Note:
Before using this information and the product it supports, be sure to read the general information under "Notices" on page 763.

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

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

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

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

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

World Wide Web:

If you would like a reply, be sure to include your name, address, telephone number, or FAX number. Make sure to include the following in your comment or note:
• Title and order number of this document
• Page number or topic related to your comment

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

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

## Figures ................................... xv

## Tables ................................... xvii

## About this document ............................. xx

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

## Summary of changes .................................... xxx

## Chapter 1. General programming information ................... 1

**Overview of Distributed Protocol Interface (DPI) versions 1.1 and 2.0** ................................ 1

## Chapter 2. SNMP agent Distributed Protocol Interface version 1.1. ....... 3

**SNMP agents and subagents** .................................. 3
**Processing DPI requests** ...................................... 4
  - Processing a GET request ........................................ 4
  - Processing a SET request ........................................ 4
  - Processing a GET-NEXT request ................................. 4
  - Processing a REGISTER request ............................... 5
  - Processing a TRAP request ................................... 5
**SNMP agent DPI library routines** ................................. 6
  - mkDPIlist() ........................................... 7
  - fDPIparse() ........................................... 7
  - mkDPIregister() ....................................... 8
  - mkDPIresponse() ....................................... 8
  - mkDPIset() ........................................... 9
  - mkDPItrap() .......................................... 10
  - mkDPItrape() ......................................... 11
  - pDPIpacket() .......................................... 11
  - query_DPI_port() ...................................... 13
**Sample SNMP DPI client program for C sockets for version 1.1** ................................ 13
  - Using the DPISAMPL program .................................. 13
  - DPISAMPN NCCFLST for the SNMP manager ...................... 15
  - Compiling and linking the DPISAMPL.C source code ............ 16
  - dpiSample table MIB descriptions ............................. 16
  - The DPISAMPL.C source code .................................. 16

## Chapter 3. SNMP agent Distributed Protocol Interface version 2.0 ........ 41

**SNMP agents and subagents** .................................. 41
**DPI agent requests** ........................................... 42
**SNMP DPI version 2.0 library** ................................ 43
**SNMP DPI Version 2.0 API** ................................... 43
Compiling and linking DPI Version 2.0 ................................................................. 44
Compiling and linking DPI Version 2.0: UNIX System Services environment ............. 44
Compiling and linking DPI Version 2.0: MVS environment ........................................ 45
DPI Version 1.x base code considerations .............................................................. 45
Migrating your SNMP DPI subagent to Version 2.0 .................................................. 45
  Required actions for migrating your SNMP DPI subagent to Version 2.0 ............... 45
  Recommended actions for migrating your SNMP DPI subagent to Version 2.0 .......... 46
  snmp_dpi_xxxx_packet structures name changes .............................................. 47
SNMP DPI environment variables ............................................................................. 48
SNMP DPI subagent programming concepts .............................................................. 48
Specifying the SNMP DPI API ................................................................................. 49
DPI subagent connect processing .............................................................................. 49
DPI subagent OPEN request ....................................................................................... 49
DPI subagent REGISTER request ............................................................................... 50
DPI subagent GET processing ..................................................................................... 51
DPI subagent SET processing ..................................................................................... 52
DPI subagent GETNEXT processing .......................................................................... 53
DPI subagent GETBULK processing request ............................................................. 54
DPI subagent TRAP request ....................................................................................... 54
DPI subagent ARE_YOU_THERE request .................................................................. 55
DPI subagent UNREGISTER request .......................................................................... 55
DPI subagent CLOSE request ..................................................................................... 56
Multithreading programming considerations ........................................................... 56
Functions, data structures, and constants ................................................................... 57
Basic DPI API functions ........................................................................................... 58
  The DPIdebug() function ....................................................................................... 59
  The DPI_PACKET_LEN() macro ............................................................................ 60
  The fDPIparse() function ....................................................................................... 61
  The fDPIset() function .......................................................................................... 62
  The mkDPIAreYouThere() function ....................................................................... 63
  The mkDPIclose() function .................................................................................... 64
  The mkDPIopen() function ..................................................................................... 65
  The mkDPIregister() function .............................................................................. 67
  The mkDPIresponse() function ............................................................................. 69
  The mkDPIset() function ....................................................................................... 71
  The mkDPItrap() function ..................................................................................... 73
  The mkDPIunregister() function ........................................................................... 75
  The pDPIpacket() function .................................................................................... 76
Transport-related DPI API functions ......................................................................... 77
  The DPIawait_packet_from_agent() function ..................................................... 78
  The DPIconnect_to_agent_TCP() function ......................................................... 80
  The DPIconnect_to_agent_UNIXstream() function ............................................. 82
  The DPIdisconnect_from_agent() function ........................................................ 84
  The DPIget_fd_for_handle() function .................................................................. 85
  The DPIsend_packet_to_agent() function ........................................................... 86
  The lookup_host() function ................................................................................ 88
  The lookup_host6() function ............................................................................... 89
DPI structures ............................................................................................................. 90
  The snmp_dpi_close_packet structure ................................................................ 91
  The snmp_dpi_get_packet structure ..................................................................... 92
  The snmp_dpi_hdr structure .................................................................................. 93
  The snmp_dpi_next_packet structure .................................................................. 95
  The snmp_dpi_resp_packet structure ................................................................... 96
  The snmp_dpi_set_packet structure ..................................................................... 97
  The snmp_dpi_ureg_packet structure .................................................................... 99
  The snmp_dpi_u64 structure .................................................................................. 100
DPI OPEN character set selection .......................................................................... 101
SNMP DPI constants, values, return codes, and include file .................................. 101
DPI CLOSE reason codes ........................................................................................... 102
DPI packet types ....................................................................................................... 102
DPI RESPONSE error codes ...................................................................................... 102
DPI UNREGISTER reason codes ........................................................................ 103
DPI SNMP value types .................................................................................. 103
Value representation of DPI SNMP value types ........................................ 104
Value ranges and limits for DPI SNMP value types ................................... 105
Return codes from DPI transport-related functions .................................... 105
The snmp_dpi.h include file ......................................................................... 106
    snmp_dpi.h include parameters .............................................................. 106
    snmp_dpi.h include description .............................................................. 106
DPI subagent example .................................................................................. 107
Overview of subagent processing ............................................................... 107
SNMP DPI: Connecting to the agent ........................................................... 109
SNMP DPI: Registering a subtree with the agent ......................................... 111
SNMP DPI: Processing requests from the agent .......................................... 113
SNMP DPI: Processing a GET request ....................................................... 116
SNMP DPI: Processing a GETNEXT request ............................................. 119
SNMP DPI: Processing a SET/COMMIT/UNDO request ............................ 122
SNMP DPI: Processing an UNREGISTER request ........................................ 125
SNMP DPI: Processing a CLOSE request .................................................. 126
SNMP DPI: Generating a TRAP ................................................................. 126

Chapter 4. Running the sample SNMP DPI client program for version 2.0 . . . 129
Using the sample SNMP DPI client program .............................................. 129
Compiling and linking the dpi_mvs_sample.c source code .......................... 129
DPI Simple-MIB descriptions ..................................................................... 130

