</td>
</tr>
<tr>
<td>10225</td>
<td>The IOVCNT parameter is greater than 120, for a READV, RECVMSG, SENDMSG, or WRITEV call.</td>
<td>End the call.</td>
<td>Correct the IOVCNT parameter.</td>
</tr>
<tr>
<td>10226</td>
<td>Not valid COMMAND parameter specified for a GETIBMOPT call.</td>
<td>End the call.</td>
<td>Correct the COMMAND parameter of the GETIBMOPT call.</td>
</tr>
<tr>
<td>10229</td>
<td>A call was issued on an APITYPE=3 connection without an ECB or REQAREA parameter.</td>
<td>End the call.</td>
<td>Add an ECB or REQAREA parameter to the call.</td>
</tr>
<tr>
<td>10300</td>
<td>Termination is in progress for either the CICS transaction or the socket interface.</td>
<td>End the call.</td>
<td>None.</td>
</tr>
<tr>
<td>10330</td>
<td>A SELECT call was issued without a MAXSOC value and a TIMEOUT parameter.</td>
<td>End the call.</td>
<td>Correct the call by adding a TIMEOUT parameter.</td>
</tr>
<tr>
<td>10331</td>
<td>A call that is not valid was issued while in SRB mode.</td>
<td>End the call.</td>
<td>Get out of SRB mode and reissue the call.</td>
</tr>
<tr>
<td>10332</td>
<td>A SELECT call is invoked with a MAXSOC value greater than that which was returned in the INITAPI function (MAXSNO field).</td>
<td>End the call.</td>
<td>Correct the MAXSOC parameter and reissue the call.</td>
</tr>
<tr>
<td>Error code</td>
<td>Problem description</td>
<td>System action</td>
<td>Programmer’s response</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------------------</td>
<td>------------------------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>10334</td>
<td>An error was detected in creating the data areas required to process the socket call.</td>
<td>End the call.</td>
<td>Call the IBM Software Support Center.</td>
</tr>
<tr>
<td>10999</td>
<td>An abend has occurred in the subtask.</td>
<td>Write message EZY1282E to the system console. End the subtask and post the TRUE ECB.</td>
<td>If the call is correct, call your system programmer.</td>
</tr>
<tr>
<td>20000</td>
<td>An unknown function code was found in the call.</td>
<td>End the call.</td>
<td>Correct the SOC-FUNCTION parameter.</td>
</tr>
<tr>
<td>20001</td>
<td>The call passed an incorrect number of parameters.</td>
<td>End the call.</td>
<td>Correct the parameter list.</td>
</tr>
<tr>
<td>20002</td>
<td>The user ID associated with the program linking EZACIC25 does not have the proper authority to execute a CICS EXTRACT EXIT.</td>
<td>End the call.</td>
<td>Start the CICS socket interface before executing this call.</td>
</tr>
<tr>
<td>20003</td>
<td>The CICS socket interface is not in operation.</td>
<td>End the call.</td>
<td>Contact the CICS system programmer. Ensure that the user ID being used is permitted to have at least UPDATE access to the EXITPROGRAM resource.</td>
</tr>
<tr>
<td>20004</td>
<td>The CICS socket TRUE failed to suspend the task.</td>
<td>End the call.</td>
<td>Call the IBM Software Support Center.</td>
</tr>
<tr>
<td>20005</td>
<td>The socket task was purged by CICS while the task was being suspended by the CICS socket TRUE.</td>
<td>End the call.</td>
<td>None.</td>
</tr>
</tbody>
</table>
Appendix C. GETSOCKOPT/SETSOCKOPT command values

You can use the following table to determine the decimal or hexadecimal value associated with the GETSOCKOPT/SETSOCKOPT OPTNAMES supported by the APIs discussed in this document.

The command names are shown with underscores for the assembler language. The underscores should be changed to dashes if using the COBOL programming language.

Languages that cannot easily handle binary values, such as COBOL, should use the decimal value associated with the command where necessary.

The hexadecimal value can be used in Macro, Assembler and PL/I programs.

Table 26. GETSOCKOPT/SETSOCKOPT command values for Macro, Assembler, COBOL and PL/I

<table>
<thead>
<tr>
<th>Command name</th>
<th>Decimal value</th>
<th>Hex value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP_ADD_MEMBERSHIP</td>
<td>1048581</td>
<td>X'00100005'</td>
</tr>
<tr>
<td>IP_ADD_SOURCE_MEMBERSHIP</td>
<td>1048588</td>
<td>X'0010000C'</td>
</tr>
<tr>
<td>IP_BLOCK_SOURCE</td>
<td>1048586</td>
<td>X'0010000A'</td>
</tr>
<tr>
<td>IP_DROP_MEMBERSHIP</td>
<td>1048582</td>
<td>X'00100006'</td>
</tr>
<tr>
<td>IP_DROP_SOURCE_MEMBERSHIP</td>
<td>1048589</td>
<td>X'0010000D'</td>
</tr>
<tr>
<td>IP_MULTICAST_IF</td>
<td>1048583</td>
<td>X'00100007'</td>
</tr>
<tr>
<td>IP_MULTICAST_LOOP</td>
<td>1048580</td>
<td>X'00100004'</td>
</tr>
<tr>
<td>IP_MULTICAST_TTL</td>
<td>1048579</td>
<td>X'00100003'</td>
</tr>
<tr>
<td>IP_UNBLOCK_SOURCE</td>
<td>1048587</td>
<td>X'0010000B'</td>
</tr>
<tr>
<td>IPV6_ADDR_PREFERENCES</td>
<td>65568</td>
<td>X'00010020'</td>
</tr>
<tr>
<td>IPV6_JOIN_GROUP</td>
<td>65541</td>
<td>X'00010005'</td>
</tr>
<tr>
<td>IPV6_LEAVE_GROUP</td>
<td>65542</td>
<td>X'00010006'</td>
</tr>
<tr>
<td>IPV6_MULTICAST_HOPS</td>
<td>65545</td>
<td>X'00010009'</td>
</tr>
<tr>
<td>IPV6_MULTICAST_IF</td>
<td>65543</td>
<td>X'00010007'</td>
</tr>
<tr>
<td>IPV6_MULTICAST_LOOP</td>
<td>65540</td>
<td>X'00010004'</td>
</tr>
<tr>
<td>IPV6_UNICAST_HOPS</td>
<td>65539</td>
<td>X'00010003'</td>
</tr>
<tr>
<td>IPV6_V6ONLY</td>
<td>65546</td>
<td>X'0001000A'</td>
</tr>
<tr>
<td>MCAST_BLOCK_SOURCE</td>
<td>1048620</td>
<td>X'0010002C'</td>
</tr>
<tr>
<td>MCAST_JOIN_GROUP</td>
<td>1048616</td>
<td>X'00100028'</td>
</tr>
<tr>
<td>MCAST_JOIN_SOURCE_GROUP</td>
<td>1048618</td>
<td>X'0010002A'</td>
</tr>
<tr>
<td>MCAST_LEAVE_GROUP</td>
<td>1048617</td>
<td>X'00100029'</td>
</tr>
<tr>
<td>MCAST_LEAVE_SOURCE_GROUP</td>
<td>1048619</td>
<td>X'0010002B'</td>
</tr>
<tr>
<td>MCAST_UNBLOCK_SOURCE</td>
<td>1048621</td>
<td>X'0010002D'</td>
</tr>
<tr>
<td>SO_BROADCAST</td>
<td>32</td>
<td>X'000000020'</td>
</tr>
<tr>
<td>SO_ERROR</td>
<td>4103</td>
<td>X'00010007'</td>
</tr>
<tr>
<td>SO_LINGER</td>
<td>128</td>
<td>X'00000080'</td>
</tr>
</tbody>
</table>
### Table 26. GETSOCKOPT/SETSOCKOPT command values for Macro, Assembler, COBOL and PL/I (continued)

<table>
<thead>
<tr>
<th>Command name</th>
<th>Decimal value</th>
<th>Hex value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SO_KEEPALIVE</td>
<td>8</td>
<td>X'00000008'</td>
</tr>
<tr>
<td>SO_OOBINLINE</td>
<td>256</td>
<td>X'00001000'</td>
</tr>
<tr>
<td>SO_RCVBUF</td>
<td>4098</td>
<td>X'00001002'</td>
</tr>
<tr>
<td>SO_RCVTIMEO</td>
<td>4102</td>
<td>X'00001006'</td>
</tr>
<tr>
<td>SO_REUSEADDR</td>
<td>4</td>
<td>X'00000004'</td>
</tr>
<tr>
<td>SO_SNDBUF</td>
<td>4097</td>
<td>X'00001001'</td>
</tr>
<tr>
<td>SO_SNDTIMEO</td>
<td>4101</td>
<td>X'00001005'</td>
</tr>
<tr>
<td>SO_TYPE</td>
<td>4104</td>
<td>X'00001008'</td>
</tr>
<tr>
<td>TCP_KEEPALIVE</td>
<td>2147483654</td>
<td>X'80000008'</td>
</tr>
<tr>
<td>TCP_NODELAY</td>
<td>2147483649</td>
<td>X'80000001'</td>
</tr>
</tbody>
</table>

### Table 27. GETSOCKOPT/SETSOCKOPT optname value for C programs

<table>
<thead>
<tr>
<th>Option name</th>
<th>Decimal value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP_ADD_MEMBERSHIP</td>
<td>5</td>
</tr>
<tr>
<td>IP_ADD_SOURCE_MEMBERSHIP</td>
<td>12</td>
</tr>
<tr>
<td>IP_BLOCK_SOURCE</td>
<td>10</td>
</tr>
<tr>
<td>IP_DROP_MEMBERSHIP</td>
<td>6</td>
</tr>
<tr>
<td>IP_DROP_SOURCE_MEMBERSHIP</td>
<td>13</td>
</tr>
<tr>
<td>IP_MULTICAST_IF</td>
<td>7</td>
</tr>
<tr>
<td>IP_MULTICAST_LOOP</td>
<td>4</td>
</tr>
<tr>
<td>IP_MULTICAST_TTL</td>
<td>3</td>
</tr>
<tr>
<td>IP_UNBLOCK_SOURCE</td>
<td>11</td>
</tr>
<tr>
<td>MCAST_BLOCK_SOURCE</td>
<td>44</td>
</tr>
<tr>
<td>MCAST_JOIN_GROUP</td>
<td>40</td>
</tr>
<tr>
<td>MCAST_JOIN_SOURCE_GROUP</td>
<td>42</td>
</tr>
<tr>
<td>MCAST_LEAVE_GROUP</td>
<td>41</td>
</tr>
<tr>
<td>MCAST_LEAVE_SOURCE_GROUP</td>
<td>43</td>
</tr>
<tr>
<td>MCAST_UNBLOCK_SOURCE</td>
<td>45</td>
</tr>
<tr>
<td>SO_ACCEPTCONN</td>
<td>2</td>
</tr>
<tr>
<td>SO_BROADCAST</td>
<td>32</td>
</tr>
<tr>
<td>SO_CLUSTERCONNTYPE</td>
<td>16385</td>
</tr>
<tr>
<td>SO_DEBUG</td>
<td>1</td>
</tr>
<tr>
<td>SO_ERROR</td>
<td>4103</td>
</tr>
<tr>
<td>SO_KEEPALIVE</td>
<td>8</td>
</tr>
<tr>
<td>SO_LINGER</td>
<td>128</td>
</tr>
<tr>
<td>SO_OOBINLINE</td>
<td>256</td>
</tr>
<tr>
<td>SO_RCVBUF</td>
<td>4098</td>
</tr>
<tr>
<td>SO_REUSEADDR</td>
<td>4</td>
</tr>
<tr>
<td>SO_SNDBUF</td>
<td>4097</td>
</tr>
</tbody>
</table>
Table 27. GETSOCKOPT/SETSOCKOPT optname value for C programs (continued)

<table>
<thead>
<tr>
<th>Option name</th>
<th>Decimal value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SO_TYPE</td>
<td>4104</td>
</tr>
<tr>
<td>TCP_KEEPALIVE</td>
<td>8</td>
</tr>
<tr>
<td>TCP_NODELAY</td>
<td>1</td>
</tr>
</tbody>
</table>
Appendix D. CICS sockets messages

This topic contains CICS socket interface messages.

EZY1218—EZY1366

EZY1218E: mm/dd/yy hh:mm:ss PROGRAM programname DISABLED TRANID= transactionid PARTNER INET ADDR= inetaddress PORT= portnumber

Explanation

The Listener checked the status of the program associated with the transaction. It was not enabled.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

programname is the name of the program that is associated with the transaction requested by the connecting client.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action

Listener continues.

Operator response

Use CEMT to determine and correct the status of the program.

System programmer response

None.

Module

EZACIC02

Destination

LISTENER
EZY1219E: mm/dd/yy hh:mm:ss UNEXPECTED eventtype EVENT IN LISTENER transactionid FROM CLIENT IP ADDRESS ipaddress PORT portnumber

Explanation

The CICS Listener was notified about an unexpected event.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

eventtype is the type of event: READ, WRITE, or EXCEPTION.

transactionid is the name of the Listener’s CICS transaction.

ipaddress is the remote IP address of the client.

portnumber is the remote port number of the client.

System action

The Listener closes the connection and continues processing.

Operator response

Contact the system programmer.

System programmer response

If the event type is EXCEPTION, investigate whether or not the client is attempting to send out-of-band data. If necessary, have the client avoid sending out-of-band data. If the event type is not EXCEPTION or the client is not attempting to send out-of-band data, then contact the IBM Software Support Center.

Module

EZACIC02

Destination

LISTENER

EZY1220E: mm/dd/yy hh:mm:ss READ FAILURE ON CONFIGURATION FILE PHASE=phase EIBRESP2=response

Explanation

EZACIC21 was unable to read the IP CICS Sockets configuration file, EZACONFG.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

phase is the IP CICS Sockets initialization phase.
response is the response from CICS when reading the IP CICS Sockets configuration file.

**System action**

If the ABEND code is AEXY, then the listener ends normally. Otherwise, the listener ends with an ABEND code of EZAL.

**Operator response**

Notify the CICS system programmer.

**System programmer response**

Use the EIBRESP2 value to determine the problem and correct the file. See [http://www-01.ibm.com/software/htp/cics/library/](http://www-01.ibm.com/software/htp/cics/library/) for information about EIBRESP2 values. If the EIBRESP2 value is zero, then the EZACONFG file has been defined as remote. If this is the configuration file you want, then verify that no CICS Sockets programs can run directly in the file owning region. This can cause the file to become disabled. Ensure that EZACIC20 is not in the file owning region PLT, and that the EZAC and EZAO transactions are unable to run directly in the file owning region. Attempts to open the file will fail if the file is defined with a value of YES specified in the ADD, DELETE, or UPDATE parameters in the CICS file definition in more than one CICS region.

**Module**

EZACIC21

**Destination**

INITIALIZATION

**EZY1221E: mm/dd/yy hh:mm:ss CICS SOCKETS ENABLE FAILURE EIBRCODE BYTE2 = resp_code**

**Explanation**

The attempt to enable the task related user exit (TRUE) failed.

*mm/dd/yy* is the date (month/day/year) of the message.

*hh:mm:ss* is the time (hours:minutes:seconds) of the message.

*resp_code* is the CICS response code from attempting to enable IP CICS Sockets Task Related User Exit (TRUE).

**System action**

Terminate the transaction.

**Operator response**

Notify the CICS system programmer.
System programmer response

Use the EIBRESP2 value to determine the problem and correct the file. An EIBRCODE BYTE2 value of 20 indicates the TRUE is already enabled. This will occur if you disable the interface using EZAO,STOP,CICS transaction and then immediately issue EZAO,START,CICS transaction before the Task Related User Exit (TRUE) is completely disabled from the previous EZAO,STOP,CICS transaction. See [http://www-01.ibm.com/software/htp/cics/library/](http://www-01.ibm.com/software/htp/cics/library/) for information about EIBRCODEs.

Module

EZACIC21

Destination

INITIALIZATION

EZY1222E: mm/dd/yy hh:mm:ss CICS/SOCKETS REGISTRATION FAILURE RETURN code= return_code

Explanation

The attempt to register the CICS Sockets Feature to z/OS failed.

System action

Terminate the transaction.

Operator response

Contact your System Administrator.

System programmer response

See the [z/OS MVS Programming: Product Registration](https://www.ibm.com/support/knowledgecenter/SSLTBW_2.2.0/com.ibm.zos.v2r2.bios.doc/contents/ProductRegistration.html) for information about the values for return_code.

Module

EZACIC21

Destination

INITIALIZATION

EZY1223E: mm/dd/yy hh:mm:ss CICS/SOCKETS ATTACH FAILURE RETURN CODE = return_code REASON CODE = reason_code

Explanation

An attempt to attach one of the pool subtasks failed.
System action

Stop attaching pool subtasks. The size of the pool is determined by the number of subtasks successfully attached.

Operator response

Contact the CICS system programmer.

System programmer response

See the z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN for information about the values for return_code and reason_code and make appropriate adjustments to your CICS environment.

Module

EZACIC21

Destination

INITIALIZATION

EZY1224I: mm/dd/yy hh:mm:ss CICS/SOCKETS INITIALIZATION SUCCESSFUL USING tasking_method

Explanation

The CICS socket interface has completed initialization successfully.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

tasking_method is the tasking method used to support the EZASOKET calls. The possible methods are:

Reusable MVS subtasks

Signifies that the IP CICS socket interface is using MVS subtasks from the pool generated according to the value specified on the NTASKS configuration parameter.

Non-reusable MVS subtasks

Signifies that the IP CICS socket interface is attaching an MVS subtask for each IP CICS Sockets-enabled application because NTASKS=0.

Open Transaction Environment

Signifies that the IP CICS socket interface is enabled to use CICS Open Transaction Environment. All EZASOKET calls will be processed on an Open API, L8, TCB. Programs calling EZASOKET should be coded to threadsafe programming standards and defined to CICS as CONCURRENCY(THREADSAFE) to benefit from this environment.

System action

Continue with execution.
Operator response

None.

System programmer response

None.

**Module**

EZACIC21

**Destination**

INITIALIZATION

**EZY1225E: mm/dd/yy hh:mm:ss STARTBR FAILURE ON CICS/SOCKETS CONFIGURATION FILE PHASE=xx EIBRESP2=rrrrrr**

**Explanation**

The STARTBR command used for the configuration file has failed.

**System action**

Terminate the transaction.

**Operator response**

Contact the CICS system programmer.

**System programmer response**

Use the EIBRESP2 value to determine the problem. Check the CICS definition of the Configuration file to ensure the browse operation is permitted. See [http://www-01.ibm.com/software/htp/cics/library/](http://www-01.ibm.com/software/htp/cics/library/) for information about EIBRESP2 values.

**Module**

EZACIC21

**Destination**

INITIALIZATION

**EZY1226E: mm/dd/yy hh:mm:ss READNEXT FAILURE ON CICS/SOCKETS CONFIGURATION FILE PHASE=xx EIBRESP2=rrrrrr**

**Explanation**

The READNEXT command used for the configuration file has failed.
System action

Terminate the transaction.

Operator response

Contact the CICS system programmer.

System programmer response

Use the EIBRESP2 value to determine the problem. Check the CICS definition of the Configuration file to ensure the browse operation is permitted. See [http://www-01.ibm.com/software/htp/cics/library/](http://www-01.ibm.com/software/htp/cics/library/) for information about EIBRESP2 values.

Module

EZACIC21

Destination

INITIALIZATION

EZY1227E: mm/dd/yy hh:mm:ss CICS/SOCKETS INVALID LISTENER TRANID = tran

Explanation

The Listener transaction tran was not defined to CICS.

System action

Terminate Listener Initialization.

Operator response

Use CICS facilities to define the Listener transaction and program. Then use EZAO to start the Listener.

System programmer response

None.

Module

EZACIC21

Destination

INITIALIZATION
EZY1228E: *mm/dd/yy hh:mm:ss* CICS/ SOCKETS LISTENER TRANSACTION tran DISABLED

**Explanation**

The Listener transaction tran could not be started because it was disabled.

**System action**

Terminate Listener Initialization.

**Operator response**

Use CICS facilities to enable the transaction and then start the Listener using EZAO.

**System programmer response**

None.

**Module**

EZACIC21

**Destination**

INITIALIZATION

---

EZY1229E: *mm/dd/yy hh:mm:ss* CICS SOCKETS LISTENER TRANSACTION tran NOT AUTHORIZED

**Explanation**

The Listener transaction tran could not be started because it was not authorized.

**System action**

Terminate Listener Initialization.

**Operator response**

Use CICS facilities to authorize starting the Listener transaction and then start the Listener using EZAO.

**System programmer response**

None.

**Module**

EZACIC21

**Destination**

INITIALIZATION
EZY1246E: mm/dd/yy hh:mm:ss CICS SOCKETS LISTENER
PROGRAM ID mmmmmmmmm INVALID

Explanation

The Listener transaction could not be started because program mmmmmmmmm is not defined.

System action

Terminate Listener Initialization.

Operator response

If the program ID is correct, use CICS facilities to define it. If it is not correct, use the EZAC transaction to correct the CICS Sockets Configuration file.

System programmer response

None.

Module

EZACIC21

Destination

INITIALIZATION

EZY1247E: mm/dd/yy hh:mm:ss CICS SOCKETS LISTENER
PROGRAM ID mmmmmmmmm DISABLED

Explanation

The Listener transaction could not be started because program mmmmmmmmm is disabled.

System action

Terminate Listener Initialization.

Operator response

Use CICS facilities to enable the program and then use EZAO to start the Listener.

System programmer response

None.

Module

EZACIC21

Destination

INITIALIZATION
EZY1250E: mm/dd/yy hh:mm:ss CICS/SOCKETS LISTENER tran NOT ON CONFIGURATION FILE

Explanation
The Listener transaction tran is not defined on the CICS Sockets configuration file.

System action
Terminate Listener Initialization.

Operator response
If the Listener transaction name is correct, use the EZAC transaction to define it on the CICS Configuration file. If the name is not correct, correct it on the EZAO transaction.

System programmer response
None.

Module
EZACIC21

Destination
INITIALIZATION

EZY1251E: mm/dd/yy hh:mm:ss CICS SOCKETS MODULE mmmmmmmmm ABEND xxxx

Explanation
The CICS Sockets module mmmmmmmmm has abended.

System action
Terminate the transaction.

Operator response
Contact the IBM Software Support Center.

System programmer response
None.

Module
EZACIC21

Destination
INITIALIZATION
**EZY1252E**: `mm/dd/yy hh:mm:ss UNABLE TO LOAD EZASOH03
ERROR CODE= error_code REASON CODE= reason_code

Explanation

During CICS Sockets initialization, the attempt to load module EZASOH03 failed.

**System action**

Terminate Initialization.

**Operator response**

Contact the CICS system programmer.

**System programmer response**

See the [z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU](https://www.ibm.com) for information about the values for `error_code` and `reason_code` to determine why the module would not load. Also, look for associated MVS messages.

**Module**

EZACIC21

**EZY1253E**: `mm/dd/yy hh:mm:ss CICS/sockets listener tran
not on configuration file

Explanation

An EZAO STOP LISTENER transaction was run with an invalid Listener name.

**System action**

Present the panel to correct the name.

**Operator response**

Correct the name and retry termination.

**System programmer response**

None.

**Module**

EZACIC22

**Destination**

TERMINATION
EZY1254E: \textit{mm/dd/yy hh:mm:ss} CACHE FILE ERROR RESP2
VALUE ***** CALL # *

\textbf{Explanation}

An error occurred on a cache file operation.

\textbf{System action}

Return to the calling program with an error response.

\textbf{Operator response}

Contact the CICS system programmer.

\textbf{System programmer response}


\textbf{Module}

EZACIC25

\textbf{Destination}

DOMAIN NAME SERVER FUNCTION

---

EZY1255E: \textit{mm/dd/yy hh:mm:ss} TEMPORARY STORAGE ERROR RESP2 VALUE ***** CALL # *

\textbf{Explanation}

An error occurred on a temporary storage operation in EZACIC25.

\textbf{System action}

Return to the calling program with an error response.

\textbf{Operator response}


\textbf{System programmer response}

None.

\textbf{Module}

EZACIC25
Destination

DOMAIN NAME SERVER FUNCTION

EZY1256E: mm/dd/yy hh:mm:ss CICS SOCKETS INTERFACE NOT ENABLED PRIOR TO LISTENER STARTUP

Explanation

An attempt to start a Listener was made when the CICS socket interface was inactive.

System action

Return error and terminate transaction EZAO.

Operator response

Use transaction EZAO to start the CICS socket interface prior to starting the Listener.

System programmer response

None.

Module

EZACIC21

Destination

INITIALIZATION

EZY1258I: module ENTRY POINT IS address

Explanation

This message displays the entry point address of a module.

module is the name of the module.

address is the entry point address of the module.

System action

Processing continues.

Operator response

None.

System programmer response

None.
Module

EZACIC01, EZACIC02

**EZY1259E: mm/dd/yy hh:mm:ss IOCTL CALL FAILURE**
**TRANSACTION=transactionid TASKID=tasknumber ERRNO=errno**

**Explanation**

Listener transaction transactionid experienced a failure on the IOCTL call.

In the message text:

- **mm/dd/yy**
  - The date (month/day/year) of the message.

- **hh:mm:ss**
  - The time (hours:minutes:seconds) of the message.

- **transactionid**
  - The name of the transaction under which the Listener is executing.

- **tasknumber**
  - The CICS task number of the Listener task.

- **errno**
  - is the UNIX System Services return code. These return codes are listed in the sockets and sockets extended return codes (ERRNOs) in z/OS Communications Server: IP and SNA Codes.

**System action**

If the error is during initialization of the Listener, then the Listener transaction transactionid terminates. Otherwise, the Listener closes the socket that was being processed and resumes normal processing.

**Operator response**

Use the errno value to determine the cause of the failure.

**System programmer response**

None.

Module

EZACIC02

Destination

LISTENER

**EZY1260E: mm/dd/yy hh:mm:ss EZACIC03 ATTACH FAILED**
**GPR15=xxxxxxxx ERRNO=errno TRAN=tran TASK=cicstask**

**Explanation**

An ATTACH for an MVS subtask has failed. The reason code is in GPR 15.
errno is the UNIX System Services return code. These return codes are listed in the sockets and sockets extended return codes (ERRNOs) in [z/OS Communications Server: IP and SNA Codes]

**System action**

The task related user exit (TRUE) for this transaction is disabled. The transaction abends with an AEY9.

**Operator response**

Contact the CICS system programmer.

**System programmer response**

Determine the cause for the ATTACH failure and correct.

**Module**

EZACIC01

**Destination**

TASK RELATED USER EXIT (TRUE)

EZY1261I: mm/dd/yy hh:mm:ss EZACIC03 ATTACH SUCCESSFUL, TCB ADDRESS= tcbaddr TERM=term TRAN=tran TASK=cicstask

**Explanation**

An ATTACH for an MVS subtask was successful. This message is produced only for Listeners and for those tasks that cannot be accommodated within the pool of reusable tasks.

**Result:** If you specify the character L as the last character in the subtask ID parameter of an INITAPI socket command, then the IP CICS Socket task related user exit (TRUE) assumes that the CICS transaction is a listener causing the TRUE to attach a new task to support the listener's socket commands.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

tcbaddr is the address of the Task Control Block (TCB) being attached.

term is the CICS terminal ID associated with the CICS transaction identified by tran.

tran is the name of the CICS transaction that was requested.

cicstask is the task number of the CICS transaction identified by tran.

**System action**

Processing continues.
Operator response

If this message happens frequently, increase the size of the reusable task pool, NTASKS, for this CICS. Increasing NTASKS appropriately will prevent overhead incurred with attaching the subtask. See the EZACICD TYPE parameter in for information the NTASKS value.

System programmer response

None.

Module

EZACIC01

Destination

TASK RELATED USER EXIT (TRUE)

EZY1262E: mm/dd/yy hh:mm:ss GWA ADDRESS INVALID
UEPGAA=xxxxxxxx TRAN=tran TASK=cicstask

Explanation

The task related user exit (TRUE) detected an invalid GWA address.

System action

The TRUE is disabled and the task abends with an AEY9.

Operator response

Use EZAO to stop (immediate) and start the CICS socket interface. If the problem repeats, contact the IBM Software Support Center.

System programmer response

None.

Module

EZACIC01

Destination

TASK RELATED USER EXIT (TRUE)

EZY1263E: mm/dd/yy hh:mm:ss TIE ADDRESS INVALID
UEPGAA=xxxxxxxx TRAN=tran TASK=cicstask

Explanation

The task related user exit (TRUE) detected an invalid TIE address.
System action

The TRUE is disabled and the task abends with an AEY9.

Operator response

Use EZAO to stop (immediate) and start the CICS socket interface. If the problem repeats, contact the IBM Software Support Center.

System programmer response

None.

Module

EZACIC01

Destination

TASK RELATED USER EXIT (TRUE)

EZY1264E: mm/dd/yy hh:mm:ss FLAG WORD ADDRESS
INVALID UEPFLAGS= xxxxxxxx ERRNO=errno TRAN=tran
TASK=cicstask

Explanation

The task related user exit (TRUE) detected an invalid flag word address.

errno is the UNIX System Services return code. These return codes are listed in the sockets and sockets extended return codes (ERRNOs) in z/OS Communications Server: IP and SNA Codes.

System action

The TRUE is disabled and the task abends with an AEY9.

Operator response

Use EZAO to stop (immediate) and start the CICS socket interface. If the problem repeats, contact the IBM Software Support Center.

System programmer response

None.

Module

EZACIC01

Destination

TASK RELATED USER EXIT (TRUE)
**EZY1265E:** \( \text{mm/dd/yy hh:mm:ss} \) CICS VERSION UNSUPPORTED
GWACIVRM=xxxx ERRNO=errno TRAN=tran TASK=cicstask

**Explanation**

The task related user exit (TRUE) detected a version of CICS which it does not support. The CICS version must be 3 or above.

\( \text{errno} \) is the UNIX System Services return code. These return codes are listed in the sockets and sockets extended return codes (ERRNOs) in [z/OS Communications Server: IP and SNA Codes](#).

**System action**

The TRUE is disabled and the task abends with an AEY9.

**Operator response**

Contact the CICS system programmer.

**System programmer response**

The CICS socket interface requires CICS V3R3 or later.

**Module**

EZACIC01

**Destination**

TASK RELATED USER EXIT (TRUE)

---

**EZY1267E:** \( \text{mm/dd/yy hh:mm:ss} \) ROUTING TASK FUNCTION INVALID UERTIFD=xx ERRNO=errno TRAN=tran TASK=cicstask

**Explanation**

The task related user exit (TRUE) detected an invalid routing task function.

\( \text{errno} \) is the UNIX System Services return code. These return codes are listed in the sockets and sockets extended return codes (ERRNOs) in [z/OS Communications Server: IP and SNA Codes](#).

**System action**

The TRUE is disabled and the task abends with an AEY9.

**Operator response**

If this happens repeatedly, use EZAO to STOP (immediate) the CICS socket interface and then START it. If it still happens, contact the IBM Software Support Center.

**System programmer response**

None.
Module
EZACIC01

Destination
TASK RELATED USER EXIT (TRUE)

EZY1268E: mm/dd/yy hh:mm:ss SAVE AREA ADDRESS INVALID
UEPHSMA= xxxxxxxx ERRNO(errno) TRAN=tran TASK=cicstask

Explanation
The task related user exit (TRUE) detected an invalid save area address.

errno is the UNIX System Services return code. These return codes are listed in the sockets and sockets extended return codes (ERRNOs) in z/OS Communications Server: IP and SNA Codes.

System action
The TRUE is disabled and the task abends with an AEY9.

Operator response
Contact the IBM Software Support Center.

System programmer response
None.

Module
EZACIC01

Destination
TASK RELATED USER EXIT (TRUE)

EZY1269E: mm/dd/yy hh:mm:ss PARM LIST ADDRESS INVALID
GPR1= xxxxxxxx ERRNO(errno) TRAN=tran TASK=cicstask

Explanation
The task related user exit (TRUE) detected an invalid parameter list on a call request from the CICS application program.

errno is the UNIX System Services return code. These return codes are listed in the sockets and sockets extended return codes (ERRNOs) in z/OS Communications Server: IP and SNA Codes.

System action
The TRUE is disabled and the task abends with an AEY9.
Operator response

Check the application program calls to the CICS socket interface to ensure that each call has the correct number and type of parameters.

System programmer response

None.

Module

EZACIC01

Destination

TASK RELATED USER EXIT (TRUE)

EZY1270E: mm/dd/yy hh:mm:ss PARM nn ADDRESS INVALID
ADDRESS= xxxxxxxx ERRNO=errno TRAN=tran TASK=cicstask

Explanation

The task related user exit (TRUE) detected an invalid parameter address on a call request from the CICS application program. nn is the number of the parameter.

errno is the UNIX System Services return code. These return codes are listed in the sockets and sockets extended return codes (ERRNOs) in [z/OS Communications Server: IP and SNA Codes]

System action

The TRUE is disabled and the task abends with an AEY9.

Operator response

Check the application program calls to the CICS socket interface to ensure that the parameter addresses are valid (not zero). This problem is most common in assembler language and C applications.

System programmer response

None.

Module

EZACIC01

Destination

TASK RELATED USER EXIT (TRUE)
EZY1271E: mm/dd/yy hh:mm:ss TOKERR=xxxxxxxx
ERRNO=errno TRAN=tran TASK=cicstask
Explanation

The task related user exit (TRUE) detected a token error on an internal token used to coordinate CICS transaction activity with TCP/IP activity.

errno is the UNIX System Services return code. These return codes are listed in the sockets and sockets extended return codes (ERRNOs) in z/OS Communications Server: IP and SNA Codes.

System action

The TRUE is disabled and the task abends with an AEY9.

Operator response

Contact the IBM Software Support Center.

System programmer response

None.

Module

EZACIC01

Destination

TASK RELATED USER EXIT (TRUE)

EZY1272E: mm/dd/yy hh:mm:ss INVALID SOCKET/FUNCTION
CALL FUNCTION= xxxx ERRNO=errno TRAN=tran
TASK=cicstask
Explanation

A call to EZASOKET specified in invalid function.

errno is the UNIX System Services return code. These return codes are listed in the sockets and sockets extended return codes (ERRNOs) in z/OS Communications Server: IP and SNA Codes.

System action

The TRUE is disabled and the task abends with an AEY9.

Operator response

Correct the call and try again.

System programmer response

None.
Module
EZACIC01

Destination
task related user exit (TRUE)

EZY1273E: mm/dd/yy hh:mm:ss IUCV SOCK/FUNC TABLE INVALID FUNCTION= xxxx ERRNO=errno TRAN=tran TASK=cicstask

Explanation
A call to EZACICAL specified a function that was not valid.

errno is the UNIX System Services return code. These return codes are listed in the sockets and sockets extended return codes (ERRNOs) in z/OS Communications Server: IP and SNA Codes.

System action
The TRUE is disabled and the task abends with an AEY9.

Operator response
Correct the call and try again.

System programmer response
None.

Module
EZACIC01

Destination
TASK RELATED USER EXIT (TRUE)

EZY1274E: mm/dd/yy hh:mm:ss INCORRECT EZASOKET PARM COUNT FUNCTION= xxxx ERRNO=errno TRAN=tran TASK=cicstask

Explanation
A call to EZASOKET specified an invalid number of parameters.

errno is the UNIX System Services return code. These return codes are listed in the sockets and sockets extended return codes (ERRNOs) in z/OS Communications Server: IP and SNA Codes.

System action
The TRUE is disabled and the task abends with an AEY9.
Operator response

Correct the call and try again.

System programmer response

None.

Module

EZACIC01

Destination

TASK RELATED USER EXIT (TRUE)

EZY1275E: mm/dd/yy hh:mm:ss MONITOR CALLS NOT SUPPORTED UERTFID=xx ERRNO=errno TRAN=tran TASK=cicstask

Explanation

The task related user exit (TRUE) detected a monitor call which is not supported for this version of CICS.

erno is the UNIX System Services return code. These return codes are listed in the sockets and sockets extended return codes (ERRNOs) in z/OS Communications Server: IP and SNA Codes.

System action

The TRUE is disabled and the task abends with an AEY9.

Operator response

Contact the IBM Software Support Center.

System programmer response

None.

Module

EZACIC01

Destination

TASK RELATED USER EXIT (TRUE)

EZY1276E: mm/dd/yy hh:mm:ss EDF CALLS NOT SUPPORTED UERTFID=xx ERRNO=errno TRAN=tran TASK=cicstask

Explanation

The task related user exit (TRUE) detected an EDF (Execute Diagnostic Facility) call. This TRUE does not support EDF calls.
**errno** is the UNIX System Services return code. These return codes are listed in the sockets and sockets extended return codes (ERRNOs) in [z/OS Communications Server: IP and SNA Codes](#).

### System action

The TRUE is disabled and the task abends with an AEY9.

### Operator response

Contact the IBM Software Support Center.

### System programmer response

None.

**Module**

EZACIC01

**Destination**

TASK RELATED USER EXIT (TRUE)

**EZY1277I: mm/dd/yy hh:mm:ss EZACIC03 DETACHED TCB ADDRESS=xxxxxxx ERRNO=errno TRAN=tran TASK=cicstask**

**Explanation**

An attached subtask is terminating.

**errno** is the UNIX System Services return code. These return codes are listed in the sockets and sockets extended return codes (ERRNOs) in [z/OS Communications Server: IP and SNA Codes](#).

### System action

The TRUE detaches the MVS subtask.

### Operator response

None.

### System programmer response

None.

**Module**

EZACIC01

**Destination**

TASK RELATED USER EXIT (TRUE)
EZY1278I: mm/dd/yy hh:mm:ss EZACIC03 DETACH SUCCESSFUL TCB ADDRESS= xxxxxxxx TRAN=tran TASK=cicstask

Explanation
An attached subtask is terminating.

System action
The TRUE detaches the MVS subtask.

Operator response
None.

System programmer response
None.

Module
EZACIC01

Destination
TASK RELATED USER EXIT (TRUE)

EZY1279E: mm/dd/yy hh:mm:ss INVALID SYNC PT COMMAND DISP=xx TRAN=tran TASK=cicstask

Explanation
The task related user exit (TRUE) Detected an invalid Sync Point command.

System action
Disable the TRUE and return to the caller.

Operator response
Contact the IBM Software Support Center.

System programmer response
None.

Module
EZACIC01

Destination
TASK RELATED USER EXIT (TRUE)
**EZY1280E: mm/dd/yy hh:mm:ss INVALID RESYNC COMMAND**

**Explanation**

The task related user exit (TRUE) Detected an invalid Resync command.

**System action**

Disable the TRUE and return to the caller.

**Operator response**

Contact the IBM Software Support Center.

**System programmer response**

None.

**Module**

EZACIC01

---

**EZY1282E: mm/dd/yy hh:mm:ss 10999 ABEND reasonxx**

**Explanation**

The ESTAE processing in EZACIC03 could not be completed because of reasonxx.

**System action**

Allow the ABEND to percolate.

**Operator response**


**System programmer response**

None.

**Module**

EZACIC03

**Destination**

MVS SUBTASK

---

**EZY1285E: mm/dd/yy hh:mm:ss CICS/SOCKETS LISTENER TRANSACTION tran NOT ON CONFIGURATION FILE**

**Explanation**

The Listener attempting to start does not have a description record on the CICS Sockets configuration file.
System action
Listener terminates.

Operator response
Contact CICS system programmer.

System programmer response
Add the Listener to the configuration file using EZAC and try again.

Module
EZACIC02

Destination
LISTENER

EZY1286E: mm/dd/yy hh:mm:ss READ FAILURE ON CICS/ SOCKETS CONFIGURATION FILE TRANSACTION= tran EIBRESP2= rrrrr

Explanation
The Listener could not read the configuration file.

System action
Listener terminates.

Operator response
Contact CICS system programmer.

System programmer response
Use the CICS APR to interpret the value of EIBRESP2. If the file is not known to CICS, perform the installation steps for the configuration file.


Module
EZACIC02

Destination
LISTENER
EZY1287E: mm/dd/yy hh:mm:ss EZYCIC02 GETMAIN FAILURE
FOR VARIABLE STORAGE TRANSACTION= tran
EIBRESP2=rrrr

Explanation
EZACIC02 could not obtain the variable storage it requires to execute.

System action
Listener terminates.

Operator response
Contact CICS system programmer.

System programmer response
Use the CICS APR to interpret the value of EIBRESP2. Correct your CICS configuration as indicated.


Module
EZACIC02

Destination
LISTENER

EZY1288E: mm/dd/yy hh:mm:ss CICS SOCKETS MODULE
mmmmmmmm ABEND aaaa

Explanation
An abend has occurred in module mmmmmmm of the CICS socket interface.

System action
Listener terminates.

Operator response

System programmer response
None.

Module
EZACIC02
Destination
LISTENER

**EZY1289I: mm/dd/yy hh:mm:ss CICS LISTENER TRANSACTION tran taskno TERMINATING**

**Explanation**

The Listener is ending. This could be a normal shutdown situation or a failure related to the Listener socket. If it is the latter, a previous message described the failure.

In the message text:

*mm/dd/yy*

The date (month/day/year) of the message.

*hh:mm:ss*

The time (hours:minutes:seconds) of the message.

*tran*

The listener transaction ID.

*taskno*

The CICS task number assigned to the listener transaction ID.

**Example**

EZY1289I 02/19/09 13:51:39 CICS/SOCKETS LISTENER TRANSACTION CSKM TERMINATING

**System action**

The Listener ends.

**Operator response**

None.

**User response**

Not applicable.

**System programmer response**

None.

**Problem determination**

Not applicable.

**Source**

z/OS Communications Server TCP/IP: CICS Listener

**Module**

EZACIC02
Routing code
1

Descriptor code
2

Automation

This message is sent to the system console and to the CICS transient data queue that is specified by the IP CICS Sockets ERRORTD configuration option.

**EZY1290I:** mm/dd/yy hh:mm:ss LISTENER TRANSACTION *tran* STARTING

**Explanation**

Transaction *tran*, Listener program EZACIC02 has been given control.

**System action**

Listener *tran* continues.

**Operator response**

None.

**System programmer response**

None.

**Module**

EZACIC02

**Destination**

LISTENER

**EZY1291I:** mm/dd/yy hh:mm:ss LISTENER TRANSACTION *transactionid* TASKID= *taskno* ACCEPTING REQUESTS VIA PORT *port*

**Explanation**

The specified transaction can now receive connection requests on the specified port.

This message is issued when any of the following events occur:

- The listener is initialized and was able to connect to its TCP/IP.
- The listener reconnects to its TCP/IP after its TCP/IP has been restarted.
- The listener's socket descriptor table is no longer full and the table is now accepting client connections.
In the message text:

*mm/dd/yy*

The date (month/day/year) of the message.

*hh:mm:ss*

The time (hours:minutes:seconds) of the message.

*transactionid*

The name of the listener's transaction that can now accept new client connections.

*taskno*

The task number assigned by CICS.

*port*

The port number on which the listener identified by the *transactionid* value is listening.

**Example**

EZY1291I 01/19/06 10:07:33 LISTENER TRANSACTION=CSKL TASKID=0000079L ACCEPTING REQUESTS VIA PORT 3010

**System action**

The listener transaction continues.

**Operator response**

No action needed.

**User response**

None.

**System programmer response**

No action needed.

**Problem determination**

None.

**Source**

Not applicable.

**Module**

EZACIC02

**Routing code**

Not applicable.

**Descriptor code**

Not applicable.
EZY1292E: mm/dd/yy hh:mm:ss CANNOT START LISTENER, TRUE NOT ACTIVE TRANSACTION= tran TASKID= cicstask EIBRCODE BYTE3=rr

Explanation

The initialization of the CICS socket interface did not complete successfully and this Listener cannot continue.

System action

Listener transaction tran terminates.

Operator response

If EZAO is being used to start the Listener, ensure that the CICS socket interface has successfully completed initialization first. If this happens during automatic initialization, look for other messages which would indicate why the initialization of the CICS socket interface failed.

See the for information about EIBRCODEs.

System programmer response

None.

Module

EZACIC02

Destination

LISTENER

EZY1293E: mm/dd/yy hh:mm:ss INITAPI CALL FAILURE TRANSACTION=tran TASKID= cicstask ERRNO=errno

Explanation

Listener transaction tran experienced a failure on the INITAPI call.

errno is the UNIX System Services return code. These return codes are listed in the sockets and sockets extended return codes (ERRNOs) in z/OS Communications Server: IP and SNA Codes.

System programmer response

None.

System action

Listener transaction tran terminates.