Chapter 5. SNMP manager API ................................................................. 131
SNMP protocol ............................................................................................ 131
The SNMP manager API overview ............................................................ 132
The SNMP notification API overview ......................................................... 132
SNMP manager API functions ................................................................... 132
    snmpAddVarBind – Adds a VarBind to the SnmpVarBinds structure ........ 133
    snmpBuildPDU – Builds an SNMP PDU .................................................. 133
    snmpBuildSession – Creates a session ................................................... 135
    snmpCreateVarBinds – Creates a VarBind structure ......................... 136
    snmpFreeDecodedPDU - Free the decoded PDU ................................. 137
    snmpFreeOID - Free an OID string ...................................................... 137
    snmpFreePDU – Frees the resources of a PDU ...................................... 137
    snmpFreeVarBinds – Frees the VarBinds structure ............................ 138
    snmpGetLastErrorInfo - Get the error information from the PDU response 138
    snmpGetNumberOfVarBinds – Get the number of VarBinds attached to the PDU 139
    snmpGetOID – Get the OID from the VarBind structure .................. 139
    snmpGetRequestId – Get the PDU’s requestId value ........................ 140
    snmpGetSockFd – Get the socket’s file descriptor ............................... 140
    snmpGetValue – Get the value from the VarBind structure ................ 140
    snmpInitialize – Initialize the manager environment ..................... 142
    snmpSendRequest – Send the snmpPDU request to an agent .............. 143
    snmpSetLogFunction – Set the logging level ....................................... 144
    snmpSetLogLevel – Set the logging level ........................................... 145
    snmpSetRequestId – Set the PDU’s requestId value .......................... 145
    snmpTerminate – Release the resources .............................................. 146
    snmpTerminateSession – Terminate a session ...................................... 146
    snmpValueCreateBits – Create an smiValue of type Bits ................... 147
    snmpValueCreateCounter32 – Create an smiValue of type Counter32 ... 147
    snmpValueCreateCounter64 – Create an smiValue of type Counter64 ... 148
    snmpValueCreateGauge32 – Create an smiValue of type Gauge32 ....... 148
    snmpValueCreateInteger – Create an smiValue of type Integer .......... 148
    snmpValueCreateInteger32 – Create an smiValue of type Integer32 .... 149
    snmpValueCreateIPAddr – Create an smiValue of type IPAddr ............. 149
    snmpValueCreateNSApAddr – Create an smiValue of type NSApAddr .... 150
Chapter 6. Resource Reservation Setup Protocol API (RAPI) ........................................ 161
  API outline ........................................................................................................ 162
  Compiling and linking RAPI applications ....................................................... 162
  Running RAPI applications ........................................................................... 162
  Event upcall ...................................................................................................... 163
    rapi_event_rtn_t - Event upcall .................................................................... 163
  Client library services ..................................................................................... 165
    rapi_release - Remove a session ................................................................... 166
    rapi_reserve - Make, modify, or delete a reservation .................................... 166
    rapi_sender - Specify sender parameters .................................................... 167
    rapi_session - Create a session .................................................................... 169
    rapi_version - RAPI version ......................................................................... 170
  RAPI formatting routines ................................................................................ 170
    rapi_fmt_adspec - Format an adspec ............................................................ 170
    rapi_fmt_filtspec - Format a filtspec ......................................................... 171
    rapi_fmt_flowspec - Format a flowspec ..................................................... 172
    rapi_fmt_tspec - Format a tspec .................................................................. 173
  RAPI objects ..................................................................................................... 173
    RAPI objects - Flowspecs .............................................................................. 174
    RAPI objects - Sender tspecs ....................................................................... 174
    RAPI objects - Adspecs ................................................................................ 174
    RAPI objects - Filter specs and sender templates ........................................ 175
  RAPI asynchronous event handling ................................................................. 175
    rapi_dispatch - Dispatch API event ............................................................. 176
    rapi_getfd - Get file descriptor .................................................................... 177
  RAPI error handling ........................................................................................ 177
    RAPI error codes .......................................................................................... 178
    RSVP error codes ......................................................................................... 179
  RAPI header files ............................................................................................ 179
    RAPI header files: Integer and floating point types ...................................... 179
    The <rapi.h> header ..................................................................................... 180
    Integrated services data structures and macros .......................................... 186

Chapter 7. X Window System interface in the z/OS Communications Server
  environment ...................................................................................................... 195
  X Window System and Motif .......................................................................... 195
  DLL support for the X Window System .......................................................... 196
  How the X Window System interface works in the MVS environment .......... 197
  X Window System programming considerations .......................................... 198
  Running an X Window System or Motif DLL-enabled application ............... 199
  X Window System environment variables .................................................... 200
  Motif environment variables ........................................................................ 201
  EBCDIC/ASCII translation in the X Window System ..................................... 202
  Standard clients supplied with MVS z/OS UNIX X Window System support ... 202
Chapter 8. Remote procedure calls in the z/OS Communications Server environment 207
The RPC interface ........................................... 207
Portmapper and rpcbind .................................... 209
  Contacting portmapper or rpcbind ................... 210
  Portmapper and rpcbind target assistance ........... 210
  Registering with rpcbind .............................. 211
  Deregistering with rpcbind ............................ 212
  Obtaining address lists from the rpcbind server ..... 212
RPC servers in a CINET environment .................. 213
RPCGEN command ........................................... 214
clnt_stat enumerated type ............................... 216
Porting RPC applications .................................. 216
  Remapping file names with MANIFEST.H .............. 216
  Accessing system return messages .................... 217
  Printing system return messages ..................... 217
  Enumerations .......................................... 217
  Header files for remote procedure calls .............. 217
Compiling and linking RPC applications ............... 217
  Compatibility considerations when compiling and linking RPC applications ....... 217
  Sample compile cataloged procedure additions ....... 218
  Compiling and linking RPC applications: Nonreentrant modules ...... 218
  Compiling and linking RPC applications: Reentrant modules ...... 218
RPC global variables ....................................... 218
  rpc_createerr .......................................... 219
  svc_fds .................................................. 220
  svc_fdset ............................................... 221
Remote procedure and external data representation calls 222
  auth_destroy() ......................................... 223
  authnone_create() ...................................... 224
  authunix_create() ...................................... 225
  authunix_create_default() ............................. 226
callrpc() .................................................. 227
clnt_broadcast() .......................................... 229
clnt_call() ............................................... 231
clnt_control() ............................................ 233
clnt_create() ............................................. 234
clnt_destroy() ............................................ 235
clnt_freeres() ............................................. 236
clnt_geterr() ............................................. 237
clnt_pcreateerror() ....................................... 238
clnt_permo() .............................................. 239
clnt_perror() ............................................. 240
clnt_spcreateerror() ..................................... 241
clnt_spermo() ............................................. 242
clnt_sperro() ............................................. 243
clnttcp_create() ......................................... 244
clntudp_create() ......................................... 245
clntraw_create() ......................................... 247
get_myaddress() .......................................... 249
getrpcport() .............................................. 250
pmap_getmaps() ........................................... 251
pmap_getport() ........................................... 252
pmap_rmtcall() ........................................... 253
pmap_set() ................................................ 255
pmap_unset() ............................................. 256
registerrpc() ............................................ 257
svc_destroy() ............................................. 259
svc_freeargs() ............................................ 260
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>svc_getargs()</td>
<td>261</td>
</tr>
<tr>
<td>svc_getcaller()</td>
<td>262</td>
</tr>
<tr>
<td>svc_getreq()</td>
<td>263</td>
</tr>
<tr>
<td>svc_getreqset()</td>
<td>264</td>
</tr>
<tr>
<td>svc_register()</td>
<td>265</td>
</tr>
<tr>
<td>svc_run()</td>
<td>267</td>
</tr>
<tr>
<td>svc_sendreply()</td>
<td>268</td>
</tr>
<tr>
<td>svc_unregister()</td>
<td>269</td>
</tr>
<tr>
<td>svcerr_auth()</td>
<td>270</td>
</tr>
<tr>
<td>svcerr_decode()</td>
<td>271</td>
</tr>
<tr>
<td>svcerr_noproc()</td>
<td>272</td>
</tr>
<tr>
<td>svcerr_noprogs()</td>
<td>273</td>
</tr>
<tr>
<td>svcerr_progvers()</td>
<td>274</td>
</tr>
<tr>
<td>svcerr_systemerr()</td>
<td>275</td>
</tr>
<tr>
<td>svcerr_weakauth()</td>
<td>276</td>
</tr>
<tr>
<td>svcraw_create()</td>
<td>277</td>
</tr>
<tr>
<td>svctcp_create()</td>
<td>278</td>
</tr>
<tr>
<td>svctcp_create()</td>
<td>279</td>
</tr>
<tr>
<td>xdr_accepted_reply()</td>
<td>280</td>
</tr>
<tr>
<td>xdr_array()</td>
<td>281</td>
</tr>
<tr>
<td>xdr_authunix_parms()</td>
<td>282</td>
</tr>
<tr>
<td>xdr_bool()</td>
<td>283</td>
</tr>
<tr>
<td>xdr_bytes()</td>
<td>284</td>
</tr>
<tr>
<td>xdr_callspace()</td>
<td>285</td>
</tr>
<tr>
<td>xdr_callmsg()</td>
<td>286</td>
</tr>
<tr>
<td>xdr_char()</td>
<td>287</td>
</tr>
<tr>
<td>xdr_destroy()</td>
<td>288</td>
</tr>
<tr>
<td>xdr_double()</td>
<td>289</td>
</tr>
<tr>
<td>xdr_enum()</td>
<td>290</td>
</tr>
<tr>
<td>xdr_float()</td>
<td>291</td>
</tr>
<tr>
<td>xdr_free()</td>
<td>292</td>
</tr>
<tr>
<td>xdr_getpos()</td>
<td>293</td>
</tr>
<tr>
<td>xdr_inline()</td>
<td>294</td>
</tr>
<tr>
<td>xdr_int()</td>
<td>295</td>
</tr>
<tr>
<td>xdr_long()</td>
<td>296</td>
</tr>
<tr>
<td>xdr_opaque()</td>
<td>297</td>
</tr>
<tr>
<td>xdr_opaque_auth()</td>
<td>298</td>
</tr>
<tr>
<td>xdr_pmap()</td>
<td>299</td>
</tr>
<tr>
<td>xdr_pmaplist()</td>
<td>300</td>
</tr>
<tr>
<td>xdr_pointer()</td>
<td>301</td>
</tr>
<tr>
<td>xdr_reference()</td>
<td>302</td>
</tr>
<tr>
<td>xdr_rejected_reply()</td>
<td>303</td>
</tr>
<tr>
<td>xdr_replyspace()</td>
<td>304</td>
</tr>
<tr>
<td>xdr_replyspace()</td>
<td>305</td>
</tr>
<tr>
<td>xdr_setpos()</td>
<td>306</td>
</tr>
<tr>
<td>xdr_short()</td>
<td>307</td>
</tr>
<tr>
<td>xdr_string()</td>
<td>308</td>
</tr>
<tr>
<td>xdr_text_char()</td>
<td>309</td>
</tr>
<tr>
<td>xdr_u_char()</td>
<td>310</td>
</tr>
<tr>
<td>xdr_u_int()</td>
<td>311</td>
</tr>
<tr>
<td>xdr_u_long()</td>
<td>312</td>
</tr>
<tr>
<td>xdr_u_short()</td>
<td>313</td>
</tr>
<tr>
<td>xdr_union()</td>
<td>314</td>
</tr>
<tr>
<td>xdr_vector()</td>
<td>315</td>
</tr>
<tr>
<td>xdr_void()</td>
<td>316</td>
</tr>
<tr>
<td>xdr_wrapstring()</td>
<td>317</td>
</tr>
<tr>
<td>xdrwrap_mem_create()</td>
<td>318</td>
</tr>
<tr>
<td>xdrrec_create()</td>
<td>319</td>
</tr>
<tr>
<td>xdrrec_create()</td>
<td>320</td>
</tr>
<tr>
<td>xdrrec_endofrecord()</td>
<td>321</td>
</tr>
<tr>
<td>xdrrec_eof()</td>
<td>322</td>
</tr>
<tr>
<td>xdrrec_skiprecord()</td>
<td>323</td>
</tr>
<tr>
<td>xdrstdio_create()</td>
<td>324</td>
</tr>
</tbody>
</table>
### Chapter 9. Remote procedure calls in the z/OS UNIX System Services environment