Operator response

Use the errno value to determine the cause of the failure.
Module
EZACIC02
Destination
LISTENER

EZY1294E: mm/dd/yy hh:mm:ss SOCKET CALL FAILURE
TRANSACTION= tran TASKID= cicstask ERRNO= errno
Explanation
Listener transaction tran experienced a failure on the SOCKET call.

errno is the UNIX System Services return code. These return codes are listed in the
sockets and sockets extended return codes (ERRNOs) in z/OS Communications
Server: IP and SNA Codes

System programmer response
None.

System action
Listener transaction tran terminates.

Operator response
Use the errno value to determine the cause of the failure.

Module
EZACIC02
Destination
LISTENER

EZY1295E: mm/dd/yy hh:mm:ss BIND CALL FAILURE
TRANSACTION= tran TASKID= cicstask ERRNO= errno
Explanation
Listener transaction tran experienced a failure on the BIND call.

errno is the UNIX System Services return code. These return codes are listed in the
sockets and sockets extended return codes (ERRNOs) in z/OS Communications
Server: IP and SNA Codes

System action
Listener transaction tran terminates.
Operator response

Use the `errno` value to determine the cause of the failure.

Note:
1. An `ERRNO=13` could indicate that the port and jobname specified in the PORT statement in `hlq.TCPIP.PROFILE` does not match the port and jobname used by the CICS Listener.
2. An `ERRNO=48` could indicate that the port is not reserved in `hlq.TCPIP.PROFILE`.

System programmer response

None.

Module

EZACIC02

Destination

LISTENER

**EZY1296E:** `mm/dd/yy hh:mm:ss LISTEN CALL FAILURE`  
**TRANSACTION=** `tran`  
**TASKID=** `cicstask`  
**ERRNO=** `errno`

Explanation

Listener transaction `tran` experienced a failure on the LISTEN call.

`errno` is the UNIX System Services return code. These return codes are listed in the sockets and sockets extended return codes (ERRNOs) in [z/OS Communications Server: IP and SNA Codes](#).

System action

Listener transaction `tran` terminates.

Operator response

Use the `errno` value to determine the cause of the failure.

System programmer response

None.

Module

EZACIC02

Destination

LISTENER
EZY1297E: mm/dd/yy hh:mm:ss GETCLIENTID CALL FAILURE
TRANSACTIOM=tran TASKID= cicstask ERRNO=errno

Explanation

Listener transaction tran experienced a failure on the GETCLIENTID call.

errno is the UNIX System Services return code. These return codes are listed in the sockets and sockets extended return codes (ERRNOs) in z/OS Communications Server: IP and SNA Codes.

System action

Listener transaction tran terminates.

Operator response

Use the errno value to determine the cause of the failure.

System programmer response

None.

Module

EZACIC02

Destination

LISTENER

EZY1298E: mm/dd/yy hh:mm:ss CLOSE FAILURE TRANID= tran
TASKID= cicstask ERRNO= errno

Explanation

Listener transaction tran experienced a failure on the CLOSE call.

errno is the UNIX System Services return code. These return codes are listed in the sockets and sockets extended return codes (ERRNOs) in z/OS Communications Server: IP and SNA Codes.

System action

Listener transaction tran continues.

Operator response

Use the errno value to determine the cause of the failure.

System programmer response

None.
Module
EZACIC02

Destination
LISTENER

EZY1299E: mm/dd/yy hh:mm:ss SELECT CALL FAILURE
TRANSACTION= tran TASKID= xxxxx ERRNO= errno

Explanation
Listener transaction tran experienced a failure on the SELECT call.

errno is the UNIX System Services return code. These return codes are listed in the
sockets and sockets extended return codes (ERRNOs) in z/OS Communications
Server: IP and SNA Codes.

System action
Listener transaction tran terminates.

Operator response
Use the errno value to determine the cause of the failure.

System programmer response
None.

Module
EZACIC02

Destination
LISTENER

EZY1300E: mm/dd/yy hh:mm:ss RECV FAILURE TRANSID=
transactionid TASKID= tasknumber ERRNO= errno INET
ADDR=inetaddress PORT=portnumber

Explanation
The Listener transaction transactionid experienced a failure on the RECV call.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the Listener transaction performing the RECV Socket.

tasknumber is the CICS task number assigned to the CICS transaction transactionid.
errno is the UNIX System Services return code. These return codes are listed in the sockets and sockets extended return codes (ERRNOs) in [z/OS Communications Server: IP and SNA Codes]

inetaddress is the internet address of the connecting client.

portnumber is the connecting client’s port number.

**System action**

The Listener transaction transactionid continues.

**Operator response**

Use the errno value to determine the cause of the failure.

**System programmer response**

None.

**Module**

EZACIC02

**Destination**

LISTENER

**EZY1301E: mm/dd/yy hh:mm:ss CONNECTION CLOSED BY CLIENT TRANSACTION= transactionid PARTNER INET ADDR= ipaddr PORT= port**

**Explanation**

A remote client connected to the CICS Listener but then closed the connection before sending the entire amount of data required by the Listener as determined by the MINMSGL standard Listener configuration parameter or the MSGLEN enhanced Listener configuration parameter.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the transaction name of the CICS Listener.

ipaddr is the internet address of the remote client.

port is the port number of the remote client.

**System action**

The Listener transaction transactionid continues.

**Operator response**

Correct the client program.
System programmer response
None.

Module
EZACIC02

Destination
LISTENER

EZY1302I: mm/dd/yy hh:mm:ss READ TIMEOUT PARTNER INET ADDR= inetaddress PORT= portnumber LISTENER TRANID= tran_id TASKID= task_id

Explanation
The initial message from the client did not arrive within the read timeout value specified for this Listener in the CICS Sockets configuration file.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client’s port number.

tran_id is the name of the listener’s transaction.

task_id is the task number of the listener’s transaction.

Example
EZY1302I 02/24/09 16:13:16 READ TIMEOUT PARTNER INET ADDR=9.42.105.102 PORT=1030 LISTENER TRANID= CSKM TASKID= 0000085L

System action
The Listener closes the connection socket and does not attempt to start a server transaction.

Operator response
Determine the cause of the delay and correct it.

System programmer response
None.

Problem determination
Not applicable.
Source

z/OS Communications Server TCP/IP: LISTENER

Module

EZACIC02

Routing code

10

Descriptor code

12

Automation

This message is sent to the CICS transient data queue that is specified by the IP CICS Sockets ERRORTD configuration option.

EZY1303I: mm/dd/yy hh:mm:ss EZACIC02 GIVESOCKET TIMEOUT TRANS transactionid PARTNER INET ADDR=inetaddress PORT=portnumber

Explanation

The started server transaction did not perform the takesocket within the timeout value specified for this Listener in the CICS Sockets configuration file.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client’s port number.

System action

Send an error message to the client and close the socket.

Operator response

Determine the reason for the delay in the server transaction. Possible causes are an overloaded CICS system or excessive processing in the server transaction before the takesocket is issued. Correct the situation and try again.

System programmer response

None.
Module
EZACIC02

Destination
LISTENER

**EZY1304I: mm/dd/yy hh:mm:ss UNEXPECTED INPUT EVENT TRANSACTION** transactionid **PARTNER INET ADDR=inetaddress PORT=portnumber**

**Explanation**

The Listener received data from the client after the end of the transaction input message.

*mm/dd/yy* is the date (month/day/year) of the message.

*hh:mm:ss* is the time (hours:minutes:seconds) of the message.

*transactionid* is the name of the transaction that was requested by the connecting client.

*inetaddress* is the internet address of the connecting client.

*portnumber* is the connecting client’s port number.

**System action**

The Listener ignores this data.

**Operator response**

Ensure that the minimum message length specification for this Listener in the CICS Sockets Configuration file is correct. If it is, determine why the client is sending this additional data.

**System programmer response**

None.

Module
EZACIC02

Destination
LISTENER
EZY1305E: mm/dd/yy hh:mm:ss UNEXPECTED EXCEPTION EVENT TRANS transactionid PARTNER INET ADDR=inetaddress PORT=portnumber

Explanation

The Listener received an exception event on this connection other than the event showing a successful takesocket was issued by the server.

*mm/dd/yy* is the date (month/day/year) of the message.

*hh:mm:ss* is the time (hours:minutes:seconds) of the message.

*transactionid* is the name of the transaction that was requested by the connecting client.

*inetaddress* is the internet address of the connecting client.

*portnumber* is the connecting client’s port number.

**System action**

Ignore the event.

**Operator response**

Ensure the client is not doing anything that would cause an exception event such as the use of out-of-band data.

**System programmer response**

None.

**Module**

EZACIC02

**Destination**

LISTENER

EZY1306E: mm/dd/yy hh:mm:ss SECURITY EXIT mmmmmmmmm 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.

**Operator response**

Use CICS RDO to define the security exit.
System programmer response
None.

Module
EZACIC02

Destination
LISTENER

**EZY1307E**: `mm/dd/yy hh:mm:ss MAXIMUM # OF SOCKETS USED TRANS= tran TASKID= cicstask ERRNO= errno`

**Explanation**
All of the sockets allocated to Listener transaction `xxxx` are in use.

`errno` is the UNIX System Services return code. These return codes are listed in the sockets and sockets extended return codes (ERRNOs) in z/OS Communications Server: IP and SNA Codes.

System action
The ACCEPT call is delayed until a socket is available.

Operator response
Use the EZAC transaction to increase the number of sockets allocated Listener `tran` and then stop and restart Listener transaction `tran`.

System programmer response
None.

Module
EZACIC02

Destination
LISTENER

**EZY1308E**: `mm/dd/yy hh:mm:ss ACCEPT CALL FAILURE TRANSACTION= tran TASKID= cicstask ERRNO= errno`

**Explanation**
Listener transaction `tran` experienced a failure on the ACCEPT call.

`errno` is the UNIX System Services return code. These return codes are listed in the sockets and sockets extended return codes (ERRNOs) in z/OS Communications Server: IP and SNA Codes.
System action

Listener transaction tran terminates.

Operator response

Use the errno value to determine the cause of the failure.

System programmer response

None.

Module

EZACIC02

Destination

LISTENER

EZY1309E: mm/dd/yy hh:mm:ss GIVESOCKET FAILURE TRANS
transactionid TASKID=tasknumber ERRNO=errno INET
ADDR=inetaddress PORT=portnumber

Explanation

The Listener transaction transactionid experienced a failure on the GIVESOCKET call.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction that was requested by the connecting client.

tasknumber is the CICS task number assigned to the CICS transaction transactionid.

errno is the UNIX System Services return code. These return codes are listed in the sockets and sockets extended return codes (ERRNOs) in z/OS Communications Server: IP and SNA Codes.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client’s port number.

System action

The Listener transaction transactionid terminates.

Operator response

Use the errno value to determine the cause of the failure.
System programmer response

None.

Module

EZACIC02

Destination

LISTENER

EZY1310E: mm/dd/yy hh:mm:ss IC VALUE NOT NUMERIC
TRANID=transactionid PARTNER INET ADDR=inetaddress
PORT=portnumber

Explanation

The interval specified in the transaction input message contains one or more non-numeric characters.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client's port number.

System action

The interval is ignored, and the transaction is started immediately.

Operator response

Correct the client program which is sending this transaction input message.

System programmer response

None.

Module

EZACIC02

Destination

LISTENER
EZY1311E: \textit{mm/dd/yy hh:mm:ss CICS TRANID} \textit{transactionid} \textbf{NOT AUTHORIZED PARTNER INET ADDR}={\textit{inetaddress}} \textbf{PORT}={\textit{portnumber}}

\textbf{Explanation}

The transaction name specified in the transaction input message is not RSL authorized.

\textit{mm/dd/yy} is the date (month/day/year) of the message.

\textit{hh:mm:ss} is the time (hours:minutes:seconds) of the message.

\textit{transactionid} is the name of the transaction that was requested by the connecting client.

\textit{inetaddress} is the internet address of the connecting client.

\textit{portnumber} is the connecting client’s port number.

\textbf{System action}

The transaction is not started.

\textbf{Operator response}

Correct the CICS transaction definition if the transaction should be authorized or the client program if it is sending the wrong transaction name.

\textbf{System programmer response}

None.

\textbf{Module}

EZACIC02

\textbf{Destination}

LISTENER

EZY1312E: \textit{mm/dd/yy hh:mm:ss SECURITY EXIT} \textit{mmmmmmmmmm} \textbf{CANNOT BE LOADED TRANID}={\textit{tran}} \textbf{TASKID}={\textit{cicstask}}

\textbf{Explanation}

Listener transaction \textit{tran} experienced a failure when it attempted to load security exit program \textit{mmmmmmmmmm}.

\textbf{System action}

Listener transaction \textit{tran} continues but the server transaction associated with this transaction input message is not started.
Operator response

Use CEMT to determine the status of the exit program and correct whatever problems are found.

System programmer response

None.

Module

EZACIC02

Destination

LISTENER

EZY1313E: mm/dd/yy hh:mm:ss LISTENER NOT AUTHORIZED TO ACCESS SECURITY EXIT mmmmmmmmm TRANID= tran TASKID=xxxxxxxx

Explanation

Listener transaction tran is not authorized to access security exit program mmmmmmmmm.

System action

Listener transaction tran continues but the server transaction associated with this transaction input message is not started.

Operator response

If the security exit program name is incorrect, use EZAC to correct the definition of this Listener on the CICS Sockets Configuration file. If the security exit program is correct, use the CICS RDO facility to authorize Listener transaction xxxx to use security exit program mmmmmmmmm.

System programmer response

None.

Module

EZACIC02

Destination

LISTENER

EZY1314E: mm/dd/yy hh:mm:ss SECURITY EXIT mmmmmmmmm IS DISABLED TRANID= tran TASKID=xxxxxxxx

Explanation

Security exit program mmmmmmmmm is disabled.
System action

Listener transaction tran continues but the server transaction associated with this transaction input message is not started.

Operator response

Use CEMT to enable the security exit program.

System programmer response

None.

Module

EZACIC02

Destination

LISTENER

**EZY1315E: mm/dd/yy hh:mm:ss INVALID TRANSID transactionid**
PARTNER INET ADDR=inetaddress PORT=portnumber

Explanation

The transaction input message from the client specified transaction transactionid but this transaction is not defined to CICS.

*mm/dd/yy* is the date (month/day/year) of the message.

*hh:mm:ss* is the time (hours:minutes:seconds) of the message.

*transactionid* is the name of the transaction that was requested by the connecting client. The *transactionid* field will be blank if no printable name was passed by the client or the security exit.

*inetaddress* is the internet address of the connecting client.

*portnumber* is the connecting client’s port number.

System action

The Listener transaction continues but the server transaction associated with this transaction input message is not started.

Operator response

If the transaction name is incorrect, correct the client program. If the transaction name is correct, correct the CICS transaction definition.

System programmer response

If *transactionid* is blank, then there is a possible mismatch because the Listener is expecting the first message segment to start with a transaction name but it does not. A packet trace might be helpful in determining whether there is such a
mismatch. For example, if the packet trace shows that the first message segment
starts with X'160300' or X'160301' then possibly a clienthello message was received,
which indicates that there is an Application Transparent Transport Layer Security
(AT-TLS) policy on the client side of the TCP connection but no matching AT-TLS
policy (or AT-TLS is not enabled) on the Listener side of the TCP connection. This
would need to be addressed by the AT-TLS administrator. See Application
Transparent Transport Layer Security Data Protection in z/OS Communications
Server: IP Configuration Guide and Steps for diagnosing AT-TLS problems in z/OS
Communications Server: IP Diagnosis Guide for more information.

Module

EZACIC02

Destination

LISTENER

EZY1316E: mm/dd/yy hh:mm:ss TRANSID transactionid IS
DISABLED PARTNER INET ADDR=inetaddress
PORT=portnumber

Explanation

Transaction transactionid is disabled.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction that was requested by the connecting
client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client’s port number.

System action

The Listener transaction continues but the server transaction associated with this
transaction input message is not started.

Operator response

Use CEMT to enable the server transaction.

System programmer response

None.

Module

EZACIC02
Destination
LISTENER

EZY1317E: mm/dd/yy hh:mm:ss TRANSID transactionid IS NOT AUTHORIZED PARTNER INET ADDR=inetaddress PORT=portnumber

Explanation
The Listener transaction transactionid is not authorized to start the transaction name specified in the transaction input message.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client’s port number.

System action
The transaction is not started.

Operator response
Authorize Listener transaction transactionid to start the transaction.

System programmer response
None.

Module
EZACIC02

Destination
LISTENER

EZY1318E: mm/dd/yy hh:mm:ss TD START SUCCESSFUL QUEUEID= que

Explanation
The Listener transaction started a server transaction through transient data queue que

System action
Listener transaction continues and the server transaction is ready to start.
Operator response

None.

System programmer response

None.

Module

EZACIC02

Destination

LISTENER

EZY1319E: mm/dd/yy hh:mm:ss QIDERR FOR TD DESTINATION queuename PARTNER INET ADDR=inetaddress PORT=portnumber

Explanation

The Listener transaction was unable to start a CICS transaction through transient data queue queuename. DFHRESP was QIDERR.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

queuename is the name of the transient data queue that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client’s port number.

System action

The Listener transaction continues.

Operator response

If the queue name is incorrect, correct the client program sending this transaction input message. If the queue name is correct, correct the CICS Destination Control Table.

System programmer response

None.

Module

EZACIC02
Destination

LISTENER

EZY1320E: mm/dd/yy hh:mm:ss I/O ERROR FOR TD
DESTINATION queuename PARTNER INET ADDR=inetaddress
PORT=portnumber

Explanation

The Listener transaction was unable to start a CICS transaction through transient data queue queuename. DFHRESP was IOERR.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

queuename is the name of the transient data queue that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client’s port number.

System action

The Listener transaction continues.

Operator response

Contact the CICS system programmer.

System programmer response

None.

Module

EZACIC02

Destination

LISTENER

EZY1321E: mm/dd/yy hh:mm:ss LENGTH ERROR FOR TD
DESTINATION queuename PARTNER INET ADDR=inetaddress
PORT=portnumber

Explanation

The Listener transaction was unable to start a CICS transaction through transient data queue queuename. DFHRESP was LENGERR.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.
queue name is the name of the transient data queue that was requested by the connecting client.

inet address is the internet address of the connecting client.

port number is the connecting client’s port number.

**System action**

The Listener transaction continues.

**Operator response**

Contact the CICS system programmer. The minimum length for this queue should be greater than 72.

**System programmer response**

Change definition of Transient Data Queue to accommodate length of this message.

**Module**

EZACIC02

**Destination**

LISTENER

**EZY1322E**: mm/dd/yy hh:mm:ss TD DESTINATION queue name DISABLED PARTNER INET ADDR=inet address PORT=port number

**Explanation**

The Listener transaction was unable to start a CICS transaction through transient data queue queue name. DFHRESP was DISABLED.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

queue name is the name of the transient data queue that was requested by the connecting client.

inet address is the internet address of the connecting client.

port number is the connecting client’s port number.

**System action**

The Listener transaction continues.

**Operator response**

Use CEMT to enable the destination.
System programmer response

None.

Module

EZACIC02

Destination

LISTENER

EZY1323E: mm/dd/yy hh:mm:ss TD DESTINATION queuename OUT OF SPACE PARTNER INET ADDR=inetaddress PORT=portnumber

Explanation

The Listener transaction was unable to start a CICS transaction through transient data queue queuename. DFHRESP was NOSPACE.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

queuename is the name of the transient data queue that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client’s port number.

System action

The Listener transaction continues.

Operator response

Contact the CICS system programmer.

System programmer response

Allocate space for this Transient Data Queue.

Module

EZACIC02

Destination

LISTENER
**EZY1324E**: *mm/dd/yy hh:mm:ss* TD START FAILED QUEUE
**ID=** *queue name*  **PARTNER INET ADDR=** *inet address*  **PORT=** *port number*

**Explanation**

The Listener transaction was unable to start a CICS transaction through transient data queue *queue name*.

*mm/dd/yy* is the date (month/day/year) of the message.

*hh:mm:ss* is the time (hours:minutes:seconds) of the message.

*queue name* is the name of the transient data queue that was requested by the connecting client.

*inet address* is the internet address of the connecting client.

*port number* is the connecting client’s port number.

**System action**

The Listener transaction continues.

**Operator response**

Contact the CICS system programmer.

**System programmer response**

Determine the problem with the Transient Data Queue and correct it.

**Module**

EZACIC02

**Destination**

LISTENER

**EZY1325I**: *mm/dd/yy hh:mm:ss* START SUCCESSFUL
**TRANID=** *transaction id*  **PARTNER INET ADDR=** *inet address*  **PORT=** *port number*

**Explanation**

The Listener transaction was able to start a CICS transaction *transaction id* transient data queue.

*mm/dd/yy* is the date (month/day/year) of the message.

*hh:mm:ss* is the time (hours:minutes:seconds) of the message.

*transaction id* is the name of the transaction that was requested by the connecting client.
**inetaddress** is the internet address of the connecting client.

**portnumber** is the connecting client’s port number.

**System action**

The Listener transaction continues.

**Operator response**

None.

**System programmer response**

None.

**Module**

EZACIC02

**Destination**

LISTENER

**EZY1326E: mm/dd/yy hh:mm:ss START I/O ERROR**

**TRANID=transactionid PARTNER INET ADDR=inetaddress PORT=portnumber**

**Explanation**

The Listener transaction was unable to start a CICS transaction **transactionid**. DFHRESP was IOERR.

**mm/dd/yy** is the date (month/day/year) of the message.

**hh:mm:ss** is the time (hours:minutes:seconds) of the message.

**transactionid** is the name of the transaction that was requested by the connecting client.

**inetaddress** is the internet address of the connecting client.

**portnumber** is the connecting client’s port number.

**System action**

The Listener transaction continues.

**Operator response**

Contact the CICS system programmer.

**System programmer response**

Determine the cause of the I/O error and correct it.
Module
EZACIC02

Destination
LISTENER

EZY1327E: mm/dd/yy hh:mm:ss START TRANSACTION ID transactionid INVALID PARTNER INET ADDR=inetaddress PORT=portnumber
Explanation
The Listener transaction was unable to start a CICS transaction transactionid. DFHRESP was TRANSIDERR.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client’s port number.

System action
The Listener transaction continues.

Operator response
Contact the CICS system programmer.

System programmer response
Check the transaction definition in RDO to ensure it is correct.

Module
EZACIC02

Destination
LISTENER

EZY1328E: mm/dd/yy hh:mm:ss START TRANSACTION ID transactionid NOT AUTHORIZED PARTNER INET ADDR=inetaddress PORT=portnumber
Explanation
The Listener transaction was unable to start a CICS transaction transactionid. DFHRESP was NOTAUTH.
**EZY1329E: mm/dd/yy hh:mm:ss START FAILED (99)**

*TRANSID=transactionid*  
*PARTNER INET ADDR=inetaddress*  
*PORT=portnumber*

**Explanation**

The Listener transaction was unable to start a CICS transaction *transactionid*.  
DFHRESP was 99.

*mm/dd/yy* is the date (month/day/year) of the message.  
*hh:mm:ss* is the time (hours:minutes:seconds) of the message.  
*transactionid* is the name of the transaction that was requested by the connecting client.  
*inetaddress* is the internet address of the connecting client.  
*portnumber* is the connecting client’s port number.

**System action**

The Listener transaction continues.
Operator response

Contact the CICS system programmer.

System programmer response

Check the transaction definition in RDO. Look for associated messages in the MSGUSR queue, which might indicate why the transaction would not start.

Module

EZACIC02

Destination

LISTENER

EZY1330E: mm/dd/yy hh:mm:ss IC START SUCCESSFUL
TRANID=transactionid PARTNER INET ADDR=inetaddress
PORT=portnumber

Explanation

The Listener transaction was able to start a CICS transaction transactionid.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client’s port number.

System action

The Listener transaction continues.

Operator response

None.

System programmer response

None.

Module

EZACIC02

Destination

LISTENER
EZY1331E: *mm/dd/yy hh:mm:ss* IC START I/O ERROR
TRANID=*
PARTNER INET ADDR=*
PORT=* PORT=* 

**Explanation**

Listener transaction was unable to start a CICS transaction *transactionid*. DFHRESP was IOERR.

*mm/dd/yy* is the date (month/day/year) of the message.

*hh:mm:ss* is the time (hours:minutes:seconds) of the message.

*transactionid* is the name of the transaction that was requested by the connecting client.

*inetaddress* is the internet address of the connecting client.

*portnumber* is the connecting client's port number.

**System action**

Listener transaction continues.

**Operator response**

Contact the CICS system programmer.

**System programmer response**

Look for other messages in the MSGUSR queue, which provide specific information on the I/O error and correct the problem.

**Module**

EZACIC02

**Destination**

LISTENER

EZY1332E: *mm/dd/yy hh:mm:ss* IC START INVALID REQUEST
TRANID=*
PARTNER INET ADDR=*
PORT=* PORT=* 

**Explanation**

Listener transaction was unable to start a CICS transaction *transactionid*. DFHRESP was INVREQ.

*mm/dd/yy* is the date (month/day/year) of the message.

*hh:mm:ss* is the time (hours:minutes:seconds) of the message.

*transactionid* is the name of the transaction that was requested by the connecting client.
**System action**

Listener transaction continues.

**Operator response**

Collect the messages written to the console and MSGUSR queue, client input data, and a SOCKAPI component trace and contact the IBM Software Support Center.

**System programmer response**

None.

**Module**

EZACIC02

**Destination**

LISTENER

**EZY1333E**: mm/dd/yy hh:mm:ss IC START FAILED

TRANID=transactionid PARTNER INET ADDR=inetaddress

PORT=portnumber

**Explanation**

Listener transaction was unable to start a CICS transaction *transactionid*.

*mm/dd/yy* is the date (month/day/year) of the message.

*hh:mm:ss* is the time (hours:minutes:seconds) of the message.

*transactionid* is the name of the transaction that was requested by the connecting client.

*inetaddress* is the internet address of the connecting client.

*portnumber* is the connecting client’s port number.

**System action**

Listener transaction continues.

**Operator response**

Contact the CICS system programmer.

**System programmer response**

Check the RDO definition of the transaction. Collect the messages written to the console and MSGUSR queue, client input data, and a SOCKAPI component trace.
and contact the IBM Software Support Center.

**Module**

EZACIC02

**Destination**

LISTENER

**EZY1334E**: *mm/dd/yy hh:mm:ss INVALID USER
TRANID=*transactionid* PARTNER INET ADDR = *inetaddress*
PORT = *portnumber* USERID = *userid*

**Explanation**

This message indicates that the user security exit has given the Listener an invalid USERID field.

*mm/dd/yy* is the date (month/day/year) of the message.

*hh:mm:ss* is the time (hours:minutes:seconds) of the message.

*transactionid* is the name of the transaction that was requested by the connecting client.

*inetaddress* is the internet address of the connecting client.

*portnumber* is the connecting client's port number.

*userid* is the user ID assigned by the user security exit.

**System action**

The server transaction that is identified by the *transactionid* value does not start.

**Operator response**

Correct the user ID that is not valid in the user security exit.

**System programmer response**

None.

**Module**

EZACIC02

**Destination**

LISTENER
EZY1335E: mm/dd/yy hh:mm:ss WRITE FAILED ERRNO=errno
TRANID=transactionid. PARTNER INET ADDR/inetaddress
PORT=portnumber

Explanation

Listener transaction had a failure on a WRITE command.

*mm/dd/yy* is the date (month/day/year) of the message.

*hh:mm:ss* is the time (hours:minutes:seconds) of the message.

*errno* is the UNIX System Services return code. These return codes are listed in the sockets and sockets extended return codes (ERRNOs) in [z/OS Communications Server: IP and SNA Codes](http://www.ibm.com/servers/eserver/zseries/zoscomm/index.html).

*transactionid* is the name of the transaction that was requested by the connecting client.

*inetaddress* is the internet address of the connecting client.

*portnumber* is the connecting client’s port number.

System action

The Listener transaction continues.

Operator response

Use the *errno* value to determine the cause of the failure.

System programmer response

None.

Module

EZACIC02

Destination

LISTENER

EZY1336E: mm/dd/yy hh:mm:ss TAKESOCKET FAILURE TRANS
transactionid TASKID=tasknumber ERRNO=errno INET
ADDR/inetaddress PORT=portnumber

Explanation

The Listener transaction had a failure on a TAKESOCKET command.

*mm/dd/yy* is the date (month/day/year) of the message.

*hh:mm:ss* is the time (hours:minutes:seconds) of the message.
errno is the UNIX System Services return code. These return codes are listed in the sockets and sockets extended return codes (ERRNOs) in z/OS Communications Server: IP and SNA Codes.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client’s port number.

**System action**

The Listener transaction continues.

**Operator response**

Use the errno value to determine the cause of the failure.

**System programmer response**

None.

**Module**

EZACIC02

**Destination**

LISTENER

**EZY1337E: mm/dd/yy hh:mm:ss CICS IN QUIESCE, LISTENER TERMINATING TRANSID= tran TASKID= cicstask**

**Explanation**

Listener transaction tran is terminating because it detected a CICS quiesce in progress.

**System action**

Listener transaction tran terminates.

**Operator response**

None.

**System programmer response**

None.

**Module**

EZACIC02
Destination

LISTENER

EZY1338E: mm/dd/yy hh:mm:ss PROGRAM programname NOT FOUND TRANID=transactionid PARTNER INET ADDR=inetaddress PORT=portnumber

Explanation

The Listener checked the status of the program associated with the transaction. It was not found.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

programname is the name of the program which is associated with the transaction requested by the connecting client.

transactionid is the name of the transaction that was requested by the connecting client.

inetaddress is the internet address of the connecting client.

portnumber is the connecting client’s port number.

System action

Listener continues.

Operator response

If transactionid is incorrect, correct the client program that sent the transaction input message. If the transaction ID is correct, check the transaction and program definitions in CICS.

System programmer response

None.

Module

EZACIC02

EZY1339E: mm/dd/yy hh:mm:ss EXIT PROGRAM (EZACIC01) IS NOT ENABLED. DISABLE IGNORED TERM=term TRAN=tranxxx

Explanation

A termination of the CICS socket interface was requested but the interface is not enabled.

System action

The termination request is ignored.
Operator response
None.

System programmer response
None.

Module
EZACIC22

Destination
TERMINATION

EZY1340E: mm/dd/yy hh:mm:ss API ALREADY QUIESCING DUE TO PREVIOUS REQ. EZAO IGNORED TERM=term TRAN=tranxxx

Explanation
A request for a quiesce of the CICS socket interface has been made but one is already in progress.

System action
Ignore the second request.

Operator response
None.

System programmer response
None.

Module
EZACIC22

Destination
TERMINATION

EZY1341E: mm/dd/yy hh:mm:ss API ALREADY IN IMMED MODE DUE TO PREV. REQ. EZAO IGNORED TERM=term TRAN=tranxxx

Explanation
A request for an immediate of the CICS socket interface has been made but one is already in progress.
System action

Ignore the second request.

Operator response

None.

System programmer response

None.

Module

EZACIC22

Destination

TERMINATION

EZY1342I: mm/dd/yy hh:mm:ss DISABLE DELAYED UNTIL ALL USING TASKS COMPLETE TERM=termid TRAN=transid

Explanation

A quiesce is in progress and is waiting for all outstanding CICS tasks to complete using the CICS socket interface.

When an IP CICS interface is being shut down the following actions occur:

- All listeners are posted to end.
- If the interface is configured as OTE=NO, then all non-listener tasks have their MVS subtask posted and their CICS task ends.
- If the interface is configured as OTE=YES, then any non-listener transaction that is running a blocking socket command is forced to end by a CICS FORCE PURGE action.

See the information about the “TYPE=CICS setting for the TYPE parameter” on page 55 for information about the OTE configuration option.

In the message text:

- mm/dd/yy
  - The date (month/day/year) of the message.
- hh:mm:ss
  - The time (hours:minutes:seconds) of the message.
- termid
  - The CICS terminal ID on which the IP CICS socket shutdown is occurring.
- transid
  - The CICS transaction ID that requested that the IP CICS socket be shut down.

System action

The system continues to shut down.
Operator response

None.

System programmer response

None.

Module

EZACIC22

Destination

TERMINATION

**EZY1343I**: *mm/dd/yy hh:mm:ss CICS/SOCKETS INTERFACE IMMEDIATELY DISABLED*. *TERM=*term* TRAN=*tranxxx

**Explanation**

A request for the immediate ending of the CICS socket interface has been successfully completed.

In the message text:

*mm/dd/yy*

The date (month/day/year) of the message.

*hh:mm:ss*

The time (hours:minutes:seconds) of the message.

*term*

The terminal ID from which the command to end the CICS socket interface was issued.

*tran*

The transaction ID that is ending the CICS socket interface.

**Example**

EZY1343I 02/19/09 13:52:50 CICS/SOCKETS INTERFACE IMMEDIATELY DISABLED. TERM= TRAN=EZAP

**System action**

The CICS socket interface ends.

**Operator response**

None.

**System programmer response**

None.

**Problem determination**

Not applicable.
Source

z/OS Communications Server TCP/IP: CICS socket interface termination

Module

EZACIC22

Routing code

1

Descriptor code

2

Automation

This message is sent to the system console and to the CICS transient data queue that is specified by the IP CICS Sockets ERRORTD configuration option.

**EZY1344I**: mm/dd/yy hh:mm:ss CICS/ SOCKETS INTERFACE QUIESCENTLY DISABLED TERM=term TRAN=tranxxx

**Explanation**

A request for the deferred ending of the CICS socket interface has been successfully completed.

In the message text:

- **mm/dd/yy**: The date (month/day/year) of the message.
- **hh:mm:ss**: The time (hours:minutes:seconds) of the message.
- **term**: The terminal ID from which the command to end the CICS socket interface was issued.
- **tran**: The transaction ID that is ending the CICS socket interface.

**Example**

EZY1344I 02/19/09 13:52:21 CICS/ SOCKETS INTERFACE QUIESCENTLY DISABLED. TERM= TRAN=EZAP

**System action**

The CICS socket interface ends.

**Operator response**

None.

**System programmer response**

None.
Problem determination

Not applicable.

Source

z/OS Communications Server TCP/IP: CICS socket interface termination

Module

EZACIC22

Routing code

1

Descriptor code

2

Automation

This message is sent to the system console and to the CICS transient data queue that is specified by the IP CICS Sockets ERRORTD configuration option.

EZY1345E: mm/dd/yy hh:mm:ss CICS/SOCKETS WLM REGISTER FAILURE. RETURN CODE = return_code, GROUP = groupname, LISTNER = list

Explanation

The CICS Listener received an error response when attempting to register WLM group with the Workload manager.

  mm/dd/yy hh:mm:ss
  Date and time of the message.

  return_code
  The return code from the WLM registration.

  groupname
  Name of the WLM group.

  list
  Name of the CICS Listener.

System action

The Listener continues initialization but will not use groupname to participate in workload connection balancing.

Operator response

Verify that the WLM group name is correct and correctly defined to the Workload manager. If it is incorrect, either change it in the EZACICD TYPE=LISTENER macro that was used to define the Listener, or change it via the EZAC transaction. See the z/OS MVS Programming: Workload Management Services for more information about return_code.
System programmer response

None

Module

EZACIC12

EZY1346E: mm/dd/yy hh:mm:ss CICS SOCKETS WLM DEREGISTER FAILED RETURN CODE = return_code, GROUP = groupname, LISTENER = list

Explanation

The CICS Listener received an error response when attempting to deregister WLM group with the Workload manager.

mm/dd/yy hh:mm:ss
Date and time of the message.

return_code
The return code from the WLM deregistration.

groupname
Name of the WLM group.

list
Name of the CICS Listener.

System action

The Listener continues termination.

Operator response

See the z/OS MVS Programming: Workload Management Services for more information about return_code.

System programmer response

None.

Module

EZACIC12

EZY1347I: mm/dd/yy hh:mm:ss PROGRAM programname ASSUMED TO BE AUTOINSTALLED TRANID=transactionid IP ADDR=inetaddress PORT=portnumber

Explanation

The Listener checked the status of the program associated with the transaction. It was not found. Because program autoinstall is active in the CICS region, the Listener assumes that the program definition will automatically be installed by CICS.

mm/dd/yy
The date (month/day/year) of the message.
hh:mm:ss
The time (hours:minutes:seconds) of the message.

programname
The name of the undefined program which is associated with the transaction requested by the connecting client.

transactionid
The name of the transaction that was requested by the connecting client.

inetaddress
The internet address of the connecting client.

portnumber
The connecting client’s port number.

System action
Listener continues.

Operator response
None.

System programmer response
Verify that the program name in the transaction definition is correct. Verify that the program is intended to be autoinstalled rather than explicitly defined in the PPT.

Module
EZACIC02

Destination
LISTENER

EZY1348E: mm/dd/yy hh:mm:ss INVALID SOCKET FUNCTION function ERRNO errno TRAN tranid TASK taskid

Explanation
The task related user exit (TRUE) detected an invalid socket function on a call request from the CICS application program.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

function is the invalid socket function.

erro is the UNIX System Services return code. These return codes are listed in the sockets and sockets extended return codes (ERRNOs) in z/OS Communications Server: IP and SNA Codes.

tranid is the name of the CICS transaction.

taskid is the CICS task ID number.
System action

The TRUE is disabled and the task abends with an AEY9 CICS abend code.

Operator response

Correct the invalid socket function and try again.

The most probable errno is 10011 "INVALID SOCKET FUNCTION". If the socket function name appears correct, ensure that the application padded the function call with blanks.

System programmer response

None.

Module

EZACIC01

Destination

Task Related User Exit (TRUE)

EZY1349E: mm/dd/yy hh:mm:ss UNABLE TO OPEN CONFIGURATION FILE TRANSACTION=transactionid EIBRESP2=eibresp2

Explanation

The CICS Listener received an abnormal response from CICS when attempting to open the CICS Sockets configuration file (EZACONFG) using an EXEC CICS SET FILE call.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the transaction under which the Listener is executing.

eibresp2 is the EIBRESP2 value returned by CICS on the EXEC CICS SET FILE call as described in [http://www-01.ibm.com/software/htp/cics/library/](http://www-01.ibm.com/software/htp/cics/library/)

System action

The Listener ends.

Operator response

Contact the CICS system programmer.

System programmer response

Use the to interpret the value of EIBRESP2. If the file is not known to CICS, perform the installation steps for the configuration file.
Module

EZACIC02

Destination

LISTENER

EZY1350E: mm/dd/yy hh:mm:ss NOT AUTHORIZED TO USE api_function, action IGNORED. TERM=termid TRAN=transid

Explanation

The IP CICS socket interface uses a CICS EXTRACT EXIT command to determine whether the IP CICS Sockets Task Related User Exit (TRUE) is enabled. This action is performed by IP CICS socket interface initialization and shutdown programs, the Listener, and by any user application linking to the IP CICS domain name server module.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

api_function is the CICS command performed.

action is the action intended.

- ENABLE means the IP CICS socket interface is being enabled.
- DISABLE means the IP CICS socket interface is being disabled.
- STARTUP means the IP CICS socket interface is being started.

termid is the terminal ID where the transaction receiving the error is executing.

transid is the name of the transaction that is incurring the security violation.

System action

- If the TRUE is being enabled when the IP CICS socket interface is initializing, then the enable action is ignored and the interface is not activated.
- If the TRUE is being disabled when the IP CICS socket interface is shutting down, then the disable action is ignored and the interface remains active.
- If the IP CICS socket interface is being started, then the startup action is ignored and the interface remains inactive.

Operator response

Contact the CICS system programmer.

System programmer response

Ensure that the user ID being used is allowed at least UPDATE access to the EXITPROGRAM resource.

Module

EZACIC02, EZACIC21, EZACIC22
**Destination**

Listener, Initialization, Shutdown

**EZY1351E: mm/dd/yy hh:mm:ss EXIT PROGRAM (EZACIC01) IS NOT ENABLED, action IGNORED. TERM=termid TRAN=transid**

**Explanation**

The IP CICS socket interface uses a CICS ENABLE PROGRAM command to enable the IP CICS Sockets Task Related User Exit (TRUE). This action is performed by IP CICS socket interface initialization.

*mm/dd/yy* is the date (month/day/year) of the message.

*hh:mm:ss* is the time (hours:minutes:seconds) of the message.

*action* is the action intended.

- ENABLE means the IP CICS socket interface is being enabled.
- DISABLE means the IP CICS socket interface is being disabled.

*termid* is the terminal ID where the transaction receiving the error is executing.

*transid* is the name of the transaction that is incurring the security violation.

**System action**

The IP CICS socket interface is not initialized.

**Operator response**

Contact the CICS system programmer.

**System programmer response**

Ensure that the user ID being used is allowed at least UPDATE access to the EXITPROGRAM resource.

**Module**

EZACIC21

**Destination**

Initialization

**EZY1352E: mm/dd/yy hh:mm:ss SUBTASK ENDED UNEXPECTEDLY TRANSACTION= transactionid TASKID= taskid**

**Explanation**

The current tasks CICS Sockets subtask ended unexpectedly. This is probably caused by an ABEND of the subtask.

*mm/dd/yy* is the date (month/day/year) of the message.
hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the CICS transaction whose subtask ended unexpectedly.

taskid is the CICS task number of the task whose subtask ended unexpectedly.

**System action**

The CICS socket interface is disabled for the current task. Any subsequent CICS Sockets calls by that task will result in CICS ABEND code AEY9. Other tasks are not affected.

**Operator response**

Contact the CICS system programmer.

**System programmer response**

Check the console log for previous messages that explain what happened to the subtask.

**Module**

EZACIC01

**Destination**

TASK RELATED USER EXIT (TRUE)

**EZY1353E: mm/dd/yy hh:mm:ss COMMA MISSING AFTER IC TRANS ID = transactionid PARTNER IP ADDR = inetaddress PORT = portnumber**

**Explanation**

The listener did not find a comma delimiter after the interval control (IC) start type indicator in the client's transaction request message.

In the message text:

- **mm/dd/yy**
  - The date (month/day/year) of the message.

- **hh:mm:ss**
  - The time (hours:minutes:seconds) of the message.

- **transactionid**
  - The name of the transaction that was requested by the connecting client.

- **inetaddress**
  - The internet address of the connecting client.

- **portnumber**
  - The connecting client's port number.
Example

An example of a transaction request message for the standard listener:
SCCS,DATA,IC000010
EZY1258I 10/11/05 14:01:55 EZACIC02 ENTRY POINT IS 17CB2028
EZY1258I 10/11/05 14:01:55 EZACIC01 ENTRY POINT IS 177E2518
EZY1291I 10/11/05 14:01:56 LISTENER TRANSACTION= CSKL TASKID= 0000032L
ACCEPTING REQUESTS VIA PORT 3010
EZY1353E 10/11/05 14:02:56 COMMA MISSING AFTER IC TRANSACTION ID= SCCS
PARTNER INET ADDR=10.1.1.2 PORT= 1076

System action

The listener does not start the transaction specified by the client's transaction request message and ends the connection. This message is also returned to the client.

Operator response

Ensure that a comma delimiter separates the IC start type and the IC start time. See “IBM listener input format” on page 142 for information about the client's transaction request message.

User response

Not applicable.

System programmer response

None.

Problem determination

Not applicable.

Source

Module
EZACIC02
Routing code
Not applicable.
Descriptor code
Not applicable.

EZY1354I: mm/dd/yy hh:mm:ss CICS/ SOCKETS CICS TRACING
IS status
Explanation

This message shows the status of changing IP CICS Sockets CICS tracing and is issued when one of the following occurs:
• The operator issued the EZAO,START,TRACE transaction.
• The operator issued the EZAO,STOP,TRACE transaction.
• The CICS Master User Trace Flag is specified as OFF and the IP CICS Sockets TRACE configuration is specified as YES.

`mm/dd/yy` is the date (month/day/year) of the message.

`hh:mm:ss` is the time (hours:minutes:seconds) of the message.

`status` is the status of CICS tracing for the IP CICS socket interface.
• ENABLED indicates that the IP CICS socket interface will generate CICS trace data when CICS tracing is active.
• DISABLED indicates that the IP CICS socket interface will not generate CICS trace data.

**System action**

When `status` is ENABLED, IP CICS Sockets will generate CICS trace data when CICS tracing is active. When `status` is DISABLED, IP CICS Sockets will not generate CICS trace data.

**Operator response**

None.

**System programmer response**

None.

**Module**

EZACIC00, EZACIC01

**Destination**

TRC00000, SUB05100

**EZY1355I: mm/dd/yy hh:mm:ss CICS/ SOCKETS TCBLIM EXCEEDS MAXOPENTCBS**

**Explanation**

IP CICS Sockets has determined that the value specified for TCBLIM exceeds the value of MAXOPENTCBS allowed at the time the interface was enabled. TCBLIM will be forced to the same value as MAXOPENTCBS.

`mm/dd/yy` is the date (month/day/year) of the message.