- Deviations from Sun RPC 4.0
- Using z/OS UNIX System Services RPC
- Support for 64-bit integers
- UDP transport protocol CLIENT handles
- RPC restrictions

### Chapter 10. Network Computing System

- NCS and the Network Computing Architecture
- NCS components
  - Remote procedure call runtime library
  - Location broker
  - Network interface definition language compiler
- MVS implementation of NCS
- NCS system IDL data sets
- NCS C header data sets and the Pascal include data set
- NCS RPC run-time library
- NCS portability issues
  - NCS defines NCSDEFS.H
  - Required user-defined USERDEFS.H
- NCS: Preprocessing, compiling, and linking
  - NCS preprocessor programs
  - Compiling and linking NCS programs
- Running UUID@GEN
- NCS sample programs
  - The NCSSMP sample program
  - NCS sample redefines
- Compiling, linking, and running the sample BINOP program
  - Setting up the sample BINOP program
  - Compiling the sample BINOP program
  - Linking the sample BINOP program
  - Running the sample BINOP program
- Compiling, linking, and running the NCSSMP program
  - Setting up the NCSSMP program
  - Compiling the NCSSMP program
  - Linking the NCSSMP program
  - Running the NCSSMP program
- Compiling, linking, and running the sample BANK program
  - Setting up the sample BANK program
  - Compiling the sample BANK program
  - Linking the sample BANK program
  - Running the sample BANK program

### Chapter 11. Running the sample mail filter program

- Compiling and linking the lf_smpl.c source code
- Specifying filters in the sendmail configuration file
- Running the sample mail filter program
- Library control functions
Chapter 12. Policy API (PAPI) ........................................ 383
API outline for retrieving data from Policy Agent ................. 383
Compiling and linking PAPI applications ......................... 383
Running PAPI applications ......................................... 384
PAPI return codes .................................................. 384
PAPI client library services ........................................ 385
PAPI: Connecting and retrieving data ................................ 386
papi_connect - Connect to Policy Agent ......................... 386
papi_debug - Set debug capability .............................. 387
papi_disconnect - Disconnect from the Policy Agent ......... 387
papi_free_perf_data - Free retrieved QoS performance data .... 388
papi_get_perf_data - Retrieve QoS performance data .......... 388
PAPI helper functions .............................................. 391
papi_get_action_perf_by_id - Obtain performance information on the action specified by the action ID 391
papi_get_action_perf_info - Obtain performance information on a particular action .......................... 391
papi_get_actions_count - Obtain number of actions in the policy performance data ......................... 392
papi_get_policy_instance - Obtain policy instance number for policies in the policy performance data .... 393
papi_get_rule_perf_by_id - Obtain performance information on the rule specified by the rule ID ...... 393
papi_get_rule_perf_info - Obtain performance information on a particular rule ............................... 394
papi_get_rules_count - Obtain number of rules in the policy performance data ................................. 395
papi_strerror - Return string describing PAPI return code value ........................................................................ 395

Chapter 13. FTP Client Application Programming Interface (API) 397
FTP client API compatibility considerations ....................... 398
FTP client API guidelines and requirements ..................... 398
Java call formats ................................................. 400
COBOL, C, REXX, assembler, and PL/I call formats .......... 401
Converting parameter descriptions ................................ 402
z/OS FTP client behavior when invoked from the FTP client API .... 402
FTP Client Application Interface (FCAI) control block ........ 404
FTP Client Application Interface (FCAI) stem variables ...... 412
Predefined REXX variables ....................................... 413
Sending requests to the FTP client API .......................... 418
<table>
<thead>
<tr>
<th>Chapter 15. Application Transparent Transport Layer Security (AT-TLS)</th>
<th>637</th>
</tr>
</thead>
<tbody>
<tr>
<td>CICS transaction considerations</td>
<td>638</td>
</tr>
<tr>
<td>Using the SIOCTTLSCTL ioctl</td>
<td>639</td>
</tr>
<tr>
<td>Starting AT-TLS on a connection</td>
<td>639</td>
</tr>
<tr>
<td>Stopping AT-TLS on a connection</td>
<td>639</td>
</tr>
<tr>
<td>Requesting AT-TLS queries and additional functions</td>
<td>640</td>
</tr>
<tr>
<td>Steps for implementing an aware server application</td>
<td>640</td>
</tr>
<tr>
<td>Steps for implementing a controlling server application</td>
<td>640</td>
</tr>
<tr>
<td>Coding the SIOCTTLSCTL ioctl</td>
<td>643</td>
</tr>
<tr>
<td>SIOCTTLSCTL (X'C038D90B')</td>
<td>644</td>
</tr>
<tr>
<td>SIOCTTLSCTL ioctl return values</td>
<td>652</td>
</tr>
<tr>
<td>SIOCTTLSCTL ioctl coding examples</td>
<td>654</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Understanding how clients interface to DCAS</td>
<td>657</td>
</tr>
<tr>
<td>Interfacing with the DCAS: Defining the format for request and response specifications</td>
<td>658</td>
</tr>
<tr>
<td>Configuring the DCAS server to work with your solution</td>
<td>661</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 17. Miscellaneous programming interfaces</th>
<th>663</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIOCSAPPLDATA IOCTL</td>
<td>663</td>
</tr>
<tr>
<td>SIOCSAPPLDATA input</td>
<td>663</td>
</tr>
<tr>
<td>SIOCSAPPLDATA output</td>
<td>664</td>
</tr>
<tr>
<td>SIOCSAPPLDATA C language example</td>
<td>664</td>
</tr>
<tr>
<td>SIOCSMOCTL IOCTL</td>
<td>665</td>
</tr>
<tr>
<td>SIOCSMOCTL input</td>
<td>665</td>
</tr>
<tr>
<td>SIOCSMOCTL output</td>
<td>666</td>
</tr>
<tr>
<td>Steps for creating an ancillary socket</td>
<td>667</td>
</tr>
<tr>
<td>Applications in a common INET environment</td>
<td>667</td>
</tr>
<tr>
<td>TCP_KeepAlive socket option</td>
<td>668</td>
</tr>
</tbody>
</table>
## Figures

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SNMP Dist Prog Interface subagent sample</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>Remote procedure call (client)</td>
<td>208</td>
</tr>
<tr>
<td>3</td>
<td>Remote procedure call (server)</td>
<td>209</td>
</tr>
<tr>
<td>4</td>
<td>RPC client program sample</td>
<td>328</td>
</tr>
<tr>
<td>5</td>
<td>RPC server program sample</td>
<td>330</td>
</tr>
<tr>
<td>6</td>
<td>RPC raw data stream program sample</td>
<td>332</td>
</tr>
<tr>
<td>7</td>
<td>Macro to maintain IBM System/370 portability</td>
<td>345</td>
</tr>
<tr>
<td>8</td>
<td>NCSDEFS.H and USERDEFS.H include statements</td>
<td>345</td>
</tr>
<tr>
<td>9</td>
<td>Message header and records</td>
<td>482</td>
</tr>
<tr>
<td>10</td>
<td>NMI monitoring request format</td>
<td>485</td>
</tr>
<tr>
<td>11</td>
<td>NMsec_ACTIVATE_IPTUNMANUAL request format</td>
<td>493</td>
</tr>
<tr>
<td>12</td>
<td>NMsec_ACTIVATE_IPTUNMANUAL request format</td>
<td>494</td>
</tr>
<tr>
<td>13</td>
<td>NMsec_DEACTIVATE_IPTUNMANUAL request format</td>
<td>494</td>
</tr>
<tr>
<td>14</td>
<td>NMsec_DEACTIVATE_IPTUNMANUAL request format</td>
<td>494</td>
</tr>
<tr>
<td>15</td>
<td>NMsec_DEACTIVATE_IKETUN request format</td>
<td>495</td>
</tr>
<tr>
<td>16</td>
<td>NMsec_REFRESH_IPTUNMANUAL request format</td>
<td>495</td>
</tr>
<tr>
<td>17</td>
<td>NMsec_REFRESH_IKETUN request format</td>
<td>495</td>
</tr>
<tr>
<td>18</td>
<td>NMsec_GET_STACKINFO response format</td>
<td>496</td>
</tr>
<tr>
<td>19</td>
<td>NMsec_GET_SUMMARY response format</td>
<td>499</td>
</tr>
<tr>
<td>20</td>
<td>NMsec_GET_IPFLTCURR, NMsec_GET_IPFLTDEFAULT, and NMsec_GET_IPFLTPOICY response format</td>
<td>502</td>
</tr>
<tr>
<td>21</td>
<td>NMsec_GET_PORTTRAN response format</td>
<td>509</td>
</tr>
<tr>
<td>22</td>
<td>NMsec_GET_IPTUNMANUAL response format</td>
<td>509</td>
</tr>
<tr>
<td>23</td>
<td>NMsec_GET_IPTUNMANUALSTACK response format</td>
<td>513</td>
</tr>
<tr>
<td>24</td>
<td>NMsec_GET_IPTUNMANUALSTACK response format</td>
<td>518</td>
</tr>
<tr>
<td>25</td>
<td>NMsec_GET_IKETUN response format</td>
<td>520</td>
</tr>
<tr>
<td>26</td>
<td>NMsec_GET_IKETUNCASCADE response format</td>
<td>524</td>
</tr>
<tr>
<td>27</td>
<td>NMsec_GET_IPINTERFACES response format</td>
<td>525</td>
</tr>
<tr>
<td>28</td>
<td>NMsec_GET_IKENSINFO response format</td>
<td>525</td>
</tr>
<tr>
<td>29</td>
<td>Tunnel control response format</td>
<td>527</td>
</tr>
<tr>
<td>30</td>
<td>NMsec_GET_CLIENTINFO response format</td>
<td>533</td>
</tr>
<tr>
<td>31</td>
<td>CTE layout</td>
<td>583</td>
</tr>
<tr>
<td>32</td>
<td>SIOCTTLSCTL with TTLS_Query_Only</td>
<td>642</td>
</tr>
<tr>
<td>33</td>
<td>SIOCTTLSCTL with TTLS_Init_Connection.</td>
<td>643</td>
</tr>
<tr>
<td>34</td>
<td>SIOCTTLSCTL with TTLS_Reset_Session or TTLS_Reset_Cipher</td>
<td>643</td>
</tr>
<tr>
<td>35</td>
<td>MVS X Window System application to server</td>
<td>687</td>
</tr>
<tr>
<td>36</td>
<td>Resources specified for a typical X Window System application</td>
<td>689</td>
</tr>
</tbody>
</table>
Tables