`hh:mm:ss` is the time (hours:minutes:seconds) of the message.

**System action**

IP CICS Sockets TCBLIM will default to the value of MAXOPENTCBS. IP CICS Sockets processing continues.
Operator response

Contact the CICS system programmer.

System programmer response

Adjust the value specified by the TCBLIM configuration option using one or more of the following methods:

- Specify an appropriate TCBLIM value on the EZACICD TYPE=CICS,TCBLIM= macro.
- Specify an appropriate TCBLIM value using the EZAC Configuration transaction.
- Specify an appropriate TCBLIM value dynamically by using the EZAO Operator transaction.
- Specify an appropriate MAXOPENTCBS value using the CICS System Initialization parameters.
- Specify an appropriate MAXOPENTCBS value using the CICS Master Terminal transaction, CEMT SET DISPATCHER MAXOPENTCBS.

See the following sections in:

- “Building the configuration data set with EZACICD” on page 52 for information about using the EZACICD macro.
- “Customizing the configuration transaction (EZAC)” on page 68 for information about the EZAC Configuration transaction.
- “Using the SET function” on page 112 and “Using the INQUIRE function” on page 109 for information about the EZAO Operator transaction.
- “TYPE parameter for EZACICD” on page 54 for a description of the TCBLIM parameter.


Module

EZACIC21

Destination

Initialization

**EZY1356E: mm/dd/yy hh:mm:ss CICS/ SOCKETS TCBLIM HAS BEEN REACHED**

Explanation

The number of IP CICS Sockets-enabled CICS tasks using an Open API, L8, TCB is equal to the value specified by the TCBLIM configuration option.

*mm/dd/yy* is the date (month/day/year) of the message.

*hh:mm:ss* is the time (hours:minutes:seconds) of the message.
System action

The IP CICS socket interface will suspend any new tasks until one of the following actions occur:

- The IP CICS Sockets TCBLIM value is increased.
- Existing transactions using IP CICS Sockets end.

This message will be issued only when the interface detects that it has reached TCBLIM. EZY1360I will be issued when this condition is relieved.

Operator response

Contact the CICS system programmer.

System programmer response

Use the CICS Master Terminal transaction, CEMT INQ TASK HVALUE(ATTCBLIM), to determine which IP CICS Sockets-enabled CICS transactions are subject to TCBLIM. Either take action to reduce the IP CICS Sockets work load or increase the IP CICS Socket TCBLIM configuration option. You can use the EZAO,SET,CICS Operator transaction to dynamically increase TCBLIM. The new value you set for the TCBLIM configuration option must be less than or equal to the value specified by MAXOPENTCBS.

Module

EZACIC01

Destination

SUB16000

EZY1357I: mm/dd/yy hh:mm:ss TRANSIENT DATA QUEUE SPECIFIED ON ERRORTD IS NOT DEFINED TO CICS

Explanation

IP CICS Sockets has determined that the CICS transient data queue specified by the ERRORTD configuration option was not defined to the CICS region where the IP CICS socket interface is enabled.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

System action

The CSMT transient data queue will be used for reporting all IP CICS Sockets interface messages. CSMT is the default CICS transient data queue name.

Operator response

Contact the CICS system programmer.
System programmer response

Ensure that the CICS transient data queue specified by the ERRORTD configuration option is properly defined to CICS.

See "Defining the TCPM transient data queue for CICS TCP/IP" on page 38 for more information.

Module

EZACIC21

Destination

Initialization

EZY1358E: 10999 ABEND - IP CICS SOCKETS USING OTE

Explanation

IP CICS Sockets has incorrectly called the MVS subtask wrapper module when the interface was enabled to use CICS Open Transaction Environment.

System action

The IP CICS socket interface will stop.

Operator response

Contact the CICS system programmer.

System programmer response


Module

EZACIC03

Destination

MVS SUBTASK

EZY1359I: mm/dd/yy hh:mm:ss CICS/ SOCKETS APPLICATIONS WILL USE THE QR TCB

Explanation

IP CICS Sockets has determined that CICS FORCEQR=YES is specified.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.
System action

CICS will force all user application programs, including those enabled to IP CICS Sockets, that are specified as threadsafe to run under the CICS Quasi-Reentrant (QR) TCB, as if they were specified as quasi-reentrant programs.

Operator response

Contact the CICS system programmer.

System programmer response

If you do not want to incur the overhead of CICS switching Open API-enabled tasks back to the QR TCB, then change the value of FORCEQR to NO. See [http://www-01.ibm.com/software/htp/cics/library/](http://www-01.ibm.com/software/htp/cics/library/) for more details about the following information:

- FORCEQR CICS system initialization parameter.
- CICS master terminal transaction that is used to dynamically change the FORCEQR setting.

Module

EZACIC21

Destination

Initialization

**EZY1360I: mm/dd/yy hh:mm:ss CICS/ SOCKETS TCBLIM CONDITION HAS BEEN RELIEVED**

Explanation

IP CICS Sockets enable transactions are no longer suspended due to TCBLIM.

*mm/dd/yy* is the date (month/day/year) of the message.

*hh:mm:ss* is the time (hours:minutes:seconds) of the message.

System action

Any new or suspended IP CICS Sockets work will now be processed without being suspended due to IP CICS Sockets being at TCBLIM.

Operator response

None.

System programmer response

None.

Module

EZACIC01
Destination

SUB16000, Task termination

**EZY1361E: mm/dd/yy hh:mm:ss CICS/TS OPEN TRANSACTION ENVIRONMENT SUPPORT IS NOT AVAILABLE**

**Explanation**

The IP CICS Sockets OTE configuration parameter is specified as YES. IP CICS Sockets determined that the CICS environment that is required to support the exploitation of CICS Open Transaction Environment by IP CICS Sockets is not available.

*mm/dd/yy* is the date (month/day/year) of the message.

*hh:mm:ss* is the time (hours:minutes:seconds) of the message.

**System action**

The IP CICS socket interface is not enabled to use CICS Open Transaction Environment.

**Operator response**

Contact the system programmer.

**System programmer response**

Perform one of the following:

- Upgrade the level of CICS to support Open Transaction Environment. The CICS Open Transaction Environment requires CICS/TS V2R2 or later.
- Change the IP CICS socket interface configuration to use MVS subtasks when configuring it by using the EZAC configuration transaction or the EZACICD macro.

**Module**

EZACIC21

**Destination**

Initialization

**EZY1362E: mm/dd/yy hh:mm:ss CICS/SOCKETS START OF LISTENER transactionid FAILED RESP1= resp1 RESP2=resp2**

**Explanation**

CICS Sockets attempted to start the specified listener, but the EXEC CICS START command failed with the RESP1 and RESP2 values listed in the message text.

*mm/dd/yy* is the date (month/day/year) of the message.

*hh:mm:ss* is the time (hours:minutes:seconds) of the message.
**transactionid** is the transaction name of the listener that the CICS Sockets attempted to start.

**resp1** is the RESP1 value returned by the EXEC CICS START transaction.

**resp2** is the RESP2 value returned by the EXEC CICS START transaction.

**System action**
The CICS Listener does not start.

**Operator response**
None.

**System programmer response**

- If the RESP2 value is 8 or 9, then the problem is related to the USERID parameter in the definition of the listener. Verify that the USERID parameter is correct. See Chapter 2, “Setting up and configuring CICS TCP/IP,” on page 23 for a description of the USERID parameter.
- If the RESP2 value is 8, then the USERID parameter of the listener definition specifies a user ID that is not known to RACF. Therefore, either change the USERID parameter or define the user ID to RACF.
- If the RESP2 value is 9, then the user ID under which the EXEC CICS START was issued does not have SURROGAT security access to the user ID that is specified in the USERID parameter. For example, if the failure occurs during CICS PLT processing, then the PLT user ID does not have SURROGAT security access to the listener's user ID. See http://www-01.ibm.com/software/cics/library/ for more information.

**Module**
EZACIC21

**Destination**
INITIALIZATION

**EZY1363I**: *mm/dd/yy hh:mm:ss LISTENER transactionid taskno HAD threads THREADS ACTIVE WHEN STACK tcpname ENDED*

**Explanation**
This message displays the number of listener threads that were active when the TCP/IP stack that is specified ended. This message is followed by one or more EZY1368I messages that describe the clients that are affected.

In the message text:

*mm/dd/yy*
  The date (month/day/year) of the message.
hh:mm:ss
The time (hours:minutes:seconds) of the message.

transactionid
The listener's transaction ID.

taskno
The task number assigned by CICS.

threads
The number of threads that were active when the specified TCP/IP stack ended.

tcpname
The TCP/IP procedure name with which the listener had affinity.

Example
Following is an example of the messages that are displayed when the stack has ended while the listener was processing data.

EZY1369E 01/10/06 12:59:32 LISTENER CSKL 10295 IS DELAYED, STACK TCPCS IS UNAVAILABLE
EZY1363I 01/10/06 12:59:33 LISTENER CSKL 10295 HAD 5 THREADS ACTIVE WHEN STACK TCPCS ENDED
EZY1367I 01/10/06 12:59:33 SOCK# IP ADDRESS PORT CHILD
EZY1368I 01/10/06 12:59:33 2 10.11.1.2 10245 PAYR
EZY1368I 01/10/06 12:59:33 12 2001:DB8:10::11:2:1 21069
EZY1368I 01/10/06 12:59:33 15 10.91.1.1 10245 INVN
EZY1368I 01/10/06 12:59:33 19 10.81.1.1 21212 ACCT
EZY1368I 01/10/06 12:59:33 999 2001:DB8:10::11:1:2 00901 ORDR

System action
Processing continues.

Operator response
No action needed.

User response
No action needed.

System programmer response
No action needed.

Problem determination
Not applicable.

Source
z/OS Communications Server TCP/IP: CICS Socket Interface and API

Module
EZACIC02
Routing code

10

Descriptor code

12

Automation

This message is sent to the CICS transient data queue that is specified by the IP CICS Sockets ERRORTD configuration option.

EZY1364I: mm/dd/yy hh:mm:ss LISTENER transactionid DETECTED THAT TTLS IS status ON STACK tcpname

Explanation

The CICS Listener is defined with a GETTID parameter of YES which indicates that the listener is requested to attempt to obtain the connecting client certificates and user IDs from Application Transparent Transport Layer Security (AT-TLS). If status is DISABLED, then AT-TLS is disabled in the TCP/IP stack. Therefore, the listener is unable to obtain client certificates and user IDs as requested by the GETTID parameter. If status is ENABLED, then AT-TLS has been enabled in the TCP/IP stack, making it possible for the listener to obtain client certificates and user IDs.

mm/dd/yy is the date (month/day/year) of the message.

hh:mm:ss is the time (hours:minutes:seconds) of the message.

transactionid is the name of the listeners CICS transaction.

status is the status of AT-TLS in the TCP/IP stack. status is either DISABLED or ENABLED.

tcpname is the name of the TCP/IP stack.

System action

The listener continues its normal processing, which includes attempting to obtain client certificates and User IDs.

Operator response

Contact the system programmer.

System programmer response

No response is needed if status is ENABLED. If status is DISABLED, then verify that the GETTID parameter of YES is correct in the listener definition. If so, request that your AT-TLS administrator investigate why AT-TLS is not enabled in the TCP/IP stack. See Chapter 2, “Setting up and configuring CICS TCP/IP,” on page 23 for a description of the GETTID parameter.
Module
EZACIC02

Destination
LISTENER

EZY1365E: mm/dd/yy hh:mm:ss LISTENER transactionid taskno IS NOT ACCEPTING REQUESTS ON PORT port

Explanation

The listener identified by the specified transaction ID and task number cannot process inbound connections because the listener's socket descriptor table is full.

In the message text:

mm/dd/yy
   The date (month/day/year) of the message.

hh:mm:ss
   The time (hours:minutes:seconds) of the message.

transactionid
   The name of the listener's transaction that cannot accept new client connections.

taskno
   The task number assigned by CICS.

port
   The port number on which the specified listener is listening.

Example
EZY1365E 01/19/06 10:07:33 LISTENER CSKL 0000079 IS NOT ACCEPTING REQUESTS AT PORT 3010

System action

The listener does not accept new connections until the number of socket descriptors currently being processed by the listener is less than the value specified by the lesser of either the system MAXFILEPROC parameter or the listener user ID's FILEPROCMAX parameter.

Operator response

Contact the system programmer.

User response

No action needed.
System programmer response

Perform any of the following actions as appropriate:

- If the ERRORTD log indicates that the child server transaction failed to take the client's given socket, then investigate the CICS region where the child server transaction runs.

  See the steps for diagnosing TCP/IP clients that are unable to connect in z/OS Communications Server: IP Diagnosis Guide for information about diagnosing child server transactions problems.


- If the listeners NUMSOCK value is greater than or equal to the value specified by the MAXFILEPROC parameter, then perform one of the following actions:
  - Set the NUMSOCK value to be less than the MAXFILEPROC value using either the EZACICD macro or the EZAC configuration transaction and then restart the listener. See the information about “Configuring the CICS TCP/IP environment” on page 52 for more information about using the EZACICD macro and the EZAC configuration transaction.
  - Set the MAXFILEPROC value to be greater than the NUMSOCK value using the SETOMVS system command. See the SETOMVS command information in z/OS MVS System Commands for information about dynamically changing the MAXFILEPROC option that z/OS UNIX System Services is currently using.

- If the listener user ID FILEPROCMAX value is less than the value specified by the NUMSOCK parameter, set the FILEPROCMAX value to be greater than the value specified by the NUMSOCK parameter. For more information about the FILEPROCMAX specification, see the documentation provided for the SAF product that is in use on your system. If you are using RACF, see the information about the FILEPROCMAX parameter in z/OS Security Server RACF Security Administrator’s Guide.

Problem determination

See the system programmer response.

Source

z/OS Communications Server TCP/IP: CICS Socket Interface and API

Module

EZACIC02

Routing code

1

Descriptor code

2
Automation

This message is sent to the system console and to the CICS transient data queue that is specified by the IP CICS Sockets ERRORTD configuration option.

**EZY1366E: mm/dd/yy hh:mm:ss CICS/SOCKETS LISTENER TRANSACTION tranid IS ALREADY ACTIVE**

**Explanation**

The IP CICS Sockets Listener determined that another listener with the same transaction ID is already active.

*mm/dd/yy* is the date (month/day/year) of the message.

*hh:mm:ss* is the time (hours:minutes:seconds) of the message.

*tranid* is the CICS transaction identifier of the duplicate IP CICS Sockets Listener.

**System action**

The IP CICS Sockets Listener that issued this message ends.

**Operator response**

Contact the system programmer.

**System programmer response**

Change the Listeners CICS transaction identifier or port number to ensure that the definition is unique. See [Chapter 2, “Setting up and configuring CICS TCP/IP,” on page 23](#) for more information about configuring the IP CICS Sockets Listener.

**Module**

EZACIC02

**Destination**

Initialization

**EZY1367I: mm/dd/yy hh:mm:ss SOCK# IP ADDRESS PORT CHILD**

**Explanation**

The listener was processing client connections when its TCP/IP stack ended. This message is issued when the listener has accepted sockets that were not taken by child server tasks. This message is a header message for the EZY1368I detail messages that follow. This message accompanies an EZY1363I message.

In the message text:

*mm/dd/yy*

The date (month/day/year) of the message.
hh:mm:ss

The time (hours:minutes:seconds) of the message.

Example

Following is an example of the messages displayed when the stack has ended while the listener was processing data.

EZY1369E 01/10/06 12:59:32 LISTENER CSKL 10295 IS DELAYED, STACK TCPCS IS UNAVAILABLE
EZY1363I 01/10/06 12:59:33 LISTENER CSKL 10295 HAD 5 THREADS ACTIVE WHEN STACK TCPCS ENDED
EZY1367I 01/10/06 12:59:33 SOCK# IP ADDRESS PORT CHILD
EZY1368I 01/10/06 12:59:33 2 10.11.1.2 10245 PAYR
EZY1368I 01/10/06 12:59:33 12 2001:DB8:10::11:2:1 21089
EZY1368I 01/10/06 12:59:33 15 10.91.1.1 10245 INWN
EZY1368I 01/10/06 12:59:33 19 10.81.1.1 21212 ACCT
EZY1368I 01/10/06 12:59:33 999 2001:DB8:10::11:1:2 00901 ORDR

System action

Processing continues.

Operator response

No action needed.

User response

No action needed.

System programmer response

No action needed.

Problem determination

Not applicable.

Source

z/OS Communications Server TCP/IP: CICS Socket Interface and API

Module

EZACIC02

Routing code

10

Descriptor code

12

Automation

This message is sent to the CICS transient data queue that is specified by the IP CICS Sockets ERRORTD configuration option.
EZY1368I: mm/dd/yy hh:mm:ss sock# ipaddr port tran

Explanation

The listener was processing client connections when its TCP/IP stack ended. This message is issued when the listener has accepted sockets that were not taken by child server tasks. One EZY1368I message is issued for each client connection that is being processed.

In the message text:

mm/dd/yy
The date (month/day/year) of the message.

hh:mm:ss
The time (hours:minutes:seconds) of the message.

sock#
The listener's socket number.

ipaddr
The client's IP address.

port
The client's port number.

tran
The child server's transaction ID. A blank child server transaction ID indicates that the ID has not yet been determined.

Example

Following is an example of the messages displayed when the stack has ended while the listener was processing data.

EZY1369E 01/10/06 12:59:32 LISTENER CSKL 10295 IS DELAYED, STACK TCPCS IS UNAVAILABLE
EZY1363I 01/10/06 12:59:33 LISTENER CSKL 10295 HAD 5 THREADS ACTIVE WHEN STACK TCPCS ENDED
EZY1367I 01/10/06 12:59:33 SOCK# IP ADDRESS PORT CHILD
EZY1368I 01/10/06 12:59:33 2 10.11.1.2 10245 PAYR
EZY1368I 01/10/06 12:59:33 12 2001:DB8:10::11:2:1 21089
EZY1368I 01/10/06 12:59:33 15 10.91.1.1 10245 INWN
EZY1368I 01/10/06 12:59:33 19 10.81.1.1 21212 ACCT
EZY1368I 01/10/06 12:59:33 999 2001:DB8:10::11:1:2 00901 ORDR

System action

Processing continues.

Operator response

No action needed.

User response

No action needed.

System programmer response

No action needed.
Problem determination

Not applicable.

Source

z/OS Communications Server TCP/IP: CICS Socket Interface and API

Module

EZACIC02

Routing code

10

Descriptor code

12

Automation

This message is sent to the CICS transient data queue that is specified by the IP CICS Sockets ERRORTD configuration option.

**EZY1369E**: *mm/dd/yy hh:mm:ss LISTENER transactionid taskno IS DELAYED, STACK tcpname IS UNAVAILABLE.*

Explanation

The TCP/IP stack assigned to the specified listener is not active.

In the message text:

*mm/dd/yy*

The date (month/day/year) of the message.

*hh:mm:ss*

The time (hours:minutes:seconds) of the message.

*transactionid*

The listener's transaction ID.

*taskno*

The task number assigned by CICS.

*tcpname*

The TCP/IP procedure name with which the listener had affinity.

Example

The following is an example of the messages displayed when the stack has ended while the listener was processing data.

EZY1369E 01/10/06 12:59:32 LISTENER CSKL 10295 IS DELAYED, STACK TCPCS IS UNAVAILABLE
EZY1363I 01/10/06 12:59:33 LISTENER CSKL 10295 HAD 5 THREADS ACTIVE WHEN STACK TCPCS ENDED
EZY1367I 01/10/06 12:59:33 SOCK# IP ADDRESS PORT CHILD
EZY1368I 01/10/06 12:59:33 2 10.11.1.2 10245 PAYR
EZY1368I 01/10/06 12:59:33 12 2001:DB8:10::11:2:1 21089
System action

The listener releases any resources and connects to the TCP/IP stack specified by the tcpname value. If the connection fails because the stack is not active, then the listener delays using the time value specified by its RTYTIME configuration option and attempts to reconnect. See the “TYPE=LISTENER setting for the TYPE parameter” on page 59 for information about setting the listener's RTYTIME value.

Operator response

Start or restart the TCP/IP address space specified by the tcpname value.

User response

No action needed.

System programmer response

No action needed.

Problem determination

Not applicable.

Source

z/OS Communications Server TCP/IP: CICS Socket Interface and API Module

EZACIC02

Routing code

1

Descriptor code

2

Automation

This message is sent to the system console and to the CICS transient data queue that is specified by the IP CICS Sockets ERRORTD configuration option.
**EZY1370I**: *mm/dd/yy hh:mm:ss LISTENER transactionid NUMSOCK numsock IS EQUAL TO OR GREATER THAN MAXFILEPROC maxfileproc*

**Explanation**

A listener startup run-time check determined that the z/OS UNIX System Services MAXFILEPROC value was less than or equal to the listener's NUMSOCK value. The listener's accept processing pauses when the number of sockets that are supported by this listener exceeds the MAXFILEPROC value. No new connections are accepted until the number of sockets that are supported by this listener is less than the MAXFILEPROC value.

In the message text:

- **mm/dd/yy**
  - The date (month/day/year) of the message.
- **hh:mm:ss**
  - The time (hours:minutes:seconds) of the message.
- **transactionid**
  - The listener's transaction ID.
- **numsock**
  - The number of sockets supported by this listener.
- **maxfileproc**
  - The maximum number of descriptors for files, sockets, directories, and any other file-system objects that can be concurrently active or allocated by a single process.

**Example**

```
EZY1370I 01/19/06 10:07:33 LISTENER CSKL NUMSOCK 2000 IS EQUAL TO OR GREATER THAN MAXFILEPROC 250
```

**System action**

Processing continues.

**Operator response**

Contact the system programmer.

**User response**

No action needed.

**System programmer response**

Perform one of the following actions:

- Set the NUMSOCK value to be less than the MAXFILEPROC value using either the EZACICD macro or the EZAC configuration transaction, and then restart the listener. See the information about "Configuring the CICS TCP/IP environment" on page 52 for more information about using the EZACICD macro and the EZAC configuration transaction.
- Set the MAXFILEPROC value to be greater than the NUMSOCK value using the SETOMVS system command. See the SETOMVS command information in z/OS.
MVS System Commands for information about dynamically changing the MAXFILEPROC option that z/OS UNIX System Services is currently using.

Problem determination

Not applicable.

Source

z/OS Communications Server TCP/IP: CICS Socket Interface and API

Module

EZACIC02

Routing code

10

Descriptor code

12

Automation

This message is sent to the CICS transient data queue that is specified by the IP CICS Sockets ERRORTD configuration option.

EZY1371E: mm/dd/yy hh:mm:ss AUTOMATIC APPLDATA
REGISTRATION FAILED FOR TRANSACTION= transactionid
TASKNO= taskno ERRNO= errno

Explanation

The automatic registration of application data failed for the reason described by the errno value.

In the message text:

mm/dd/yy

The date (month/day/year) of the message.

hh:mm:ss

The time (hours:minutes:seconds) of the message.

transactionid

The listener's transaction ID.

taskno

The task number assigned by CICS.

errno

errno is the UNIX System Services return code. These return codes are listed in
the sockets and sockets extended return codes (ERRNOs) in z/OS
Communications Server: IP and SNA Codes.
Example
EZY1371E 07/01/06 10:07:33 AUTOMATIC APPLDATA REGISTRATION FAILED FOR TRANSACTION= CSKL TASKNO= 0000022L ERRNO= 55

System action

The application continues.

Operator response

Contact the system programmer.

User response

Not applicable.

System programmer response

See the information about automatically registering application data in z/OS Communications Server: IP Programmer's Guide and Reference for information about the socket commands affected by the automatic registration of application data.

errno is the UNIX System Services return code. See the sockets and sockets extended return codes (ERRNOs) information in z/OS Communications Server: IP and SNA Codes for the action that you should take based on the SIOCSAPPLDATA IOCTL socket command return code.

Problem determination

See the system programmer response.

Source

z/OS Communications Server TCP/IP: CICS Socket Interface and API

Module

EZACIC01, EZACIC02

Routing code

10

Descriptor code

12

Automation

This message is sent to the CICS transient data queue that is specified by the IP CICS Sockets ERRORTD configuration option.
Appendix E. Sample programs

This topic contains the following samples:

- EZACICSC - An IPv4 child server, see "EZACICSC" on page 577
- EZACICSS - An IPv4 iterative server, see "EZACICSS" on page 577
- EZACIC6C - An IPv6 child server, see "EZACIC6C" on page 595
- EZACIC6S - An IPv6 iterative server, see "EZACIC6S" on page 604
- EZACICAC - An assembler child server, see "EZACICAC" on page 623
- EZACICAS - An assembler iterative server, see "EZACICAS" on page 630
- SELECTEX - The SELECTEX socket call, see "SELECTEX" on page 644

EZACICSC

The following COBOL socket program is in the SEZAINST data set.

Figure 178. EZACICSC IPv4 child server sample

```
* Communications Server for z/OS, Version 1, Release 9
* Copyright: Licensed Materials - Property of IBM
* "Restricted Materials of IBM"
* 5694-A01
* Copyright IBM Corp. 1993, 2007
* US Government Users Restricted Rights - Use, duplication or disclosure restricted by
  GSA ADP Schedule Contract with IBM Corp.
* Status: CSV1R9
* $MOD(EZACICSC),COMP(CICS),PROD(TCPIP):
* $SEG(EZACICSC)
* Module Name : EZACICSC
* Description :
  This is a sample CICS/TCP application program. It issues
  TAKESOCKET to obtain the socket passed from MASTER
  SERVER and perform dialog function with CLIENT program.
IDENTIFICATION DIVISION.
PROGRAM-ID. EZACICSC.
ENVIRONMENT DIVISION.
DATA DIVISION.
* WORKING-STORAGE SECTION.
  77 TASK-START PIC X(40)
     VALUE IS 'TASK STARTING THRU CICS/TCP/IP INTERFACE'.
  77 TAKE-ERR PIC X(24)
     VALUE IS 'TAKESOCKET FAIL'.
```
77 TAKE-SUCCESS PIC X(24) VALUE 'TAKESOCKET SUCCESSFUL'.
77 READ-ERR PIC X(24) VALUE 'READ SOCKET FAIL'.
77 READ-SUCCESS PIC X(24) VALUE 'READ SOCKET SUCCESSFUL'.
77 WRITE-ERR PIC X(24) VALUE 'WRITE SOCKET FAIL'.
77 WRITE-END-ERR PIC X(32) VALUE 'WRITE SOCKET FAIL - PGM END MSG'.
77 WRITE-SUCCESS PIC X(25) VALUE 'WRITE SOCKET SUCCESSFUL'.
77 CLOS-ERR PIC X(24) VALUE 'CLOSE SOCKET FAIL'.
77 CLOS-SUCCESS PIC X(24) VALUE 'CLOSE SOCKET SUCCESSFUL'.
77 INVREQ-ERR PIC X(24) VALUE 'INTERFACE IS NOT ACTIVE'.
77 IOERR-ERR PIC X(24) VALUE 'IOERR OCCURRS'.
77 LENGERR-ERR PIC X(24) VALUE 'LENGERR ERROR'.
77 ITEMERR-ERR PIC X(24) VALUE 'ITEMERR ERROR'.
77 NOSPACE-ERR PIC X(24) VALUE 'NOSPACE CONDITION'.
77 QIDERR-ERR PIC X(24) VALUE 'QIDERR CONDITION'.
77 ENDDATA-ERR PIC X(30) VALUE 'RETRIEVE DATA CAN NOT BE FOUND'.
77 WRKEND PIC X(20) VALUE 'CONNECTION END'.
77 WRITE-SW PIC X(1) VALUE 'N'.
77 FORCE-ERROR-MSG PIC X(1) VALUE 'N'.

01 SOKET-FUNCTIONS.
  02 SOKET-ACCEPT PIC X(16) VALUE 'ACCEPT'.
  02 SOKET-BIND PIC X(16) VALUE 'BIND'.
  02 SOKET-CLOSE PIC X(16) VALUE 'CLOSE'.
  02 SOKET-CONNECT PIC X(16) VALUE 'CONNECT'.
  02 SOKET-FCNTL PIC X(16) VALUE 'FCNTL'.
  02 SOKET-GETCLIENTID PIC X(16) VALUE 'GETCLIENTID'.
  02 SOKET-GETHOSTBYADDR PIC X(16) VALUE 'GETHOSTBYADDR'.
  02 SOKET-GETHOSTBYNAME PIC X(16) VALUE 'GETHOSTBYNAME'.
  02 SOKET-GETHOSTID PIC X(16) VALUE 'GETHOSTID'.
  02 SOKET-GETHOSTNAME PIC X(16) VALUE 'GETHOSTNAME'.
  02 SOKET-GETPEERNAME PIC X(16) VALUE 'GETPEERNAME'.
  02 SOKET-GETSOCKNAME PIC X(16) VALUE 'GETSOCKNAME'.
  02 SOKET-GETSOCKOPT PIC X(16) VALUE 'GETSOCKOPT'.
  02 SOKET-GIVESOCKET PIC X(16) VALUE 'GIVESOCKET'.
  02 SOKET-INITAPI PIC X(16) VALUE 'INITAPI'.
  02 SOKET-IOCTL PIC X(16) VALUE 'IOCTL'.
  02 SOKET-LISTEN PIC X(16) VALUE 'LISTEN'.
  02 SOKET-READ PIC X(16) VALUE 'READ'.
  02 SOKET-RECV PIC X(16) VALUE 'RECV'.
  02 SOKET-RECVFROM PIC X(16) VALUE 'RECVFROM'.
  02 SOKET-SELECT PIC X(16) VALUE 'SELECT'.
  02 SOKET-SEND PIC X(16) VALUE 'SEND'.
  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 'DATA RECEIVED'.

*---------------------------------------------------------------*
* program's variables *
*---------------------------------------------------------------*
77 SUBTRACE PIC X(8) VALUE 'CONTRACE'.
77 RESPONSE PIC 9(9) COMP.
77 TASK-FLAG PIC X(1) VALUE '0'.
77 TAKE-socket PIC 9(8) COMP.
77 SOCKID PIC 9(4) COMP.
77 SOCKID-FWD PIC 9(8) COMP.
PROCEDURE DIVISION.

MOVE 'Y' TO WRITE-SW.

EXEC CICS HANDLE CONDITION INVREQ (INVREQ-ERR-SEC)
  IOERR (IOERR-SEC)
  ENDDATA (ENDDATA-SEC)
  LENGERR (LENGERR-SEC)
  NOSPACE (NOSPACE-ERR-SEC)
  QIDERR (QIDERR-SEC)
  ITEMERR (ITEMERR-SEC)
END-EXEC.

PERFORM INITIAL-SEC THRU INITIAL-SEC-EXIT,
TAKESOCKET-SEC THRU TAKESOCKET-SEC-EXIT.

MOVE '0' TO TASK-FLAG.
PERFORM CLIENT-TASK THRU CLIENT-TASK-EXIT
VARYING CNT FROM 1 BY 1 UNTIL TASK-FLAG = '1'.

CLOSE- SOCK.

*---------------------------------------------------------------*
*   CLOSE 'accept descriptor'                             *
*---------------------------------------------------------------*

CALL 'EZASOKET' USING SOKET-CLOSE SOCKID
          ERRNO RETCODE.

IF RETCODE < 0 THEN
  MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
  MOVE CLOS-ERR TO ERR-MSG
  MOVE SOCKID TO ERR-SOCKET
  MOVE RETCODE TO ERR-RETCODE
  MOVE ERRNO TO ERR-ERRNO
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 SOCKET-TAKESOCKET SOCKID
          CLIENTID-LSTN ERRNO RETCODE.
IF RETCODE < 0 THEN
  MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
  MOVE TAKE-ERR TO ERR-MSG
  MOVE SOCKID TO ERR-SOCKET
  MOVE RETCODE TO ERR-RETCODE
  MOVE ERRNO TO ERR-ERRNO
  MOVE CICS-ERR-AREA TO CICS-MSG-AREA
  PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
  GO TO PGM-EXIT
ELSE
  MOVE SPACES TO CICS-MSG-AREA
  MOVE WRITE-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 'EZACICO4' USING TCP-BUF TCPLENG.
  CALL 'EZASOKEI' USING SOKET-WRITE SOCKID TCPLENG
  TCP-BUF ERRNO RETCODE.

  IF RETCODE < 0 THEN
    MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
    MOVE WRITE-ERR TO ERR-MSG
    MOVE SOCKID TO ERR-SOCKET
    MOVE RETCODE TO ERR-RETCODE
    MOVE ERRNO TO ERR-ERRNO
    MOVE CICS-ERR-AREA TO CICS-MSG-AREA
    PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
    GO TO PGM-EXIT
  ELSE
    MOVE WRITE-SUCCESS TO CICS-MSG-AREA
    PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
  TAKESOCKET-SEC-EXIT.
  EXIT.

CLIENT-TASK.
*---------------------------------------------------------------------*
  * Issue ‘RECV’ socket to receive input data from client *
  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*---------------------------------------------------------------------*

  MOVE LOW-VALUES TO TCP-BUF.
  MOVE 200 TO TCPLENG.
  MOVE ZEROS TO RECV-FLAG.
  CALL 'EZASOKEI' USING SOKET-RECV SOCKID
  RECV-FLAG TCPLENG TCP-BUF ERRNO RETCODE.

  IF RETCODE < 0 THEN
    MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
    MOVE READ-ERR TO ERR-MSG
    MOVE SOCKID TO ERR-SOCKET
    MOVE RETCODE TO ERR-RETCODE
    MOVE ERRNO TO ERR-ERRNO
    MOVE CICS-ERR-AREA TO CICS-MSG-AREA
    PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
    GO TO PGM-EXIT
  ELSE
    MOVE READ-SUCCESS TO CICS-MSG-AREA
    PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.

  * REMOVE FOLLOWING STATEMENT IF USING EBCDIC CLIENT
  * CALL 'EZACIC05' USING TCP-BUF TCPLENG.
  *
  * DETERMINE WHETHER THE CLIENT IS FINISHED SENDING DATA
* IF TCP-BUF-H = 'END' OR TCP-BUF-H = 'end' THEN
  MOVE '1' TO TASK-FLAG
  PERFORM CLIENT-TALK-END THRU CLIENT-TALK-END-EXIT
  GO TO CLIENT-TASK-EXIT.

  IF RETCODE = 0 THEN
    MOVE '1' TO TASK-FLAG
    GO TO CLIENT-TASK-EXIT.

** ECHO RECEIVING DATA
**-----------------------------------------------
** MOVE TCP-BUF TO CICS-MSG-AREA.
** PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
** MOVE RETCODE TO TCPLENG.
** * REMOVE FOLLOWING STATEMENT IF USING EBCDIC CLIENT
** CALL 'EZACIC04' USING TCP-BUF TCPLENG.
** CALL 'EZASOKET' USING SOKET-WRITE SOCKID TCPLENG
** TCP-BUF ERRNO RETCODE.
** IF RETCODE < 0 THEN
**  MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
**  MOVE WRITE-ERR TO ERR-MSG
**  MOVE SOCKID TO ERR-SOCKET
**  MOVE RETCODE TO ERR-RETCODE
**  MOVE ERRNO TO ERR-ERRNO
**  MOVE CICS-ERR-AREA TO CICS-MSG-AREA
**  PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
**  GO TO PGM-EXIT ELSE
**  MOVE WRITE-SUCCESS TO CICS-MSG-AREA
**  PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
** CLIENT-TASK-EXIT.
** EXIT.

WRITE-CICS.
  MOVE 78 TO CLENG.
  MOVE EIBTASKN TO TASK-NUMBER.
  IF WRITE-SW = 'Y' THEN
    IF INTERFACE-IS-THREADSAFE THEN
      IF FORCE-ERROR-MSG = 'Y' THEN
        EXEC CICS WRITEQ TD QUEUE('CSMT') FROM(TD-MSG)
        LENGTH(CLENG) NOHANDLE END-EXEC
      ELSE
        NEXT SENTENCE
      END-EXEC
    ELSE
      EXEC CICS WRITEQ TD QUEUE('CSMT') FROM(TD-MSG)
      LENGTH(CLENG) NOHANDLE END-EXEC
    ELSE
      NEXT SENTENCE.
      MOVE SPACES TO CICS-MSG-AREA.
    END-EXEC
  END-EXEC

WRITE-CICS-EXIT.
EXIT.

CLIENT-TALK-EXIT.
MOVE LOW-VALUES TO TCP-BUF.
MOVE WRKEND TO TCP-BUF CICS-MSG-AREA.
MOVE 50 TO TCPLENG.

* REMOVE FOLLOWING STATEMENT IF USING EBCDIC CLIENT
* CALL 'EZACIC04' USING TCP-BUF TCPLENG.
  CALL 'EZASOKET' USING SOKET-WRITE SOCKID TCPLENG
  TCP-BUF ERRNO RETCODE.
  IF RETCODE < 0 THEN
    MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
    MOVE WRITE-END-ERR TO ERR-MSG
    MOVE SOCKID TO ERR-SOCKET

576 z/OS V2R1.0 Communications Server: IP CICS Sockets Guide
The following COBOL socket program is in the SEZAINST data set.
Figure 179. EZACICSS IPv4 iterative server sample

*******************************************************************************
+ Communications Server for z/OS, Version 1, Release 9
+ *
+ Copyright: Licensed Materials - Property of IBM
+ "Restricted Materials of IBM"
+ 5694-A01
+ Copyright IBM Corp. 1977, 2007
+ *
+ US Government Users Restricted Rights -
+ Use, duplication or disclosure restricted by
+ GSA ADP Schedule Contract with IBM Corp.
+ *
+ Status: CSV1R9
+ *
+ $MOD(EZACICSS),COMP(CICS),PROD(TCPPIP):
+
+*******************************************************************************
+ $SEG(EZACICSS)
+
+ Module Name : EZACICSS
+
+ Description : This is a sample server program. It establishes a connection between
+ CICS & TCPIP to process client requests. The server expects the data received
+ from a host / workstation in ASCII. All responses sent by the server to the
+ CLIENT are in ASCII. This server is started using CECI or via the LISTENER.
+ CECI START TRANS(xxxx) from(yyyy)
+ where xxxx is this servers CICS transaction id and yyyy is the port this server will listen on.
+ It processes request received from clients for updates to a hypothetical
+ DB2 database. Any and all references to DB2 or SQL are commented out as this
+ sample is to illustrate CICS Sockets.
+ A client connection is broken when the client transmits and 'END' token to the
+ server. All processing is terminated when an 'TRM' token is received from a
+ client.
+
+ LOGIC : 1. Establish server setup
+ a). TRUE Active
+ b). CAF Active
+ 2. Assign user specified port at start up or use the program declared default.
+ 3. Initialize the Socket.
+ 4. Bind the port.
+ 5. Set Bit Mask to accept incoming read request.
+ 6. Process request from clients.
+ a). Wait for connection
+ b). Process request until 'END' token is receive from client.
+ c). Close connection.
+ note: The current client request ends when the client closes the connection or sends an
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(B) 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(B) DISPLAY.
  05 FILLER PIC X(13) VALUE IS SPACES.

01 CLOSE-ERR.
  05 CLOSE-ERR-M PIC X(30)
     VALUE IS 'CLOSE SOCKET DESCRIPTION FAILED'.
  05 FILLER PIC X(9) VALUE IS ' ERRNO = '.
  05 CLOSE-ERRNO PIC 9(B) 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.
    08 DAINSERT VALUE 'INSERT'.
    08 DADELETE VALUE 'DELETE'.
    08 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 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(16) 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 = '.

580 z/OS V2R1.0 Communications Server: IP CICS Sockets Guide
05 SELECT-ERRNO PIC 9(8) DISPLAY.
05 FILLER PIC X(14)
   VALUE IS SPACES.

01 SQL-ERROR.
  05 FILLER PIC X(35)
     VALUE IS 'SQLERR -PROG TERMINATION,SQLCODE = '.
  05 SQL-ERR-CODE PIC -9(9).
  05 FILLER PIC X(11)
     VALUE IS SPACES.

01 SOCKET-ERR.
  05 SOCKET-ERR-M PIC X(25)
     VALUE IS 'SOCKET CALL FAIL - SOCKET'.
  05 FILLER PIC X(9)
     VALUE IS ' ERRNO = '.
  05 SOCKET-ERRNO PIC 9(8) DISPLAY.
  05 FILLER PIC X(13)
     VALUE IS SPACES.

01 TAKE-ERR.
  05 TAKE-ERR-M PIC X(17)
     VALUE IS 'TAKESOCKET FAILED'.
  05 FILLER PIC X(9)
     VALUE IS ' ERRNO = '.
  05 TAKE-ERRNO PIC 9(8) DISPLAY.
  05 FILLER PIC X(21)
     VALUE IS SPACES.

01 WRITE-ERR.
  05 WRITE-ERR-M PIC X(33)
     VALUE IS 'WRITE SOCKET FAIL'.
  05 FILLER PIC X(9)
     VALUE IS ' ERRNO = '.
  05 WRITE-ERRNO PIC 9(8) DISPLAY.
  05 FILLER PIC X(21)
     VALUE IS SPACES.

*---------------------------------------------------------------*
* PROGRAM'S CONSTANTS                                          *
*---------------------------------------------------------------*
77 CTOB PIC X(4) VALUE 'CTOB'.
77 DEL-ID PIC X(1) VALUE '.
77 BACKLOG PIC 9(8) COMP VALUE 5.
77 NONZERO-FWRD PIC 9(8) VALUE 256.
77 TCP-FLAG PIC 9(8) COMP VALUE 0.
77 SOCK-TYPE PIC 9(8) COMP VALUE 1.
77 AF-INET PIC 9(8) COMP VALUE 2.
77 NUM-FDS PIC 9(8) COMP VALUE 5.
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 INADOR-ANY.
  05 FILLER PIC 9(8) BINARY VALUE 0.

01 SOCKET-FUNCTIONS.
  02 SOKET-ACCEPT PIC X(16) VALUE 'ACCEPT '.
  02 SOKET-BIND PIC X(16) VALUE 'BIND '.
  02 SOKET-CLOSE PIC X(16) VALUE 'CLOSE '.
  02 SOKET-CONNECT PIC X(16) VALUE 'CONNECT '.
  02 SOKET-FCNTL PIC X(16) VALUE 'FCNTL '.
  02 SOKET-GETCLIENTID PIC X(16) VALUE 'GETCLIENTID '.
  02 SOKET-GETHOSTBYADDR PIC X(16) VALUE 'GETHOSTBYADDR '.
  02 SOKET-GETHOSTBYNAME PIC X(16) VALUE 'GETHOSTBYNAME '.
  02 SOKET-GETHOSTID PIC X(16) VALUE 'GETHOSTID '.
  02 SOKET-GETHOSTNAME PIC X(16) VALUE 'GETHOSTNAME '.
  02 SOKET-GETPEERNAME PIC X(16) VALUE 'GETPEERNAME '.
  02 SOKET-GETNAMEINFO PIC X(16) VALUE 'GETNAMEINFO '.
  02 SOKET-GETSOCKNAME PIC X(16) VALUE 'GETSOCKNAME '.
  02 SOKET-GETSOCKOPT PIC X(16) VALUE 'GETSOCKOPT '.
  02 SOKET-GIVESOCKET PIC X(16) VALUE 'GIVESOCKET '.
  02 SOKET-INITAPI PIC X(16) VALUE 'INITAPI '.
  02 SOKET-IOCTL PIC X(16) VALUE 'IOCTL '.
  02 SOKET-LISTEN PIC X(16) VALUE 'LISTEN '.
  02 SOKET-NSTOP PIC X(16) VALUE 'NSTOP '.

Appendix E. Sample programs  581
02 SOKET-READ PIC X(16) VALUE 'READ '.
02 SOKET-RECV PIC X(16) VALUE 'RECV '.
02 SOKET-RECFROM PIC X(16) VALUE 'RECFROM '.
02 SOKET-SELECT PIC X(16) VALUE 'SELECT '.
02 SOKET-SELECTEX PIC X(16) VALUE 'SELECTEX '.
02 SOKET-SEND PIC X(16) VALUE 'SEND '.
02 SOKET-SENDO TO PIC X(16) VALUE 'SENDO TO '.
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-TERM API PIC X(16) VALUE 'TERM API '.
02 SOKET-WRITE PIC X(16) VALUE 'WRITE '.

*---------------------------------------------------------------*
* PROGRAM'S VARIABLES                                          *
*---------------------------------------------------------------*
77 PROTOCOL PIC 9(8) COMP VALUE 0.
77 SRV-SOCKID PIC 9(4) COMP VALUE 0.
77 SRV-SOCKID-FWD PIC 9(8) COMP VALUE 0.
77 CLI-SOCKID PIC 9(4) COMP VALUE 0.
77 CLI-SOCKID-FWD PIC 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 9(8) COMP.
77 WSPTR PIC 9(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 9(8) COMP.
77 TRANS PIC X(4).