1. Components of DPI version 2.0 ........................................... 43
2. Environment variables for the SNMP DPI .................................. 48
3. SNMP manager API debug levels ........................................... 159
4. Environment variables for the X Window System interface ............ 200
5. Environment variables for Motif ............................................ 201
6. Callback return values ....................................................... 376
7. PAPI function return codes ................................................ 384
8. Programming requirements for the FTP client API ..................... 398
9. FCAI control block .......................................................... 404
10. FCAI_Version field value ................................................... 406
11. FCAI_TraceIt field value .................................................... 407
12. FCAI_TraceCAPI field value .............................................. 407
13. FCAI_TraceStatus field value .......................................... 407
14. FCAI_Result field value ................................................... 407
15. FCAI_Status field values .................................................. 408
16. FCAI_IE field values ........................................................ 408
17. FCAI stem variables .......................................................... 412
18. Predefined REXX variables ................................................ 413
19. FTP client API for REXX return codes .................................. 442
20. FTP client CREATE request return codes ................................ 443
21. FTP client INIT request return codes .................................. 446
22. FTP client SCMD request return codes .................................. 447
23. FTP client POLL request return codes .................................. 449
24. FTP client GETL_FIND request return codes ......................... 452
25. FTP client GETL_COPY request return codes ......................... 453
26. FTP client SET_TRACE request return codes ......................... 455
27. FTP client SET_REQUEST_TIMER request return codes ............ 456
28. FCAI_Map structure elements ............................................ 457
29. FTP client GET_FCAI_MAP request return codes .................... 458
30. FTP client TERM request return codes .................................. 459
31. FTP client API for REXX trace return codes .......................... 460
32. NMsecMessageHdr structure .............................................. 479
33. Input record descriptor ..................................................... 481
34. Output record descriptor ................................................... 481
35. NMsecRecordHdr structure ................................................ 482
36. NMsecSecDesc structure ................................................... 482
37. NMsecCascadingSecDesc structure ..................................... 483
38. Valid input filter specifications for request types .................... 487
39. NMsecInFilter structure .................................................... 488
40. NMsecTunnel field descriptions ......................................... 493
41. NMsecPolicySource data ................................................... 496
42. NMsecStack structure ........................................................ 496
43. NMsecStackExclAddr structure ........................................... 498
44. NMsecStatistics structure .................................................. 499
45. NMsecIPFilter structure .................................................... 502
46. NMsec_GET_PORTTRAN structure ....................................... 509
47. NMsecIP Tunnel structure .................................................. 509
48. NMsecIPManualTunnel structure ....................................... 512
49. NMsecIPDynamicTunnel structure ..................................... 513
50. NMsecIPDynamicStack structure ...................................... 517
51. NMsecIPDynamicIKE structure ......................................... 519
52. NMsecIKE Tunnel structure ................................................ 521
53. IKE tunnel statistics ........................................................ 523
54. NMsecInterface structure .................................................. 525
55. NMsec_GET_IKENSINFO structure ...................................... 526
<table>
<thead>
<tr>
<th>Number</th>
<th>Section Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>117</td>
<td>TCP KeepAlive time values</td>
<td>669</td>
</tr>
<tr>
<td>118</td>
<td>SO_CLUSTERCONNTYPE socket option benefits</td>
<td>671</td>
</tr>
<tr>
<td>119</td>
<td>TCP well-known port assignments</td>
<td>675</td>
</tr>
<tr>
<td>120</td>
<td>Well-known UDP port assignments</td>
<td>676</td>
</tr>
<tr>
<td>121</td>
<td>Environment variables for X Window System Interface V11r4</td>
<td>696</td>
</tr>
<tr>
<td>122</td>
<td>Building X client modules based on X11 functions</td>
<td>699</td>
</tr>
<tr>
<td>123</td>
<td>Building X client modules based on Xt Intrinsics and Athena Toolkit functions.</td>
<td>700</td>
</tr>
<tr>
<td>124</td>
<td>Opening and closing display</td>
<td>701</td>
</tr>
<tr>
<td>125</td>
<td>Creating and destroying windows</td>
<td>701</td>
</tr>
<tr>
<td>126</td>
<td>Manipulating windows</td>
<td>702</td>
</tr>
<tr>
<td>127</td>
<td>Changing window attributes</td>
<td>702</td>
</tr>
<tr>
<td>128</td>
<td>Obtaining window information</td>
<td>703</td>
</tr>
<tr>
<td>129</td>
<td>Properties and atoms</td>
<td>703</td>
</tr>
<tr>
<td>130</td>
<td>Manipulating window properties</td>
<td>703</td>
</tr>
<tr>
<td>131</td>
<td>Setting window selections</td>
<td>704</td>
</tr>
<tr>
<td>132</td>
<td>Manipulating colormaps</td>
<td>704</td>
</tr>
<tr>
<td>133</td>
<td>Manipulating color cells</td>
<td>704</td>
</tr>
<tr>
<td>134</td>
<td>Creating and freeing pixmaps</td>
<td>704</td>
</tr>
<tr>
<td>135</td>
<td>Manipulating graphics contexts</td>
<td>704</td>
</tr>
<tr>
<td>136</td>
<td>Clearing and copying areas</td>
<td>705</td>
</tr>
<tr>
<td>137</td>
<td>Drawing lines</td>
<td>706</td>
</tr>
<tr>
<td>138</td>
<td>Filling areas</td>
<td>706</td>
</tr>
<tr>
<td>139</td>
<td>Loading and freeing fonts</td>
<td>706</td>
</tr>
<tr>
<td>140</td>
<td>Querying character string sizes</td>
<td>707</td>
</tr>
<tr>
<td>141</td>
<td>Drawing text</td>
<td>707</td>
</tr>
<tr>
<td>142</td>
<td>Transferring images</td>
<td>707</td>
</tr>
<tr>
<td>143</td>
<td>Manipulating cursors</td>
<td>708</td>
</tr>
<tr>
<td>144</td>
<td>Handling window manager functions</td>
<td>708</td>
</tr>
<tr>
<td>145</td>
<td>Manipulating keyboard settings</td>
<td>709</td>
</tr>
<tr>
<td>146</td>
<td>Controlling the screen saver</td>
<td>709</td>
</tr>
<tr>
<td>147</td>
<td>Manipulating hosts and access control</td>
<td>710</td>
</tr>
<tr>
<td>148</td>
<td>Handling events</td>
<td>710</td>
</tr>
<tr>
<td>149</td>
<td>Enabling and disabling synchronization.</td>
<td>710</td>
</tr>
<tr>
<td>150</td>
<td>Using default error handling</td>
<td>711</td>
</tr>
<tr>
<td>151</td>
<td>Communicating with window managers</td>
<td>711</td>
</tr>
<tr>
<td>152</td>
<td>Manipulating keyboard event functions</td>
<td>712</td>
</tr>
<tr>
<td>153</td>
<td>Manipulating regions</td>
<td>713</td>
</tr>
<tr>
<td>154</td>
<td>Using cut and paste buffers</td>
<td>713</td>
</tr>
<tr>
<td>155</td>
<td>Querying visual types</td>
<td>714</td>
</tr>
<tr>
<td>156</td>
<td>Manipulating images</td>
<td>714</td>
</tr>
<tr>
<td>157</td>
<td>Manipulating bit maps</td>
<td>714</td>
</tr>
<tr>
<td>158</td>
<td>Using the resource manager</td>
<td>715</td>
</tr>
<tr>
<td>159</td>
<td>Manipulating display functions</td>
<td>715</td>
</tr>
<tr>
<td>160</td>
<td>Extension routines</td>
<td>718</td>
</tr>
<tr>
<td>161</td>
<td>MIT extensions to X</td>
<td>718</td>
</tr>
<tr>
<td>162</td>
<td>Associate table functions</td>
<td>720</td>
</tr>
<tr>
<td>163</td>
<td>Miscellaneous utility routines</td>
<td>720</td>
</tr>
<tr>
<td>164</td>
<td>Authorization routines</td>
<td>722</td>
</tr>
<tr>
<td>165</td>
<td>X Intrinsic header file names</td>
<td>724</td>
</tr>
<tr>
<td>166</td>
<td>Xt Intrinsics routines</td>
<td>724</td>
</tr>
<tr>
<td>167</td>
<td>Athena widget routines</td>
<td>734</td>
</tr>
<tr>
<td>168</td>
<td>Athena header file names</td>
<td>736</td>
</tr>
<tr>
<td>169</td>
<td>Motif header file names</td>
<td>737</td>
</tr>
</tbody>
</table>
About this document

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.

The information in this document supports both IPv6 and IPv4. Unless explicitly noted, information describes IPv4 networking protocol. IPv6 support is qualified within the text.

This document 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 the 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 use by an experienced programmer familiar with the IBM® z/OS® operating system and commands, and with the TCP/IP protocols.

This document is written for programmers interested in high-level application functions that can be used to program applications in a TCP/IP environment. These functions involve user authentication, distributed databases, distributed processing, network management, and device sharing.

Before using this document, you should be familiar with the MVS™ operating system and the IBM Time Sharing Option (TSO).

Depending on the design and function of your application, you should be familiar with the C programming language.

In addition, z/OS Communications Server and any required programming products should already be installed and customized for your network.

How this document is organized

The document contains the following topics:

- Chapter 1, “General programming information,” on page 1 provides an overview of Distributed Protocol Interface (DPI) versions 1.1 and 2.0.
- Chapter 2, “SNMP agent Distributed Protocol Interface version 1.1,” on page 3 provides information about SNMP agent DPI version 1.1 agents and subagents, compile and link samples, descriptions of DPI library routines, and a sample client program for C sockets.
- Chapter 3, “SNMP agent Distributed Protocol Interface version 2.0,” on page 41 provides SNMP agent DPI version 2.0 programming information.
Chapter 4, “Running the sample SNMP DPI client program for version 2.0,” on page 129 explains how to run the sample SNMP DPI client program, dpi_mvs_sample.c, installed in /usr/lpp/tcpip/samples.

Chapter 5, “SNMP manager API,” on page 131 describes how to use this API to build SNMP management applications to retrieve SNMP management data.

Chapter 6, “Resource Reservation Setup Protocol API (RAPI),” on page 161 describes the API for requesting enhanced Quality of Service (QoS).

Chapter 7, “X Window System interface in the z/OS Communications Server environment,” on page 195 describes the X Window System API.

Chapter 8, “Remote procedure calls in the z/OS Communications Server environment,” on page 207 describes the high-level remote procedure calls (RPCs) implemented in TCP/IP, including the RPC programming interface to the C language and communication between processes.

Chapter 9, “Remote procedure calls in the z/OS UNIX System Services environment,” on page 337 provides information on use of UNIX System Services RPC and deviations from Sun RPC 4.0.

Chapter 10, “Network Computing System,” on page 341 describes the NCS tools used for heterogeneous distributed computing.

Chapter 11, “Running the sample mail filter program,” on page 367 explains how to run the sample mail filter program, if_smpl.c.

Chapter 12, “Policy API (PAPI),” on page 383 describes the Policy Agent API (PAPI).

Chapter 13, “FTP Client Application Programming Interface (API),” on page 397 describes the callable application programming interface to the z/OS FTP client.

Chapter 14, “Network management interfaces,” on page 475 describes the interfaces that allow network monitor and management applications to obtain information about their network operations, for both TCP/IP and VTAM®.

Chapter 15, “Application Transparent Transport Layer Security (AT-TLS),” on page 637 describes Application Transparent Transport Layer Security (AT-TLS), which creates a secure session at the TCP/IP layer on behalf of an application.