01 CLIENTID-LSTN.
   05 CID-DOMAIN-LSTN PIC 9(8) COMP VALUE 2.
   05 CID-LSTN-INFO.
      10 CID-NAME-LSTN PIC X(8).
      10 CID-SUBTNAM-LSTN PIC X(8).
   05 CID-RES-LSTN PIC X(20) VALUE LOW-VALUES.

01 INIT-SUBTASKID.
   05 SUBTASKNO PIC X(7) VALUE LOW-VALUES.
   05 SUBT-CHAR PIC A(1) VALUE 'L'.

01 IDENT.
   05 TCPNAME PIC X(8) VALUE 'TCPCS '.
   05 ADSNAME PIC X(8) VALUE 'EZACIC6S'.

01 MAXSOC PIC 9(4) BINARY VALUE 0.
01 MAXSNO PIC 9(8) BINARY VALUE 0.
01 NFDS PIC 9(8) BINARY.

01 PORT-RECORD.
   05 PORT PIC X(4).
   05 FILLER PIC X(36).

01 SELECT-CSOCKET.
   05 READMASK PIC X(4) VALUE LOW-VALUES.
   05 DUMYMASK PIC X(4) VALUE LOW-VALUES.
   05 REPLY-RDMASK PIC X(4) VALUE LOW-VALUES.
   05 REPLY-RDMASK-FF PIC X(4).

01 SOCKADDR-IN.
   05 SAIN-FAMILY PIC 9(4) BINARY VALUE 0.
      88 SAIN-FAMILY-IS-AFINET VALUE 2.
   05 SAIN-DATA PIC X(14).
   05 SAIN-SIN REDefINES SAIN-DATA.
      10 SAIN-SIN-PORT PIC 9(4) BINARY.
      10 SAIN-SIN-ADDR PIC 9(8) BINARY.
      10 FILLER PIC X(8).
01 SOCKET-CONV.
   05 SOCKET-TBL OCCURS 6 TIMES.
      10 SOCC-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 MSTDIME 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.

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 SQL-STATEMENTS: SQL COMMUNICATION AREA

* ***************************************************** *
* 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-MGRDPT 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.

EXEC CICS PUSH HANDLE END-EXEC.

EXEC CICS HANDLE CONDITION INEXITREQ(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
   END-IF.

Appendix E. Sample programs 585
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-SUCCTO MSG-AREA.

PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.

COMPUTE NFDS = NUM-FDS + 1.

MOVE LOW-VALUES TO READMASK.
MOVE 6 TO TCPLENG.

CALL 'EZACIC06' USING CTOB
    READMASK
    SOCKET-CONV
    TCPLENG
    RETCODE.

IF RETCODE = -1
    THEN
        MOVE BITMASK-ERR TO MSG-AREA
        PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT
    ELSE
        PERFORM ACCEPT-CLIENT-REQ THRU
        ACCEPT-CLIENT-REQ-EXIT
        UNTIL TASK-TERM
END-IF.

PERFORM CLOSE-SOCKET THRU CLOSE-SOCKET-EXIT.

MOVE TCP-SERVER-OFF TO MSG-AREA.

PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.

*---------------------------------------------------------------*
**
*   END OF PROGRAM *
**
*---------------------------------------------------------------*

PGM-EXIT.

EXEC CICS
    RETURN
END-EXEC.

GOBACK.

*---------------------------------------------------------------*
**
*   TRUE IS NOT ENABLED *
**
*---------------------------------------------------------------*

TCP-TRUE-REQ.

MOVE TCP-EXIT-ERR TO MSG-AREA.
PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
GO TO PGM-EXIT.
DB2-CALL ATTACH FACILITY IS NOT ENABLED

DB2-TRUE-REQ.
MOVE DB2-CAF-ERR TO MSG-AREA.
PERFORM HANDLE-TCPICICS THRU HANDLE-TCPICICS-EXIT.
GO TO PGM-EXIT.

LISTENER STARTED TASK

LISTENER-STARTED-TASK.
MOVE CLIENTID-PARM TO CID-LSTN-INFO.
MOVE GIVE-TAKE-SOCKET TO SOCK-TO-RECV-FWD.
CALL 'EZASOKET' USING SOKET-TAKESOCKET
  SOCK-TO-RECV
  CLIENTID-LSTN
  ERRNO
  RETCODE.

IF RETCODE < 0 THEN
  MOVE ERRNO TO TAKE-ERRNO
  MOVE TAKE-ERR TO MSG-AREA
  PERFORM HANDLE-TCPICICS THRU HANDLE-TCPICICS-EXIT
  GO TO PGM-EXIT
ELSE
  MOVE BUFFER-LENG TO TCPLENG
  MOVE START-MSG TO TCP-BUF
  MOVE RETCODE TO SRV-SOCKID
  CALL 'EZACICO4' USING TCP-BUF TCPLENG
  CALL 'EZASOKET' USING SOKET-WRITE
    SRV-SOCKID
    TCPLENG
    TCP-BUF
    ERRNO
    RETCODE

IF RETCODE < 0 THEN
  MOVE ERRNO TO WRITE-ERRNO
  MOVE WRITE-ERR TO MSG-AREA
  PERFORM HANDLE-TCPICICS THRU HANDLE-TCPICICS-EXIT
  GO TO PGM-EXIT
ELSE
  CALL 'EZASOKET' USING SOKET-CLOSE
    SRV-SOCKID
    ERRNO
    RETCODE

IF RETCODE < 0 THEN
  MOVE ERRNO TO CLOSE-ERRNO
  MOVE CLOSE-ERR TO MSG-AREA
  PERFORM HANDLE-TCPICICS THRU HANDLE-TCPICICS-EXIT
  GO TO PGM-EXIT
ELSE
  NEXT SENTENCE
END-IF
END-IF.

MOVE LOW-VALUES TO TCP-BUF.

LISTENER-STARTED-TASK-EXIT.
EXIT.

*---------------------------------------------------------------------*
*    START SERVER PROGRAM                                               *
*---------------------------------------------------------------------*

INIT-SOCKET.

MOVE EIBTASKN TO SUBTASKNO.

CALL 'EZASOKET' USING SOKET-INITAPI
    MAXSOC
    IDENT
    INIT-SUBTASKID
    MAXSNO
    ERRNO
    RETCODE.

IF RETCODE < 0 THEN
    MOVE ERRNO TO INIT-ERRNO
    MOVE INITAPI-ERR TO MSG-AREA
    PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT
    GO TO PGM-EXIT
ELSE
    MOVE INIT-MSG TO MSG-AREA
    PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT
END-IF.

INIT-SOCKET-EXIT.
EXIT.

SOCKET-BIND-LSTN.

MOVE -1 TO SRV-SOCKID-FWD.

*---------------------------------------------------------------------*
*    CREATING A SOCKET TO ALLOCATE                                    *
*    AN OPEN SOCKET FOR INCOMING CONNECTIONS                          *
*---------------------------------------------------------------------*

CALL 'EZASOKET' USING SOKET-SOCKET
    AF-INET
    SOCK-TYPE
    PROTOCOL
    ERRNO
    RETCODE.

IF RETCODE < 0 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 '1' TO SOCKET-CHAR(RETCODE + 1)
END-IF.

*---------------------------------------------------------------------*
*    BIND THE SOCKET TO THE SERVICE PORT                               *
*    TO ESTABLISH A LOCAL ADDRESS FOR PROCESSING INCOMING             *
*    CONNECTIONS.                                                     *
*---------------------------------------------------------------------*

MOVE AF-INET TO SAIN-FAMILY.
MOVE INADDR-ANY TO SAIN-SIN-ADDR.
MOVE PORT TO SAIN-SIN-PORT.

CALL 'EZASOKET' USING SOKET-BIND
CALL 'EZASOKET' USING SOKET-LISTEN
SRV-SOCKID
BACKLOG
ERRNO
RETCODE.

IF RETCODE < 0 THEN
  MOVE ERRNO TO LISTEN-ERRNO
  MOVE LISTEN-ERR TO MSG-AREA
  PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT
  GO TO PGM-EXIT.

CALL 'EZASOKET' USING SOKET-SELECT
NFDS
TIMEVAL
READMASK
DUMYMASK
DUMYMASK
DUMYMASK
REPLY-RDMASK
DUMYMASK
DUMYMASK
ERRNO
RETCODE.

IF RETCODE < 0 THEN
  MOVE ERRNO TO SELECT-ERRNO
  MOVE SELECT-ERR TO MSG-AREA
  PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT
  GO TO PGM-EXIT.

IF RETCODE = 0 THEN GO TO ACCEPT-CLIENT-REQ-EXIT.

CALL 'EZASOKET' USING SOKET-ACCEPT
SRV-SOCKID
SOCKADDR-IN
ERRNO
RETCODE.

IF RETCODE < 0 THEN
  MOVE ERRNO TO ACCEPT-ERRNO
  MOVE ACCEPT-ERR TO MSG-AREA
  PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT
  GO TO PGM-EXIT.

ACCEPT-CLIENT-REQ-EXIT.
EXIT.
RETCODE.

IF RETCODE < 0 THEN
  MOVE ERRNO TO ACCEPT-ERRNO
  MOVE ACCEPT-ERR TO MSG-AREA
  PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT
  GO TO PGM-EXIT.

MOVE RETCODE TO CLI-SOCKID.

PERFORM ACCEPT-RECV THRU ACCEPT-RECV-EXIT
  UNTIL TASK-END OR TASK-TERM.

MOVE DB2END TO MSG-AREA.

PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.

CALL 'EZASOKET' USING SOKET-CLOSE
  CLI-SOCKID
  ERRNO
  RETCODE.

IF RETCODE < 0 THEN
  MOVE ERRNO TO CLOSE-ERRNO
  MOVE CLOSE-ERR TO MSG-AREA
  PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.

IF NOT TASK-TERM
  MOVE '0' TO TASK-FLAG.

ACCEPT-CLIENT-REQ-EXIT.

EXIT.

*--------------------------------------------------------------*
**
* RECEIVING DATA THROUGH A SOCKET BY ISSUING 'RECVFROM' *
* COMMAND. *
**
*--------------------------------------------------------------*

ACCEPT-RECV.

MOVE 'T' TO TCP-INDICATOR.
MOVE BUFFER-LENG TO TCPLENG.
MOVE LOW-VALUES TO TCP-BUF.

CALL 'EZASOKET' USING SOKET-RECVFROM
  CLI-SOCKID
  TCP-FLAG
  TCPLENG
  TCP-BUF
  SOCKADDR-IN
  ERRNO
  RETCODE.

IF RETCODE EQUAL 0 AND TCPLENG EQUAL 0
  THEN NEXT SENTENCE
ELSE
  IF RETCODE = 0
    THEN MOVE ERRNO TO RECVFROM-ERRNO
    MOVE RECVFROM-ERR TO MSG-AREA
    PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT
    MOVE '1' TO TASK-FLAG
  ELSE
    CALL 'EZACIC05' USING TCP-BUF TCPLENG
    IF TCP-BUF-H = LOW-VALUES OR SPACES
      THEN MOVE NULL-DATA TO MSG-AREA
      PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT
    ELSE
      IF TCP-BUF-H = 'END'
        THEN MOVE '1' TO TASK-FLAG
      ELSE IF TCP-BUF-H = 'TRM'
        THEN MOVE '2' TO TASK-FLAG

590  z/OS V2R1.0 Communications Server: IP CICS Sockets Guide
ELSE PERFORM TALK-CLIENT THRU
TALK-CLIENT-EXIT
END-IF
END-IF
END-IF
END-IF
END-IF.

ACCEPT-RECV-EXIT.
EXIT.

**********************************************************
** PROCESSES TALKING TO CLIENT THAT WILL UPDATE DB2 **
** TABLES. **
**********************************************************
** DATA PROCESS: **
** **
** INSERT REC - INS,X81,TEST DEPT,A0213B,Y94 **
** UPDATE REC - UPD,X81,,A1234C, **
** DELETE REC - DEL,X81,, **
** END CLIENT - END,(end client connection ) **
** END SERVER - TRM,(terminate server ) **
** **
**********************************************************

TALK-CLIENT.

UNSTRING TCP-BUF DELIMITED BY DEL-ID OR ALL '*' INTO IN-ACT
   IN-DEPTNO
   IN-DEPTN
   IN-MGRNO
   IN-ADMREDEPT.

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-ADMREDEPT)
      *** 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
 ELSE
  MOVE KEYWORD-ERR TO MSG-AREA
  PERFORM HANDLE-TCPCICS THRU
  HANDLE-TCPCICS-EXIT
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 TO
QUEUE ('CSMT')
FROM (TCPCICS-MSG-AREA)
LENGTH (LENG)
NOHANDLE
END-EXEC

******************************************************************************
** WRITE THE DB2 MESSAGE TO CLIENT.  **
******************************************************************************

MOVE TCPCICS-MSG-2 TO TCP-BUF
CALL 'EZACIC04' USING TCP-BUF TCPLENG
CALL 'EZASOKET' USING SOKET-WRITE
CLI-SOCKID
TCPLENG
TCP-BUF
ERRNO
RETCODE

MOVE LOW-VALUES TO TCP-BUF
TCP-INDICATOR
DB2-ACT

IF RETCODE < 0 THEN

MOVE ERRNO TO WRITE-ERRNO
MOVE WRITE-ERR TO MSG-AREA
PERFORM HANDLE-TCPCICS THRU
HANDLE-TCPCICS-EXIT
MOVE '1' TO TASK-FLAG
END-IF.

END-IF.

TALK-CLIENT-EXIT.
EXIT.

******************************************************************************
** CLOSE ORIGINAL SOCKET DESCRIPTOR **
******************************************************************************

CLOSE-SOCKET.

CALL 'EZASOKET' USING SOKET-CLOSE
SRV-SOCKID
ERRNO
RETCODE.

IF RETCODE < 0 THEN

MOVE ERRNO TO CLOSE-ERRNO
MOVE CLOSE-ERR TO MSG-AREA
PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.

CLOSE-SOCKET-EXIT.
EXIT.

******************************************************************************
** SEND TCP/IP ERROR MESSAGE **
******************************************************************************

HANDLE-TCPCICS.
MOVE LENGTH OF TCPCICS-MSG-AREA TO LENG.

EXEC CICS ASKTIME
  ABSTIME (TSTAMP)
  NOHANDLE
END-EXEC.

EXEC CICS FORMATTIME
  ABSTIME (TSTAMP)
  MMDDYY (MSGDATE)
  TIME (MSGTIME)
  DATESEP ('/')
  TIMESEP (':')
  NOHANDLE
END-EXEC.

EXEC CICS WRITEQ TO
  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 'EZACICO4' USING TCP-BUF TCPLENG
  MOVE ' ' TO TCP-INDICATOR
  CALL 'EZASOKET' USING SOKET-WRITE
    CLI-SOCKID
    TCPLENG
    TCP-BUF
    ERRNO
    RETCODE
  IF RETCODE < 0
    THEN
      MOVE ERRNO TO WRITE-ERRNO
      MOVE WRITE-ERR TO MSG-AREA
  EXEC CICS WRITEQ TO
    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 TO
  QUEUE ('CSMT')
  FROM (TCPCICS-MSG-AREA)
  RESP (RESPONSE)
  LENGTH (LENG)
END-EXEC.

MOVE LOW-VALUES TO TCP-BUF.
MOVE TCPCICS-MSG-2 TO TCP-BUF.

CALL 'EZACIC04' USING TCP-BUF TCPLENG.

CALL 'EZASOKET' USING SOKET-WRITE
  CLI-SOCKID
  TCPLENG
  TCP-BUF
  ERRNO
  RETCODE.

IF RETCODE < 0 THEN
  MOVE ERRNO TO WRITE-ERRNO
  MOVE WRITE-ERR TO MSG-AREA
  PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
GO TO PGM-EXIT.

SQL-ERROR-ROU-EXIT.
EXIT.

**---------------------------------------------------------------**
*  
*  OTHER ERRORS (HANDLE CONDITION)  
*  
*  **---------------------------------------------------------------**

INVREQ-ERR-SEC.
MOVE TCP-EXIT-ERR TO MSG-AREA.
PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
GO TO PGM-EXIT.

IDERR-SEC.
MOVE IDERR-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.

594  z/OS V2R1.0 Communications Server: IP CICS Sockets Guide
The following COBOL socket program is in the SEZAINST data set.