Chapter 17, “Miscellaneous programming interfaces,” on page 663 describes programming interfaces including the TCP_KeepAlive function.

Appendix A, “TCP/IP in the sysplex,” on page 671 introduces the SO_CLUSTERCONNTYPE socket option for obtaining sysplex connection endpoint characteristics.

Appendix B, “Well-known port assignments,” on page 675 lists the well-known port assignments for transport protocols TCP and UDP.

Appendix C, “Programming interfaces for providing classification data to be used in differentiated services policies,” on page 679 provides information on the Differentiated Services (DS) aspect of QoS and the passing of application classification data on SENDMSG.


Appendix E, “Related protocol specifications,” on page 743 lists the related protocol specifications for TCP/IP.
How to use this document

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

Determining whether a publication is current

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

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

How to contact IBM service

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

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

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

Outside 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 789.
Conventions and terminology that are used in this document

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

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

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

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

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

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

Clarification of notes

Information traditionally qualified as Notes is further qualified as follows:

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

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

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

Commands

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

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

Parameters

The following types of parameters are used in syntax diagrams.

Required

Required parameters are displayed on the main path.

Optional

Optional parameters are displayed below the main path.

Default

Default parameters are displayed above the main path.

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

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

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

Syntax examples

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

Optional values

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

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).

```
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.

```
DEFAULT
OPERAND
```

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.

```
variable
```

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
```

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 773, 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 collections.

<table>
<thead>
<tr>
<th>Titles</th>
<th>Order Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS V1R11 Collection</td>
<td>SK3T-4269</td>
<td>This is the CD collection shipped with the z/OS product. It includes the libraries for z/OS V1R11, in both BookManager and PDF formats.</td>
</tr>
<tr>
<td>z/OS Software Products Collection</td>
<td>SK3T-4270</td>
<td>This CD includes, in both BookManager and PDF formats, the libraries of z/OS software products that run on z/OS but are not elements and features, as well as the <em>Getting Started with Parallel Sysplex®</em> bookshelf.</td>
</tr>
<tr>
<td>z/OS V1R11 and Software Products DVD Collection</td>
<td>SK3T-4271</td>
<td>This collection includes the libraries of z/OS (the element and feature libraries) and the libraries for z/OS software products in both BookManager and PDF format. This collection combines SK3T-4269 and SK3T-4270.</td>
</tr>
<tr>
<td>z/OS Licensed Product Library</td>
<td>SK3T-4307</td>
<td>This CD includes the licensed documents in both BookManager and PDF format.</td>
</tr>
<tr>
<td>IBM System z® Redbooks Collection</td>
<td>SK3T-7876</td>
<td>The Redbooks® selected for this CD series are taken from the IBM Redbooks inventory of over 800 books. All the Redbooks that are of interest to the zSeries® platform professional are identified by their authors and are included in this collection. The zSeries subject areas range from e-business application development and enablement to hardware, networking, Linux®, solutions, security, parallel sysplex, and many others.</td>
</tr>
</tbody>
</table>

**Other documents**

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

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

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

<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>GC24-3376</td>
</tr>
<tr>
<td>Understanding LDAP</td>
<td>SG24-4986</td>
</tr>
</tbody>
</table>
## Redbooks

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

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

## Where to find related information on the Internet

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

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

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

**IBM Communications Server product support**

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

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

**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, Redpapers, and Technotes

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

**IBM Systems Center flashes**

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

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

**RFCs**

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

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

**Internet drafts**

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

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

Information about Web addresses can also be found in information APAR III1334.
Note: Any pointers in this publication to Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

DNS Web sites

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

USENET news groups
  comp.protocols.dns.bind

BIND mailing lists
  http://www.isc.org/ml-archives/

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

  BIND 9 Users (This list might not be maintained indefinitely.)
  • Subscribe by sending mail to bind9-users-request@isc.org.
  • Submit questions or answers to this forum by sending mail to bind9-users@isc.org.

The z/OS Basic Skills Information Center

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

Specifically, the z/OS Basic Skills Information Center is intended to achieve the following objectives:
  • Provide basic education and information about z/OS without charge
  • Shorten the time it takes for people to become productive on the mainframe
  • Make it easier for new people to learn z/OS

To access the z/OS Basic Skills Information Center, open your Web browser to the following Web site, which is available to all users (no login required):
  http://publib.boulder.ibm.com/infocenter/zoslnctr/v1r7/index.jsp

How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this document or any other z/OS Communications Server documentation, do one of the following:
  • Go to the z/OS contact page at http://www.ibm.com/systems/z/os/zos/webqs.html. You can enter and submit your comments in the form provided at this Web site.
  • Send your comments by e-mail to comsvrcf@us.ibm.com. Be sure to include the name of the document, the part number of the document, the version of z/OS
Communications Server, and, if applicable, the specific location of the text that you are commenting on (for example, a section number, a page number or a table number).
Summary of changes

Summary of changes
for SC31-8787-11
z/OS Version 1 Release 11

This document contains information previously presented in SC31-8787-10, which support z/OS Version 1 Release 10.

New information

- FTP access to UNIX named pipes, see “Predefined REXX variables” on page 413.
- IPSec enhancements, see “Local IPSec NMI” on page 476.
- NSS private key and certificate services for XML appliances, see “NMsec_GET_CLIENTINFO” on page 533.
- The Network Management Interfaces (NMI) has added the real time collection of trace data from the OSAENTA traces, see “Real-time TCP/IP network monitoring NMI” on page 540.
- Network management interface enhancements - stack configuration data, see the following:
  - “Real-time TCP/IP network monitoring NMI” on page 540
  - “TCP/IP callable NMI (EZBNMIFR)” on page 598
- AT-TLS enhancements, see the following:
  - “Real-time TCP/IP network monitoring NMI” on page 540
  - “SIOCTTLSCTL (X'C038D90B')” on page 644
- Network management interface enhancements - sysplex networking data, see “TCP/IP callable NMI (EZBNMIFR)” on page 598.
- Network management interface enhancements - detailed CSM usage, see “SNA network monitoring NMI” on page 620.

Deleted information

- The SYSTCPDA packet records type 1, 2, and 3 are no longer created by TCP/IP and are no longer described in this document.

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

This document has been enabled for the following types of advanced searches in the online z/OS Library Center: commands.

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

Summary of changes
for SC31-8787-10
z/OS Version 1 Release 10
This document contains information previously presented in SC31-8787-09, which support z/OS Version 1 Release 9.

**New information**

- FTP client API for Java™, see "Java call formats" on page 400.
- Network Management Interface enhancements, see Chapter 14, “Network management interfaces;” on page 475.
- IPSec RFC Currency, see Chapter 14, “Network management interfaces,” on page 475.
- Defensive filtering, see “IPSec NMI request messages” on page 484.
- DataPower® and z/OS security integration, see Network security services (NSS) network management NMI on page 531.
- Packet trace enhancements, see “Processing the CTE records for SYSTCPDA and SYSTCPOT” on page 553.
- Optimize asynchronous socket receive use of 64-bit shared memory, see “SIOCSMOCTL IOCTL” on page 665.

**Changed information**

- Enhanced rpcbind application registration control, see Portmapper and rpcbind target assistance on page 210.

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

This document has been enabled for the following types of advanced searches in the online z/OS Library Center: commands.

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

**Summary of changes for SC31-8787-09**

*z/OS Version 1 Release 9*

This document contains information previously presented in SC31-8787-08, which support z/OS Version 1 Release 8.

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

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

**New information**
- SNMP manager API, see Chapter 5, “SNMP manager API” on page 131.
- IPSec network management interface support and network security services, see “Local IPSec NMI” on page 476.
- FTP Kerberos single sign on support, see Table 69 on page 563 and Table 72 on page 567.
- Enable application identifier in NMI, SMF, and Netstat, see “SIOCSAPPLDATA IOCTL” on page 663.

**Changed information**

- Enable AT-TLS for FTP client and server, see “Real-time SMF data NMI (SYSTCPSPM) record formats” on page 558.
- Enable AT-TLS for the TN3270E Telnet server, see “Using the SIOCTTLSCTL ioctl” on page 639.

**Deleted information**

- The TN3270E Telnet server running in the TCP/IP address space is removed.
- The APPC Application Suite is removed from the z/OS V1R9 Communications Server product and therefore documentation describing APPC Application Suite support has been deleted.

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

This document has been enabled for the following types of advanced searches in the online z/OS Library Center: commands.

You might notice changes in the style and structure of some content in this document—for example, headings that use uppercase for the first letter of initial words only, and procedures that have a different look and format. The changes are ongoing improvements to the consistency and retrievability of information in our documents.
Chapter 1. General programming information

The information presented in this reference applies only to IPv4, AF_INET sockets unless specified as IPv6.

For the fundamental technical information you need to know before you attempt to work with the application programming interfaces (APIs) that are provided with TCP/IP, see z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference.

The modules generated by the new compiler are similar to those produced by the AD/Cycle® compiler.

Overview of Distributed Protocol Interface (DPI) versions 1.1 and 2.0

Two levels of Distributed Protocol Interface (DPI) are supported by z/OS Communications Server. The following shows some support differences between the two versions:

• Support provided by DPI Version 1.1
  – Was supported on earlier releases of TCP/IP and continues to be supported by the SNMP agent; existing subagents written with DPI Version 1.1 still run with no changes required.
  – Is intended for C socket API users, not z/OS UNIX C socket users.
  – Supports connections from subagents using TCP sockets.
  – Is documented in RFC 1228.

• Support provided by DPI Version 2.0:
  – Is supported in TCP/IP z/OS UNIX and above.
  – Contains more functions that make writing a subagent easier.
  – Supports both SNMP Version 1 and Version 2 protocols.
  – Is used by z/OS UNIX C socket users but not standard C socket users.
  – Supports connections from subagents using TCP sockets and UNIX Stream sockets.
  – Is documented in RFC 1592.

While DPI Version 1.1 can continue to be used by existing subagents, users who are writing new subagents or modifying old ones should consider upgrading to DPI Version 2.0 to take advantage of the SNMP Version 2 protocols and the greater functionality of DPI Version 2.0.

Although the SNMP agent that is included with z/OS Communications Server is now enabled to support SNMP Version 3 (SNMPv3), no changes are required to subagents written with either DPI Version 1.1 or Version 2.0. SNMPv3 did not introduce any new protocol data unit (PDU) types. Support for the SNMPv3 framework is handled by the SNMP agent.

Users of DPI Version 1.1 must compile using the DPI library routines provided in SEZADPIL and the version of the header file, snmp_dpx.h, that is provided in SEZACMAC. When an included header file exists as a member of an MVS
partitioned data set, the underscore (_) in the header file name is changed to an at
sign (@) when the header file is located during the compiling of a program.
Therefore, header file snmp_dpx.h can be found as member SNMP@DPX in the
SEZACMAC data set. See Chapter 2, “SNMP agent Distributed Protocol Interface
version 1.1,” on page 3 for additional details.

Users of DPI Version 2.0 must compile using the DPI library routines provided in
the directory /usr/lpp/tcpip/snmp/build/libdpi20 and the DPI Version 2.0 copy of the
header file snmp_dpi.h in /usr/lpp/tcpip/snmp/include. Additional details are in
Chapter 3, “SNMP agent Distributed Protocol Interface version 2.0,” on page 41.

For information about migrating an existing subagent from DPI Version 1.1 to DPI
Version 2.0, see “Required actions for migrating your SNMP DPI subagent to
Version 2.0” on page 45.
Chapter 2. SNMP agent Distributed Protocol Interface version 1.1

The simple network management protocol (SNMP) agent Distributed Protocol Interface (DPI) permits you to dynamically add, delete, or replace management variables in the local management information base (MIB) without recompiling the SNMP agent. The DPI protocol is also supported by SNMP agents on other IBM platforms. This makes it easier to port subagents between those platforms.

For more information about the DPI interface, see RFC 1228. See Appendix E, “Related protocol specifications,” on page 743 for information about accessing RFCs.

SNMP agents and subagents

To allow the subagents to perform their functions, the SNMP agent binds to an arbitrarily chosen TCP port and listens for connection requests from subagents. A well-known port is not used. Every invocation of the SNMP agent potentially results in a different TCP port being used.

Agents, or SNMP servers, are responsible for performing the network management functions requested by the network management stations.

A subagent provides an extension to the functionality provided by the SNMP agent. The subagent allows you to define your own MIB variables, which are useful in your environment, and register them with the SNMP agent. When requests for these variables are received by the SNMP agent, the agent passes the request to the subagent and returns a response to the agent. The SNMP agent creates an SNMP response packet and sends the response to the remote network management station that initiated the request. The existence of the subagent is transparent to the network management station.

A subagent of the SNMP agent determines the port number by sending a GET request for an MIB variable, which represents the value of the TCP port. The subagent is not required to create and parse SNMP packets, because the DPI application programming interface (API) has a library routine query_DPI_port(). After the subagent obtains the value of the DPI TCP port, it should make a TCP connection to the appropriate port. After a successful socket connect() call, the subagent registers the set of variables it supports with the SNMP agent. For information about the connect() call, see the connect() call information in the z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference. When all variable classes are registered, the subagent waits for requests from the SNMP agent.

If connections to the SNMP agent are restricted by the security product, then the security product user ID associated with the subagent must be permitted to the agent’s security product resource name for the connection to be accepted. See the Simple Network Management Protocol (SNMP) information in the z/OS Communications Server: IP Configuration Guide for more information about security product access between subagents and the z/OS Communications Server SNMP agent.
Processing DPI requests

The SNMP agent can initiate three DPI requests: GET, SET, and GET-NEXT. These requests correspond to the three SNMP requests that a network management station can make. The subagent responds to a request with a response packet. The response packet can be created using the mkDPIresponse() library routine, which is part of the DPI API library.

The SNMP subagent can only initiate two requests: REGISTER and TRAP. A REGISTER request indicates to the SNMP agent which MIB variables are supported by the subagent. A TRAP request notifies the SNMP agent of an asynchronous event that should be sent to network management stations.

Processing a GET request

The DPI packet is parsed to get the object ID of the requested variable. If the specified object ID of the requested variable is not supported by the subagent, the subagent returns an error indication of SNMP_NO_SUCH_NAME. Name, type, or value information is not returned. For example:

```c
unsigned char *cp;

cp = mkDPIresponse(SNMP_NO_SUCH_NAME,0);
```

If the object ID of the variable is supported, an error is not returned and the name, type, and value of the object ID are returned using the mkDPIset() and mkDPIresponse() routines. The following is an example of an object ID, whose type is string, being returned.

```c
char *obj_id;
unsigned char *cp;
struct dpi_set_packet *ret_value;
char *data;

data = "a string to be returned";
ret_value = mkDPIset(obj_id,SNMP_TYPE_STRING,
                     strlen(data)+1,data);

cp = mkDPIresponse(0,ret_value);
```

Processing a SET request

Processing a SET request is similar to processing a GET request, but the SNMP agent passes additional information to the subagent. This additional information consists of the type, length, and value to be set.

If the object ID of the variable is not supported, the subagent returns an error indication of SNMP_NO_SUCH_NAME. If the object ID of the variable is supported, but cannot be set, an error indication of SNMP_READ_ONLY is returned. If the object ID of the variable is supported, and is successfully set, the message SNMP_NO_ERROR is returned.

Processing a GET-NEXT request

Parsing a GET-NEXT request yields two parameters: the object ID of the requested variable and the reason for this request. This allows the subagent to return the name, type, and value of the next supported variable, whose name lexicographically follows that of the passed object ID.

Subagents can support several different groups of the MIB tree. However, the subagent cannot jump from one group to another. You must determine the reason
for the request to then determine the path to traverse in the MIB tree. The second parameter contains this reason and is the group prefix of the MIB tree that is supported by the subagent.

If the object ID of the next variable supported by the subagent does not match this group prefix, the subagent must return SNMP_NO_SUCH_NAME. If required, the SNMP agent calls on the subagent again and passes a different group prefix.

For example, if you have two subagents, the first subagent registers two group prefixes, A and C, and supports variables A.1, A.2, and C.1. The second subagent registers the group prefix B, and supports variable B.1.

When a remote management station begins dumping the MIB, starting from A, the following sequence of queries is performed:

Subagent 1 gets called:

get-next(A,A) == A.1
get-next(A.1,A) == A.2
get-next(A.2,A) == error(no such name)

Subagent 2 is then called:

get-next(A.2,B) == B.1
get-next(B.1,B) == error(no such name)

Subagent 1 is then called:

get-next(B.1,C) == C.1
get-next(C.1,C) == error(no such name)

Processing a REGISTER request
A subagent must register the variables that it supports with the SNMP agent. Packets can be created using the mkDPIregister() routine.