Figure 180. EZACIC6C IPv6 child server sample

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EZACIC6C.
ENVIRONMENT DIVISION.
DATA DIVISION.
* WORKING-STORAGE SECTION.
  77 TASK-START PIC X(40)
      VALUE IS 'TASK STARTING THRU CICS/TCPIP INTERFACE '.
  77 GNI-ERR PIC X(24)
      VALUE IS ' GETNAMEINFO FAIL '.
  77 GNI-SUCCESS PIC X(24)
      VALUE IS ' GETNAMEINFO SUCCESSFUL '.
  77 GPN-ERR PIC X(24)
      VALUE IS ' GETPEERNAME FAIL '.
  77 GPN-SUCCESS PIC X(24)
      VALUE IS ' GETPEERNAME SUCCESSFUL '.
  77 TAKE-ERR PIC X(24)
      VALUE IS ' TAKESOCKET FAIL '.
  77 TAKE-SUCCESS PIC X(24)
      VALUE IS ' TAKESOCKET SUCCESSFUL '.
  77 READ-ERR PIC X(24)
      VALUE IS ' READ SOCKET FAIL '.
  77 READ-SUCCESS PIC X(24)
      VALUE IS ' READ SOCKET SUCCESSFUL '.
  77 WRITE-ERR PIC X(24)
      VALUE IS ' WRITE SOCKET FAIL '.
  77 WRITE-END-ERR PIC X(32)
      VALUE IS ' WRITE SOCKET FAIL - PGM END MSG '.
  77 WRITE-SUCCESS PIC X(25)
      VALUE IS ' WRITE SOCKET SUCCESSFUL '.
  77 CLOS-ERR PIC X(24)
      VALUE IS ' CLOSE SOCKET FAIL '.
```
77 CLOS-SUCCESS PIC X(24) VALUE IS 'CLOSE SOCKET SUCCESSFUL'.
77 INVREQ-ERR PIC X(24) VALUE IS 'INTERFACE IS NOT ACTIVE'.
77 IOERR-ERR PIC X(24) VALUE IS 'IOERR OCCURRS'.
77 LENGERR-ERR PIC X(24) VALUE IS 'LENGERR ERROR'.
77 ITEMERR-ERR PIC X(24) VALUE IS 'ITEMERR ERROR'.
77 NOSPACE-ERR PIC X(24) VALUE IS 'NOSPACE CONDITION'.
77 QIDERR-ERR PIC X(24) VALUE IS 'QIDERR CONDITION'.
77 ENDDATA-ERR PIC X(30) VALUE IS 'RETRIEVE DATA CAN NOT BE FOUND'.
77 WRKEND PIC X(20) VALUE 'CONNECTION END'.
77 WRITE-SW PIC X(1) VALUE 'N'.
77 FORCE-ERROR-MSG PIC X(1) VALUE 'N'.

01 SOKET-FUNCTIONS.
 02 SOKET-ACCEPT PIC X(16) VALUE 'ACCEPT'.
 02 SOKET-BIND PIC X(16) VALUE 'BIND'.
 02 SOKET-CLOSE PIC X(16) VALUE 'CLOSE'.
 02 SOKET-CONNECT PIC X(16) VALUE 'CONNECT'.
 02 SOKET-FCNTL PIC X(16) VALUE 'FCNTL'.
 02 SOKET-GETCLIENTID PIC X(16) VALUE 'GETCLIENTID'.
 02 SOKET-GETHOSTBYADDR PIC X(16) VALUE 'GETHOSTBYADDR'.
 02 SOKET-GETHOSTBYNAME PIC X(16) VALUE 'GETHOSTBYNAME'.
 02 SOKET-GETHOSTID PIC X(16) VALUE 'GETHOSTID'.
 02 SOKET-GETHOSTNAME PIC X(16) VALUE 'GETHOSTNAME'.
 02 SOKET-GETPEERNAME PIC X(16) VALUE 'GETPEERNAME'.
 02 SOKET-GETNAMEINFO PIC X(16) VALUE 'GETNAMEINFO'.
 02 SOKET-GETSOCKNAME PIC X(16) VALUE 'GETSOCKNAME'.
 02 SOKET-GETSOCKOPT PIC X(16) VALUE 'GETSOCKOPT'.
 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-IOTCL PIC X(16) VALUE 'IOTCL'.
 02 SOKET-LISTEN PIC X(16) VALUE 'LISTEN'.
 02 SOKET-NTOP PIC X(16) VALUE 'NTOP'.
 02 SOKET-READ PIC X(16) VALUE 'READ'.
 02 SOKET-RECV PIC X(16) VALUE 'RECV'.
 02 SOKET-RECVFROM PIC X(16) VALUE 'RECVFROM'.
 02 SOKET-SELECT PIC X(16) VALUE 'SELECT'.
 02 SOKET-SEND PIC X(16) VALUE 'SEND'.
 02 SOKET-SENDTO PIC X(16) VALUE 'SENDTO'.
 02 SOKET-SETSOCKOPT PIC X(16) VALUE 'SETSOCKOPT'.
 02 SOKET-IOCTLSOCKET PIC X(16) VALUE 'IOCTLSOCKET'.
 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 WRKRM PIC X(14) VALUE IS 'DATA RECEIVED'.

*---------------------------------------------------------------*
* program's variables                                         *
*---------------------------------------------------------------*

77 SUBTRACE PIC X(8) VALUE 'CONTRACE'.
77 RESPONSE PIC 9(9) COMP.
77 TASK-FLAG PIC X(1) VALUE '0'.
77 TAKE-SOCKET PIC 9(8) COMP.
77 DTAZ-LENGTH PIC 9(04).
77 NTOP-FAMILY PIC 9(8) COMP.
77 NTOP-LENGTH PIC 9(4) COMP.
77 SOCKID PIC 9(4) COMP.
77 SOCKID-FWD PIC 9(8) COMP.
77 ERRNO PIC 9(8) COMP.
77 RETCODE PIC 9(8) COMP.
01 TCP-BUF.
 05 TCP-BUF-H PIC X(3) VALUE IS SPACES.
 05 TCP-BUF-DATA PIC X(197) VALUE IS SPACES.
 07 TCPLEN PIC 9(8) COMP.
 07 RECVPIC X(9) COMP.
 07 REC-FLAG PIC 9(8) COMP.
 07 CLG ENDPCV 9(4) COMP.
Appendix E. Sample programs 597
10 PEER-SING-PORT PIC 9(4) BINARY.
10 PEER-SING-FLOWINFO PIC 9(8) BINARY.
10 PEER-SING-ADDR.
  15 FILLER PIC 9(16) BINARY.
  15 FILLER PIC 9(16) BINARY.
10 PEER-SING-SCOPEID PIC 9(8) BINARY.

* TRANSACTION INPUT MESSAGE FROM THE LISTENER
*
01 TCPSOCKET-PARM.
  05 GIVE-TAKE-SOCKET PIC 9(8) COMP.
  05 LSTN-NAME PIC X(8).
  05 LSTN-SUBTASKNAME PIC X(8).
  05 CLIENT-IN-DATA PIC X(35).
  05 THREADSAFE-INDICATOR PIC X(1).
  88 INTERFACE-IS-THREADSAFE VALUE '1'.
  05 SOCKADDR-IN.
   10 SOCK-FAMILY PIC 9(4) BINARY.
     88 SOCK-FAMILY-IS-AFINET VALUE 2.
     88 SOCK-FAMILY-IS-AFINET6 VALUE 19.
   10 SOCK-DATA PIC X(26).
   10 SOCK-SIN REDEFINES SOCK-DATA.
     15 SOCK-SIN-PORT PIC 9(4) BINARY.
     15 SOCK-SIN-ADDR PIC 9(8) BINARY.
     15 FILLER PIC X(8).
     15 FILLER PIC X(12).
   10 SOCK-SIN6 REDEFINES SOCK-DATA.
     15 SOCK-SIN6-PORT PIC 9(4) BINARY.
     15 SOCK-SIN6-FLOWINFO PIC 9(8) BINARY.
     15 FILLER PIC 9(16) BINARY.
     15 FILLER PIC 9(16) BINARY.
     15 FILLER PIC 9(16) BINARY.
   15 SOCK-SIN6-SCOPEID PIC 9(8) BINARY.
   05 FILLER PIC X(68).
   05 CLIENT-IN-DATA-LENGTH PIC 9(4) COMP.
   05 CLIENT-IN-DATA-2 PIC X(999).

PROCEDURE DIVISION.
MOVE 'Y' TO WRITE-SW.
EXEC CICS HANDLE CONDITION INVREQ (INVREQ-ERR-SEC)
   IOERR (IOERR-SEC)
   ENDDATA (ENDDATA-SEC)
   NOSPACE (NOSPACE-ERR-SEC)
   QIDERR (QIDERR-SEC)
   ITEMERR (ITEMERR-SEC)
END-EXEC.
EXEC CICS IGNORE CONDITION LENGERR
END-EXEC.
PERFORM INITIAL-SEC THRU INITIAL-SEC-EXIT.
PERFORM TAKE_SOCKET-SEC THRU TAKE_SOCKET-SEC-EXIT.
PERFORM GET-PEER-NAME THRU GET-PEER-NAME-EXIT.
PERFORM GET-NAME-INFO THRU GET-NAME-INFO-EXIT.
MOVE '0' TO TASK-FLAG.
PERFORM CLIENT-TASK THRU CLIENT-TASK-EXIT
   VARYING CNT FROM 1 BY 1 UNTIL TASK-FLAG = '1'.
CLOSE-SOCK.

CALL 'EZASOKET' USING SOKET-CLOSE SOCKID
   ERNO RETCODE.
IF RETCODE < 0 THEN
   MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
   MOVE CLOS-ERR TO ERR-MSG
   MOVE SOCKID TO ERR-SOCKET
   MOVE RETCODE TO ERR-RETCODE
   MOVE ERRNO TO ERR-ERNO
MOVE CICS-ERR-AREA TO CICS-MSG-AREA
ELSE
   MOVE CLOS-SUCCESS TO CICS-MSG-AREA.
   PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
PGM-EXIT.

IF RETCODE < 0 THEN
   EXEC CICS ABEND ABCODE('SRV6') END-EXEC.

MOVE SPACES TO CICS-MSG-AREA.
MOVE 'END OF EZACIC6C PROGRAM' TO CICS-MSG-AREA.
EXEC CICS RETURN END-EXEC.
GOBACK.

*-------------------------------------------------------------------*
* RECEIVE PASSED PARAMETER WHICH ARE CID                         *
*-------------------------------------------------------------------*
INITIAL-SEC.

MOVE SPACES TO CICS-MSG-AREA.
MOVE 50 TO MSGLENG.
MOVE 'SRV6 TRANSACTION START UP ' TO CICS-MSG-AREA.
PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.

* PREPARE TO RECEIVE AND ENHANCED TIM
* MOVE 1153 TO CLENG.
INITIALIZE TCPSOCKET-PARM.

EXEC CICS RETRIEVE INTO(TCPSOCKET-PARM)
   LENGTH(CLENG)
END-EXEC.

MOVE 'LISTENER ADDR SPACE ' TO DETAIL-FIELD.
MOVE SPACES TO DETAIL-DATA.
MOVE LSTN-NAME TO DETAIL-DATA.
MOVE CICS-DETAIL-AREA TO CICS-MSG-AREA.
PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.

MOVE 'LISTENER TASK ID ' TO DETAIL-FIELD.
MOVE SPACES TO DETAIL-DATA.
MOVE LSTN-SUBTASKNAME TO DETAIL-DATA.
MOVE CICS-DETAIL-AREA TO CICS-MSG-AREA.
PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.

IF CLIENT-IN-DATA-LENGTH <= 0
   MOVE 'TIM IS STANDARD' TO CICS-MSG-AREA
   PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
   MOVE 'CLIENT IN DATA ' TO DETAIL-FIELD
   MOVE SPACES TO DETAIL-DATA
   MOVE CLIENT-IN-DATA TO DETAIL-DATA
   MOVE CICS-DETAIL-AREA TO CICS-MSG-AREA
   PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
ELSE
   MOVE 'TIM IS ENHANCED' TO CICS-MSG-AREA
   PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
   MOVE 'CLIENT IN DATA ' TO DETAIL-FIELD
   MOVE SPACES TO DETAIL-DATA
   MOVE CLIENT-IN-DATA TO DETAIL-DATA
   MOVE CICS-DETAIL-AREA TO CICS-MSG-AREA
   PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
   MOVE 'CLIENT IN DATA 2 LEN' TO DETAIL-FIELD
   MOVE SPACES TO DETAIL-DATA
   MOVE CLIENT-IN-DATA-LENGTH TO DATA2-LENGTH
   MOVE DATA2-LENGTH TO DETAIL-DATA
   MOVE CICS-DETAIL-AREA TO CICS-MSG-AREA
   PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
   MOVE 'CLIENT IN DATA 2 ' TO DETAIL-FIELD
MOVE SPACES TO DETAIL-DATA
MOVE CLIENT-IN-DATA-2 TO CICS-DATA2-AREA
MOVE DATA-2-FOR-MSG TO DETAIL-DATA
MOVE CICS-DETAIL-AREA TO CICS-MSG-AREA
PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.

INITIAL-SEC-EXIT.
EXIT.

*---------------------------------------------------------------*
**
* Perform TCP SOCKET functions by passing socket command to *
* EZASOKET routine. SOCKET command are translated to pre- *
* define integer. *
**
*---------------------------------------------------------------*

TAKESOCKET-SEC.

*---------------------------------------------------------------*
**
* Issue 'TAKESOCKET' call to acquire a socket which was *
* given by the LISTENER program. *
**
*---------------------------------------------------------------*

* MOVE AF-INET TO CID-DOMAIN-LSTN CID-DOMAIN-APPL.
  MOVE SOCK-FAMILY TO CID-DOMAIN-LSTN CID-DOMAIN-APPL.
  MOVE LSTN-NAME TO CID-NAME-LSTN.
  MOVE LSTN-SUBTASKNAME TO CID-SUBTASKNAME-LSTN.
  MOVE GIVE-TAKE-SOCKET TO TAKE-SOCKET SOCKID SOCKID-FWD.
  CALL 'EZASOKET' USING SOCKET-TAKESOCKET SOCKID
  CLIENTID-LSTN ERRNO RETCODE.

IF RETCODE < 0 THEN
  MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
  MOVE TAKE-ERR TO ERR-MSG
  MOVE RETCODE TO ERR-RETCODE
  MOVE ERRNO TO ERR-ERRNO
  MOVE CICS-ERR-AREA TO CICS-MSG-AREA
  PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
  GO TO PGM-EXIT
ELSE
  MOVE SPACES TO CICS-MSG-AREA
  MOVE TAKE-SUCCESS TO CICS-MSG-AREA
  PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
  MOVE SPACES TO CICS-MSG-AREA.
  IF SOCK-FAMILY-IS-AFINET
    MOVE 'TOOK AN AF_INET SOCKET' TO CICS-MSG-AREA
    PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
    MOVE SPACES TO DETAIL-DATA
    MOVE 'AF_INET ADDRESS IS ' TO DETAIL-FIELD
    MOVE SOCK-FAMILY TO NTOP-FAMILY
    MOVE 16 TO NTOP-LENGTH
    CALL 'EZASOKET' USING SOCKET-NTOP
      NTOP-FAMILY
      SOCK-SIN-ADDR
      DETAIL-DATA
      NTOP-LENGTH
      ERRNO
      RETCODE
  ELSE
    MOVE 'TOOK AN AF_INET6 SOCKET' TO CICS-MSG-AREA
    PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
    MOVE 'AF_INET6 ADDRESS IS ' TO DETAIL-FIELD
    MOVE SOCK-FAMILY TO NTOP-FAMILY
    MOVE 45 TO NTOP-LENGTH
    CALL 'EZASOKET' USING SOCKET-NTOP
      NTOP-FAMILY
      SOCK-SIN6-ADDR
      DETAIL-DATA
      NTOP-LENGTH
      ERRNO
      RETCODE.
  END-IF
END-IF
MOVE CICS-DETAIL-AREA TO CICS-MSG-AREA.
PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.

MOVE RETCODE TO SOCKID.
MOVE SPACES TO TCP-BUF.
MOVE TASK-START TO TCP-BUF.
MOVE 50 TO TCPLENG.

* REMOVE FOLLOWING STATEMENT IF USING EBCDIC CLIENT *
* CALL 'EZACICO4' USING TCP-BUF TCPLENG.

CALL 'EZASOKET' USING SOCKET-WRITE SOCKID TCPLENG
TCP-BUF ERRNO RETCODE.

IF RETCODE < 0 THEN
  MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
  MOVE WRITE-ERR TO ERR-MSG
  MOVE SOCKID TO ERR-SOCKET
  MOVE RETCODE TO ERR-RETCODE
  MOVE ERRNO TO ERR-ERRNO
  MOVE CICS-ERR-AREA TO CICS-MSG-AREA
  PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
  GO TO PGM-EXIT
ELSE
  MOVE WRITE-SUCCESS TO CICS-MSG-AREA
  PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
  TAKESOCKET-SEC-EXIT.
  EXIT.

GET-PEER-NAME.
  CALL 'EZASOKET' USING SOCKET-GETPEERNAME
  SOCKID PEER-NAME ERRNO RETCODE.
  IF RETCODE < 0 THEN
    MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
    MOVE GPN-ERR TO ERR-MSG
    MOVE SOCKID TO ERR-SOCKET
    MOVE RETCODE TO ERR-RETCODE
    MOVE ERRNO TO ERR-ERRNO
    MOVE CICS-ERR-AREA TO CICS-MSG-AREA
    PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
    GO TO PGM-EXIT
  ELSE
    MOVE GPN-SUCCESS TO CICS-MSG-AREA
    PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
    GET-PEER-NAME-EXIT.
    EXIT.

GET-NAME-INFO.
  IF PEER-FAMILY-IS-AFINET
    MOVE 16 TO NAME-LEN
  ELSE
    MOVE 28 TO NAME-LEN.
  MOVE SPACES TO HOST-NAME.
  MOVE 256 TO HOST-NAME-LEN.
  MOVE SPACES TO SERVICE-NAME.
  MOVE 32 TO SERVICE-NAME-LEN.
  CALL 'EZASOKET' USING SOCKET-GETNAMEINFO
  PEER-NAME NAME-LEN
  HOST-NAME HOST-NAME-LEN
  SERVICE-NAME SERVICE-NAME-LEN
  NAME-INFO-FLAGS
  ERRNO RETCODE.
  IF RETCODE < 0 THEN
    MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
    MOVE GNI-ERR TO ERR-MSG
    MOVE SOCKID TO ERR-SOCKET
    MOVE RETCODE TO ERR-RETCODE
    MOVE ERRNO TO ERR-ERRNO
    MOVE CICS-ERR-AREA TO CICS-MSG-AREA
    PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
    GO TO PGM-EXIT
  ELSE
    MOVE GNI-SUCCESS TO CICS-MSG-AREA
    PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
    GET-NAME-INFO-EXIT.
    EXIT.

CLIENT-TASK.
MOVE LOW-VALUES TO TCP-BUF.
MOVE 200 TO TCPLENG.
MOVE ZEROS TO RECV-FLAG.

CALL 'EZASOKET' USING SOKET-RECV SOCKID
      RECV-FLAG TCPLENG TCP-BUF ERRNO RETCODE.

IF RETCODE < 0 THEN
   MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
   MOVE READ-ERR TO ERR-MSG
   MOVE SOCKID TO ERR-SOCKET
   MOVE RETCODE TO ERR-RETCODE
   MOVE ERRNO TO ERR-ERRNO
   MOVE CICS-ERR-AREA TO CICS-MSG-AREA
   PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
   GO TO PGM-EXIT
ELSE:
   MOVE READ-SUCCESS TO CICS-MSG-AREA
   PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.

* REMOVE FOLLOWING STATEMENT IF USING EBCDIC CLIENT
* CALL 'EZACIC05' USING TCP-BUF TCPLENG.

* DETERMINE WHETHER THE CLIENT IS FINISHED SENDING DATA
* IF TCP-BUF-H = 'END' OR TCP-BUF-H = 'end' THEN
   MOVE '1' TO TASK-FLAG
   PERFORM CLIENT-TALK-END THRU CLIENT-TALK-END-EXIT
   GO TO CLIENT-TASK-EXIT.

IF RETCODE = 0 THEN
   MOVE '1' TO TASK-FLAG
   GO TO CLIENT-TASK-EXIT.

** ECHO RECEIVING DATA
**

MOVE TCP-BUF TO CICS-MSG-AREA.
PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.

MOVE RETCODE TO TCPLENG.

* REMOVE FOLLOWING STATEMENT IF USING EBCDIC CLIENT
* CALL 'EZACIC04' USING TCP-BUF TCPLENG.
CALL 'EZASOKET' USING SOKET-WRITE SOCKID TCPLENG
      TCP-BUF ERRNO RETCODE.

IF RETCODE < 0 THEN
   MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
   MOVE WRITE-ERR TO ERR-MSG
   MOVE SOCKID TO ERR-SOCKET
   MOVE RETCODE TO ERR-RETCODE
   MOVE ERRNO TO ERR-ERRNO
   MOVE CICS-ERR-AREA TO CICS-MSG-AREA
   PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
   GO TO PGM-EXIT
ELSE:
   MOVE WRITE-SUCCESS TO CICS-MSG-AREA
   PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.

CLIENT-TASK-EXIT.

EXIT.

WRITE-CICS.
MOVE 78 TO CLENG.
MOVE EIBTASKN TO TASK-NUMBER.
IF WRITE-SW = 'Y' THEN
IF INTERFACE-IS-THREADSAFE THEN
  IF FORCE-ERROR-MSG = 'Y' THEN
    EXEC CICS WRITEQ TO QUEUE('CSMT') FROM(TD-MSG)
    LENGTH(CLENG) NOHANDLE
    END-EXEC
  ELSE
    NEXT SENTENCE
  ELSE
    EXEC CICS WRITEQ TO QUEUE('CSMT') FROM(TD-MSG)
    LENGTH(CLENG) NOHANDLE
    END-EXEC
  ELSE
    NEXT SENTENCE.
MOVE SPACES TO CICS-MSG-AREA.
WRITE-CICS-EXIT.
EXIT.

CLIENT-TALK-END.
  MOVE LOW-VALUES TO TCP-BUF.
  MOVE WRKEND TO TCP-BUF CICS-MSG-AREA.
  MOVE 50 TO TCPLENG.
  * REMOVE FOLLOWING STATEMENT IF USING EBCDIC CLIENT
  *
  CALL 'EZACIC04' USING TCP-BUF TCPLENG.
  CALL 'EZASOKET' USING SOKET-WRITE SOCKID TCPLENG
  TCP-BUF ERRNO RETCODE.
  IF RETCODE < 0 THEN
    MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
    MOVE WRITE-END-ERR TO ERR-MSG
    MOVE SOCKID TO ERR-SOCKET
    MOVE ERRNO TO ERR-RETCODE
    MOVE CICS-ERR-AREA TO CICS-MSG-AREA
    PERFORM WRITE-CICS THRU WRITE-CICS-EXIT
    GO TO PGM-EXIT.
  END-EXEC.

CLIENT-TALK-END-EXIT.
EXIT.

INVREQ-ERR-SEC.
  MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
  MOVE INVREQ-ERR TO CICS-MSG-AREA.
  PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
  GO TO PGM-EXIT.

IOERR-SEC.
  MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
  MOVE IOERR-ERR TO CICS-MSG-AREA.
  PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
  GO TO PGM-EXIT.

LENGERR-SEC.
  MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
  MOVE LENGERR-ERR TO CICS-MSG-AREA.
  PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
  GO TO PGM-EXIT.

NOSPACE-ERR-SEC.
  MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
  MOVE NOSPACE-ERR TO CICS-MSG-AREA.
  PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
  GO TO PGM-EXIT.

QIDERR-SEC.
  MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
  MOVE QIDERR-ERR TO CICS-MSG-AREA.
  PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
  GO TO PGM-EXIT.

ITEMERR-SEC.
  MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
  MOVE ITEMERR-ERR TO CICS-MSG-AREA.
  PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
  GO TO PGM-EXIT.

ENDDATA-SEC.
  MOVE 'Y' TO WRITE-SW FORCE-ERROR-MSG
  MOVE ENDDATA-ERR TO CICS-MSG-AREA.
  PERFORM WRITE-CICS THRU WRITE-CICS-EXIT.
  GO TO PGM-EXIT.
EZACIC6S

The following COBOL socket program is in the SEZAINST data set.

Figure 181. EZACIC6S IPv6 iterative server sample

***********************************************************************
*  Communications Server for z/OSVersion 1, Release 9
*  Copyright: Licensed Materials - Property of IBM
*  "Restricted Materials of IBM"
*  5694-A01
*  Copyright IBM Corp. 2003, 2007
*  US Government Users Restricted Rights -
*  Use, duplication or disclosure restricted by
*  GSA ADP Schedule Contract with IBM Corp.
*  Status: CSV1R9
*  $MOD(EZACIC6S),COMP(CICS),PROD(TCPIP):
***********************************************************************

$SEG(EZACIC6S)

* Module Name : EZACIC6S
* Description : This is a sample server program. It establishes a connection between
* CICS & TCPIP to process client requests. The server expects the data received
* from a host / workstation in ASCII. All responses sent by the server to the
* CLIENT are in ASCII. This server is started using CECI or via the LISTENER.
* CECI START TRANS(xxxx) from(yyyy)
* where xxxx is this servers CICS transaction id and yyy is the
* port this server will listen on.
* It processes request received from clients for updates to a hypothetical
* DB2 database. Any and all references to DB2 or SQL are commented out as this
* sample is to illustrate CICS Sockets.
* A client connection is broken when the client transmits and 'END' token to the
* server. All processing is terminated when an 'TRM' token is received from a
* client.

* LOGIC : 1. Establish server setup
* a). TRUE Active
* b). CAF Active
* 2. Assign user specified port at start up or use the program
* declared default.
* 3. Initialize the AF_INET6 socket.
* 4. Bind the port and 'in6addr_any.'
* 5. Set Bit Mask to accept incoming
* read request.
* 6. Process request from clients.
IDENTIFICATION DIVISION.
PROGRAM-ID. EZACIC6S.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

* messages *

77 BITMASK-ERR PIC X(30)
   VALUE IS 'BITMASK CONVERSION - FAILED '.
77 ENDDATA-ERR PIC X(30)
   VALUE IS 'RETRIEVE DATA CAN NOT BE FOUND'.
77 INIT-MSG PIC X(30)
   VALUE IS 'INITAPI COMPLETE '.
77 IOERR-ERR PIC X(30)
   VALUE IS 'IOERR OCCURRS '.
77 ITEMERR-ERR PIC X(30)
   VALUE IS 'ITEMERR ERROR '.
77 KEYWORD-ERR PIC X(30)
   VALUE IS 'INPUT KEYWORD ERROR '.
77 LENGERR-ERR PIC X(30)
   VALUE IS 'LENGERR ERROR '.
77 NOSPACE-ERR PIC X(30)
   VALUE IS 'NOSPACE CONDITION '.
77 NULL-DATA PIC X(30)
   VALUE IS 'READ NULL DATA '.
77 QIDERR-ERR PIC X(30)
   VALUE IS 'TRANSIENT DATA QUEUE NOT FOUND'.
77 START-MSG PIC X(30)
   VALUE IS 'SERVER PROGRAM IS STARTING '.
77 TCP-EXIT-ERR PIC X(30)
   VALUE IS 'SERVER STOPPED:TRUE NOT ACTIVE'.
77 TCP-SERVER-OFF PIC X(30)
   VALUE IS 'SERVER IS ENDING '.
77 TS-INVREQ-ERR PIC X(30)
   VALUE IS 'WRITE TS FAILED - INVREQ '.
77 TS-NOTAUTH-ERR PIC X(30)
   VALUE IS 'WRITE TS FAILED - NOTAUTH '.
77 TS-IOERR-ERR PIC X(30)
   VALUE IS 'WRITE TS FAILED - IOERR '.
77 WRITETS-ERR PIC X(30)
   VALUE IS 'WRITE TS FAILED '.

01 ACCEPT-ERR.
   05 ACCEPT-ERR-M PIC X(25)
      VALUE IS 'SOCKET CALL FAIL - ACCEPT'.
   05 FILLER PIC X(9)
      VALUE IS ' ERRNO = '.
   05 ACCEPT-ERRNO PIC 9(8) DISPLAY.
   05 FILLER PIC X(13)
      VALUE IS SPACES.

01 NTOP-ERR.
   05 NTOP-ERR-M PIC X(23)
      VALUE IS 'SOCKET CALL FAIL - NTOP'.
   05 FILLER PIC X(9)
      VALUE IS ' ERRNO = '.
   05 NTOP-ERRNO PIC 9(8) DISPLAY.
   05 FILLER PIC X(13)
      VALUE IS SPACES.

Appendix E. Sample programs 605
01 NTOP-OK.
  05 NTOP-OK-M PIC X(21) VALUE 'ACCEPTED IP ADDRESS: '.
  05 NTOP-PRESENTABLE-ADDR PIC X(45) DISPLAY VALUE IS SPACES.

01 GNI-ERR.
  05 GNI-ERR-M PIC X(30) VALUE 'SOCKET CALL FAIL - GETNAMEINFO'.
  05 FILLER PIC X(9) VALUE ' ERRNO = '.
  05 GNI-ERRNO PIC 9(8) DISPLAY.
  05 FILLER PIC X(13) VALUE IS SPACES.

01 GNI-HOST-NAME-OK.
  05 FILLER PIC X(19) VALUE 'CLIENTS HOST NAME: '.
  05 GNI-HOST-NAME PIC X(255) DISPLAY VALUE IS SPACES.

01 GNI-SERVICE-NAME-OK.
  05 FILLER PIC X(22) VALUE 'CLIENTS SERVICE NAME: '.
  05 GNI-SERVICE-NAME PIC X(32) DISPLAY VALUE IS SPACES.

01 GPN-ERR.
  05 GPN-ERR-M PIC X(30) VALUE 'SOCKET CALL FAIL - GETPEERNAME'.
  05 FILLER PIC X(9) VALUE ' ERRNO = '.
  05 GPN-ERRNO PIC 9(8) DISPLAY.
  05 FILLER PIC X(13) VALUE IS SPACES.

01 BIND-ERR.
  05 BIND-ERR-M PIC X(25) VALUE 'SOCKET CALL FAIL - BIND'.
  05 FILLER PIC X(9) VALUE ' 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 'CLOSE SOCKET DESCRIPTOR FAILED'.
  05 FILLER PIC X(9) VALUE ' ERRNO = '.
  05 CLOSE-ERRNO PIC 9(8) DISPLAY.
  05 FILLER PIC X(8) VALUE IS SPACES.

01 DB2END.
  05 FILLER PIC X(16) VALUE 'DB2 PROCESS ENDS'.
  05 FILLER PIC X(39) VALUE IS SPACES.

01 DB2-CAF-ERR.
  05 FILLER PIC X(24) VALUE 'CONNECT NOT ESTABLISHED '
  05 FILLER PIC X(30) VALUE 'ATTACHMENT FACILITY NOT ACTIVE'.
  05 FILLER PIC X(1) VALUE IS SPACES.

01 DB2MSG.
  05 DB2-ACT PIC X(6) VALUE SPACES.
  08 DAINSERT VALUE 'INSERT'.
  08 DADELETE VALUE 'DELETE'.
  08 DAUPDATE VALUE 'UPDATE'.
  05 DB2M PIC X(18) VALUE ' COMPLETE - #ROWS '.
  05 DB2M-VAR PIC X(10).
  05 FILLER PIC X(2) VALUE SPACES.
  05 DB2CODE PIC -(9)9.
Appendix E. Sample programs

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 'WRITE SOCKET FAIL'.
   05 FILLER PIC X(9) VALUE ' ERRNO = '.
   05 WRITE-ERRNO PIC 9(8) DISPLAY.
   05 FILLER PIC X(21) VALUE IS SPACES.

---------------------------------------------------------------------
* PROGRAM'S CONSTANTS *
---------------------------------------------------------------------
77 CTOB PIC X(4) VALUE 'CTOB'.
77 DEL-ID PIC X(1) VALUE ','.
77 BACKLOG PIC 9(8) COMP VALUE 5.
77 NONZERO-FWD PIC 9(8) VALUE 256.
77 TCP-FLAG PIC 9(8) COMP VALUE 0.
77 SOCK-TYPE PIC 9(8) COMP VALUE 1.
77 AF-INET6 PIC 9(8) COMP VALUE 19.
77 NUM-FDS PIC 9(8) COMP VALUE 5.
77 LOM PIC 9(4) COMP VALUE 4.
77 CECI-LENG PIC 9(8) COMP VALUE 5.
77 BUFFER-LENG PIC 9(8) COMP VALUE 55.
77 GWLENG PIC 9(4) COMP VALUE 256.
77 DEFAULT-PORT PIC X(4) VALUE '????'.
88 DEFAULT-SPECIFIED VALUE '1950'.

---------------------------------------------------------------------
* 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 LENG PIC 9(4) COMP.
77 WSLENG PIC 9(4) COMP.
77 RESPONSE PIC 9(9) COMP.

---------------------------------------------------------------------
* PROGRAM'S FUNCTIONS *
---------------------------------------------------------------------
01 SOKET-FUNCTIONS.
   02 SOKET-ACCEPT PIC X(16) VALUE 'ACCEPT '.
   02 SOKET-BIND PIC X(16) VALUE 'BIND '.
   02 SOKET-CLOSE PIC X(16) VALUE 'CLOSE '.
   02 SOKET-CONNECT PIC X(16) VALUE 'CONNECT '.
   02 SOKET-FCNTL PIC X(16) VALUE 'FCNTL '.
   02 SOKET-GETCLIENTID PIC X(16) VALUE 'GETCLIENTID '.
   02 SOKET-GETHOSTBYADDR PIC X(16) VALUE 'GETHOSTBYADDR '.
   02 SOKET-GETHOSTBYNAME PIC X(16) VALUE 'GETHOSTBYNAME '.
   02 SOKET-GETHOSTID PIC X(16) VALUE 'GETHOSTID '.
   02 SOKET-GETHOSTNAME PIC X(16) VALUE 'GETHOSTNAME '.
   02 SOKET-GETPEERNAME PIC X(16) VALUE 'GETPEERNAME '.
   02 SOKET-GETNAMEINFO PIC X(16) VALUE 'GETNAMEINFO '.
   02 SOKET-GETSOCKNAME PIC X(16) VALUE 'GETSOCKNAME '.
   02 SOKET-GETSOCKOPT PIC X(16) VALUE 'GETSOCKOPT '.
   02 SOKET-GIVESOCKET PIC X(16) VALUE 'GIVESOCKET '.
   02 SOKET-INITAPI PIC X(16) VALUE 'INITAPI '.
   02 SOKET-IOCTL PIC X(16) VALUE 'IOCTL '.
   02 SOKET-LISTEN PIC X(16) VALUE 'LISTEN '.
   02 SOKET-NTOP PIC X(16) VALUE 'NTOP '.
   02 SOKET-READ PIC X(16) VALUE 'READ '.
   02 SOKET-RECV PIC X(16) VALUE 'RECV '.
   02 SOKET-RECVFROM PIC X(16) VALUE 'RECVFROM '.
   02 SOKET-SELECT PIC X(16) VALUE 'SELECT '.
   02 SOKET-SEND PIC X(16) VALUE 'SEND '.
   02 SOKET-SENDTO PIC X(16) VALUE 'SENDTO '.
   02 SOKET-SETSOCKOPT PIC X(16) VALUE 'SETSOCKOPT '.
   02 SOKET-SHUTDOWN PIC X(16) VALUE 'SHUTDOWN '.
   02 SOKET-SOCKET PIC X(16) VALUE 'SOCKET '.
   02 SOKET-TAKE_SOCKET PIC X(16) VALUE 'TAKE_SOCKET '.
   02 SOKET-TERM_API PIC X(16) VALUE 'TERMAPI '.
   02 SOKET-WRITE PIC X(16) VALUE 'WRITE '.

---------------------------------------------------------------------
* PROGRAM'S VARIABLES *
---------------------------------------------------------------------
77 PROTOCOL PIC 9(8) COMP VALUE 0.
77 SRV-SOCKID PIC 9(4) COMP VALUE 0.
77 SRV-SOCKID-FWD PIC 9(8) COMP VALUE 0.
77 CLI-SOCKID PIC 9(4) COMP VALUE 0.
77 CLI-SOCKID-FWD PIC 9(8) COMP VALUE 0.
77 LENG PIC 9(4) COMP.
77 WSLENG PIC 9(4) COMP.
77 RESPONSE PIC 9(9) COMP.
Appendix E. Sample programs
01 NTOP-FAMILY PIC 9(8) BINARY.
01 PTON-FAMILY PIC 9(8) BINARY.
01 PRESENTABLE-ADDR PIC X(45) VALUE SPACES.
01 PRESENTABLE-ADDR-LEN PIC 9(4) BINARY VALUE 45.
01 NUMERIC-ADDR.
   05 FILLER PIC 9(16) BINARY VALUE 0.
   05 FILLER PIC 9(16) BINARY VALUE 0.
01 NAME-LEN PIC 9(8) BINARY.
01 HOST-NAME PIC X(255).
01 HOST-NAME-LEN PIC 9(8) BINARY.
01 SERVICE-NAME PIC X(32).
01 SERVICE-NAME-LEN PIC 9(8) BINARY.
01 NAME-INFO-FLAGS PIC 9(8) BINARY VALUE 0.
01 NI-NOFQDN PIC 9(8) BINARY VALUE 1.
01 NI-NUMERICHOST PIC 9(8) BINARY VALUE 2.
01 NI-NAMEREQD PIC 9(8) BINARY VALUE 4.
01 NI-NUMERICSERV PIC 9(8) BINARY VALUE 8.
01 NI-DGRAM PIC 9(8) BINARY VALUE 16.
01 HOST-NAME-CHAR-COUNT PIC 9(4) COMP.
01 HOST-NAME-UNSTRUNG PIC X(255) VALUE SPACES.
01 SERVICE-NAME-CHAR-COUNT PIC 9(4) COMP.
01 SERVICE-NAME-UNSTRUNG PIC X(32) VALUE SPACES.
01 SOCKET-CONV.
   05 SOCKET-TBL OCCURS 6 TIMES.
      10 SOCK-CHAR PIC X(1) VALUE '0'.
01 TCPCICS-MSG-AREA.
   02 TCPCICS-MSG-1.
      05 MSGDATE PIC 9(8).
      05 FILLER PIC X(2) VALUE SPACES.
      05 MSGTIME PIC 9(8).
      05 FILLER PIC X(2) VALUE SPACES.
      05 MODULE PIC X(10) VALUE 'EZACIC6S: '.
   02 TCPCICS-MSG-2.
      05 MSG-AREA PIC X(55) VALUE SPACES.
01 TCP-INPUT-DATA PIC X(85) VALUE LOW-VALUES.
01 TCPSOCKET-PARM REDEFINES TCP-INPUT-DATA.
   05 GIVE-TAKE-SOCKET PIC 9(8) COMP.
   05 CLIENTID-PARM.
      10 LSTN-NAME PIC X(8).
      10 LSTN-SUBTASKNAME PIC X(8).
   05 CLIENT-DATA-FLD.
      10 CLIENT-IN-DATA PIC X(35).
      10 FILLER PIC X(1).
   05 TCPSOCKADDR-IN.
      10 SOCK-FAMILY PIC 9(4) BINARY.
         88 SOCK-FAMILY-IS-AFINET VALUE 2.
         88 SOCK-FAMILY-IS-AFINET6 VALUE 19.
      10 SOCK-DATA PIC X(26).
      10 SOCK-SIN REDEFINES SOCK-DATA.
         15 SOCK-SIN-PORT PIC 9(4) BINARY.
         15 SOCK-SIN-ADDR PIC 9(8) BINARY.
      10 FILLER PIC X(8).
      15 FILLER PIC X(12).
      10 SOCK-SIN6 REDEFINES SOCK-DATA.
         15 SOCK-SIN6-PORT PIC 9(4) BINARY.
         15 SOCK-SIN6-FLOWINFO PIC 9(8) BINARY.
      15 FILLER PIC X(16) BINARY.
      20 FILLER PIC X(16) BINARY.
      15 SOCK-SIN6-SCOPEID PIC 9(8) BINARY.
   05 FILLER PIC X(68).
   05 CLIENT-IN-DATA-LENGTH PIC 9(4) COMP.
   05 CLIENT-IN-DATA-2 PIC X(999).
01 SOCK-TO-RECV-FWD.
   02 FILLER PIC 9(4) BINARY.
   02 SOCK-TO-RECV PIC 9(4) BINARY.
01 TIMEVAL.
   02 TVSEC PIC 9(8) COMP VALUE 180.
02 TVUSEC PIC 9(8) COMP VALUE 0.
01 ZERO-Parm PIC X(16) VALUE LOW-VALUES.
01 ZERO-Fld REDEFINES ZERO-Parm.
  02 ZERO-8 PIC X(8).
  02 ZERO-DUM PIC X(2).
  02 ZERO-HWRD PIC 9(4) COMP.
  02 ZERO-FWRD PIC 9(8) COMP.

* *********************************************** *
* INPUT FORMAT FOR UPDATING THE SAMPLE DB2 TABLE *
* *********************************************** *
01 INPUT-DEPT.
  05 IN-ACT PIC X(3).
  05 IN-DEPTNO PIC X(3).
  05 IN-DEPTN PIC X(36).
  05 IN-MGRNO PIC X(6).
  05 IN-ADMRDEPT PIC X(3).

* SQL STATEMENTS: SQL COMMUNICATION AREA *
* ---------------------------------------- *
*** EXEC SQL INCLUDE SQLCA END-EXEC.

* SQL STATEMENTS: DEPARTMENT TABLE CREATE STATEMENT FOR DB2 *
* -------------------------------------------------------- *
* CREATE TABLE TCPCICS.DEPT *
* (DEPTNO CHAR(3), *
* DEPTNAME CHAR(36), *
* MGRNO CHAR(6), *
* ADMRDEPT CHAR(3)); *

* 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
  EDC
  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-TCPCCICS THRU HANDLE-TCPCCICS-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.
* 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
    ELSE
        MOVE PORT TO PORT-RECORD
        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 'EZACICO6' USING CTOB
    READMASK
    SOCKET-CONV
    TCPLENG
    RETCODE.

IF RETCODE = -1
    THEN
        MOVE BITMASK-ERR TO MSG-AREA
        PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT
    ELSE
        PERFORM ACCEPT-CLIENT-REQ THRU
            ACCEPT-CLIENT-REQ-EXIT
        UNTIL TASK-TERM
    END-IF.

PERFORM CLOSE-SOCKET THRU CLOSE-SOCKET-EXIT.

MOVE TCP-SERVER-OFF TO MSG-AREA.
PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.

******************************************************************************
******************************************************************************
******************************************************************************

** Appendix E. Sample programs **
PGM-EXIT.

EXEC CICS
RETURN
END-EXEC.
GOBACK.

TCP-TRUE-REQ.
MOVE TCP-EXIT-ERR TO MSG-AREA.
PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
GO TO PGM-EXIT.

DB2-TRUE-REQ.
MOVE DB2-CAF-ERR TO MSG-AREA.
PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
GO TO PGM-EXIT.

LISTENER-STARTED-TASK.
MOVE CLIENTID-PARM TO CID-LSTN-INFO.
MOVE GIVE-TAKE-SOCKET TO SOCK-TO-RECV-FWD.
CALL 'EZASOKET' USING SOKET-TAKESOCKET
          SOCK-TO-RECV
          CLIENTID-LSTN
          ERRNO
          RETCODE.

IF RETCODE < 0 THEN
  MOVE ERRNO TO TAKE-ERRNO
  MOVE TAKE-ERR TO MSG-AREA
  PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT
  GO TO PGM-EXIT
ELSE
  MOVE BUFFER-LENG TO TCPLENG
  MOVE START-MSG TO TCP-BUF
  MOVE RETCODE TO SRV-SOCKID
  CALL 'EZACIC04' USING TCP-BUF TCPLENG
  CALL 'EZASOKET' USING SOKET-WRITE
          SRV-SOCKID
          TCPLENG
          TCP-BUF
          ERRNO
          RETCODE

IF RETCODE < 0 THEN
  MOVE ERRNO TO WRITE-ERRNO
  MOVE WRITE-ERR TO MSG-AREA
  PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT

614  z/OS V2R1.0 Communications Server: IP CICS Sockets Guide
GO TO PGM-EXIT
ELSE

CALL 'EZASOKET' USING SOKET-CLOSE
SRV-SOCKID
ERRNO
RETCODE

IF RETCODE < 0 THEN
    MOVE ERRNO TO CLOSE-ERRNO
    MOVE CLOSE-ERR TO MSG-AREA
    PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT
    GO TO PGM-EXIT ELSE NEXT SENTENCE
END-IF
END-IF.

MOVE LOW-VALUES TO TCP-BUF.

LISTENER-STARTED-TASK-EXIT.
EXIT.

*---------------------------------------------------------------*
* START SERVER PROGRAM                                          *
*---------------------------------------------------------------*

INIT-SOCKET.

MOVE EIBTASKN TO SUBTASKNO.

CALL 'EZASOKET' USING SOKET-INITAPI
MAXSOC
IDENT
INIT-SUBTASKID
MAXSNO
ERRNO
RETCODE.

IF RETCODE < 0 THEN
    MOVE ERRNO TO INIT-ERRNO
    MOVE INITAPI-ERR TO MSG-AREA
    PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT
    GO TO PGM-EXIT ELSE NEXT SENTENCE
END-IF.

INIT-SOCKET-EXIT.
EXIT.

SOCKET-BIND-LSTN.

MOVE -1 TO SRV-SOCKID-FWD.

*---------------------------------------------------------------*
* CREATING A SOCKET TO ALLOCATE                                 *
* AN OPEN SOCKET FOR INCOMING CONNECTIONS                        *
*---------------------------------------------------------------*

CALL 'EZASOKET' USING SOKET-SOCKET
AF-INET6
SOCK-TYPE
PROTOCOL
ERRNO
RETCODE.
IF RETCODE < 0 THEN
  MOVE ERRNO TO SOCKET-ERRNO
  MOVE SOCKET-ERR TO MSG-AREA
  PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT
  GO TO PGM-EXIT
ELSE
  MOVE RETCODE TO SRV-SOCKID
  MOVE '1' TO SOCK-CHAR(RETCODE + 1)
END-IF.

*--------------------------------------------------------------*
**
* BIND THE SOCKET TO THE SERVICE PORT *
* TO ESTABLISH A LOCAL ADDRESS FOR PROCESSING INCOMING *
* CONNECTIONS. *
**
*--------------------------------------------------------------*

MOVE AF-INET6 TO SAIN-FAMILY.
MOVE ZEROS TO SAIN-SIN6-FLOWINFO.
MOVE IN6ADDR-ANY TO SAIN-SIN6-ADDR.
MOVE ZEROS TO SAIN-SIN6-SCOPEID.
MOVE PORT TO SAIN-SIN6-PORT.

CALL 'EZASOKET' USING SOKET-BIND
  SRV-SOCKID
  SOCKADDR-IN
  ERRNO
  RETCODE.

IF RETCODE < 0 THEN
  MOVE ERRNO TO BIND-ERRNO
  MOVE BIND-ERR TO MSG-AREA
  PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT
  GO TO PGM-EXIT.

*--------------------------------------------------------------*
**
* CALL THE LISTEN COMMAND TO ALLOWS SERVERS TO *
* PREPARE A SOCKET FOR INCOMING CONNECTIONS AND SET MAXIMUM *
* CONNECTIONS. *
**
*--------------------------------------------------------------*

CALL 'EZASOKET' USING SOKET-LISTEN
  SRV-SOCKID
  BACKLOG
  ERRNO
  RETCODE.

IF RETCODE < 0 THEN
  MOVE ERRNO TO LISTEN-ERRNO
  MOVE LISTEN-ERR TO MSG-AREA
  PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT
  GO TO PGM-EXIT.

SOCKET-BIND-LSTN-EXIT.
EXIT.

*--------------------------------------------------------------*
**
* SOCKET HAS BEEN SET UP, THEN CALL 'ACCEPT' TO *
* ACCEPT A REQUEST WHEN A CONNECTION ARRIVES. *
* THIS SAMPLE PROGRAM WILL ONLY USE 5 SOCKETS. *
**
*--------------------------------------------------------------*

ACCEPT-CLIENT-REQ.

CALL 'EZASOKET' USING SOKET-SELECT
  NFDS
  TIMEVAL
  READMASK
  DUMYMASK
  DUMYMASK
IF RETCODE < 0 THEN
  MOVE ERRNO TO SELECT-ERRNO
  MOVE SELECT-ERR TO MSG-AREA
  PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT
  GO TO PGM-EXIT.
IF RETCODE = 0 THEN GO TO ACCEPT-CLIENT-REQ-EXIT.

*--------------------------------------------------------------*
* ACCEPT REQUEST                                             *
*--------------------------------------------------------------*

CALL 'EZASOKET' USING SOCKET-Accept
  SRV-SOCKID
  SOCKADDR-IN
  ERRNO
  RETCODE.

IF RETCODE < 0 THEN
  MOVE ERRNO TO ACCEPT-ERRNO
  MOVE ACCEPT-ERR TO MSG-AREA
  PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT
  GO TO PGM-EXIT.
MOVE RETCODE TO CLI-SOCKID.
PERFORM GET-NAME-INFO THRU GET-NAME-INFO-EXIT.
PERFORM ACCEPT-RECV THRU ACCEPT-RECV-EXIT
  UNTIL TASK-END OR TASK-TERM.
MOVE DB2END TO MSG-AREA.
PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
CALL 'EZASOKET' USING SOCKET-CLOSE
  CLI-SOCKID
  ERRNO
  RETCODE.

IF RETCODE < 0 THEN
  MOVE ERRNO TO CLOSE-ERRNO
  MOVE CLOSE-ERR TO MSG-AREA
  PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
IF NOT TASK-TERM
  MOVE '0' TO TASK-FLAG.

ACCEPT-CLIENT-REQ-EXIT.
EXIT.

*--------------------------------------------------------------*
* DETERMINE THE CONNECTED HOST NAME BY ISSUING THE             *
* GETNAMEINFO COMMAND.                                         *
*--------------------------------------------------------------*

GET-NAME-INFO.

  MOVE SAIN-SIN6-ADDR TO NUMERIC-ADDR.
  MOVE 45 TO PRESENTABLE-ADDR-LEN.
  MOVE SPACES TO PRESENTABLE-ADDR.
  CALL 'EZASOKET' USING SOCKET-N TOP AF-INET6
    NUMERIC-ADDR
    PRESENTABLE-ADDR PRESENTABLE-ADDR-LEN
ERRNO RETCODE.

IF RETCODE < 0 THEN
  MOVE ERRNO TO NTOP-ERRNO
  MOVE NTOP-ERR TO MSG-AREA
  PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.

  MOVE PRESENTABLE-ADDR TO NTOP-PRESENTABLE-ADDR.
  MOVE NTOP-OK TO MSG-AREA.
  PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.

CALL 'EZASOKET' USING SOKET-GETPEERNAME
  CLI-SOCKID
  SOCKADDR-PEER
  ERRNO
  RETCODE.

IF RETCODE < 0 THEN
  MOVE ERRNO TO GPN-ERRNO
  MOVE GPN-ERR TO MSG-AREA
  PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT
  GO TO PGM-EXIT.

MOVE 28 TO NAME-LEN.
MOVE 255 TO HOST-NAME-LEN.
MOVE 32 TO SERVICE-NAME-LEN.
MOVE ZEROS TO NAME-INFO-FLAGS.
CALL 'EZASOKET' USING SOKET-GETNAMEINFO
  SOCKADDR-PEER
  NAME-LEN
  HOST-NAME
  HOST-NAME-LEN
  SERVICE-NAME
  SERVICE-NAME-LEN
  NAME-INFO-FLAGS
  ERRNO
  RETCODE.

IF RETCODE < 0 THEN
  MOVE ERRNO TO GNI-ERRNO
  MOVE GNI-ERR TO MSG-AREA
  PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.

MOVE 0 TO HOST-NAME-CHAR-COUNT.
INSPECT HOST-NAME TALLYING HOST-NAME-CHAR-COUNT
  FOR CHARACTERS BEFORE X'00'.
UNSTRING HOST-NAME DELIMITED BY X'00'
  INTO HOST-NAME-UNSTRUNG
  COUNT IN HOST-NAME-CHAR-COUNT.
STRING HOST-NAME-UNSTRUNG DELIMITED BY ''
  INTO GNI-HOST-NAME.
MOVE GNI-HOST-NAME-OK TO MSG-AREA.
PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.

MOVE 0 TO SERVICE-NAME-CHAR-COUNT.
INSPECT SERVICE-NAME TALLYING SERVICE-NAME-CHAR-COUNT
  FOR CHARACTERS BEFORE X'00'.
UNSTRING SERVICE-NAME DELIMITED BY X'00'
  INTO SERVICE-NAME-UNSTRUNG
  COUNT IN SERVICE-NAME-CHAR-COUNT.
STRING SERVICE-NAME-UNSTRUNG DELIMITED BY ''
  INTO GNI-SERVICE-NAME.
MOVE GNI-SERVICE-NAME-OK TO MSG-AREA.
PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.

DISPLAY 'HOST NAME = ' HOST-NAME.
DISPLAY 'SERVICE = ' SERVICE-NAME.

GET-NAME-INFO-EXIT.
EXIT.*--------------------------------------------------------------*
**
** RECEIVING DATA THROUGH A SOCKET BY ISSUING 'RECVFROM' COMMAND. *
*-------------------------------------------------------------------*
ACCEPT-RECV.

MOVE 'T' TO TCP-INDICATOR.
MOVE BUFFER-LENG TO TCPLENG.
MOVE LOW-VALUES TO TCP-BUF.

CALL 'EZASOKET' USING SOKET-RECVFROM
CLI-SOCKID
TCP-FLAG
TCPLENG
TCP-BUF
SOCKADDR-IN
ERRNO
RETCODE.

IF RETCODE EQUAL 0 AND TCPLENG EQUAL 0
    THEN NEXT SENTENCE
ELSE
    IF RETCODE < 0
        THEN
            MOVE ERRNO TO RECVFROM-ERRNO
            MOVE RECVFROM-ERR TO MSG-AREA
            PERFORM HANDLE-TCPCICS THRU
            HANDLE-TCPCICS-EXIT
            MOVE '1' TO TASK-FLAG
        ELSE
            CALL 'EZACIC05' USING TCP-BUF TCPLENG
            IF TCP-BUF-H = LOW-VALUES OR SPACES
                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
END-IF.

ACCEPT-RECV-EXIT.
EXIT.

**********************************************************
** PROCESSES TALKING TO CLIENT THAT WILL UPDATE DB2 **
** TABLES. **
**********************************************************
** DATA PROCESS: **
**
** INSERT REC - INS,X81,TEST DEPT,A0213B,Y94 **
** UPDATE REC - UPD,X81,,A1234C, **
** DELETE REC - DEL,X81,, **
** END CLIENT - END,(end client connection ) **
** END SERVER - TRM,(terminate server ) **
**
**********************************************************

TALK-CLIENT.

UNSTRING TCP-BUF DELIMITED BY DEL-ID OR ALL '*'
    INTO IN-ACT
    IN-DEPTNO
    IN-DEPTN
    IN-MGRNO
    IN-ADMRDEPT.

IF IN-ACT EQUAL 'END'
    THEN
        MOVE '1' TO TASK-FLAG
    ELSE
        IF IN-ACT EQUAL 'U' OR EQUAL 'UPD'

Appendix E. Sample programs  619
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
ENDIF
ENDIF
ENDIF.

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 TO
  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 'EZACICO4' USING TCP-BUF TCPLENG
CALL 'Ezasocket' USING SOKET-WRITE
            CLI-SOCKID
            TCPLENG
            TCP-BUF
            ERRNO
            RETCODE
MOVE LOW-VALUES TO TCP-BUF
            TCP-INDICATOR
            DB2-ACT
IF RETCODE < 0
THEN
  MOVE ERRNO TO WRITE-ERRNO
  MOVE WRITE-ERR TO MSG-AREA
  PERFORM HANDLE-TCPCICS THRU
        HANDLE-TCPCICS-EXIT
  MOVE '1' TO TASK-FLAG
ENDIF
ENDIF.
TALK-CLIENT-EXIT.
EXIT.

*---------------------------------------------------------------*
**
* CLOSE ORIGINAL SOCKET DESCRIPTOR *
**
*---------------------------------------------------------------*
CLOSE-SOCKET.

CALL 'EZASOKET' USING SOKET-CLOSE
SRV-SOCKID
ERRNO
RETCODE.

IF RETCODE < 0 THEN
  MOVE ERRNO TO CLOSE-ERRNO
  MOVE CLOSE-ERR TO MSG-AREA
  PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
CLOSE-SOCKET-EXIT.
EXIT.

*---------------------------------------------------------------*
**
* SEND TCP/IP ERROR MESSAGE *
**
*---------------------------------------------------------------*
HANDLE-TCPCICS.

MOVE LENGTH OF TCPCICS-MSG-AREA TO LENG.

EXEC CICS ASKTIME
  ABSTIME (TSTAMP)
  NOHANDLE
END-EXEC.

EXEC CICS FORMATTIME
  ABSTIME (TSTAMP)
  MMDDYY (MSGDATE)
  TIME (MSGTIME)
  DATESEP ('/')
  TIMESEP (':')
  NOHANDLE
END-EXEC.

EXEC CICS WRITEQ TD
  QUEUE ('CSMT')
  FROM (TCPCICS-MSG-AREA)
  RESP (RESPONSE)
  LENGTH (LENG)
END-EXEC.

IF RESPONSE = DFHRESP(NORMAL)
  THEN NEXT SENTENCE
ELSE
  IF RESPONSE = DFHRESP(INVREQ)
    THEN MOVE TS-INVREQ-ERR TO MSG-AREA
  ELSE
    IF RESPONSE = DFHRESP(NOTAUTH)
      THEN MOVE TS-NOTAUTH-ERR TO MSG-AREA
    ELSE
      IF RESPONSE = DFHRESP(IOERR)
        THEN MOVE TS-IOERR-ERR TO MSG-AREA
      ELSE MOVE WRITETS-ERR TO MSG-AREA
      END-IF
    END-IF
  END-IF
END-IF.

IF TCP-INDICATOR = 'T' THEN
  MOVE BUFFER-LENG TO TCPLENG
MOVE LOW-VALUES TO TCP-BUF
MOVE TCPCICS-MSG-2 TO TCP-BUF

CALL 'EZACIC04' USING TCP-BUF TCPLENG
MOVE '' TO TCP-INDICATOR

CALL 'EZASOKET' USING SOKET-WRITE
  CLI-SOCKID
  TCPLENG
  TCP-BUF
  ERRNO
  RETCODE

IF RETCODE < 0 THEN
  MOVE ERRNO TO WRITE-ERRNO
  MOVE WRITE-ERR TO MSG-AREA
  EXEC CICS WRITEQ TO
    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 TO
  QUEUE ('CSMT')
  FROM (TCPCICS-MSG-AREA)
  RESP (RESPONSE)
  LENGTH (LENG)
END-EXEC.

MOVE LOW-VALUES TO TCP-BUF.
MOVE TCPCICS-MSG-2 TO TCP-BUF.

CALL 'EZACIC04' USING TCP-BUF TCPLENG.

CALL 'EZASOKET' USING SOKET-WRITE
  CLI-SOCKID
  TCPLENG
  TCP-BUF
  ERRNO
  RETCODE.

IF RETCODE < 0 THEN
  MOVE ERRNO TO WRITE-ERRNO
  MOVE WRITE-ERR TO MSG-AREA
  PERFORM HANDLE-TCPCICS THRU HANDLE-TCPCICS-EXIT.
END-IF.

GO TO PGM-EXIT.

SQL-ERROR-ROU-EXIT.

622  z/OS V2R1.0 Communications Server: IP CICS Sockets Guide
EXIT.

*---------------------------------------------------------------*
* OTHER ERRORS (HANDLE CONDITION) *
*---------------------------------------------------------------*

INVREQ-ERR-SEC.
   MOVE TCPIP-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 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.

EZACICAC

The following Assembler socket program is in the SEZAINST data set.
**Figure 182. EZACICAC assembler child server sample**

***********************************************************************
* Module Name: EZACICAC - This is a very simple child server *
* Copyright: Licensed Materials - Property of IBM *
* "Restricted Materials of IBM"
* 5694-A01
* Copyright IBM Corp. 2003, 2007
* US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
* Status: CSV1R9
* LANGUAGE: ASSEMBLER
* ATTRIBUTES: NON-REUSEABLE
* REGISTER USAGE:
  * R 1 = *
  * R 2 = *
  * R 3 = *
  * R 4 = *
  * R 5 = *
  * R 6 = *
  * R 7 = *
  * R 8 = *
  * R 9 = *
  * R10 = *
  * R11 = *
  * R12 = *
  * R13 = *
  * R14 = *
  * R15 = *
* INPUT:
* OUTPUT:
* $MOD(EZACICAC),COMP(CICS),PROD(TCPIP):
***********************************************************************

DFHEISTG DSECT
SOCSTG DS 0F PROGRAM STORAGE
* Storage to format messages
* TDMSG DS 0F WRITEQ TD Message area
TDDATE DS CL8 MM/DD/YY
TDTIME DS CL8 HH:MM:SS
TDTEXT DS CL40 TDTEXT
* ORG TDTEXT
TDTEXT0 DS OCL40
TDCMD DS CL16 COMMAND ISSUED
TDRESULT DS CL24 SUCCESSFUL/UNSUCCESSFUL
TDMSG EQU * End of message
TDMSGL EQU TDMSG-TDMSG Length of TD message text
* Message to display the clients host name
* ORG TDTEXT
TDHOSTMSG DS OCL40
TDHOSTLIT DS CL9
TDHOST DS CL31
*
* Message to display the clients service name
* 
* ORG TDTTEXT
TDSERVMGS DS OCL40
TDSERVFLIT DS CL8
TDSERV DS CL32
*
TDLEN DS H Length of TD message text
*
* Working storage fields
*
CLENG DS H Length of data to RETRIEVE
UTIME DS PL8 ABSTIME data area
DNORK DS D Double work work area
UNPKWRK DS CL15 For packing/unpacking
PARMLIST DS 20F Parm list for EZASOKET calls
*
SOCDESC DS H Socket Descriptor
*
ERRNO DS F ERRNO
RETCODE DS F Return code
*
* Storage to map the clientid structure.
*
CLIENTID DS OCL40
GIVE.DOM DS F Domain of socket given/taken
AS_NAME DS CL8 Address space name
TASK_ID DS CL8 Task identifier
DS CL20 Reserved
*
* Storage to address the Transaction Input Message from the Listener.
*
SOKTIM DS OCL1153
SOKDESC DS F Socket descriptor given
SOKLASID DS CL8 Listener address space name
SOKLIDI DS CL8 Listener task identifier
SOKDATAI DS CL35 Client input data
SOKTSI DS CL1 Threadsafe inidicator
SOKADDR DS OF Clients socket address
SOKFAM DS H Address family
SOK_DATA DS 0C Protocol specific area
SOKFLEN EQU *-SOKADDR Start of AF_INET unique area
SOK.Sin DS 0C
SOK Sin.PORT DS H Clients port number
SOK_SIN_CIPAD DS F Clients INET address (netid)
DS CLB Reserved area not used
DS 2OF
SOK_SIN_LEN EQU *-SOK_SIN Length of AF_INET area
ORG SOK_DATA Start of AF_INET unique area
SOK.Sin DS 0C
SOK_SIN.Port DS H Clients port number
SOK_SIN_FILOINFO DS CL4 Flow information
SOK_SIN_CIPAD DS OCL6 Clients INET address (netid)
SOK_SIN_SCOPE_ID DS CL4 Scope Id
SOK_SIN_FLEN EQU *-SOK SIn Length of AF_INET6 area
ORG SOK_DATA Start of AF_INET6 unique area
SOK.Sin DS 0C
SOK_SIN6_PORT DS H Clients port number
SOK_SIN6_FILOINFO DS CL4 Flow information
SOK_SIN6_CIPAD DS CL6 Clients INET address (netid)
SOK_SIN6_SCOPE_ID DS CL4 Scope Id
SOK_SIN6_FLEN EQU *-SOK_SIN6 Length of AF_INET6 area
ORG SOK_DATA Start of AF_INET6 unique area
SOK_SIN6 DS CL68 Reserved
SOKDATA1 DS H Length of data area 2
SOKDATA2 DS CL999 Data area 2
*
* Program storage marker
*
SOCSTGE EQU * End of Program Storage
SOCSTGL EQU SOCSTGE-SOCSTG Length of Program Storage
*
* Beginning of program
*
EZACICAC CSECT
EZACICAC AMODE ANY Addressing mode ...
EZACICAC RMODE ANY Residency mode ...
SOC0000 DS 0H
B SOC00100 Branch to startup address
DC CL17'EZACICAC-EYECATCH'
SOC00100 DS 0H Beginning of program
LA R10,SOCSTG Address Pgm Dynamic Stg
USING SOCSTG,R10 Tell Assembler about storage
MVC TDTTEXT(40),STARTED_MSG Move STARTED message to TD area
BAL R7,WRITEQ Write to TD Queue
MVC CLENG,=H'72'  Length for standard listener
MVC CLENG,=H'1153' Length for enhanced listener

* Retrieve the Task Input Message (TIM) from the Listener
* EXEC CICS RETRIEVE INTO(SOKTIM) LENGTH(CLENG)
* Issue the 'TAKESOCKET' call to acquire the socket which was
* given by the listener program.
* XC CLIENTID,CLIENTID Clear the clientid structure
MVC GIVE_DOM=2,SOKFAM Based on the AF in the TIM
MVC AS_NAME,SOKLASID Set the address space name
MVC TASK_ID,SOKLID and the subtask identifier
MVC SOCDESC,SOKDESC+2 and the socket descriptor.
* CALL EZASOKET,(SOCTSOCK,SOCDESC,CLIENTID, X
ERRNO,RETCODE),VL,MF=(E,PARMLIST)
L R5,ERRNO Capture the ERRNO and
L R6,RETCODE the return code.
C R6,=F'0'  Is the call successful?
BL SOCERR No! Go display error and terminate
MVC SOCDESC,RETCODE+2 Yes, format the return code and
MVC TDcmd,SOCtSOCK the API function performed.
MVC TDRESULT(24),SUCC Move SUCCESSFUL msg to TD area
MVC TDCMD,SOCDESC,PEERADDR Move message to TD area
BAL R7,WRITEQ Write to TD Queue

* Remove the following call to EZACICO4 if using an EBCDIC client.
* CALL EZACICO4,(TCP_BUF,TCPLENG),VL,MF=(E,PARMLIST)
* Notify client the the child subtask has started.
* CALL EZASOKET,(SOCWRITE,SOCDESC,TCPLENG,TCP_BUF, X
ERRNO,RETCODE),VL,MF=(E,PARMLIST)
* L R5,ERRNO Capture the ERRNO and
L R6,RETCODE the return code.
C R6,=F'0'  Is the call successful?
BL SOCERR No! Go display error and terminate
MVC TDcmd,SOCWRITE the API function performed.
MVC TDRESULT(24),SUCC Move SUCCESSFUL msg to TD area
MVC TDCMD,TCPLENG,PEERADDR Move message to TD area
BAL R7,WRITEQ Write to TD Queue

* Get our peers' socket address
* CALL EZASOKET,(SOCADDR,SOCDESC,PEERADDR, X
ERRNO,RETCODE),VL,MF=(E,PARMLIST)
* L R5,ERRNO Capture the ERRNO and
L R6,RETCODE the return code.
C R6,=F'0'  Is the call successful?
BL SOCERR No! Go display error and terminate
MVC TDcmd,SOCADDR the API function performed.
MVC TDRESULT(24),SUCC Move SUCCESSFUL msg to TD area
MVC TDCMD,PEERADDR Move message to TD area
BAL R7,WRITEQ Write to TD Queue

* Get our client's host name and service name
* L R8,=F'16' Set the sockaddr length to IPv4
CLC SOKFAM,=AL2(AF_INET) Is the client AF_INET ?
BE SET SOCKADDR_LEN Yes. Go store the length.
L R8,=F'28'  Set the sockaddr length to IPv6
SET_SOCKADDR_LEN DS OH
ST R8,PEERADDR_LEN Save the value of the sockaddr length
L R8,=F'0'  Clear the
ST R8,ONI_FLAGS flags
XC PEER_HOSTNAME,PEER_HOSTNAME Clear the host name storage
L R8,=F'255' Set the length of
ST R8,PEER_HOSTNAMELEN the host name storage

626  z/OS V2R1.0 Communications Server: IP CICS Sockets Guide
XC PEER_SERVICENAME,PEER_SERVICENAME Clear the service name storage
L R8,=F'32' Set the length of
ST R8,PEER__SERVICENAMELEN the service name storage
*
CALL EZASOKET,(SOCGNI,PEERADDR,PEERADDR_LEN, X
PEER_HOSTNAME,PEER_HOSTNAMELEN, X
PEER_SERVICENAME,PEER_SERVICENAMELEN, X
GNI_FLAGS, X
ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
L R5,ERRNO Capture the ERRNO and
L R6,RETCODE the return code.
C R6,=F'0' Is the call successful?
BL SOCERR No! Go display error and terminate
MVC TDCMD,SOCGNI the API function performed.
MVC TRESULT(24),SUCCE Move SUCCESSFUL msg to TD area
MVC TDTEXT(40),TDTXT0 Move message to TD area
BAL R7,WRITEQ Write to TD Queue
*
* Display the host name
*
MVC TDHOSTLIT,=C'HOSTNAME=' Move message to TD area
MVC TDHOST(L'TDHOST),PEER_HOSTNAME
MVC TDTEXT(40),TDHOSTMSG Move message to TD area
BAL R7,WRITEQ Write to TD Queue
*
* Display the service name
*
MVC TDHOSTLIT,=C'SERVICE=' Move message to TD area
MVC TDSERV(L'TDSERV),PEER_SERVICENAME
MVC TDTEXT(40),TDSERVMSG Move message to TD area
BAL R7,WRITEQ Write to TD Queue
*
* Receive data from the client
*
AGAIN1 DS 0H
*
XC TCP_BUF,TCP_BUF Clear the buffer storage
*
CALL EZASOKET,(SOCRECV,SOCDESC,RECV_FLAG,TCPLENG,TCP_BUF, X
ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
L R5,ERRNO Capture the ERRNO and
L R6,RETCODE the return code.
C R6,=F'0' Is the call successful?
BL SOCERR No! Go display error and terminate
MVC TDCMD,SOCRECV the API function performed.
MVC TRESULT(24),SUCCE Move SUCCESSFUL msg to TD area
MVC TDTEXT(40),TDTXT0 Move message to TD area
BAL R7,WRITEQ Write to TD Queue
*
* Remove the following call to EZACIC05 if using an EBCDIC client.
*
CALL EZACIC05,(TCP_BUF,TCPLENG),VL,MF=(E,PARMLIST)
*
* Determine whether the client is finished sending data
*
CLC TCP_BUF_H,=C'END'
BE SIGNAL_CLOSING
CLC TCP_BUF_H,=C'end'
BE SIGNAL_CLOSING
*
* Remove the following call to EZACIC04 if using an EBCDIC client.
*
CALL EZACIC04,(TCP_BUF,TCPLENG),VL,MF=(E,PARMLIST)
*
* Echo the data received back to the client
*
CALL EZASOKET,(SOCWRITE,SOCDESC,TCPLENG,TCP_BUF, X
ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
L R5,ERRNO Capture the ERRNO and
L R6,RETCODE the return code.
C R6,=F'0' Is the call successful?
BL SOCERR No! Go display error and terminate
MVC TDCMD,SOCWRITE the API function performed.
MVC TRESULT(24),SUCCE Move SUCCESSFUL msg to TD area
MVC TDTEXT(40),TDTXT0 Move message to TD area
BAL R7,WRITEQ  Write to TD Queue

*  Go receive another message
*  B  AGAIN1
*  Tell client the connection will close.
*  SIGNAL_CLOSING DS OH
  XC  TCP_BUF,TCP_BUF  Clear the buffer storage
  MVC  TCP_BUF(L'WRKEND),WRKEND  Set the message
  L  R8,='F'50'  Set the
  ST  R8,TCPLENG  message length.
*  Remove the following call to EZACIC04 if using an EBCDIC client.
*  CALL  EZACIC04,(TCP_BUF,TCPLENG),VL,MF=(E,PARMLIST)
*  Notify the client that the connection will end.
  CALL  EZASOKET,(SOCWRITE,SOCDESC,TCPLENG,TCP_BUF,
  ERRNO,RETCODE),VL,MF=(E,PARMLIST)
  L  R5,ERRNO  Capture the ERRNO and
  L  R6,RETCODE  the return code.
  C  R6,='F'0'  Is the call successful?
  BL  SOCERR  No!  Go display error and terminate
  MVC  TDCMD,SOCWRITE  the API function performed.
  MVC  TDRESULT(24),SUCC  Move SUCCESSFUL msg to TD area
  MVC  TDTEXT(40),TDTXT0  Move message to TD area
  BAL  R7,WRITEQ  Write to TD Queue
*  Close the socket
  CALL  EZASOKET,(SOCCLOSE,SOCDESC,
  ERRNO,RETCODE),VL,MF=(E,PARMLIST)
  L  R5,ERRNO  Capture the ERRNO and
  L  R6,RETCODE  the return code.
  C  R6,='F'0'  Is the call successful?
  BL  SOCERR  No!  Go display error and terminate
  MVC  TDRESULT(24),SUCC  Move SUCCESSFUL msg to TD area
  MVC  TDTXT(40),TDTXT0  Move message to TD area
  BAL  R7,WRITEQ  Write to TD Queue
  B  SOCRET  Go return to CICS
*  Error routine for all socket calls
*  SOCERR  DS OH
  MVI  FORCEMSG,C'Y'  Indicate message should be forced
  MVC  TDTXT(40),='SOCKET ERROR'
  BAL  R7,WRITEQ  Write to TD Queue
  L  R6,RETCODE  Pick up the return code value
  L  R5,ERRNO  Pick up the ERRNO value
*  CVD  R6,DWORK  Format the return code
  UNPK  TDRETC,DWORK+4(4)  for printing to the
  OI  TDRETC+6,X'F0'  TD queue
*  CVD  R5,DWORK  Format the ERRNO
  UNPK  TDERRNO,DWORK+4(4)  for printing to the
  OI  TDERRNO+6,X'F0'  TD queue
*  MVC  TDTXT(40),TDTXTS  Move the return code and ERRNO to
  BAL  R7,WRITEQ  the TD queue.  Write to the TD queue
  B  SOCRET  Go return to CICS
*  Subroutine to write messages to the destination "CSMT" for logging
*  WRITEQ  DS OH
  CLI  SOKTSI,C'1'  Is interface using OTE?
  BNE  WRITEQ01  No, write message.
  CLI  FORCEMSG,C'Y'  Is this an error message?
  BNE  WRITEQ02  Yes, bypass writing message.
  WRITEQ01  DS OH
    EXEC  CICS  ASKTIME  ABSTIME(UTIME)
    EXEC  CICS  FORMATTIME  ABSTIME(UTIME)
DATESEP('/') DDDMYY(TODATE) X
TIME(TODTIME) TIMESEP
LA R6,TOMSGL
STH R6,TDLLEN
EXEC CICS WRITEQ TD QUEUE('CSMT') X
FROM(TDMSG) X
LENGTH(TDLLEN)
WRITEQ02 DS OH
XC TDMSG,TOMSG
BR A7 Return to caller
*
* Socket family values
* AFINET DC F'2' AF_INET
AFINET6 DC F'19' AF_INET6
AF_INET EQU 2
AF_INET6 EQU 19
*
* Socket protocol values
* SSTREAM DC F'1' socket type stream
SDATAGRM DC F'2' socket type datagram
SRAW DC F'3' socket type raw
*
* IP CICS Socket API functions
* SOCACCT DC CL16'ACCEPT'
SOCBIND DC CL16'BIND'
SOCCLOSE DC CL16'CLOSE'
SOCCONNT DC CL16'CONNECT'
SOCCNTL DC CL16'FCNTL'
SOCCOLID DC CL16'GETCLIENTID'
SOCGTHBA DC CL16'GETHOSTBYADDR'
SOCGTHBN DC CL16'GETHOSTBYNAME'
SOCGTHID DC CL16'GETHOSTID'
SOCGTHN DC CL16'GETHOSTNAME'
SOCGPNA DC CL16'GETPEERNAME'
SOCGDI DC CL16'GETNAMEINFO'
SOCFAI DC CL16'FREEADDRINFO'
SOCGAI DC CL16'GETADDRINFO'
SOCGTSN DC CL16'GETSOCKNAME'
SOCGETSO DC CL16'GETSOCKOPT'
SOCSOCK DC CL16'GIVESOCKET'
SOCINIT DC CL16'INITAPI'
SOCIOCTL DC CL16'IOCTL'
SOCLISTN DC CL16'LISTEN'
SOCTOP DC CL16'NTOP'
SOCPTON DC CL16'PTON'
SOCREAD DC CL16'READ'
SOCREADV DC CL16'READV'
SOCRCV DC CL16'RECV'
SOCRCVF DC CL16'RECVFROM'
SOCRCVM DC CL16'RECVMSG'
SOCSELECT DC CL16'SELECT'
SOCSELX DC CL16'SELEX'
SOCSEND DC CL16'SEND'
SOSENDM DC CL16'SENOMSG'
SOSENDR DC CL16'SENDTO'
SOCSSOPT DC CL16'SETSOCKOPT'
SOCSHUTD DC CL16'SHUTDOWN'
SOCSOCK DC CL16'SOCKET'
SOCTSOCK DC CL16'TAKESOCKET'
SOCTERM DC CL16'TERMAPI'
SOCWRITE DC CL16'WRITE'
SOCWRITV DC CL16'WRITEV'
ZERO DC F'0'
*
* Message(s) written to the transient data queue
* STARTED_MSG DC CL40'EZACICAC Started successfully'
STOPPED_MSG DC CL40'EZACICAC Stopped successfully'
NOCOMMAREA DC CL40'***ERROR*** NO COMMAREA PASSED!
TASK_START DC CL40'TASK STARTING THRU CICS/TCPIP INTERFACE'
WRKEND DC CL20'CONNECTION END'
*
* Message buffer for data from/to client
* TCP_BUF DS 0CL200 Buffer
TCP_BUF_H DC CL3'
TCP_BUF_DATA DC CL197' '
TCPLENG DC F'200' Length of buffer
*
* Peers sockaddr
*
PEERADDR DS GF Clients socket address
PEERFAM DS H Address family
PEER_DATA DS &C Protocol specific area
PEER_LEN EQU **PEERADDR ORG PEER_DATA Start of AF_INET unique area
PEER_SIN DS &C
PEER_SIN_PORT DS H Clients port number
PEER_SIN_ADDR DS $F Clients INET address (netid)
 DS CL8 Reserved area not used
 DS 2OF PEER_SIN#LEN EQU *-PEER_SIN Length of AF_INET area
 ORG PEER_DATA Start of AF_INET6 unique area
PEER_SIN6 DS &C
PEER_SIN6_PORT DS H Clients port number
PEER_SIN6_FLOWINFO DS CL4 Flow information
PEER_SIN6_ADDR DS CL16 Clients INET address (netid)
PEER_SIN6_SCOPE_ID DS CL4 Scope Id
PEER_SIN6#LEN EQU *-PEER_SIN6 Length of AF_INET6 area
*
PEERADDR_LEN DS F
*
* Peers HOST/SERVICE NAME/LEN
*
PEER_HOSTNAME DS CL255
PEER_HOSTNAMELEN DS F
PEER_SERVICENAME DS CL32
PEER_SERVICENAMELEN DS F
*
* Receive Flag
*
GNI_FLAGS DS F GETNAMEINFO flags
*
* Receive Flag
*
RECV_FLAG DS F RECEIVE flags
*
*
TDTEXT5 DS &C40
 DC CL10'Retcode = '
 TDRETC DC CL7' ' Printable RETCODE
 DC CL3' '
 DC CL9'ERRNO = '
 TDERRNO DC CL7' ' Printable ERRNO
 DC CL4'
*
*
SUCC DC CL24'Successful '
NOTSUCC DC CL24'Not successful '
FORCEREGS DS CL1 Used to force the message when threadsafe
LORG
*
* All done. Return to CICS...
*
SOCRET DS &H
 MVC TDTEXT(40),STOPPED MSG Move STOPPED message to TD area
 BAL R7,WRITEQ Write to TD Queue
 EXEC CICS RETURN
END

EZACICAS

The following Assembler socket program is in the SEZAINST data set.
Figure 183. EZACICAS assembler iterative server sample

ASM XOPTS(NOPROLOG)
*******************************************************************************
Module Name: EZACICAS - This is a sample iterative server
Copyright: Licensed Materials - Property of IBM
"Restricted Materials of IBM"
5694-A01
Copyright IBM Corp. 2003, 2007

US Government Users Restricted Rights - Use, duplication or disclosure restricted by
GSA ADP Schedule Contract with IBM Corp.
Status: CSV1R9

LANGUAGE: ASSEMBLER
ATTRIBUTES: NON-REUSEABLE
REGISTER USAGE:
R 1 =
R 2 =
R 3 = BASE REGISTER
R 4 = BASE REGISTER
R 5 =
R 6 = WORK
R 7 = SUBROUTINE
R 8 = WORK
R 9 = GWA REGISTER
R10 =
R11 = EIB REGISTER
R12 =
R13 = DATA REGISTER
R14 =
R15 =

INPUT:

OUTPUT:

$MOD(EZACICAS),COMP(CICS),PROD(TCPIP):
*******************************************************************************
EZACICAS CSECT
DFHEIDENT CODEREG=(3,4), Base registers for the program X
DATAREG=(13), Base register for data X
EIBREG=(11) Base register for CICS EIB
EZACICAS AMODE ANY ADDRESSING MODE ...
EZACICAS RMODE ANY RESIDENCY MODE ...
B SRV60000 Branch to startup address
DC CL1'EZACICAS-EYECATCH'
SRV60000 DS 0H Beginning of program
USING GWA0000,R9 Address GWA storage
MVC MODULE,'C'EZACICAS:
*
* Establish conditions to be ignored
* EXEC CICS IGNORE CONDITION TERMERR EOC SIGNAL NOTALLOC
* Establish conditions to be handled
* EXEC CICS HANDLE CONDITION ENDDATA(ENDDATA_ERR), X
  IOERR(IOERR_ERR), X
  LENGERR(LENGERR_ERR), X
  NOSPACE(NOSPACE_ERR), X
  QIDERR(QIDERR_ERR)
* Send message that server has started.
* XC MSGAREA,MSGAREA Clear the message buffer
  MVC MSGAREA(L'STARTOK'),STARTOK Move STARTED message
  BAL R7,HANDLE_TCPCICS Write to TD Queue
*
* Determine the CICS Applid
* EXEC CICS ASSIGN APPLID(APPLID)
* Before the server can start, determine whether the IP CICS Sockets
* interface is active.
* EXEC CICS PUSH HANDLE
  EXEC CICS HANDLE CONDITION INVEXITREQ(TCP_TRUE_REQ), NOTAUTH(NOTAUTH_ERR)
  EXEC CICS EXTRACT EXIT PROGRAM('EZACIC01'), GASET(R9) GALENGTH(GWALEN)
*
EXEC CICS POP HANDLE
* At startup, the server requires the port number which it will use
* for its passive socket.
* Invocation: <server>,<port number>
* where server is the CICS Transaction name assigned to EZACICAS
* and port number is a port to which EZACICA will bind as its
* passive socket.
* TERMINAL => SRV6 04000
* LISTENER => SRV6,04000
* CECI => CECI START TR(SRV6) FROM(04000)
* THE LEADING SPACES ARE SIGNIFICANT.
*
* XC TCP_INPUT_DATA,TCP_INPUT_DATA Clear input data area
  L R8,ZERO
  STH R8,TRMNL_LEN
  L R8,TEN
  STH R8,TRMNL_MAXLEN from the terminal
*
EXEC CICS RECEIVE INTO(TCP_INPUT_DATA) LENGTH(TRMNL_LEN) X
  MAXLENGTH(TRMNL_MAXLEN)
*
LH R8,TRMNL_LEN
  C R8,TEN from the terminal. Was it 10?
  BE USE_RECEIVED_PORT Yes, go determine the port number
*
* XC TCP_INPUT_DATA,TCP_INPUT_DATA Clear input data area
  L R8,'F'1153'
  STH R8,RETRIEVE_LEN from The Listener
  MVC TRANS,EIBTRNID Copy the passed trans
*
EXEC CICS RETRIEVE INTO(TCP_INPUT_DATA) LENGTH(RETRIEVE_LEN)
*
* Determine if the server was started by CECI or a listener.
* LH R8,RETRIEVE_LEN Load the RETRIEVED length
  C R8,CICI_LEN Is it less than 5?
  BNH USE_RETRIEVED_PORT Yes. Go use the RETRIEVE'd port
  OI TAKESOCKET_SWITCH,X'01' Otherwise indicate the server X
  was started by the Listener
*
MVC BIND_PORT(5),CLIENT_IN_DATA For the LISTEN message
  PACK DWORK(8),CLIENT_IN_DATA(5) Use port from TIM
  B CONVERT_PORT Go convert it to binary format
USE_RECEIVED_PORT DS OH
  MVC BIND_PORT(5),TCP_INPUT_DATA+5 For the LISTEN message
  PACK DWORK(8),TCP_INPUT_DATA+5(5) Use the port RECEIVE'd
  B CONVERT_PORT
USE_RETRIEVED_PORT DS OH
  MVC BIND_PORT(5),TCP_INPUT_DATA For the LISTEN message
  PACK DWORK(8),TCP_INPUT_DATA(5) Use the port RETRIEVE'd
  B CONVERT_PORT_DS OH
  CBV R8,DWORK Convert user supplied port to binary
  STH R8,PORT and save it for the passive socket
*
* If the server was started by a listener, then we must take the socket
* given. Otherwise, we should proceed with an INITAPI.
* TM TAKESOCKET_SWITCH,X'01' Do we need to use TAKESOCKET ?
LISTENER_STARTED_TASK Yes. Go issue TAKESOCKET

* Since the server was not started by a listener, we should initialize
* the IP CICS Sockets interface.

INIT_SOCKETS DS 0H
   MVC SUBTASKNO,EIBTASKN Use the CICS task number
   CALL EZASOKET,(SOCINIT,MAXSOC,IDENT,INIT_SUBTASKID,MAXSNO, X
   ERRNO,RETCODE),VL,MF=(E,PARMLIST)

   L R5,ERRNO Check for successful call
   L R6,RETCODE Check for successful call
   MVC MSGCMD,SOCINIT Show the API command
   C R6,ZERO Is it less than zero
   BL SOCERR Yes, go display error and terminate
   MVC MSGRESULT(L'SUCC),SUCC Move SUCCESSFUL msg to TD area
   BAL R7,HANDLE_TCPCICS Write to TD Queue
   MVI TERMAPI_REQUIRED_SW,C'Y' Since we did an INITAPI.

* Get an AF_INET6 socket. If unsuccessful, then get an AF_INET socket.

SOCKET_BIND_LISTEN DS 0H
   CALL EZASOKET,(SOCSOKET,AFINET6,SSTREAM,ZERO, X
   ERRNO,RETCODE),VL,MF=(E,PARMLIST)

   L R5,ERRNO Check for successful call
   L R6,RETCODE Check for successful call
   MVC MSGCMD,SOCSOKET Show the API command
   C R6,ZERO Is it less than zero
   BL GET_IPV4_SOCKET Yes, go get an IPv4 socket
   STH R6,SRV_SOCKID Save the new socket descriptor
   MVC MSGRESULT(L'SUCC),SUCC Move SUCCESSFUL msg to TD area
   BAL R7,HANDLE_TCPCICS Write to TD Queue

* Setup an IPv6 sockaddr.

   MVC SAIN_SOCK_FAMILY,=AL2(AF_INET6) Set family to AF_INET6
   XC SAIN_SOCK_SIN6_FLOWINFO Clear the sockaddr storage
   MVC SAIN_SOCK_SIN6_ADDR,IN6ADDR_ANY Use IN6ADDR_ANY
   XC SAIN_SOCK_SIN6_SCOPE_ID,SAIN_SOCK_SIN6_SCOPE_ID X
   Scope ID is zeros
   MVC SAIN_SOCK_SIN6_PORT,PORT Use the user specified port
   BIND_SERVER_SOCKET Now go issue a BIND

GET_IPV4_SOCKET DS 0H
   CALL EZASOKET,(SOCSOKET,AFINET,SSTREAM,ZERO, X
   ERRNO,RETCODE),VL,MF=(E,PARMLIST)

   L R5,ERRNO Check for successful call
   L R6,RETCODE Check for successful call
   MVC MSGCMD,SOCSOKET Show the API command
   C R6,ZERO Is it less than zero
   BL SOCERR Yes, go display error and terminate
   STH R6,SRV_SOCKID Save the new socket descriptor
   MVC MSGRESULT(L'SUCC),SUCC Move SUCCESSFUL msg to TD area
   BAL R7,HANDLE_TCPCICS Write to TD Queue

* Setup an IPv4 sockaddr

   XC SOCKADDR_IN(28),SOCKADDR_IN Clear the sockaddr storage
   MVC SAIN_SOCK_FAMILY,=AL2(AF_INET) Set family to AF_INET
   MVC SAIN_SOCK_SIN_ADDR,INADDR_ANY Use INADDR_ANY
   MVC SAIN_SOCK_SIN_PORT,PORT Use the user specified port

* Bind the socket to the service port to establish a local address for
* processing incoming connections.

BIND_SERVER_SOCKET DS 0H
   CALL EZASOKET,(SOCBIND,SRV_SOCKID,SOCKADDR_IN, X
   ERRNO,RETCODE),VL,MF=(E,PARMLIST)

   L R5,ERRNO Check for successful call
   L R6,RETCODE Check for successful call
   MVC MSGCMD,SOCBIND Show the API command
   C R6,ZERO Is it less than zero
BL SOCERR  Yes, go display error and terminate
MVC MSGRESULT(L'SUCC'),SUCC Move SUCCESSFUL msg to TD area
BAL R7,HANDLE_TCPCICS Write to TD Queue

* Call the LISTEN command to allow server to prepare a socket for
* incoming connections and set the maximum number of connections.
* MVC BACKLOG,TEN  Set backlog to 10
* CALL EZASOKET,(SOCLISTN,SRV_SOCKID,BACKLOG, X
* ERRNO,RETCODE),VL,FM=(E,PARMLIST)
* L R5,ERRNO  Check for successful call
L R6,RETCODE  Check for successful call
MVC MSGCMD,SOCLISTN
C R6,ZERO  Is it less than zero
BL SOCERR  Yes, go display error and terminate
MVC MSGRESULT(L'SUCC'),SUCC Move SUCCESSFUL msg to TD area
BAL R7,HANDLE_TCPCICS Write to TD Queue

* Show server is ready to process client connections.
* L R6,TWO  Force client socket descriptor
STH R6,CLI_SOCKID to be 2.
MVC MSGAREA(L'LISTEN_SUCC'),LISTEN_SUCC
BAL R7,HANDLE_TCPCICS Write to TD Queue

* Create a read mask for the SELECT command
* L R8,NUM_FDS  Get the number of allowed FD's
A R8,ONE and add one
ST R8,NFDS for the SELECT call.

* Determine status IP CICS Sockets Interface
* CLI GWATSTAT,GWATIMED  Are we in immediate termination
BE SOCKET  Return if so
CLI GWATSTAT,GWATQUIE Are we in quiescent termination
BNE SET_SELECT_BIT_MASK No, continue with SELECT
B CLOSEDOWN

* Create the read bitmask
* SET_SELECT_BIT_MASK DS 0H
LH R6,SRV_SOCKID  Get the servers socket descriptor
SRDL R6,5  Compute the word number
SRL R7,27  Compute the socket number within the X
mask word.
SLR R8,R8  Clear work register
LA R8,1  Set high-order bit
SLL R8,0(R7)  Create mask word
ST R8,SASAVE  Save mask word
SLL R6,2  Compute the offset
LA R7,READMASK  Address the read mask storage
LA R7,0(R6,R7)  Point to the word
OC 0(4,R7),SASAVE  Turn on bits

* SELECT client connections
* ACCEPT_CLIENT_REQ DS OH
* CALL EZASOKET,(SOCSELECT,NFDS,TIMEVAL, X
READMASK,DUMYMASK,DUMYMASK, X
REPLY_RDMASK,DUMYMASK,DUMYMASK, X
ERRNO,RETCODE),VL,FM=(E,PARMLIST)
* L R5,ERRNO  Check for successful call
L R6,RETCODE  Check for successful call
ST R6,SELECT_RETCODE  Save the SELECT return code
MVC MSGCMD,SOCSELECT
C R6,ZERO  Is it less than zero
BL SOCERR  Yes, go display error and terminate
MVC MSGRESULT(L'SUCC'),SUCC Move SUCCESSFUL msg to TD area
BAL R7,HANDLE_TCPCICS Write to TD Queue

* Check the return code to determine if any sockets are ready to be
* accepted. If RETCODE is zero then there are no sockets ready.
* L R6,SELECT_RETCODE  Retrieve the SELECT return code
Any sockets ready?

Accept the client request.

CALL EZASOCKET,(SOCACCT,SRV_SOCKID,SOCKADDR_IN, X
ERRNO,RETCODE),VL,MF=(E,PARMLIST)

L R5,ERRNO
L R6,RETCODE
MVC MSGCMD,SOCACCT
C R6,ZERO
BL SOCERR
STH R6,CLI_SOCKID
MVC MSGRESULT(L'SUCC),SUCC
BAL R7,HANDLE_TCPCICS

Get our peers' socket address

CALL EZASOCKET,(SOCGPEER,CLI_SOCKID,SOCKADDR_PEER, X
ERRNO,RETCODE),VL,MF=(E,PARMLIST)

L R5,ERRNO
L R6,RETCODE
MVC MSGCMD,SOCGPEER
C R6,ZERO
BL SOCERR
MVC MSGRESULT(L'SUCC),SUCC
BAL R7,HANDLE_TCPCICS

Get our client's host name and service name

L R8,=F'16'
CLC PEER_SOCK_FAMILY,=AL2(AF_INET)
BE SET_SOCKADDR_LEN
L R8,=F'28'
SET_SOCKADDR_LEN DS 0H
ST R8,PEERADDR_LEN
ST R8,ZERO
XC PEER_HOSTNAME,PEER_HOSTNAME
ST R8,=F'255'
ST R8,PEER_SERVICENAME,PEER_SERVICENAME
ST R8,=F'32'
CALL EZASOCKET,(SOCGNI,SOCKADDR_PEER,PEERADDR_LEN, X
PEER_HOSTNAME,PEER_HOSTNAMELEN, X
PEER_SERVICENAME,PEER_SERVICENAMELEN, X
GNI_FLAGS, X
ERRNO,RETCODE),VL,MF=(E,PARMLIST)

L R5,ERRNO
L R6,RETCODE
C R6,ZERO
BL SOCERR
MVC MSGRESULT(L'SUCC),SUCC
BAL R7,HANDLE_TCPCICS

Display the host name

MVC TDHOST(L'TDHOST),PEER_HOSTNAME
MVC MSGAREA(L'TDHOSTMSG),TDHOSTMSG
BAL R7,HANDLE_TCPCICS

Display the service name

MVC TDSERV(L'TDSERV),PEER_SERVICENAME
MVC MSGAREA(L'TDSERVMSG),TDSERVMSG
BAL R7,HANDLE_TCPCICS

Receiving data through a socket by issuing the RECVFROM command.

ACCEPT_RECEIVE DS 0H
MVI TCP_INDICATOR,C'T'
MVI TCPLENG,BUFFER_LENG
XC TCP_BUF,TCP_BUF Clear the buffer storage
*
CALL EZASOKET,(SOCRECVF,CLI_SOCKID,RCVFM_FLAG,TCPLENG, X
TCP_BUF,SOCKADDR_IN, X
ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
L R5,ERRNO Capture the ERRNO and
L R6,RETCODE the return code.
ST R6,RECVFROM_RETCODE Save the RECVFROM return code
C R6,ZERO Is the call successful?
BL RECVFROM_ERROR No!
*
* If the RECVFROM return code is zero and the number of bytes received
* is also zero, then there is nothing further to process.
*
BE CHECK_NBYTES Yes. Go check number bytes received
B RECVFROM_OK NO. Go interpret clients data
CHECK_NBYTES DS 0H
L R6,TCPLENG Check number of bytes received
C R6,ZERO Is it zero ?
BE ACCEPT_RECEIVE Yes. Go issue RECVFROM again.
B RECVFROM_OK No. Must have received something.
RECVFROM_ERROR DS 0H
MVC MSGAREA(L'RECVFROM_ERR),RECVFROM_ERR
BAL R7,HANDLE_TCPCICS 'Write to TD Queue
MVI TASK_FLAG,C'1' Force the Client connection to end
B CLOSE_CLIENT Go close clients socket
RECVFROM_OK DS 0H
*
* Interpret the clients request.
*
* Remove the following call to EZACIC05 if using an EBCDIC client.
*
CALL EZACIC05,(TCP_BUF,TCPLENG),VL,MF=(E,PARMLIST)
*
CLC TCP_BUF_H,TCP_BUF_H_LOW_VALUES Display data received
BE COMMAND_IS_LOW_VALUES from the client as blanks.
CLC TCP_BUF_H,TCP_BUF_H_SPACES Display data received from
BE COMMAND_IS_SPACES the client as blanks
CLC TCP_BUF_H,TCP_BUF_H_END End client connection?
BE SET_END Yes.
CLC TCP_BUF_H,TCP_BUF_H_TRM Terminate server?
BE SET_TERM Yes.
*
* Inform the client that the server has process the message
*
XC MSGAREA,MSGAREA
MVC MSGAREA(L'SERVER_PROC_MSG),SERVER_PROC_MSG
*
EXEC CICS SYNCPOINT
*
EXEC CICS ASKTIME ABSTIME(UTIME) NOHANDLE
EXEC CICS FORMATTIME ABSTIME(UTIME) X
DATESEP(‘/’) MMDDYY(MSGDATE) X
TIME(MSGTIME) TIMESEP(‘:’) NOHANDLE
LA R6,TCPCICS_MSG_AREA_LEN
STH R6,TDLEN
EXEC CICS WRITEQ TO QUEUE('CSMT') X
FROM(TCPCICS_MSG_AREA) X
LENGTH(TDLEN)
*
MVC TCP_BUF,TCPCICS_MSG_AREA_2
*
* Remove the following call to EZACIC04 if using an EBCDIC client.
*
CALL EZACIC04,(TCP_BUF,TCPLENG),VL,MF=(E,PARMLIST)
*
* Write the server process message back to the client
*
CALL EZASOKET,(SOCWRITE,CLI_SOCKID,TCPLENG,TCP_BUF, X
ERRNO,RETCODE),VL,MF=(E,PARMLIST)
*
L R5,ERRNO Capture the ERRNO and
L R6,RETCODE the return code.
MVC MSGCMD,SOCWRITE the API function performed.
C R6,ZERO Is the call successful?
BL TALK_CLIENT_BAD No! Go display error
MVC MSGRESULT(L'SUCC'),SUCC Move SUCCESSFUL msg to TD area
*

636 z/OS V2R1.0 Communications Server: IP CICS Sockets Guide
XC TCP_BUF,TCP_BUF
MVI TCP_INDICATOR,X'00'
B ACCEPT_RECEIVE Go receive more client data

TALK_CLIENT BAD DS 0H
MVI TASK_FLAG,C'1'
B CLOSE_CLIENT

* Process command from client
* COMMAND IS LOW VALUES DS 0H
COMMAND_IS_SPACES DS 0H
XC MSGRESULT,MSGRESULT
MVC MSGCMD,SOCRECVF
MVC MSGRESULT(37),='CLIENT COMMAND IS BLANKS OR LOWVALUES'
BAL R7,HANDLE_TCPCICS Write to TD Queue
B ACCEPT_RECEIVE Go receive more data from client

SET_END DS 0H
MVI TASK_FLAG,C'1'
B CLOSECLIENT

SET_TERM DS 0H
MVI TASK_FLAG,C'2'
B CLOSE_CLIENT

CLOSE_CLIENT DS 0H
CALL EZASOKET,(SOCCLOSE,CLI_SOCKID, X
ERRNO,RETCODE),VL,MF=(E,PARMLIST)
L R5,ERRNO Check for successful call
L R6,RETCODE Check for successful call
MVC MSGCMD,SOCCLOSE
C R6,ZERO Is it less than zero
BL SOCERR Yes, go display error and terminate
MVC MSGRESULT(L'SUCC),SUCC Move SUCCESSFUL msg to TD area
BAL R7,HANDLE_TCPCICS Write to TD Queue

* Determine whether we should select another socket
* CLI TASK_FLAG,C'2' Terminate server?
BE CLOSEDOWN Yes, go close passive socket
MVI TASK_FLAG,C'0' Reset the task flag for next client
B ACCEPT_CLIENT_REQ Go select new connection.

CLOSEDOWN DS 0H
* CLOSE SOCKET DESCRIPTOR
* SET THE SERVER SOCKET TO NOT LINGER ON THE CLOSE
* CALL EZASOKET,(SOCCLOSE,SRV_SOCKID,SOCK#SO_LINGER,ON_ZERO, X
EIGHT,ERRNO,RETCODE),VL,MF=(E,PARMLIST)
* CLOSE THE SERVER PASSIVE SOCKET
* CALL EZASOKET,(SOCCLOSE,SRV_SOCKID, X
ERRNO,RETCODE),VL,MF=(E,PARMLIST)
L R5,ERRNO Check for successful call
L R6,RETCODE Check for successful call
MVC MSGCMD,SOCCLOSE
C R6,ZERO Is it less than zero
BL SOCERR Yes, go display error and terminate
MVC MSGRESULT(L'SUCC),SUCC Move SUCCESSFUL msg to TD area
BAL R7,HANDLE_TCPCICS Write to TD Queue
CLI TERMAPI_REQUIRED_SW,C'Y' A TERMAPI needed?
BE TERM_API Yes, go issue TERMAPI
B SOCRET No, return to CICS

* Terminate IP CICS Sockets API
* TERMIN_API DS 0H
CALL EZASOKET,(SOCTERM),VL,MF=(E,PARMLIST)
MVC MSGCMD,SOCTERM
MVC MSGRESULT(L'SUCC),SUCC Move SUCCESSFUL msg to TD area
BAL R7,HANDLE_TCPCICS Write to TD Queue
B SOCRET
* Listener Started Task routine.
LISTENER_STARTED_TASK DS 0H

* Take the socket which was given by the listener.
* 
* L R8,GIVE_TAKE_SOCKET Use the socket descriptor from the
* STH R8,SOCKET_TO_TAKE TIM for the TAKESOCKET
* XC CLIENTID_LSTN,CLIENTID_LSTN Clear the clientid
* LH R8,STIM_FAMILY Get the domain from the TIM
* ST R8,CID_DOMAIN_LSTN Set the domain
* MVC CID_LSTN_INFO,CLIENTID_PARM Set the Address space and X
* subtask name.
* 
* CALL EZASOKET,(SOCTSOCK,SOCKET_TO_TAKE,CLIENTID_LSTN, X
* ERRNO,RETCODE),VL,MF=(E,PARMLIST)
* 
* L R5,ERRNO Check for successful call
* L R6,RETCODE Check for successful call
* MVC MSGCMD,SOCTSOCK Set the API name
* C R6,ZERO Is it less than zero
* BL SOCERR Yes, go display error and terminate
* STH R6,Srv_SockID Save the taken socket descriptor
* MVC MSGRESULT(L'SUCC'),SUCC Move SUCCESSFUL msg to TD area
* BAL R7,HANDLE_TCPICS Write to TD Queue
* 
* * Inform the client that the server has started.
* * 
* MVC TCPLENG,BUFFER_LENG Set the message length
* XC TCP_BUF,TCP_BUF Clear the buffer
* MVC TCP_BUF(L'STARTOK'),STARTOK Move STARTED message
* * 
* * Remove the following call to EZACIC04 if using an EBCDIC client.
* * 
* CALL EZACIC04,(TCP_BUF,TCPLENG),VL,MF=(E,PARMLIST)
* * 
* * Notify client the the child subtask has started.
* * 
* CALL EZASOKET,(SOCWRITE,SrvSockID,TCPLENG,TCP_BUF, X
* ERRNO,RETCODE),VL,MF=(E,PARMLIST)
* 
* L R5,ERRNO Capture the ERRNO and
* L R6,RETCODE the return code.
* MVC MSGCMD,SOCWRITE the API function performed.
* C R6,ZERO Is the call successful?
* BL SOCERR No! Go display error and terminate
* MVC MSGRESULT(L'SUCC'),SUCC Move SUCCESSFUL msg to TD area
* BAL R7,HANDLE_TCPICS Write to TD Queue
* 
* * Close the taken socket descriptor
* * 
* CALL EZASOKET,(SOCCLOSE,SrvSockID, X
* ERRNO,RETCODE),VL,MF=(E,PARMLIST)
* L R5,ERRNO Check for successful call
* L R6,RETCODE Check for successful call
* MVC MSGCMD,SOCCLOSE
* C R6,ZERO Is it less than zero
* BL SOCERR Yes, go display error and terminate
* MVC MSGRESULT(L'SUCC'),SUCC Move SUCCESSFUL msg to TD area
* BAL R7,HANDLE_TCPICS Write to TD Queue
* 
* * Continue with server startup
* * 
* B SOCKET_BIND_LISTEN Go continue the server startup
* * 
* * Various routines to process error conditions
* * 
* TCP_TRUE_REQ DS 0H
* 
* MVC MSGAREA(L'TCP_EXIT_MSG'),TCP_EXIT_MSG
* B SEND_ERR_MSG
* NOTAUTH_ERR DS 0H
* 
* MVC MSGAREA(L'NOTAUTH_MSG'),NOTAUTH_MSG
* B SEND_ERR_MSG
* INVREQ_ERR DS 0H
* 
* MVC MSGAREA(L'INVREQ_MSG'),INVREQ_MSG
* B SEND_ERR_MSG
* IOERR_ERR DS 0H
* 
* MVC MSGAREA(L'IOERR_MSG'),IOERR_MSG
* B SEND_ERR_MSG
* LENGERR_ERR DS 0H
* 
* MVC MSGAREA(L'LENGERR_MSG'),LENGERR_MSG

z/OS V2R1.0 Communications Server: IP CICS Sockets Guide
B SEND_ERR_MSG
NOSPACE_ERR DS OH
 MVC MSGAREA(L’NOSPACE_MSG),NOSPACE_MSG
 B SEND_ERR_MSG
QIDERR_ERR DS OH
 MVC MSGAREA(L’QIDERR_MSG),QIDERR_MSG
 B SEND_ERR_MSG
ITEMERR_ERR DS OH
 MVC MSGAREA(L’ITEMERR_MSG),ITEMERR_MSG
 B SEND_ERR_MSG
ENDDATA_ERR DS OH
 MVC MSGAREA(L’ENDDATA_MSG),ENDDATA_MSG
 B SEND_ERR_MSG
SEND_ERR_MSG DS 0H
 BAL R7,HANDLE_TCPCICS Write to TD Queue
 B SOCRET Return to CICS!

* Error on EZASOKET call

SOCERR DS OH
 MVC MSGAREA(L’MSGCMD),MSGCMD
 MVC MSGAREA+16(L’SOCKET_ERR),SOCKET_ERR
 BAL R7,HANDLE_TCPCICS Write to TD Queue

* L R6,RETCODE Pick up the RETCODE value
 L R5,ERRNO Pick up the ERRNO value
 CVD R6,DWORK Format the RETCODE
 UNPK TDRETC,DWORK+4(4) for printing to the
 OI TDRETC+6,X’F0’ TD queue

* CVD R5,DWORK Format the ERRNO
 UNPK TDERRNO,DWORK+4(4) for printing to the
 OI TDERRNO+6,X’F0’ TD queue

* MVC MSGAREA(L’TDTEXT5),TDTEXT5 Move the RETCODE and ERRNO X
 to the TD queue area
 BAL R7,HANDLE_TCPCICS Write the message to the TD queue

* B SOCRET Return to CICS

* Write a message to the "CSMT" destination queue for logging
HANDLE_TCPCICS DS OH
 EXEC CICS ASKTIME ABSTIME(UTIME) NOHANDLE
 EXEC CICS FORMATTIME ABSTIME(UTIME) X
 DATESEP(’/’) MMDDYY(MSGDATE) X
 TIME(MSGTIME) TIMESEP(’’) NOHANDLE
 LA R6,TCPCICS_MSG_AREA_LEN
 STH R6,TDLEN
 EXEC CICS WRITEQ TD QUEUE(’CSMT’) X
 FROM(TCPCICS_MSG_AREA) X
 B LENGTH(TDLEN)

* Tell the client?

* CLI TCP_INDICATOR,C’T’
 BNE HANDLE_TCPCICS_RETURN
 MVC TCPLENG,BUFFER_LEN
 XC TCP_BUF,TCP_BUF
 MVC TCP_BUF,TCPCICS_MSG_AREA_2

* Remove the following call to EZACIC04 if using an EBCDIC client.
* CALL EZACIC04,(TCP_BUF,TCPLENG),VL,MF=(E,PARMLIST)
 MVI TCP_INDICATOR,C’ ’

* Notify client the the child subtask has started.
* CALL EZASOKET,(SOCWRITE,CLI_SOCKID,TCPLENG,TCP_BUF, X
 ERNO,RETCODE),VL,MF=(E,PARMLIST)

* L R5,ERRNO Capture the ERRNO and
 L R6,RETCODE the return code.
 MVC MSGCMD,SOCWRITE the API function performed.
 C R6,ZERO Is the call successful?
 BL HANDLE_TCPCICS_RETURN
 MVC MSGRESULT(L’SUCC),SUCC Move SUCCESSFUL msg to TD area
 EXEC CICS ASKTIME ABSTIME(UTIME) NOHANDLE
EXEC CICS FORMATTIME ABSTIME(UTIME) X
DATESEP('/') MMDDYY(MSGDATE) X
TIME(MSGTIME) TIMESEP(':') NOHANDLE
LA R6,TCPCICS_MSG_AREA_LEN
STH R6,TDLEN
EXEC CICS WRITEQ TO QUEUE('CSMT') X
FROM(TCPCICS_MSG_AREA) X
LENGTH(TDLEN)
*
HANDLE_TCPCICS_RETURN DS OH
XC MSGAREA,MSGAREA
BR R7 Return to caller
*
* ALL DONE.
*
SOCRET DS OH
MVC MSGAREA(L'STOPOK'),STOPOK Move STOPPED msg to TD area
BAL R7,HANDLE_TCPCICS Write to TD Queue
EXEC CICS RETURN
*
* INITAPI parameters
*
MAXSOC DC H'0' MAXSOC value, use the default
IDENT DC GCL16bp'
TCPNAME DC CL8'TCPCS ' Name of the TCP
APPLID DC CL8'CICS ' Address space name
INIT_SUBTASKID DS OCL8 Subtask for INITAPI
SUBTASKNO DC CL7 ' from EIBTASKN
SUBT_CHAR DC CL1'L' Make server use a non-reusable subtask
MAXSNO DC F'0' Highest socket descriptor available
*
* Sockets address family
*
AFINET DC F'2' AF_INET
AFINET6 DC F'19' AF_INET6
*
* SOCKET FUNCTIONS
*
SOCACCT DC CL16'ACCEPT '
SOCBIND DC CL16'BIND '
SOCCLOSE DC CL16'CLOSE '
SOCCONNT DC CL16'CONNECT '
SOCFCNTL DC CL16'FCNTL '
SOCFAI DC CL16'FREEADDRINFO '
SOGCCID DC CL16'GETCLIENTID '
SOGDAI DC CL16'GETNAMEINFO '
SOGCNDC DC CL16'GETHOSTID '
SOGTHN DC CL16'GETHOSTNAME '
SOGPEER DC CL16'GETPEERNAME '
SOGTSN DC CL16'GETSOCKNAME '
SOGTSO DC CL16'GETSOCKOPT '
SOGSOCK DC CL16'GIVESOCKET '
SOCINIT DC CL16'INITAPI '
SOCIOCTL DC CL16'IOCTL '
SOCLISTN DC CL16'LISTEN '
SOCNTOP DC CL16'NTOP '
SOCPTON DC CL16'PTON '
SOCREAD DC CL16'READ '
SOCREADV DC CL16'READV '
SOCRECVD DC CL16'RECV '
SOCRECVF DC CL16'RECVFROM '
SOCRECVM DC CL16'RECVMSG '
SOCSECT DC CL16'SELECT '
SOCSELX DC CL16'SELECTEX '
SOCSEND DC CL16'SEND '
SOCSNOM DC CL16'SENDMSG '
SOCSENDT DC CL16'SENDTO '
SOCSETO DC CL16'SETSOCKOPT '
SOCWRITE DC CL16'WRITE '
SOCWRITEV DC CL16'WRITEV '
*
* SELECT parms
*
NUM_FDS DC F'5' Number of file descriptors
NFDS DS F

640 z/OS V2R1.0 Communications Server: IP CICS Sockets Guide
TIMEVAL DC AL4(180),AL4(0)
SELECT_CSOCKET DS 0CL12
READMASK DC XL4'00' SELECT read mask
DUMYMASK DC XL4'00' mask set to binary zeros
REPLY_RDMASK DC XL4'00' SELECT reply read mask
REPLY_RDMASK_FF DS XL4
SELECT_RETCODE DS F Sum of all ready sockets in masks
*
TCPLENG DC F'0'
*
SSTREAM DC F'1' socket type stream
ZERO DC F'0'
ONE DC F'1'
TWO DC F'2'
SIX DC F'6'
EIGHT DC F'8'
TEN DC F'10'
*
* Data for RETRIEVE
* TRANS DS CL4 Transaction retrieved
LENG DS H Length of data retrieved
CG_LEN DC F'5' Length of Port from CICS Start
TAKE_SOCKET SWITCH DC X'00' Used to drive a TAKE_SOCKET
TCP_INDICATOR DC CL1 ' ' Server task flag
*
TCP_BUF DS 0CL55 Buffer
TCP_BUF_H DC CL3 ' ' Used to pass the server commands
TCP_BUF_DATA DC CL52 ' ' Command to end the client connection
TCP_BUF_H_END DC CL3 'END' Command to terminate the server
TCP_BUF_H_LOW_VALUES DC XL3 '000000' Client sent command=low values
TCP_BUF_H_SPACES DC CL3 ' ' Client sent command=spaces
TCP_BUF_H_TRM DC CL3 'TRM' Command to terminate the server
BUFFER_LENG DC F'55' Length of buffer
*
* LISTEN parms
* BACKLOG DC F'0' Backlog for LISTEN
*
* RECVFROM parms
*
RCVFMT_FLAG DC F'0' RECVFROM flag
*
* MESSAGE(S) WRITTEN TO TRANSIENT DATA QUEUE
*
BITMASK_ERR DC CL36 'BITMASK CONVERSION - FAILED'
LISTEN_SUCC DS 0CL46
DC CL34 'READY TO ACCEPT REQUESTS ON PORT: '
BIND_PORT DC CL5 ' '
DC CL7 ' '
ENDDATA_MSG DC CL30 'RETRIEVE DATA CAN NOT BE FOUND'
IOERR_MSG DC CL12 'IOERR OCCURS'
ITEMERR_MSG DC CL13 'ITEMERR ERROR'
LENERR_MSG DC CL13 'LENERR ERROR'
NOSPACE_MSG DC CL17 'NOSPACE CONDITION'
RECVFROM_ERR DC CL36 'RECVFROM SOCKET CALL FAILED'
QIDERR_MSG DC CL30 'TRANSIENT DATA QUEUE NOT FOUND'
SERVER_PROC_MSG DC CL55 'SERVER PROCESSED MESSAGE'
SERVER_PROC_MSG_DC 0CL55 'SERVER PROCESSED MESSAGE'
SOCKET_ERR DC CL15 'EZA SOCKET ERROR!'
STARTOK DC CL27 'SERVER STARTED SUCCESSFULLY'
STOPOK DC CL27 'SERVER STOPPED SUCCESSFULLY'
TCP_EXIT_MSG DC CL31 'SERVER STOPPED: TRUE NOT ACTIVE'
NOTAUTH_MSG DC CL31 'SERVER STOPPED: NOT AUTHORIZED'
*
* Message to display the clients host name
*
TDHOSTMSG DS 0CL55
TDHOSTLIT DC CL9 'HOSTNAME='
TDHOST DC CL46 ' '
*
* Message to display the clients service name
*
TDSERVMSG DS 0CL55
TDSERVLIT DC CL8 'SERVICE='
TDSERV DC CL32 '\'
DC CL15 ' '
*
* Message to display EZASOCKET RETCODE and ERRNO
* TDTEXT5 DS CL40
  DC CL10 RETCODE = '
  TDRETC DC CL7' ' Printable RETCODE
  DC CL3'
  DC CL9'ERNO = '
  TDERRNO DC CL7' ' Printable ERRNO
  DC CL4'

* Misc
* SUCDC DC CL10'SUCCESSFUL'
NOTSUC DC CL14'NOT SUCCESSFUL'
TERMREQDC Required SW DC CL1'N'
ON_ZERO DS 0C
LINGERON DC F'1' On/Off
LINGERTIME DC F'0' Linger time

* DSECTs
  EZACICA TYPE=DSECT,AREA=GWA
  EZACICA TYPE=DSECT,AREA=TIE

DFHEISTG
SRV6SAVE DS 18F Register Save Area
SRV6STRSV DS F Save area for start subroutine

* Socket address structure
* CNOP 0,8 DOUBLEWORD BOUNDARY
  SOCCADDR_IN DS 0F Socket address structure
  S Mẹ_FAM_PIY DS H Address Family
  S㏌.SOCK_DAT_DS OC Protocol specific area
  ORG Sзависим SOCK_DATA Start of AF_INET unique area
  S依赖 SOCK_PORT DS H Port number
  S依赖 SOCK_ADDR DS CL4 IPv4 address
  ORG S зависим SOCK_DATA Start of AF_INET6 area
  S依赖 SOCK_SIN6 DS 0C
  S依赖 SOCK_SIN6_PORT DS H Port number
  S依赖 SOCK_SIN6_ADDR DS CL16 IPv6 address
  S依赖 SOCK_SIN6_SCOPE_ID DS CL4 Scope id

* Peers address structure
* CNOP 0,8 DOUBLEWORD BOUNDARY
  SOCCADDR_PEER DS 0F Socket address structure
  PEER.SOCK_FAMILIY DS H Address Family
  PEER.SOCK_DATA DS OC Protocol specific area
  ORG PEER.SOCK_DATA Start of AF_INET unique area
  PEER.SOCK_PORT DS OC
  PEER.SOCK_ADDR DS CL4 IPv4 address
  ORG PEER.SOCK_DATA Start of AF_INET6 area
  PEER.SOCK_SIN6 DS 0C
  PEER.SOCK_SIN6_PORT DS H Port number
  PEER.SOCK_SIN6_ADDR DS CL16 IPv6 address
  PEER.SOCK_SIN6_SCOPE_ID DS CL4 Scope id

* PEERADDR_LEN DS F Length of Peers sockaddr
* PEER_HOSTNAME DS CL255 Peers Host name
  PEER_HOSTNAMELEN DS F Peers Host name length
  PEER_SERVICENAME DS CL32 Peers Service name
  PEER_SERVICENAMELEN DS F Peers Service name length

* Receive Flag
* GNI_FLAGS DS F GETNAMEINFO flags

* User supplied port to listen on
* PORT DS H User supplied port
* Storage used to create a message to be written to the CSMT TD Queue

TCPCICS_MSG_AREA DS 0F       TD Message area
TCPCICS_MSG_AREA_1 DS 0C
MSGDATE DS CL8                MM/DD/YY
MSGFILR1 DS CL2
MSGTIME DS CL8                HH:MM:SS
MSGFILR2 DS CL2
MODULE DS CL10                "EZACICAS: ">
TCPCICS_MSG_AREA_2 DS 0C
MSGAREA DS CL55
ORG MSGAREA
MSGCMD DS CL16                EZASOKET command issued
MSGRESULT DS CL39             Outcome of the command issued
TCPCICS_MSG_AREA_END EQU *   End of message
TCPCICS_MSG_AREA_LEN EQU TCPCICS_MSG_AREA_END-TCPCICS_MSG_AREA   Length of TD message text

TDLEN DS H                    Length of TD message text

* Various other working storage areas

 UTIME DS PL8                  ABSTIME data area
DWORK DS D                    Double word work area
UNPKWRK DS CL15               Unpack work area
PARMLIST DS 20F

* Error numbers and return codes

ERRNO DS F                    ERRNO
RETCODE DS F                  Return Code
RECVFROM_RETCODE DS F

* Client ID from Listener to be used by the TAKESOKET command

CLIENTID_LSTN DS OCL40        Domain
CID_DOMAIN_LSTN DS F
CID_LSTN_INFO DS OCL16        Address space name
CID_NAME_LSTN DS CL8
CID_SUBNAM_LSTN DS CL8        Subtask name
CID_RES_LSTN DS CL20

SOCKET_TO_TAKE DS H           Socket descriptor to take

* Data from the CICS RECIEVE command

TRMNL_LEN DS H                Length of data RECEIVE'd
TRMNL_MAXLEN DS H

* Data from the CICS RETRIEVE command

RETRIEVE_LEN DS H            Length of data RETRIEVE'd

* Socket descriptors

SRV_SOCKID DS H               Server socket descriptor
CLI_SOCKID DS H               Client socket descriptor

* For saving RB

SAVERB DS F

* Server data

CNOP 0,8                       DOUBLEWORD BOUNDARY
TCP_INPUT_DATA DS CL85        Data retrieved
ORG TCP_INPUT_DATA

* The Listeners Task Input Message (TIM)

TCPSOCKET_PARM DS 0C
GIVE_TAKE_SOCKET DS F
CLIENTID_PARM DS OCL16
LSTN_NAME DS CL8
LSTN_SUBNAME DS CL8
CLIENT_IN_DATA DS CL35
DS CLI
SOCKADDR_TIM DS 0F

Appendix E. Sample programs  643
SELECTEX

The following sample displays COBOL code issuing the SELECTEX socket call:

This is sample COBOL code issuing the SELECTEX socket call:

*------------------------------------------------------------------*
* Here is a annotated SAMPLE code from a test tool used to test      *
* the SELECTEX:                                                   *
*------------------------------------------------------------------*

WORKING-STORAGE SECTION.

01 SELECT-BITMASK PIC 9(16) BINARY VALUE 0.
01 SELECT-BITMASK-LEN PIC 9(8) BINARY VALUE 0.
01 SELECT-CHAR-STRING PIC X(64).
01 SELECT-MAXSOC PIC 9(8) BINARY VALUE 0.
01 SELECT-TIMEOUT.
  03 SELECT-TIMEOUT-SECONDS PIC S9(8) BINARY VALUE 0.
  03 SELECT-TIMEOUT-MICROSEC PIC S9(8) BINARY VALUE 0.
01 SELECT-RSNDMSK PIC 9(16) BINARY.
01 SELECT-WSNDMSK PIC 9(16) BINARY.
01 SELECT-ESNDSMK PIC 9(16) BINARY.
01 SELECT-RRETMSK PIC 9(16) BINARY.
01 SELECT-WRETMSK PIC 9(16) BINARY.
01 SELECT-ERETMSK PIC 9(16) BINARY.
77 SELECT-ECB-PTR USAGE IS POINTER.

LINKAGE SECTION.

01 SELECT-ECB PIC 9(8) BINARY.

PROCEDURE DIVISION USING L1.

PROCESS-SELECTEX.

* GET SHARED STORAGE FOR ECB.

  EXEC CICS GETMAIN SHARED
  SET (SELECT-ECB-PTR)
  FLENGTH (4)
INITIMG ('00')
END-EXEC.
SET ADDRESS OF SELECT-ECB TO SELECT-ECB-PTR.
INITIALIZE SELECT-ECB.

* WRITE ECB ADDRESS TO TS QUEUE

* EXEC CICS WRITEQ TS QUEUE ('POSTECB@') FROM (SELECT-ECB-PTR) LENGTH (4) END-EXEC.

* SOCKET CALL SELECTEX

MOVE 10 TO SELECT-MAXSOC.
MOVE -1 TO SELECT-TIMEOUT-SECONDS.
MOVE -1 TO SELECT-TIMEOUT-MICROSEC.

MOVE read-send-mask TO SELECT-CHAR-STRING.
MOVE 64 TO SELECT-BITMASK-LEN.
CALL 'EZACIC06' USING CTOB
SELECT-BITMASK
SELECT-CHAR-STRING
SELECT-BITMASK-LEN
RETCODE.
MOVE SELECT-BITMASK TO SELECT-RSNDMSK.

MOVE write-send-mask TO SELECT-CHAR-STRING.
MOVE 64 TO SELECT-BITMASK-LEN.
CALL 'EZACIC06' USING CTOB
SELECT-BITMASK
SELECT-CHAR-STRING
SELECT-BITMASK-LEN
RETCODE.
MOVE SELECT-BITMASK TO SELECT-WSNDMSK.

MOVE exception-send-mask TO SELECT-CHAR-STRING.
MOVE 64 TO SELECT-BITMASK-LEN.
CALL 'EZACIC06' USING CTOB
SELECT-BITMASK
SELECT-CHAR-STRING
SELECT-BITMASK-LEN
RETCODE.
MOVE SELECT-BITMASK TO SELECT-ESNDMSK.

CALL 'EZASOKET' USING SOKET-SELECTEX
SELECT-MAXSOC
SELECT-TIMEOUT
SELECT-RSNDMSK
SELECT-WSNDMSK
SELECT-ESNDMSK
SELECT-RRETMHGK
SELECT-WRETMHGK
SELECT-ERETMSK
SELECT-ECB
ERRNO
RETCODE.

* FREE THE STORAGE FOR THE ECB

* EXEC CICS FREEMAIN
DATAPOINTER(SELECT-ECB-PTR)
EXEC CICS DELETEQ TS
 QUEUE ('POSTECB@')
END-EXEC.

IF RETCODE = 0 THEN
 MOVE 'SELECTEX FAILED' TO MSG1
ELSE
 MOVE 'SELECTEX PROCESSED' TO MSG1.

MOVE SELECT-RRETMSK TO SELECT-BITMASK.
CALL 'EZACIC06' USING BTOC
 SELECT-BITMASK
 SELECT-CHAR-STRING
 SELECT-BITMASK-LEN
 RETCODE.
MOVE SELECT-CHAR-STRING TO read-returned-mask.

MOVE SELECT-WRETMSK TO SELECT-BITMASK.
CALL 'EZACIC06' USING BTOC
 SELECT-BITMASK
 SELECT-CHAR-STRING
 SELECT-BITMASK-LEN
 RETCODE.
MOVE SELECT-CHAR-STRING TO write-returned-mask.

MOVE SELECT-ERETMSK TO SELECT-BITMASK.
CALL 'EZACIC06' USING BTOC
 SELECT-BITMASK
 SELECT-CHAR-STRING
 SELECT-BITMASK-LEN
 RETCODE.
MOVE SELECT-CHAR-STRING TO exception-returned-mask.

PROCESS-SELECTEX-EXIT.
EXIT.

*------------------------------------------------------------------*
* Here is the annotated SAMPLE code from a test tool used to *
* call the subroutine used to post the ECB: *
*------------------------------------------------------------------*

WORKING-STORAGE SECTION.
01 POST-ECB-ADDRESS PIC 9(8) BINARY.
01 POST-ECB-LEN PIC 9(4) BINAY.

PROCEDURE DIVISION USING L1.

PROCESS-POSTECB.

* * LOOK FOR THE ADDRESS OF THE ECB IN TEMP STORAGE *
* * MOVE 4 TO POST-ECB-LEN. *
EXEC CICS READQ TS
 ITEM (1)
 QUEUE ('POSTECB@')
 INTO (POST-ECB-ADDRESS)
 LENGTH (POST-ECB-LEN)
END-EXEC.

CALL 'POSTECB' USING POST-ECB-ADDRESS
 RETCODE.
IF RETCODE < 0 THEN
    MOVE 'POSTECB FAILED'
    TO MSG1
ELSE
    MOVE 'POSTECB PROCESSED'
    TO MSG.
PROCESS-POSTECB-EXIT.
EXIT.

*------------------------------------------------------------------*
* Here is a sample assembler program that can be used to post the *
* SELECTEX ECB: *
*------------------------------------------------------------------*

TITLE 'POSTECB'
POSTECB CSECT ,
POSTECB AMODE ANY
POSTECB RMODE ANY
POSTECB MODID EYECATCHER INFO
SAVE (14,12)
LR R9,R15
DROP R15
USING POSTECB,R15 USE ENTRY REGISTER AS BASE
POSTECB MODID EYECATCHER INFO
SAVE (14,12)
LR R9,R15
DROP R15
USING POSTECB,R9 USE R90 AS BASE REGISTER
L R12,0(R1)
L R10,0(0,R12)
L R12,0(0,R12)
L R11,NEWECB
TM 0(R12),X'80'
BO POST0100 IF YES, ISSUE POST MACRO
CS R10,R11,0(R12) IF NO, TRY QUICK POST
BC 4,POST0100 IF UNSUCCESSFUL, ISSUE POST MACRO
B POST9999 RETURN TO CALLER

POST0100 DS 0H
POST (R12),255
POST9999 DS 0H
RETURN (14,12) RETURN TO CALLER
ECBADDR DS F
NEWECB DC X'400000FF' ECB WITH POST BIT ON AND CC=255
LTORG
YREGS
END
Appendix F. Related protocol specifications

This appendix lists the related protocol specifications (RFCs) for TCP/IP. The Internet Protocol suite is still evolving through requests for comments (RFC). New protocols are being designed and implemented by researchers and are brought to the attention of the Internet community in the form of RFCs. Some of these protocols are so useful that they become recommended protocols. That is, all future implementations for TCP/IP are recommended to implement these particular functions or protocols. These become the de facto standards, on which the TCP/IP protocol suite is built.

You can request RFCs through electronic mail, from the automated Network Information Center (NIC) mail server, by sending a message to service@nic.ddn.mil with a subject line of RFC nnnn for text versions or a subject line of RFC nnnn.PS for PostScript versions. To request a copy of the RFC index, send a message with a subject line of RFC INDEX.

For more information, contact nic@nic.ddn.mil or at:

Government Systems, Inc.
Attn: Network Information Center
14200 Park Meadow Drive
Suite 200
Chantilly, VA 22021

Hard copies of all RFCs are available from the NIC, either individually or by subscription. Online copies are available at the following Web address:

http://www.rfc-editor.org/rfc.html

Draft RFCs that have been implemented in this and previous Communications Server releases are listed at the end of this topic.

Many features of TCP/IP Services are based on the following RFCs:

<table>
<thead>
<tr>
<th>RFC</th>
<th>Title and Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFC 652</td>
<td>Telnet output carriage-return disposition option D. Crocker</td>
</tr>
<tr>
<td>RFC 653</td>
<td>Telnet output horizontal tabstops option D. Crocker</td>
</tr>
<tr>
<td>RFC 654</td>
<td>Telnet output horizontal tab disposition option D. Crocker</td>
</tr>
<tr>
<td>RFC 655</td>
<td>Telnet output formfeed disposition option D. Crocker</td>
</tr>
<tr>
<td>RFC 657</td>
<td>Telnet output vertical tab disposition option D. Crocker</td>
</tr>
<tr>
<td>RFC 658</td>
<td>Telnet output linefeed disposition D. Crocker</td>
</tr>
<tr>
<td>RFC 698</td>
<td>Telnet extended ASCII option T. Mock</td>
</tr>
</tbody>
</table>

© Copyright IBM Corp. 2000, 2013
RFC 726  
Remote Controlled Transmission and Echoing Telnet option J. Postel, D. Crocker

RFC 727  
Telnet logout option M.R. Crispin

RFC 732  
Telnet Data Entry Terminal option J.D. Day

RFC 733  
Standard for the format of ARPA network text messages D. Crocker, J. Vittal, K.T. Pogran, D.A. Henderson

RFC 734  
SUPDUP Protocol M.R. Crispin

RFC 735  
Revised Telnet byte macro option D. Crocker, R.H. Gumpertz

RFC 736  
Telnet SUPDUP option M.R. Crispin

RFC 749  
Telnet SUPDUP—Output option B. Greenberg

RFC 765  
File Transfer Protocol specification J. Postel

RFC 768  
User Datagram Protocol J. Postel

RFC 779  
Telnet send-location option E. Killian

RFC 783  
TFTP Protocol (revision 2) K.R. Sollins

RFC 791  
Internet Protocol J. Postel

RFC 792  
Internet Control Message Protocol J. Postel

RFC 793  
Transmission Control Protocol J. Postel

RFC 820  
Assigned numbers J. Postel

RFC 821  
Simple Mail Transfer Protocol J. Postel

RFC 822  
Standard for the format of ARPA Internet text messages D. Crocker

RFC 823  
DARPA Internet gateway R. Hinden, A. Sheltzer

RFC 826  
Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48-bit Ethernet address for transmission on Ethernet hardware D. Plummer

RFC 854  
Telnet Protocol Specification J. Postel, J. Reynolds
RFC 855
   Telnet Option Specification J. Postel, J. Reynolds
RFC 856
   Telnet Binary Transmission J. Postel, J. Reynolds
RFC 857
   Telnet Echo Option J. Postel, J. Reynolds
RFC 858
   Telnet Suppress Go Ahead Option J. Postel, J. Reynolds
RFC 859
   Telnet Status Option J. Postel, J. Reynolds
RFC 860
   Telnet Timing Mark Option J. Postel, J. Reynolds
RFC 861
   Telnet Extended Options: List Option J. Postel, J. Reynolds
RFC 862
   Echo Protocol J. Postel
RFC 863
   Discard Protocol J. Postel
RFC 864
   Character Generator Protocol J. Postel
RFC 865
   Quote of the Day Protocol J. Postel
RFC 868
   Time Protocol J. Postel, K. Harrenstien
RFC 877
   Standard for the transmission of IP datagrams over public data networks J.T. Korb
RFC 883
   Domain names: Implementation specification P.V. Mockapetris
RFC 884
   Telnet terminal type option M. Solomon, E. Wimmers
RFC 885
   Telnet end of record option J. Postel
RFC 894
   Standard for the transmission of IP datagrams over Ethernet networks C. Hornig
RFC 896
   Congestion control in IP/TCP internetworks J. Nagle
RFC 903
   Reverse Address Resolution Protocol R. Finlayson, T. Mann, J. Mogul, M. Theimer
RFC 904
   Exterior Gateway Protocol formal specification D. Mills
RFC 919
   Broadcasting Internet Datagrams J. Mogul
RFC 922
Broadcasting Internet datagrams in the presence of subnets J. Mogul

RFC 927
TACACS user identification Telnet option B.A. Anderson

RFC 933
Output marking Telnet option S. Silverman

RFC 946
Telnet terminal location number option R. Nedved

RFC 950
Internet Standard Subnetting Procedure J. Mogul, J. Postel

RFC 952
DoD Internet host table specification K. Harrenstien, M. Stahl, E. Feinler

RFC 959
File Transfer Protocol J. Postel, J.K. Reynolds

RFC 961
Official ARPA-Internet protocols J.K. Reynolds, J. Postel

RFC 974
Mail routing and the domain system C. Partridge

RFC 1001

RFC 1002

RFC 1006
ISO transport services on top of the TCP: Version 3 M.T. Rose, D.E. Cass

RFC 1009
Requirements for Internet gateways R. Braden, J. Postel

RFC 1011
Official Internet protocols J. Reynolds, J. Postel

RFC 1013
X Window System Protocol, version 11: Alpha update April 1987 R. Scheifler

RFC 1014
XDR: External Data Representation standard Sun Microsystems

RFC 1027
Using ARP to implement transparent subnet gateways S. Carl-Mitchell, J. Quarterman

RFC 1032
Domain administrators guide M. Stahl

RFC 1033
Domain administrators operations guide M. Lottor

RFC 1034
Domain names—concepts and facilities P.V. Mockapetris
RFC 1035  
Domain names—implementation and specification P.V. Mockapetris

RFC 1038  
Draft revised IP security option M. St. Johns

RFC 1041  
Telnet 3270 regime option Y. Rekhter

RFC 1042  
Standard for the transmission of IP datagrams over IEEE 802 networks J. Postel, J. Reynolds

RFC 1043  
Telnet Data Entry Terminal option: DODIIS implementation A. Yasuda, T. Thompson

RFC 1044  
Internet Protocol on Network System’s HYPERchannel: Protocol specification K. Hardwick, J. Lekashman

RFC 1053  
Telnet X.3 PAD option S. Levy, T. Jacobson

RFC 1055  
Nonstandard for transmission of IP datagrams over serial lines: SLIP J. Romkey

RFC 1057  

RFC 1058  
Routing Information Protocol C. Hedrick

RFC 1060  
Assigned numbers J. Reynolds, J. Postel

RFC 1067  

RFC 1071  
Computing the Internet checksum R.T. Braden, D.A. Borman, C. Partridge

RFC 1072  
TCP extensions for long-delay paths V. Jacobson, R.T. Braden

RFC 1073  
Telnet window size option D. Waitzman

RFC 1079  
Telnet terminal speed option C. Hedrick

RFC 1085  
ISO presentation services on top of TCP/IP based internets M.T. Rose

RFC 1091  
Telnet terminal-type option J. VanBokkelen

RFC 1094  
NFS: Network File System Protocol specification Sun Microsystems

RFC 1096  
Telnet X display location option G. Marcy

RFC 1101  
DNS encoding of network names and other types P. Mockapetris
RFC 1112
Host extensions for IP multicasting S.E. Deering

RFC 1113
Privacy enhancement for Internet electronic mail: Part I — message encipherment and authentication procedures J. Linn

RFC 1118
Hitchhikers Guide to the Internet E. Krol

RFC 1122
Requirements for Internet Hosts—Communication Layers R. Braden, Ed.

RFC 1123
Requirements for Internet Hosts—Application and Support R. Braden, Ed.

RFC 1146
TCP alternate checksum options J. Zweig, C. Partridge

RFC 1155
Structure and identification of management information for TCP/IP-based internets M. Rose, K. McCloghrie

RFC 1156
Management Information Base for network management of TCP/IP-based internets K. McCloghrie, M. Rose

RFC 1157

RFC 1158
Management Information Base for network management of TCP/IP-based internets: MIB-II M. Rose

RFC 1166
Internet numbers S. Kirkpatrick, M.K. Stahl, M. Recker

RFC 1179
Line printer daemon protocol L. McLaughlin

RFC 1180
TCP/IP tutorial T. Socolofsky, C. Kale

RFC 1183
New DNS RR Definitions C.F. Everhart, L.A. Mamakos, R. Ullmann, P.V. Mockapetris

RFC 1184
Telnet Linemode Option D. Borman

RFC 1186
MD4 Message Digest Algorithm R.L. Rivest

RFC 1187
Bulk Table Retrieval with the SNMP M. Rose, K. McCloghrie, J. Davin

RFC 1188
Proposed Standard for the Transmission of IP Datagrams over FDDI Networks D. Katz

RFC 1190
Experimental Internet Stream Protocol: Version 2 (ST-II) C. Topolcic
RFC 1191
Path MTU discovery J. Mogul, S. Deering

RFC 1198
FYI on the X window system R. Scheifler

RFC 1207
FYI on Questions and Answers: Answers to commonly asked “experienced Internet user” questions G. Malkin, A. Marine, J. Reynolds

RFC 1208
Glossary of networking terms O. Jacobsen, D. Lynch

RFC 1213
Management Information Base for Network Management of TCP/IP-based internets: MIB-II K. McCloghrie, M.T. Rose

RFC 1215
Convention for defining traps for use with the SNMP M. Rose

RFC 1227
SNMP MUX protocol and MIB M.T. Rose

RFC 1228
SNMP-DPI: Simple Network Management Protocol Distributed Program Interface G. Carpenter, B. Wijnen

RFC 1229
Extensions to the generic-interface MIB K. McCloghrie

RFC 1230
IEEE 802.4 Token Bus MIB K. McCloghrie, R. Fox

RFC 1231
IEEE 802.5 Token Ring MIB K. McCloghrie, R. Fox, E. Decker

RFC 1236
IP to X.121 address mapping for DDN L. Morales, P. Hasse

RFC 1256
ICMP Router Discovery Messages S. Deering, Ed.

RFC 1267
Border Gateway Protocol 3 (BGP-3) K. Lougheed, Y. Rekhter

RFC 1268
Application of the Border Gateway Protocol in the Internet Y. Rekhter, P. Gross

RFC 1269
Definitions of Managed Objects for the Border Gateway Protocol: Version 3 S. Willis, J. Burruss

RFC 1270
SNMP Communications Services F. Kastenholz, ed.

RFC 1285
FDDI Management Information Base J. Case

RFC 1315
Management Information Base for Frame Relay DTEs C. Brown, F. Baker, C. Carvalho

RFC 1321
The MD5 Message-Digest Algorithm R. Rivest

Appendix F. Related protocol specifications 655
<table>
<thead>
<tr>
<th>RFC</th>
<th>Title</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1323</td>
<td>TCP Extensions for High Performance</td>
<td>V. Jacobson, R. Braden, D. Borman</td>
</tr>
<tr>
<td>1325</td>
<td>FYI on Questions and Answers: Answers to Commonly Asked “New Internet User” Questions</td>
<td>G. Malkin, A. Marine</td>
</tr>
<tr>
<td>1327</td>
<td>Mapping between X.400 (1988)/ISO 10021 and RFC 822</td>
<td>S. Hardcastle-Kille</td>
</tr>
<tr>
<td>1340</td>
<td>Assigned Numbers</td>
<td>J. Reynolds, J. Postel</td>
</tr>
<tr>
<td>1344</td>
<td>Implications of MIME for Internet Mail Gateways</td>
<td>N. Bornstein</td>
</tr>
<tr>
<td>1349</td>
<td>Type of Service in the Internet Protocol Suite</td>
<td>P. Almquist</td>
</tr>
<tr>
<td>1350</td>
<td>The TFTP Protocol (Revision 2)</td>
<td>K.R. Sollins</td>
</tr>
<tr>
<td>1351</td>
<td>SNMP Administrative Model</td>
<td>J. Davin, J. Galvin, K. McCloghrie</td>
</tr>
<tr>
<td>1352</td>
<td>SNMP Security Protocols</td>
<td>J. Galvin, K. McCloghrie, J. Davin</td>
</tr>
<tr>
<td>1353</td>
<td>Definitions of Managed Objects for Administration of SNMP Parties</td>
<td>K. McCloghrie, J. Davin, J. Galvin</td>
</tr>
<tr>
<td>1354</td>
<td>IP Forwarding Table MIB</td>
<td>F. Baker</td>
</tr>
<tr>
<td>1356</td>
<td>Multiprotocol Interconnect on X.25 and ISDN in the Packet Mode</td>
<td>A. Malis, D. Robinson, R. Ullmann</td>
</tr>
<tr>
<td>1358</td>
<td>Charter of the Internet Architecture Board (IAB)</td>
<td>L. Chapin</td>
</tr>
<tr>
<td>1363</td>
<td>A Proposed Flow Specification</td>
<td>C. Partridge</td>
</tr>
<tr>
<td>1368</td>
<td>Definition of Managed Objects for IEEE 802.3 Repeater Devices</td>
<td>D. McMaster, K. McCloghrie</td>
</tr>
<tr>
<td>1372</td>
<td>Telnet Remote Flow Control Option</td>
<td>C. L. Hedrick, D. Borman</td>
</tr>
<tr>
<td>1374</td>
<td>IP and ARP on HIPPI</td>
<td>J. Renwick, A. Nicholson</td>
</tr>
<tr>
<td>1381</td>
<td>SNMP MIB Extension for X.25 LAPB</td>
<td>D. Throop, F. Baker</td>
</tr>
<tr>
<td>1382</td>
<td>SNMP MIB Extension for the X.25 Packet Layer</td>
<td>D. Throop</td>
</tr>
<tr>
<td>1387</td>
<td>RIP Version 2 Protocol Analysis</td>
<td>G. Malkin</td>
</tr>
<tr>
<td>1388</td>
<td>RIP Version 2 Carrying Additional Information</td>
<td>G. Malkin</td>
</tr>
</tbody>
</table>
RFC 1389
RIP Version 2 MIB Extensions G. Malkin, F. Baker

RFC 1390
Transmission of IP and ARP over FDDI Networks D. Katz

RFC 1393
Traceroute Using an IP Option G. Malkin

RFC 1398
Definitions of Managed Objects for the Ethernet-Like Interface Types F. Kastenholz

RFC 1408
Telnet Environment Option D. Borman, Ed.

RFC 1413
Identification Protocol M. St. Johns

RFC 1416
Telnet Authentication Option D. Borman, ed.

RFC 1420
SNMP over IPX S. Bostock

RFC 1428
Transition of Internet Mail from Just-Send-8 to 8bit-SMTP/MIME G. Vaudreuil

RFC 1442

RFC 1443

RFC 1445
Administrative Model for version 2 of the Simple Network Management Protocol (SNMPv2) J. Galvin, K. McCloghrie

RFC 1447
Party MIB for version 2 of the Simple Network Management Protocol (SNMPv2) K. McCloghrie, J. Galvin

RFC 1448

RFC 1464
Using the Domain Name System to Store Arbitrary String Attributes R. Rosenbaum

RFC 1469
IP Multicast over Token-Ring Local Area Networks T. Pusateri

RFC 1483
Multiprotocol Encapsulation over ATM Adaptation Layer 5 Juha Heinanen

RFC 1514
Host Resources MIB P. Grillo, S. Waldbusser

RFC 1516
Definitions of Managed Objects for IEEE 802.3 Repeater Devices D. McMaster, K. McCloghrie

Appendix F. Related protocol specifications 657
RFC 1521
MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies N. Borenstein, N. Freed

RFC 1535
A Security Problem and Proposed Correction With Widely Deployed DNS Software E. Gavron

RFC 1536
Common DNS Implementation Errors and Suggested Fixes A. Kumar, J. Postel, C. Neuman, P. Danzig, S. Miller

RFC 1537
Common DNS Data File Configuration Errors P. Beertema

RFC 1540
Internet Official Protocol Standards J. Postel

RFC 1571
Telnet Environment Option Interoperability Issues D. Borman

RFC 1572
Telnet Environment Option S. Alexander

RFC 1573
Evolution of the Interfaces Group of MIB-II K. McCloghrie, F. Kastenholz

RFC 1577
Classical IP and ARP over ATM M. Laubach

RFC 1583
OSPF Version 2 J. Moy

RFC 1591
Domain Name System Structure and Delegation J. Postel

RFC 1592

RFC 1594
FYI on Questions and Answers—Answers to Commonly Asked “New Internet User” Questions A. Marine, J. Reynolds, G. Malkin

RFC 1644
T/TCP — TCP Extensions for Transactions Functional Specification R. Braden

RFC 1646
TN3270 Extensions for LUName and Printer Selection C. Graves, T. Butts, M. Angel

RFC 1647
TN3270 Enhancements B. Kelly

RFC 1652
SMTP Service Extension for 8bit-MIMEtransport J. Klensin, N. Freed, M. Rose, E. Stefferud, D. Crocker

RFC 1664
Using the Internet DNS to Distribute RFC1327 Mail Address Mapping Tables C. Allochio, A. Bonito, B. Cole, S. Giordano, R. Hagens

RFC 1693
An Extension to TCP: Partial Order Service T. Connolly, P. Amer, P. Conrad
RFC 1695
Definitions of Managed Objects for ATM Management Version 8.0 using SMIv2
M. Ahmed, K. Tesink

RFC 1701
Generic Routing Encapsulation (GRE) S. Hanks, T. Li, D. Farinacci, P. Traina

RFC 1702
Generic Routing Encapsulation over IPv4 networks S. Hanks, T. Li, D.
Farinacci, P. Traina

RFC 1706
DNS NSAP Resource Records B. Manning, R. Colella

RFC 1712
DNS Encoding of Geographical Location C. Farrell, M. Schulze, S. Pleitner D.
Baldoni

RFC 1713
Tools for DNS debugging A. Romao

RFC 1723
RIP Version 2—Carrying Additional Information G. Malkin

RFC 1752
The Recommendation for the IP Next Generation Protocol S. Bradner, A. Mankin

RFC 1766
Tags for the Identification of Languages H. Alvestrand

RFC 1771
A Border Gateway Protocol 4 (BGP-4) Y. Rekhter, T. Li

RFC 1794
DNS Support for Load Balancing T. Brisco

RFC 1819
Delgrossi, L. Berger Eds.

RFC 1826
IP Authentication Header R. Atkinson

RFC 1828
IP Authentication using Keyed MD5 P. Metzger, W. Simpson

RFC 1829
The ESP DES-CBC Transform P. Karn, P. Metzger, W. Simpson

RFC 1830
SMTP Service Extensions for Transmission of Large and Binary MIME Messages
G. Vaudreuil

RFC 1831

RFC 1832
XDR: External Data Representation Standard R. Srinivasan

RFC 1833
Binding Protocols for ONC RPC Version 2 R. Srinivasan

RFC 1850
OSPF Version 2 Management Information Base F. Baker, R. Coltun
RFC 1854
SMTP Service Extension for Command Pipelining N. Freed

RFC 1869
SMTP Service Extensions J. Klensin, N. Freed, M. Rose, E. Stefferud, D. Crocker

RFC 1870
SMTP Service Extension for Message Size Declaration J. Klensin, N. Freed, K. Moore

RFC 1876
A Means for Expressing Location Information in the Domain Name System C. Davis, P. Vixie, T. Goodwin, I. Dickinson

RFC 1883
Internet Protocol, Version 6 (IPv6) Specification S. Deering, R. Hinden

RFC 1884
IP Version 6 Addressing Architecture R. Hinden, S. Deering, Eds.

RFC 1886
DNS Extensions to support IP version 6 S. Thomson, C. Huitema

RFC 1888
OSI NSAPs and IPv6 J. Bound, B. Carpenter, D. Harrington, J. Houldsworth, A. Lloyd

RFC 1891
SMTP Service Extension for Delivery Status Notifications K. Moore

RFC 1892
The Multipart/Report Content Type for the Reporting of Mail System Administrative Messages G. Vaudreuil

RFC 1894
An Extensible Message Format for Delivery Status Notifications K. Moore, G. Vaudreuil

RFC 1901
Introduction to Community-based SNMPv2 J. Case, K. McCloghrie, M. Rose, S. Waldbusser

RFC 1902

RFC 1903

RFC 1904

RFC 1905

RFC 1906
RFC 1907

RFC 1908

RFC 1912
Common DNS Operational and Configuration Errors D. Barr

RFC 1918
Address Allocation for Private Internets Y. Rekhter, B. Moskowitz, D. Karrenberg, G.J. de Groot, E. Lear

RFC 1928

RFC 1930
Guidelines for creation, selection, and registration of an Autonomous System (AS) J. Hawkinson, T. Bates

RFC 1939
Post Office Protocol-Version 3 J. Myers, M. Rose

RFC 1981
Path MTU Discovery for IP version 6 J. McCann, S. Deering, J. Mogul

RFC 1982
Serial Number Arithmetic R. Elz, R. Bush

RFC 1985
SMTP Service Extension for Remote Message Queue Starting J. De Winter

RFC 1995
Incremental Zone Transfer in DNS M. Ohta

RFC 1996
A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY) P. Vixie

RFC 2010
Operational Criteria for Root Name Servers B. Manning, P. Vixie

RFC 2011
SNMPv2 Management Information Base for the Internet Protocol using SMIv2 K. McCloghrie, Ed.

RFC 2012

RFC 2013
SNMPv2 Management Information Base for the User Datagram Protocol using SMIv2 K. McCloghrie, Ed.

RFC 2018
TCP Selective Acknowledgement Options M. Mathis, J. Mahdavi, S. Floyd, A. Romanow

RFC 2026
The Internet Standards Process — Revision 3 S. Bradner
RFC 2030
Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI
D. Mills

RFC 2033
Local Mail Transfer Protocol J. Myers

RFC 2034
SMTP Service Extension for Returning Enhanced Error Codes N. Freed

RFC 2040

RFC 2045
Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies N. Freed, N. Borenstein

RFC 2052
A DNS RR for specifying the location of services (DNS SRV) A. Gulbrandsen, P. Vixie

RFC 2065
Domain Name System Security Extensions D. Eastlake 3rd, C. Kaufman

RFC 2066
TELNET CHARSET Option R. Gellens

RFC 2080
RIPng for IPv6 G. Malkin, R. Minnear

RFC 2096
IP Forwarding Table MIB F. Baker

RFC 2104
HMAC: Keyed-Hashing for Message Authentication H. Krawczyk, M. Bellare, R. Canetti

RFC 2119
Keywords for use in RFCs to Indicate Requirement Levels S. Bradner

RFC 2133
Basic Socket Interface Extensions for IPv6 R. Gilligan, S. Thomson, J. Bound, W. Stevens

RFC 2136
Dynamic Updates in the Domain Name System (DNS UPDATE) P. Vixie, Ed., S. Thomson, Y. Rekhter, J. Bound

RFC 2137
Secure Domain Name System Dynamic Update D. Eastlake 3rd

RFC 2163
Using the Internet DNS to Distribute MIXER Conformant Global Address Mapping (MCGAM) C. Allocchio

RFC 2168
Resolution of Uniform Resource Identifiers using the Domain Name System R. Daniel, M. Mealling

RFC 2178
OSPF Version 2 J. Moy

RFC 2181
Clarifications to the DNS Specification R. Elz, R. Bush
RFC 2205

RFC 2210
The Use of RSVP with IETF Integrated Services J. Wroclawski

RFC 2211
Specification of the Controlled-Load Network Element Service J. Wroclawski

RFC 2212
Specification of Guaranteed Quality of Service S. Shenker, C. Partridge, R. Guerin

RFC 2215
General Characterization Parameters for Integrated Service Network Elements S. Shenker, J. Wroclawski

RFC 2217
Telnet Com Port Control Option G. Clarke

RFC 2219
Use of DNS Aliases for Network Services M. Hamilton, R. Wright

RFC 2228
FTP Security Extensions M. Horowitz, S. Lunt

RFC 2230
Key Exchange Delegation Record for the DNS R. Atkinson

RFC 2233
The Interfaces Group MIB using SMIv2 K. McCloghrie, F. Kastenholz

RFC 2240
A Legal Basis for Domain Name Allocation O. Vaughn

RFC 2246
The TLS Protocol Version 1.0 T. Dierks, C. Allen

RFC 2251

RFC 2253

RFC 2254
The String Representation of LDAP Search Filters T. Howes

RFC 2261
An Architecture for Describing SNMP Management Frameworks D. Harrington, R. Presuhn, B. Wijnen

RFC 2262

RFC 2271
An Architecture for Describing SNMP Management Frameworks D. Harrington, R. Presuhn, B. Wijnen

RFC 2273
SNMPv3 Applications D. Levi, P. Meyer, B. Stewart
RFC 2274

RFC 2275
View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP) B. Wijnen, R. Presuhn, K. McCloghrie

RFC 2279
UTF-8, a transformation format of ISO 10646 F. Yergeau

RFC 2292
Advanced Sockets API for IPv6 W. Stevens, M. Thomas

RFC 2308
Negative Caching of DNS Queries (DNS NCACHE) M. Andrews

RFC 2317
Classless IN-ADDR.ARPA delegation H. Eidnes, G. de Groot, P. Vixie

RFC 2320
Definitions of Managed Objects for Classical IP and ARP Over ATM Using SMIv2 (IPOA-MIB) M. Greene, J. Luciani, K. White, T. Kuo

RFC 2328
OSPF Version 2 J. Moy

RFC 2345
Domain Names and Company Name Retrieval J. Klensin, T. Wolf, G. Oglesby

RFC 2352
A Convention for Using Legal Names as Domain Names O. Vaughn

RFC 2355
TN3270 Enhancements B. Kelly

RFC 2358
Definitions of Managed Objects for the Ethernet-like Interface Types J. Flick, J. Johnson

RFC 2373
IP Version 6 Addressing Architecture R. Hinden, S. Deering

RFC 2374
An IPv6 Aggregatable Global Unicast Address Format R. Hinden, M. O’Dell, S. Deering

RFC 2375
IPv6 Multicast Address Assignments R. Hinden, S. Deering

RFC 2385
Protection of BGP Sessions via the TCP MD5 Signature Option A. Hefferman

RFC 2389
Feature negotiation mechanism for the File Transfer Protocol P. Hethmon, R. Elz

RFC 2401
Security Architecture for Internet Protocol S. Kent, R. Atkinson

RFC 2402
IP Authentication Header S. Kent, R. Atkinson

RFC 2403
The Use of HMAC-MD5–96 within ESP and AH C. Madson, R. Glenn
RFC 2404
The Use of HMAC-SHA–1–96 within ESP and AH C. Madson, R. Glenn

RFC 2405
The ESP DES-CBC Cipher Algorithm With Explicit IV C. Madson, N. Doraswamy

RFC 2406
IP Encapsulating Security Payload (ESP) S. Kent, R. Atkinson

RFC 2407
The Internet IP Security Domain of Interpretation for ISAKMP D. Piper

RFC 2408

RFC 2409
The Internet Key Exchange (IKE) D. Harkins, D. Carrel

RFC 2410
The NULL Encryption Algorithm and Its Use With IPsec R. Glenn, S. Kent

RFC 2428
FTP Extensions for IPv6 and NATs M. Allman, S. Ostermann, C. Metz

RFC 2445
Internet Calendaring and Scheduling Core Object Specification (iCalendar) F. Dawson, D. Stenerson

RFC 2459
Internet X.509 Public Key Infrastructure Certificate and CRL Profile R. Housley, W. Ford, W. Polk, D. Solo

RFC 2460
Internet Protocol, Version 6 (IPv6) Specification S. Deering, R. Hinden

RFC 2461

RFC 2462
IPv6 Stateless Address Autoconfiguration S. Thomson, T. Narten

RFC 2463
Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification A. Conta, S. Deering

RFC 2464
Transmission of IPv6 Packets over Ethernet Networks M. Crawford

RFC 2466

RFC 2476
Message Submission R. Gellens, J. Klensin

RFC 2487
SMTP Service Extension for Secure SMTP over TLS P. Hoffman

RFC 2505
Anti-Spam Recommendations for SMTP MTAs G. Lindberg
RFC 2523
Photuris: Extended Schemes and Attributes P. Karn, W. Simpson

RFC 2535
Domain Name System Security Extensions D. Eastlake 3rd

RFC 2538
Storing Certificates in the Domain Name System (DNS) D. Eastlake 3rd, O. Gudmundsson

RFC 2539
Storage of Diffie-Hellman Keys in the Domain Name System (DNS) D. Eastlake 3rd

RFC 2540
Detached Domain Name System (DNS) Information D. Eastlake 3rd

RFC 2554
SMTP Service Extension for Authentication J. Myers

RFC 2570
Introduction to Version 3 of the Internet-standard Network Management Framework J. Case, R. Mundy, D. Partain, B. Stewart

RFC 2571
An Architecture for Describing SNMP Management Frameworks B. Wijnen, D. Harrington, R. Presuhn

RFC 2572

RFC 2573
SNMP Applications D. Levi, P. Meyer, B. Stewart

RFC 2574

RFC 2575
View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP) B. Wijnen, R. Presuhn, K. McCloghrie

RFC 2576

RFC 2578

RFC 2579
Textual Conventions for SMIv2 K. McCloghrie, D. Perkins, J. Schoenwaelder

RFC 2580
Conformance Statements for SMIv2 K. McCloghrie, D. Perkins, J. Schoenwaelder

RFC 2581
TCP Congestion Control M. Allman, V. Paxson, W. Stevens

RFC 2583
Guidelines for Next Hop Client (NHC) Developers R. Carlson, L. Winkler
RFC 2591
Definitions of Managed Objects for Scheduling Management Operations D. Levi, J. Schoenwaelder

RFC 2625
IP and ARP over Fibre Channel M. Rajagopal, R. Bhagwat, W. Rickard

RFC 2635
Don’t SPEW A Set of Guidelines for Mass Unsolicited Mailings and Postings (spam*) S. Hambridge, A. Lunde

RFC 2637

RFC 2640
Internationalization of the File Transfer Protocol B. Curtin

RFC 2665
Definitions of Managed Objects for the Ethernet-like Interface Types J. Flick, J. Johnson

RFC 2671
Extension Mechanisms for DNS (EDNS0) P. Vixie

RFC 2672
Non-Terminal DNS Name Redirection M. Crawford

RFC 2675
IPv6 Jumbograms D. Borman, S. Deering, R. Hinden

RFC 2710
Multicast Listener Discovery (MLD) for IPv6 S. Deering, W. Fenner, B. Haberman

RFC 2711
IPv6 Router Alert Option C. Partridge, A. Jackson

RFC 2740
OSPF for IPv6 R. Coltun, D. Ferguson, J. Moy

RFC 2753
A Framework for Policy-based Admission Control R. Yavatkar, D. Pendarakis, R. Guerin

RFC 2782
A DNS RR for specifying the location of services (DNS SRV) A. Gubrandsen, P. Vixix, L. Esibov

RFC 2821
Simple Mail Transfer Protocol J. Klensin, Ed.

RFC 2822
Internet Message Format P. Resnick, Ed.

RFC 2840
TELNET KERMIT OPTION J. Altman, F. da Cruz

RFC 2845
Secret Key Transaction Authentication for DNS (TSIG) P. Vixie, O. Gudmundsson, D. Eastlake 3rd, B. Wellington

RFC 2851
Textual Conventions for Internet Network Addresses M. Daniele, B. Haberman, S. Routhier, J. Schoenwaelder

Appendix F. Related protocol specifications 667
RFC 2852
Deliver By SMTP Service Extension D. Newman

RFC 2874
DNS Extensions to Support IPv6 Address Aggregation and Renumbering M. Crawford, C. Huitema

RFC 2915
The Naming Authority Pointer (NAPTR) DNS Resource Record M. Mealling, R. Daniel

RFC 2920
SMTP Service Extension for Command Pipelining N. Freed

RFC 2930
Secret Key Establishment for DNS (TKEY RR) D. Eastlake, 3rd

RFC 2941
Telnet Authentication Option T. Ts’o, ed., J. Altman

RFC 2942
Telnet Authentication: Kerberos Version 5 T. Ts’o

RFC 2946
Telnet Data Encryption Option T. Ts’o

RFC 2952
Telnet Encryption: DES 64 bit Cipher Feedback T. Ts’o

RFC 2953
Telnet Encryption: DES 64 bit Output Feedback T. Ts’o

RFC 2992
Analysis of an Equal-Cost Multi-Path Algorithm C. Hopps

RFC 3019
IP Version 6 Management Information Base for The Multicast Listener Discovery Protocol B. Haberman, R. Worzella

RFC 3060
Policy Core Information Model—Version 1 Specification B. Moore, E. Ellesson, J. Strassner, A. Westerinen

RFC 3152
Delegation of IP6.ARPA R. Bush

RFC 3164
The BSD Syslog Protocol C. Lonvick

RFC 3207
SMTP Service Extension for Secure SMTP over Transport Layer Security P. Hoffman

RFC 3226
DNSSEC and IPv6 A6 aware server/resolver message size requirements O. Gudmundsson

RFC 3291
Textual Conventions for Internet Network Addresses M. Daniele, B. Haberman, S. Routhier, J. Schoenwaelder

RFC 3363
RFC 3376
Internet Group Management Protocol, Version 3 B. Cain, S. Deering, I. Kouvelas, B. Fenner, A. Thyagarajan

RFC 3390
Increasing TCP’s Initial Window M. Allman, S. Floyd, C. Partridge

RFC 3410
Introduction and Applicability Statements for Internet-Standard Management Framework J. Case, R. Mundy, D. Partain, B. Stewart

RFC 3411

RFC 3412

RFC 3413

RFC 3414

RFC 3415
View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP) B. Wijnen, R. Presuhn, K. McCloghrie

RFC 3416

RFC 3417

RFC 3418

RFC 3419
Textual Conventions for Transport Addresses M. Daniele, J. Schoenwaelder

RFC 3484
Default Address Selection for Internet Protocol version 6 (IPv6) R. Draves

RFC 3493

RFC 3513
Internet Protocol Version 6 (IPv6) Addressing Architecture R. Hinden, S. Deering

RFC 3526
More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE) T. Kivinen, M. Kojo
RFC 3542
Advanced Sockets Application Programming Interface (API) for IPv6 W. Richard Stevens, M. Thomas, E. Nordmark, T. Jinmei

RFC 3566
The AES-XCBC-MAC-96 Algorithm and Its Use With IPsec S. Frankel, H. Herbert

RFC 3569
An Overview of Source-Specific Multicast (SSM) S. Bhattacharyya, Ed.

RFC 3584

RFC 3602
The AES-CBC Cipher Algorithm and Its Use with IPsec S. Frankel, R. Glenn, S. Kelly

RFC 3629
UTF-8, a transformation format of ISO 10646 R. Kermode, C. Vicisano

RFC 3658
Delegation Signer (DS) Resource Record (RR) O. Gudmundsson

RFC 3678
Socket Interface Extensions for Multicast Source Filters D. Thaler, B. Fenner, B. Quinn

RFC 3715
IPsec-Network Address Translation (NAT) Compatibility Requirements B. Aboba, W. Dixon

RFC 3810

RFC 3947
Negotiation of NAT-Traversal in the IKE T. Kivinen, B. Swander, A. Huttunen, V. Volpe

RFC 3948
UDP Encapsulation of IPsec ESP Packets A. Huttunen, B. Swander, V. Volpe, L. DiBurro, M. Stenberg

RFC 4001
Textual Conventions for Internet Network Addresses M. Daniele, B. Haberman, S. Routhier, J. Schoenaerd

RFC 4007
IPv6 Scoped Address Architecture S. Deering, B. Haberman, T. Jinmei, E. Nordmark, B. Zill

RFC 4022
Management Information Base for the Transmission Control Protocol (TCP) R. Raghunarayan

RFC 4106
The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP) J. Viega, D. McGrew

RFC 4109
Algorithms for Internet Key Exchange version 1 (IKEv1) P. Hoffman
RFC 4113
Management Information Base for the User Datagram Protocol (UDP) B. Fenner, J. Flick

RFC 4191
Default Router Preferences and More-Specific Routes R. Draves, D. Thaler

RFC 4217
Securing FTP with TLS P. Ford-Hutchinson

RFC 4292
IP Forwarding Table MIB B. Haberman

RFC 4293
Management Information Base for the Internet Protocol (IP) S. Routhier

RFC 4301
Security Architecture for the Internet Protocol S. Kent, K. Seo

RFC 4302
IP Authentication Header S. Kent

RFC 4303
IP Encapsulating Security Payload (ESP) S. Kent

RFC 4304
Extended Sequence Number (ESN) Addendum to IPsec Domain of Interpretation (DOI) for Internet Security Association and Key Management Protocol (ISAKMP) S. Kent

RFC 4307
Cryptographic Algorithms for Use in the Internet Key Exchange Version 2 (IKEv2) J. Schiller

RFC 4308
Cryptographic Suites for IPsec P. Hoffman

RFC 4434
The AES-XCBC-PRF-128 Algorithm for the Internet Key Exchange Protocol P. Hoffman

RFC 4443
Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification A. Conta, S. Deering

RFC 4552
Authentication/Confidentiality for OSPFv3 M. Gupta, N. Melam

RFC 4678
Server/Application State Protocol v1 A. Bivens

RFC 4753
ECP Groups for IKE and IKEv2 D. Fu, J. Solinas

RFC 4754
IKE and IKEv2 Authentication Using the Elliptic Curve Digital Signature Algorithm (ECDSA) D. Fu, J. Solinas

RFC 4809

RFC 4835
Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH) V. Manral
RFC 4862
IPv6 Stateless Address Autoconfiguration S. Thomson, T. Narten, T. Jinmei

RFC 4868
Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec S. Kelly, S. Frankel

RFC 4869
Suite B Cryptographic Suites for IPsec L. Law, J. Solinas

RFC 4941
Privacy Extensions for Stateless Address Autoconfiguration in IPv6 T. Narten, R. Draves, S. Krishnan

RFC 4945
The Internet IP Security PKI Profile of IKEv1/ISAKMP, IKEv2, and PKIX B. Korver

RFC 5014
IPv6 Socket API for Source Address Selection E. Nordmark, S. Chakrabarti, J. Laganier

RFC 5095
Deprecation of Type 0 Routing Headers in IPv6 J. Abley, P. Savola, G. Neville-Neil

RFC 5175
IPv6 Router Advertisement Flags Option B. Haberman, Ed., R. Hinden

RFC 5282
Using Authenticated Encryption Algorithms with the Encrypted Payload of the Internet Key Exchange version 2 (IKEv2) Protocol D. Black, D. McGrew

RFC 5996
Internet Key Exchange Protocol Version 2 (IKEv2) C. Kaufman, P. Hoffman, Y. Nir, P. Eronen

Internet drafts

Internet drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Other groups can also distribute working documents as Internet drafts. You can see Internet drafts at [http://www.ietf.org/id.html](http://www.ietf.org/id.html)
Appendix G. Accessibility

Publications for this product are offered in Adobe Portable Document Format (PDF) and should be compliant with accessibility standards. If you experience difficulties when using PDF files, you can view the information through the z/OS Internet Library website or the z/OS Information Center. If you continue to experience problems, send an email to mhvrdfs@us.ibm.com or write to:

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. See z/OS TSO/E Primer, z/OS TSO/E User’s Guide, and z/OS ISPF User’s Guide Vol I for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

z/OS information

z/OS information is accessible using screen readers with the BookServer or Library Server versions of z/OS books in the Internet library at www.ibm.com/systems/z/os/zos/bkserv/

One exception is command syntax that is published in railroad track format, which is accessible using screen readers with the Information Center, as described in "Dotted decimal syntax diagrams."

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users accessing the Information Center using a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always
present together (or always absent together), they can appear on the same line, because they can be considered as a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that your screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1) are mutually exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, you know that your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The * symbol can be used next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is given the format 3 \* FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* \* FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol giving information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, this indicates a reference that is defined elsewhere. The string following the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you should see separate syntax fragment OP1.

The following words and symbols are used next to the dotted decimal numbers:

- A question mark (?) means an optional syntax element. A dotted decimal number followed by the ? symbol indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that syntax elements NOTIFY and UPDATE are optional; that is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

- An exclamation mark (!) means a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicate that the syntax element is the default option for all syntax elements that share the same dotted
decimal number. Only one of the syntax elements that share the same dotted decimal number can specify a ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In this example, if you include the FILE keyword but do not specify an option, default option KEEP will be applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1.1 (DELETE), the default option KEEP applies only to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

- An asterisk (*) means a syntax element that can be repeated 0 or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3*, 3 HOST, and 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

Notes:

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you could write HOST STATE, but you could not write HOST HOST.
3. The * symbol is equivalent to a loop-back line in a railroad syntax diagram.

- + means a syntax element that must be included one or more times. A dotted decimal number followed by the + symbol indicates that this syntax element must be included one or more times; that is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the * symbol, the + symbol can only repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loop-back line in a railroad syntax diagram.


Notices

This information was developed for products and services offered in the USA.

IBM may not offer all of the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user’s responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who want to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Site Counsel
IBM Corporation
P.O. Box 12195
3039 Cornwallis Road
Research Triangle Park, North Carolina 27709-2195
U.S.A

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM’s future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations might not appear.

COPYRIGHT LICENSE:

This information might contain sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing
application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_.

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.

Some portions of this publication relating to X Window System are Copyright © 1986, 1987, 1988 by Hewlett-Packard Corporation.

Permission to use, copy, modify, and distribute the M.I.T., Digital Equipment Corporation, and Hewlett-Packard Corporation portions of this software and its documentation for any purpose without fee is hereby granted, provided that the above copyright notice appears in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the names of M.I.T., Digital, and Hewlett-Packard not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. M.I.T., Digital, and Hewlett-Packard make no representation about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.


Copyright © 1988, 1993 The Regents of the University of California.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgment:
This product includes software developed by the University of California, Berkeley and its contributors.

4. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS `AS IS' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software program contains code, and/or derivatives or modifications of code originating from the software program "Popper." Popper is Copyright ©1989-1991 The Regents of the University of California. Popper was created by Austin Shelton, Information Systems and Technology, University of California, Berkeley.

Permission from the Regents of the University of California to use, copy, modify, and distribute the "Popper" software contained herein for any purpose, without fee, and without a written agreement is hereby granted, provided that the above copyright notice and this paragraph and the following two paragraphs appear in all copies. HOWEVER, ADDITIONAL PERMISSIONS MAY BE NECESSARY FROM OTHER PERSONS OR ENTITIES, TO USE DERIVATIVES OR MODIFICATIONS OF POPPER.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THE POPPER SOFTWARE, OR ITS DERIVATIVES OR MODIFICATIONS, AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE POPPER SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATIONS TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

Copyright © 1983 The Regents of the University of California.

Redistribution and use in source and binary forms are permitted provided that the above copyright notice and this paragraph are duplicated in all such forms and that any documentation, advertising materials, and other materials related to such distribution and use acknowledge that the software was developed by the University of California, Berkeley. The name of the University may not be used to endorse or promote products derived from this software without specific prior
written permission. THIS SOFTWARE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Copyright © 1991, 1993 The Regents of the University of California.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgment:
   This product includes software developed by the University of California, Berkeley and its contributors.
4. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright © 1990 by the Massachusetts Institute of Technology

Export of this software from the United States of America may require a specific license from the United States Government. It is the responsibility of any person or organization contemplating export to obtain such a license before exporting.

WITHIN THAT CONSTRAINT, permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of M.I.T. not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. Furthermore if you modify this software you must label your software as modified software and not distribute it in such a fashion that it might be confused with the original M.I.T. software. M.I.T. makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

Copyright © 1998 by the FundsXpress, INC.
Export of this software from the United States of America may require a specific license from the United States Government. It is the responsibility of any person or organization contemplating export to obtain such a license before exporting.

WITHIN THAT CONSTRAINT, permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of FundsXpress not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. FundsXpress makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

THIS SOFTWARE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.


Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND INTERNET SOFTWARE CONSORTIUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL INTERNET SOFTWARE CONSORTIUM BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Copyright © 1995-1998 Eric Young (eay@cryptsoft.com)

This package is an SSL implementation written by Eric Young (eay@cryptsoft.com). The implementation was written so as to conform with Netscape's SSL.

This library is free for commercial and non-commercial use as long as the following conditions are adhered to. The following conditions apply to all code found in this distribution, be it the RC4, RSA, lhash, DES, etc., code; not just the SSL code. The SSL documentation included with this distribution is covered by the same copyright terms except that the holder is Tim Hudson (tjh@cryptsoft.com).

Copyright remains Eric Young's, and as such any Copyright notices in the code are not to be removed. If this package is used in a product, Eric Young should be given attribution as the author of the parts of the library used. This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. All advertising materials mentioning features or use of this software must display the following acknowledgment: "This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)". The word 'cryptographic' can be left out if the routines from the library being used are not cryptographic related.

4. If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include acknowledgment: "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"

THE SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The license and distribution terms for any publicly available version or derivative of this code cannot be changed. i.e. this code cannot simply be copied and put under another distribution license [including the GNU Public License.]

This product includes cryptographic software written by Eric Young.


Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND INTERNET SOFTWARE CONSORTIUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL INTERNET SOFTWARE CONSORTIUM BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Copyright © 2004 IBM Corporation and its licensors, including Sendmail, Inc., and the Regents of the University of California.

Copyright © 1999,2000,2001 Compaq Computer Corporation

Copyright © 1999,2000,2001 Hewlett-Packard Company
Permission is hereby granted, free of charge, to any person obtaining a copy of this
software and associated documentation files (the “Software”), to deal in the
Software without restriction, including without limitation the rights to use, copy,
modify, merge, publish, distribute, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, provided that the
above copyright notice(s) and this permission notice appear in all copies of the
Software and that both the above copyright notice(s) and this permission notice
appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE
COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE
FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL
DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF
USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
NEG Li GENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN
CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used
in advertising or otherwise to promote the sale, use or other dealings in this
Software without prior written authorization of the copyright holder.

X Window System is a trademark of The Open Group.

If you are viewing this information softcopy, the photographs and color
illustrations may not appear.

You can obtain softcopy from the z/OS Collection (SK3T-4269), which contains
BookManager and PDF formats.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS
announcements can subsequently change when service for particular servers or
devices is withdrawn. Likewise, the levels of other software products supported on
a particular release of z/OS are subject to the service support lifecycle of those
products. Therefore, z/OS and its product publications (for example, panels,
samples, messages, and product documentation) can include references to
hardware and software that is no longer supported.

- For information about software support lifecycle, see: http://www-01.ibm.com/
  software/support/systemsz/lifecycle/
- For information about currently-supported IBM hardware, contact your IBM
  representative.
Policy for unsupported hardware

Various z/OS elements, such as DFSMS, HCD, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

Intel is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Java™ and all Java-based trademarks are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and Windows NT are trademarks of Microsoft Corporation in the United States, other countries, or both.

Adobe and PostScript are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other product and service names might be trademarks of IBM or other companies.
Bibliography

This bibliography contains descriptions of the documents in the z/OS Communications Server library.

z/OS Communications Server documentation is available in the following forms:

- Online at the z/OS Internet Library web page at www.ibm.com/systems/z/os/zos/bkserv/
- In softcopy on CD-ROM collections. See “Softcopy information” on page xxvi.

**z/OS Communications Server library updates**

An index to z/OS Communications Server book updates is at http://www.ibm.com/support/docview.wss?uid=swg21178966. Updates to documents are also available on RETAIN® and in information APARs (info APARs). Go to http://www.ibm.com/software/network/commserv/zos/support to view information APARs. In addition, Info APARs for z/OS documents are in z/OS and z/OS.e DOC APAR and PTF ++HOLD Documentation, which can be found at http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/Shelves/ZDOCAPAR.

**z/OS Communications Server information**

z/OS Communications Server product information is grouped by task in the following tables.

### Planning

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS Communications Server: New Function Summary</td>
<td>GC27-3664</td>
<td>This document is intended to help you plan for new IP or SNA function, whether you are migrating from a previous version or installing z/OS for the first time. It summarizes what is new in the release and identifies the suggested and required modifications needed to use the enhanced functions.</td>
</tr>
<tr>
<td>z/OS Communications Server: IPv6 Network and Application Design Guide</td>
<td>SC27-3663</td>
<td>This document is a high-level introduction to IPv6. It describes concepts of z/OS Communications Server’s support of IPv6, coexistence with IPv4, and migration issues.</td>
</tr>
</tbody>
</table>

### Resource definition, configuration, and tuning

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS Communications Server: IP Configuration Guide</td>
<td>SC27-3650</td>
<td>This document describes the major concepts involved in understanding and configuring an IP network. Familiarity with the z/OS operating system, IP protocols, z/OS UNIX System Services, and IBM Time Sharing Option (TSO) is recommended. Use this document with the z/OS Communications Server: IP Configuration Reference.</td>
</tr>
<tr>
<td>Title</td>
<td>Number</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| z/OS Communications Server: IP Configuration Reference | SC27-3651 | This document presents information for people who want to administer and maintain IP. Use this document with the z/OS Communications Server: IP Configuration Guide. The information in this document includes:  
- TCP/IP configuration data sets  
- Configuration statements  
- Translation tables  
- Protocol number and port assignments |
| z/OS Communications Server: SNA Network Implementation Guide | SC27-3672 | This document presents the major concepts involved in implementing an SNA network. Use this document with the z/OS Communications Server: SNA Resource Definition Reference. |
| z/OS Communications Server: SNA Resource Definition Reference | SC27-3675 | This document describes each SNA definition statement, start option, and macroinstruction for user tables. It also describes NCP definition statements that affect SNA. Use this document with the z/OS Communications Server: SNA Network Implementation Guide. |
| z/OS Communications Server: SNA Resource Definition Samples | SC27-3676 | This document contains sample definitions to help you implement SNA functions in your networks, and includes sample major node definitions. |
| z/OS Communications Server: IP Network Print Facility | SC27-3658 | This document is for systems programmers and network administrators who need to prepare their network to route SNA, JES2, or JES3 printer output to remote printers using TCP/IP Services. |

### Operation

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS Communications Server: IP User’s Guide and Commands</td>
<td>SC27-3662</td>
<td>This document describes how to use TCP/IP applications. It contains requests with which a user can log on to a remote host using Telnet, transfer data sets using FTP, send and receive electronic mail, print on remote printers, and authenticate network users.</td>
</tr>
<tr>
<td>z/OS Communications Server: IP System Administrator’s Commands</td>
<td>SC27-3661</td>
<td>This document describes the functions and commands helpful in configuring or monitoring your system. It contains system administrator’s commands, such as TSO Netstat, Ping, TracerTe and their UNIX counterparts. It also includes TSO and MVS commands commonly used during the IP configuration process.</td>
</tr>
<tr>
<td>z/OS Communications Server: SNA Operation</td>
<td>SC27-3673</td>
<td>This document serves as a reference for programmers and operators requiring detailed information about specific operator commands.</td>
</tr>
<tr>
<td>z/OS Communications Server: Quick Reference</td>
<td>SC27-3665</td>
<td>This document contains essential information about SNA and IP commands.</td>
</tr>
</tbody>
</table>
### Customization

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
</table>
| z/OS Communications Server: SNA Customization | SC27-3666 | This document enables you to customize SNA, and includes the following information:  
- Communication network management (CNM) routing table  
- Logon-interpret routine requirements  
- Logon manager installation-wide exit routine for the CLU search exit  
- TSO/SNA installation-wide exit routines |

### Writing application programs

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference</td>
<td>SC27-3660</td>
<td>This document describes the syntax and semantics of program source code necessary to write your own application programming interface (API) into TCP/IP. You can use this interface as the communication base for writing your own client or server application. You can also use this document to adapt your existing applications to communicate with each other using sockets over TCP/IP.</td>
</tr>
<tr>
<td>z/OS Communications Server: IP CICS Sockets Guide</td>
<td>SC27-3649</td>
<td>This document is for programmers who want to set up, write application programs for, and diagnose problems with the socket interface for CICS using z/OS TCP/IP.</td>
</tr>
<tr>
<td>z/OS Communications Server: IP IMS Sockets Guide</td>
<td>SC27-3653</td>
<td>This document is for programmers who want application programs that use the IMS TCP/IP application development services provided by the TCP/IP Services of IBM.</td>
</tr>
<tr>
<td>z/OS Communications Server: IP Programmer’s Guide and Reference</td>
<td>SC27-3659</td>
<td>This document describes the syntax and semantics of a set of high-level application functions that you can use to program your own applications in a TCP/IP environment. These functions provide support for application facilities, such as user authentication, distributed databases, distributed processing, network management, and device sharing. Familiarity with the z/OS operating system, TCP/IP protocols, and IBM Time Sharing Option (TSO) is recommended.</td>
</tr>
<tr>
<td>z/OS Communications Server: SNA Programming</td>
<td>SC27-3674</td>
<td>This document describes how to use SNA macroinstructions to send data to and receive data from (1) a terminal in either the same or a different domain, or (2) another application program in either the same or a different domain.</td>
</tr>
<tr>
<td>z/OS Communications Server: SNA Programmer’s LU 6.2 Guide</td>
<td>SC27-3669</td>
<td>This document describes how to use the SNA LU 6.2 application programming interface for host application programs. This document applies to programs that use only LU 6.2 sessions or that use LU 6.2 sessions along with other session types. (Only LU 6.2 sessions are covered in this document.)</td>
</tr>
<tr>
<td>z/OS Communications Server: SNA Programmer’s LU 6.2 Reference</td>
<td>SC27-3670</td>
<td>This document provides reference material for the SNA LU 6.2 programming interface for host application programs.</td>
</tr>
<tr>
<td>z/OS Communications Server: CSM Guide</td>
<td>SC27-3647</td>
<td>This document describes how applications use the communications storage manager.</td>
</tr>
</tbody>
</table>
### Title


### Diagnosis

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS Communications Server: IP Diagnosis Guide</td>
<td>GC27-3652</td>
<td>This document explains how to diagnose TCP/IP problems and how to determine whether a specific problem is in the TCP/IP product code. It explains how to gather information for and describe problems to the IBM Software Support Center.</td>
</tr>
<tr>
<td>z/OS Communications Server: ACF/TAP Trace Analysis Handbook</td>
<td>GC27-3645</td>
<td>This document explains how to gather the trace data that is collected and stored in the host processor. It also explains how to use the Advanced Communications Function/Trace Analysis Program (ACF/TAP) service aid to produce reports for analyzing the trace data information.</td>
</tr>
<tr>
<td>z/OS Communications Server: SNA Diagnosis Vol 1, Techniques and Procedures and z/OS Communications Server: SNA Diagnosis Vol 2, FFST Dumps and the VIT</td>
<td>GC27-3667, GC27-3668</td>
<td>These documents help you identify an SNA problem, classify it, and collect information about it before you call the IBM Support Center. The information collected includes traces, dumps, and other problem documentation.</td>
</tr>
<tr>
<td>z/OS Communications Server: SNA Data Areas Volume 1 and z/OS Communications Server: SNA Data Areas Volume 2</td>
<td>GC31-6852, GC31-6853</td>
<td>These documents describe SNA data areas and can be used to read an SNA dump. They are intended for IBM programming service representatives and customer personnel who are diagnosing problems with SNA.</td>
</tr>
</tbody>
</table>

### Messages and codes

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
</table>
| z/OS Communications Server: SNA Messages | SC27-3671 | This document describes the ELM, IKT, IST, IUT, IVT, and USS messages. Other information in this document includes:  
- Command and RU types in SNA messages  
- Node and ID types in SNA messages  
- Supplemental message-related information |
| z/OS Communications Server: IP Messages Volume 1 (EZA) | SC27-3654 | This volume contains TCP/IP messages beginning with EZA. |
| z/OS Communications Server: IP Messages Volume 2 (EZB, EZD) | SC27-3655 | This volume contains TCP/IP messages beginning with EZB or EZD. |
| z/OS Communications Server: IP Messages Volume 3 (EZY) | SC27-3656 | This volume contains TCP/IP messages beginning with EZY. |
| z/OS Communications Server: IP Messages Volume 4 (EZZ, SNM) | SC27-3657 | This volume contains TCP/IP messages beginning with EZZ and SNM. |
| z/OS Communications Server: IP and SNA Codes | SC27-3648 | This document describes codes and other information that appear in z/OS Communications Server messages. |
Index

A
abend codes
   AEY9  120
   E20L  122
   E20T  122
ACCEPT (call)  254
accept system call
   C language  171
   EZACICAL call  425
   use in server  133
accessibility  673
adapter  19
adding a z/OS UNIX system services segment  52
address
   family (domain)  137
   MVS address spaces  138
   structures
      AF_INET  137
      AF_INET6  137
address testing macros  245
addrinfo C structure  169
addrinfo interpreter parameters, on EZACIC09  416
AF parameter on call interface, on EZACIC06 call  408
AF_INET domain parameter  137, 243
AF_INET6 domain parameter  137, 243
ALTER  69
application transparent transport layer security (AT-TLS)  161
ASCII data format  161
automatic startup  107

B
BACKLOG parameter on call interface, LISTEN call  335
big endian  139
BIND (call)  257
bind system call
   C language  173
   EZACICAL call  426
   use in server  133
bit-mask on call interface, on EZACIC06 call  408
bit-mask-length on call interface, on EZACIC06 call  408
blocking/nonblocking option  182, 226
broadcast option  211
BUF parameter on call socket interface  249
   on READ  341
   on RECV  345
   on RECVFROM  348
   on SEND  366
   on SENDTO  374
   on WRITE  399

C
C language
   API  165, 203, 215, 242
   basic calls  20
   C structures
      addrinfo  169
      clientid  167

C language (continued)
C structures (continued)
   group_req  169
   group_source_req  170
   If_NameIndex  168
   ifconf  167
   ifrq  168
   ip_mreq  168
   ip_mreq_source  169
   ipv6_mreq  168
   linger  168
   NetConfHdr  168
   SetADContainer  170
   SetApplData  170
   sockaddr_in  169
   sockaddr_in6  169
   timeval  169
calls
   accept()  171
   bind()  173
   close()  178
   connect()  179
   fcntl()  181
   freeaddrinfo()  182
   gai_strerror()  183
   getaddrinfo()  184
   getclientid()  191
   gethostbyaddr()  191
   gethostbyname()  192
   gethostid()  193
   gethostname()  194
   getnameinfo()  197
   getpeername()  199
   getsockname()  201
   getssockopt()  203
   getsourcelen()  213
   givesocket()  215
   if_freenameindex()  216
   if_index()  217
   if_nameindex()  218
   if_nametoindex()  219
   inet_ntop()  220
   inet_pton()  221
   initapi()  224
   ioctl()  225
   listen()  229
   read()  229
   recv()  230
   recvfrom()  231
   select()  233
   send()  236
   sendto()  238
   setsockopt()  203
   shutdown()  242
   socket()  242
   takesocket()  244
   write()  245
compiling and linking  166
header files needed  165

© Copyright IBM Corp. 2000, 2013
C socket calls

C language

getipv4sourcefilter() 195
setipv4sourcefilter() 239
setsourcelfilter() 241

CICS 107
starting automatically 107
starting manually 108
starting with program link 122

CICS transaction processing system

defining resources in setup 26
operation with CICS TCP/IP 19

client

definition 2
socket calls used in 131
client and server processing 2

CLIENT parameter on call socket interface 249

clientid C structure 167

close system call

C language 178
EZACICAL call 427
use in child server 133
use in client 132
use in server 134

COBOL language

call basics 20
call format 424
choosing EZACICAL or Sockets Extended API 421
compilation JCL 421
EZACICAL API 423, 453

socket API calls (EZACICAL, SOKETS)

ACCEPT 425
BIND 426
CLOSE 427
CONNECT 428
FCNTL 429
GETCLIENTID 429
GETHOSTID 431
GETHOSTNAME 431
GETPEERNAME 433
GETSOCKNAME 433
GIVESOCKET 436
INITAPI 437
IOCTL 439
LISTEN 440
READ 441
RECVFROM 442
SELECT 443
SEND 445
SENDTO 446
SETSOCKOPT 447
SHUTDOWN 449
SOCKET 450
TAKESOCKET 451
WRITE 452

COBOL language call

EZASOCKET 250

COMMAND parameter on call interface, IOCTL call 324
COMMAND parameter on call socket interface 249
on EZACIC06 408
on FCNTL 269

Communications Server for z/OS, online information xxviii

COMP (COBOL USAGE) 424

concurrent server 129
defined 9
illustrated 9, 10
writing your own 133

configuration macro 52
configuration transaction 68
configuring CICS TCP/IP 23, 52

connect system call

C language 179
EZACICAL call 428
use in client 132

conversion routines 161

CONVERT 68, 74
COPY 78

CSKL transaction 141

CSKL transaction, defining in CICS 28
D

data conversion 161
data sets, modifying 68
data translation, socket interface 249, 401
  ASCII to EBCDIC 405, 418
  bit-mask to character 407
  character to bit-mask 407
  EBCDIC to ASCII 404, 416
DEFINE 80
DELETE 86
Destination Control Table 38
DFHSRT macroinstruction types 47
disability 673
DISPLAY 88
DNS
  EZACIC25, adding to RDO 29
  DNS, online information xxix
domain
  address family 137
  parameter in socket call 243
Domain Name Server cache
  cache file 98
  EZACICR macro 98
  initialization module, creating 100

e
EBCDIC data format 161
enhanced listener
  converting to 68, 74
  parameters 59
  temporary storage 27
environmental support 125
ERETMMSK parameter on call interface, on SELECT 357
ERRNO parameter on call socket interface (continued)
on ACCEPT 257
on BIND 259
on CLOSE 264
on CONNECT 267
on FCNTL 269
on FREEADDRINFO 271
on GETADDRINFO 280
on GETCLIENTID 282
on GETHOSTNMAE 289
on GETNAMEINFO 294
on GETPEERNAME 296
on GETSOCKNAME 298
on GETSOCKOPT 300
on GIVESOCKET 316
on INITAPI 322
on IOCtl 333
on LISTEN 335
on NTOP 337
on PTON 340
on READ 341
on READV 343
on RECV 346
on RECVFROM 349
on RECEVMSG 353
on SELECT 358
on SELECTEX 364
on SEND 366
on SENDMSG 371
on SENDTO 374
on SETSOCKOPT 376
on SHUTDOWN 393
ERRO parameter on call socket interface (continued)
on SOCKET 395
on TAKESOCKET 396
on WRITE 399
on WRITEV 401
errno variable 170
error check option 211
ESDNMSK parameter on call interface, on SELECT 357
event monitoring
  for listener 42
  for TRUE 40
EWOULDBLOCK error return, call interface calls
  RECV 344
  RECVFROM 346
EXEC CICS LINK 122
EXEC CICS RETRIEVE 140
EXEC CICS START 140
EZAC (configuration transaction) 68
EZAC start screen 115
EZACACHE, defining to RDO 37
EZACIC04, call interface, EBCDIC to ASCII translation 404
EZACIC05, call interface, ASCII to EBCDIC translation 405
EZACIC06 17
EZACIC06, call interface, bit-mask translation 407
EZACIC08, HOSTENT structure interpreter utility 409
EZACIC09, ADDRINFO structure interpreter utility 412
EZACIC14, call interface, EBCDIC to ASCII translation 416
EZACIC15, call interface, ASCII to EBCDIC translation 418
EZACICALC6 sample 595
EZACICALS sample 604
EZACICALC sample 623
EZACICAL 421
EZACICAL API 423, 453
EZACICAL program 423
EZACICALS sample 630
EZACICID (configuration macro) 52
EZACICIR macro 98, 100
EZACICALC0 sample 571
EZACICSE program 152
EZACICALS sample 577
EZACICALxx programs
  defining in CICS 28
  EZACICAL0 29
  EZACICAL01 30
  EZACICAL02 30
  EZACICAL03 35
  EZACICAL07 35
  EZACICAL20 30
    PLT entries 47
  EZACICAL21 30
  EZACICAL22 31
  EZACICAL23 31
  EZACICAL24 31
  EZACICAL25
    defining in RDO 31
    Domain Name Server cache 98
  EZACICAL 35
  EZACICALM 32
  EZACICALME 32
  EZACICALSC 33
  EZACICALSS 33
  summary 29
EZACONFG, defining to RDO 37
EZAO transaction
  defining in CICS 28
  manual startup/shutdown 108

Index 693
EZAP transaction defining in CICS 28
EZASOCKET 45, 156, 250

F
FCNTL (call) 268
fcntl system call
C language 181, 182
EZACICAL call 429
files, defining to RDO 36
EZACACHE 37
EZACONFG 37
FLAGS parameter on call socket interface 249
on RECV 345
on RECVFROM 347
on RECVMSG 353
on SEND 366
on SENDMSG 371
on SENDTO 373
FNDELAY flag on call interface, on FCNTL 269
FREEADDRINFO (call) 270
Functions
ALTER 69
CONVERT 74
COPY 78
DEFINE 80
DELETE 86

G
gai_strerror system call
C language 183
GETADDRINFO (call) 271
getaddrinfo system call
C language 184
GETCLIENTID (call) 280
getclientid system call
C language 191
EZACICAL call 429
use in server 133, 140
GETHOSTBYADDR (call) 282
GETHOSTBYNAME (call) 285
GETHOSTID (call) 288
gethostid system call
C language 193
EZACICAL call 431
GETHOSTNAME (call) 288
gethostname system call
C language 191, 192, 194
EZACICAL call 431
GETNAMEINFO (call) 290
getnameinfo system call
C language 197
GETPEERNAME (call) 294
getpeername system call
C language 199
EZACICAL call 433
GETSOCKNAME (call) 296
getsockname system call
C language 201, 213
EZACICAL call 433
GETSOCKOPT (call) 298
getsockopt system call
C language 203
EZACICAL call 435
GIVESOCKET (call) 314
givesocket system call
C language 215
EZACICAL call 436
use in server 134, 140
group_req structure 169
group_source_req structure 170

H
hiq.PROFILE.TCPIP data set 50
hiq.TCPIP.DATA data set 51
HOSTADD parameter on call interface, on GETHOSTBYADDR 283
HOSTENT parameter on socket call interface
on GETHOSTBYADDR 283
on GETHOSTBYNAME 286
HOSTENT structure interpreter parameters, on EZACICAL 411
HOW parameter on call interface, on SHUTDOWN 393

I
IBM Software Support Center, contacting xxi
IDENT parameter on call interface, INITAPI call 321
if_freenameindex system call
C language 216
if_indexeindex system call
C language 217
If_NameIndex C structure 168
if_nameindex system call
C language 218
if_conftime structure 167
ifreq C structure 168
immediate-no 120
immediate-yes 120
IN-BUFFER parameter on call interface, EZACICAL call 407
inet_ntop system call
C language 220
inet_aton system call
C language 221
Information APARs xxvi
initapi system call
C language 224
EZACICAL call 437
use in client 131
use in server 133
INITAPI(call) 319
INITAPIX 319
installing CICS TCP/IP 23
Internet, finding z/OS information online xxviii
Internets, TCP/IP 2
interval control 143
IOCTL (call) 322
ioctl system call
C language 225
EZACICAL call 439
IOV parameter on call socket interface 249
on READV 343
on WRITEV 400
IOVCNT parameter on call socket interface 249
on READV 343
on RECVMSG 352
on SENDMSG 370
IOVCNT parameter on call socket interface (continued) on WRITEV 400
IP protocol 3
ip_mreq C structure 168
ip_mreq_source structure 169
ipv6_mreq C structure 168
iterative server defined 9 illustrated 10, 130 socket calls in 134

J
JCL jobs 65 for C compilation 166 for CICS startup 23, 25 for COBOL compilation 421 for DNS cache file 101

K
keyboard 673

L
LENGTH parameter on call socket interface 249 on EZACIC04 405 on EZACIC05 407 on EZACIC14 418 on EZACIC15 420 license, patent, and copyright information 677 linger C structure 168 linger on close option 211 link, program 122 LISTEN (call) 334 listen system call C language 229 EZACICAL call 440 use in server 133

listener enhanced converting to 68, 74 parameters 59 temporary storage 27 input format 142 monitor control table 42 output format 144 security or transaction modules 152 standard converting to enhanced listener 68, 74 parameters 59 starting and stopping 141, 156 user-written 125 listener/server call sequence 132 listener/server, socket call (general) 133 little endian 139

M
macro, EZACICR 98 macros, address testing 245 mainframe education xxvi manifest.h C header 165 manual startup 108 MAXFILEPROC 62, 96 MAXSNO parameter on call interface, INITAPI call 322 MAXSOC parameter on call socket interface 249 on INITAPI 321 on SELECT 356 on SELECTEX 362 messages, sockets 475 modifying data sets 68 Monitor Control Table for listener 43 monitoring, event for listener 42 for TRUE 40 MSG parameter on call socket interface 249 on RECVMSG 351 on SENDMSG 369 MVS address spaces 138

N
NAME parameter on socket call interface on ACCEPT 256 on BIND 259 on CONNECT 266 on GETHOSTBYNAME 286 on GETHOSTNAME 289 on GETPEERNAME 295 on GETSOCKNAME 297 on RECVFROM 348 on SENDTO 374 NAMLEN parameter on socket call interface on GETHOSTBYNAME 286 on GETHOSTNAME 289 NBYTE parameter on call socket interface 249 on READ 341 on RECV 345 on RECVFROM 348 on SEND 366 on SENDTO 374 on WRITE 399 NetConfHdr C structure 168 network byte order 139 NTOP (call) 336

O
OPTNAME parameter on call socket interface 249 OPTVAL parameter on call socket interface 249 original COBOL application programming interface (API) 421, 453 OSI 3 OUT-BUFFER parameter on call interface, on EZACIC04 405 OUT-BUFFER parameter on call interface, on EZACIC14 418 OUT-BUFFER parameter on call interface, on EZACIC15 420 out-of-band data options in get/setsockopt call 212 sending with send call 237

P
passing sockets 134 pending activity 16 pending exception 17 pending read 17 PL/I programs, required statement 252 PLT 107

Index 695
port numbers
reserving port numbers
ports
compared with sockets
reserving port numbers
prerequisite information
program link
Program List Table
program variable definitions, call interface
assembler definition
COBOL PIC
PL/I declare
VS COBOL II PIC
programs, defining in CICS
programs, sample
PROTO parameter on call interface, on SOCKET
protocol parameter in socket call
PTON (call)
return codes
call interface
reuse local address option
RFC (request for comments)
accessing online
RRETMSK parameter on call interface, on SELECT
RSNDMSK parameter on call interface, on SELECT
S, defines socket descriptor on socket call interface
on ACCEPT
on BIND
on CLOSE
on CONNECT
on EZACIC06
on FCNTL
on FREEADDRINFO
on GETADDRINFO
on GETCLIENTID
on GETHOSTBYADDR
on GETHOSTBYNAME
on GETHOSTID
on GETHOSTNAME
on GETNAMEINFO
on GETPEERNAME
on GETSOCKNAME
on GETSOCKOPT
RET CODE parameter on call socket interface (continued)
on GIVESOCKET
on INITAPI
on IOCTL
on LISTEN
on NTOPT
on PTON
on READ
on READV
on RECV
on RECVFROM
on RECVMSG
on SELECT
on SELECTEX
on SEND
on SENDMSG
on SENDTO
on SETSOCKOPT
on SOCKET
on TAKESOCKET
on WRITE
on WRITEV
sample programs
security or transaction modules
SELECT (call)
select mask
select system call
C language
EZACICAL call
use in server
select system call
C language
EZACICAL call
use in server
SELECT sample
SELECTEX sample
SEND (call) 364
send system call
  C language 236
  EZACICAL call 445
SENDMSG (call) 367
SENDTO (call) 371
sendto system call
  C language 238
  EZACICAL call 446
server
definition 2
  socket calls in child server 132
  socket calls in concurrent server 133
  socket calls in iterative server 134
SetADContainer structure 170
SetAppIData structure 170
SETSOCKOPT (call) 375
setsockopt system call
  C language 203
  EZACICAL call 447
shortcut keys 673
SHUTDOWN (call) 391
shutdown system call
  C language 242
  EZACICAL call 449
shutdown, immediate 120
shutdown, manual 108
SNA protocols and CICS 1
SOCK_STREAM type parameter 243
sockaddr_in C structure
  format 169
  use in accept call 172
  use in bind call 174
  use in connect call 180
sockaddr_in6 C structure 169
SOCKET (call) 393
socket call interface
  on ACCEPT 256
  on BIND 258
  on CLOSE 264
  on CONNECT 266
  on FCNTL 269
  on GETPEERNAME 295
  on GETSOCKNAME 297
  on GETSOCKOPT 299
  on GIVESOCKET 315
  on IOCTL 324
  on LISTEN 335
  on READ 341
  on READV 343
  on RECV 345
  on RECVFROM 347
  on RECVMSG 351
  on SEND 366
  on SENDMSG 369
  on SENDTO 373
  on SETSOCKOPT 376
  on SHUTDOWN 393
  on WRITE 399
  on WRITEV 400
socket system call 242
  EZACICAL call 450
  use in client 132
  use in server 133
sockets
  compared with ports 7
  introduction 4
sockets (continued)
  passing 134
Sockets Extended API 4
sockets messages 475
sockets SPI blocking call 123
SOCRECV parameter on call interface, TAKESOCKET call 396
SOCTYPE parameter on call interface, on SOCKET 394
softcopy information xxvi
SRT 47
standard listener
  converting to enhanced listener 68, 74
  parameters 59
startup
  automatic 107
  manually 108
  program link 122
storage protection machines 27, 29
stub program 19
subtotal 19
SUBTASK parameter on call interface, INITAPI call 322
support, environmental 125
text diagram, how to read xxii
system recovery table 47
system services segment, adding a z/OS UNIX system services 52

T
TAKESOCKET (call) 395
takesocket system call
  C language 244
  EZACICAL call 451
  use in child server 132, 140
task control 143
Task Hangs 123
task-related user exit 19
TCP protocol 3
TCP_NODELAY 204, 211
TCP/IP
  online information xxviii
  protocol specifications 649
TCP/IP protocols 3
TCP/IP services, modifying data sets 50
TCP/IP, compared with SNA 1
TCPIP.DATA data set 51
TCPJOBNAME user id 51
TCPM td queue 38
Technotes xxvi
TERMAPI (call) 397
TIMEOUT parameter on call interface, on SELECT 356
TIMEOUT parameter on call socket interface 249
  on SELECTEX 362
timeval structure 169
trademark information 685
transaction identifier 142
transactions, defining in CICS 27
transient data 38
TRUE module
  description 19
  monitor control table 40
type (of socket) option 213
type parameter 54
  TYPE=CICS 55
  TYPE=INITIAL 55
  TYPE=LISTENER 59

Index 697
type parameter in socket call 243

U
UDP protocol 3
UNIX System Services 62
use of ADDRINFO structure interpreter, EZACIC09 412
use of HOSTENT structure interpreter, EZACIC08 409
utility programs 249, 401
  EZACIC04 404
  EZACIC05 405
  EZACIC06 407
  EZACIC08 409
  EZACIC09 412
  EZACIC14 416
  EZACIC15 418

V
VSAM cache file 98
VTAM, online information xxviii

W
WREMTMSK parameter on call interface, on SELECT 357
WRITE (call) 398
write system call
  C language 245
    EZACICAL call 452
    use in child server 133
    use in client 132
WRITEV (call) 399
WSNDMSK parameter on call interface, on SELECT 357

Z
z/OS Basic Skills Information Center xxvi
z/OS UNIX System Services 96
z/OS UNIX System Services — adding a z/OS UNIX system
  services segment 52
z/OS, documentation library listing 687
Communicating your comments to IBM

If you especially like or dislike anything about this document, use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this document. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Send your comments to us in any of the following ways:

- To send comments by FAX, use this number: 1+919-254-1258
- To send comments electronically, use this address: comsvrcf@us.ibm.com
  - To send comments by post, use this address:
    International Business Machines Corporation
    Attn: z/OS Communications Server Information Development
    P.O. Box 12195, 3039 Cornwallis Road
    Department AKCA, Building 501
    Research Triangle Park, North Carolina 27709-2195

Make sure to include the following information in your note:

- Title and publication number of this document
- Page number or topic to which your comment applies