For example:

```c
unsigned char *cp;

cp = mkDPIregister("1.3.6.1.2.1.1.2.");
```

Note: Object IDs are registered with a trailing period (.)

Processing a TRAP request
A subagent can request that the SNMP agent generate a TRAP. The subagent must provide the desired values for the generic and specific parameters of the TRAP. The subagent can optionally provide a name, type, and value parameter. The DPI API library routine mkDPItrap() can be used to generate the TRAP packet.

SNMP agent DPI header files
The following header is required to run SNMP DPI applications:

```c
snmp_dpx.h
```

This header file is installed in the SEZACMAC data set as member SNMP@DPX.

Chapter 2. SNMP agent Distributed Protocol Interface version 1.1
SNMP agent DPI: Compiling and linking

You can use several methods to compile, link-edit, and execute your TCP/IP C source program in MVS. This topic contains information about the data sets that you must include to run your C source program under MVS batch, using IBM-supplied cataloged procedures.

The following list contains partitioned data set names, which are used as examples in the following JCL statements:

**USER.MYPROG.C**
- Contains user C source programs

**USER.MYPROG.C(PROGRAM1)**
- Member PROGRAM1 in USER.MYPROG.C partitioned data set

**USER.MYPROG.H**
- Contains user #include data sets

**USER.MYPROG.OBJ**
- Contains object code for the compiled versions of user C programs in USER.MYPROG.C

**USER.MYPROG.LOAD**
- Contains link-edited versions of user programs in USER.MYPROG.OBJ

**SNMP agent DPI: Sample compile cataloged procedure additions**

Include the following in the compile step of your cataloged procedure. Cataloged procedures are included in the IBM-supplied samples for your MVS system.

- Add the following statement as the first //SYSLIB DD statement:
  
  ```
  //SYSLIB DD DSN=SEZACMAC,DISP=SHR
  ```

- Add the following //USERLIB DD statement:
  
  ```
  //USERLIB DD DSN=USER.MYPROG.H,DISP=SHR
  ```

**SNMP agent DPI: Sample link-edit cataloged procedure additions**

Include the following in the link-edit step of your cataloged procedure.

Add the following statements after the //SYSLIB DD statement:

```
// DD DSN=SEZACMTX,DISP=SHR
// DD DSN=SEZADPIL,DISP=SHR
```

**Note:** For more information about compiling and linking, see the [z/OS XL C/C++ User's Guide](#).

### SNMP DPI library routines

This topic provides the syntax, parameters, and other appropriate information for each DPI routine supported by TCP/IP.
**mkDPIlist()**

```c
#include <snmp_dpx.h>
#include <types.h>

struct dpi_set_packet *mkDPIlist(packet, oid_name, type, len, value);
struct dpi_set_packet *packet;
char *oid_name;
int type;
int len;
char *value;
```

**Parameters:**

- `packet`: A pointer to a structure dpi_set_packet, or NULL
- `oid_name`: The object identifier of the variable
- `type`: The type of the value
- `len`: The length of the value
- `value`: A pointer to the value

**Description:** The mkDPIlist() routine can be used to create the portion of the parse tree that represents a list of name and value pairs. Each entry in the list represents a name and value pair (as would normally be returned in a response packet). If the pointer `packet` is NULL, a new dpi_set_packet structure is dynamically allocated and the pointer to that structure is returned. The structure will contain the new name and value pair. If the pointer `packet` is not NULL, a new dpi_set_packet structure is dynamically allocated and chained to the list. The new structure will contain the new name and value pair. The pointer `packet` will be returned to the caller. If an error is detected, a NULL pointer is returned.

The value of `type` can be the same as for mkDPIset(). These are defined in the `snmp_dpi.h` header file.

The dpi_set_packet structure has a next pointer (0 in case of a mkDPIset() call and is also 0 upon the first mkDPIlist() call). The structure looks like this:

```c
struct dpi_set_packet {
    char *object_id;
    unsigned char type;
    unsigned short value_len;
    char *value;
    struct dpi_set_packet *next;
};
```

**fDPIparse()**

```c
#include <snmp_dpx.h>
#include <bsdtypes.h>

void fDPIparse(hdr);
struct snmp_dpi_hdr *hdr;
```

**Parameters:**

- `hdr`: Specifies a parse tree.
Description:  The fDPIparse() routine frees a parse tree that was previously created by a call to pDPIpacket(). After calling fDPIparse(), you cannot make additional references to the parse tree.

Return Values:  None.

**mkDPIregister()**

```c
#include <snmp_dpx.h>
#include <bsdtypes.h>

unsigned char *mkDPIregister(oid_name)
char *oid_name;

Parameters:
oid_name  Specifies the object identifier of the variable to be registered. Object identifiers are registered with a trailing period (.)

Description:  The mkDPIregister() routine creates a register request packet and returns a pointer to a static buffer, which holds the packet contents. The length of the remaining packet is stored in the first 2 bytes of the packet.

Return Values:  If successful, returns a pointer to a static buffer containing the packet contents. A NULL pointer is returned if an error is detected during the creation of the packet.

Example:  The following is an example of the mkDPIregister() call.
unsigned char *packet;
int len;
packet = mkDPIregister("1.3.6.1.2.1.1.");
len = *packet * 256 + *(packet + 1);
```

**mkDPIresponse()**

```c
#include <snmp_dpx.h>
#include <bsdtypes.h>

unsigned char *mkDPIresponse(ret_code, value_list)
int ret_code;
struct dpi_set_packet *value_list;

Parameters:
ret_code  Specifies the error code to be returned.
value_list  Indicates a pointer to a parse tree containing the name, type, and value information to be returned.

Description:  The mkDPIresponse() routine creates a response packet. The first parameter, ret_code, is the error code to be returned. Zero indicates no errors. Possible errors include the following:
- SNMP_BAD_VALUE
- SNMP_GEN_ERR
- SNMP_NO_ERROR
- SNMP_NO_SUCH_NAME
```
• SNMP_READ_ONLY
• SNMP_TOO_BIG

See the snmp_dpi.h header file for a description of these messages.

If ret_code does not indicate an error, the second parameter is a pointer to a parse tree created by mkDPIset(), which represents the name, type, and value of the information being returned. If an error is indicated, the second parameter is passed as a NULL pointer.

The length of the remaining packet is stored in the first 2 bytes of the packet.

Note: mkDPIresponse() always frees the passed parse tree.

Return Values: If successful, mkDPIresponse() returns a pointer to a static buffer containing the packet contents. This is the same buffer used by mkDPIregister(). A NULL pointer is returned if an error is detected during the creation of the packet.

Example: The following is an example of the mkDPIresponse() call.

unsigned char *packet;
int error_code;
struct dpi_set_packet *ret_value;

packet = mkDPIresponse(error_code, ret_value);

len = *packet * 256 + *(packet + 1);

mkDPIset()

#include <snmp_dpx.h>
#include <bsdtypes.h>

struct dpi_set_packet *mkDPIset(char *oid_name, int type, int len, char *value)

Parameters:

oid_name Specifies the object identifier of the variable.
type Specifies the type of the object identifier.
len Indicates the length of the value.
value Indicates the pointer to the first byte of the value of the object identifier.

Description: The mkDPIset() routine can be used to create the portion of a parse tree that represents a name and value pair (as would normally be returned in a response packet). It returns a pointer to a dynamically allocated parse tree representing the name, type, and value information. If an error is detected while creating the parse tree, a NULL pointer is returned.

The value of type can be one of the following, which are defined in the snmp_dpi.h header file:

• SNMP_TYPE_COUNTER
The value parameter is always a pointer to the first byte of the object ID value.

Note: The parse tree is dynamically allocated, and copies are made of the passed parameters. After a successful call to mkDPIset(), the application can dispose of the passed parameters without affecting the contents of the parse tree.

Return Values: Returns a pointer to a parse tree containing the name, type, and value information.

mkDPItrap()

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

unsigned char *mkDPItrap(generic, specific, value_list)
int generic;
int specific;
struct dpi_set_packet *value_list;
```

Parameters:
- generic: Specifies the generic field in the SNMP TRAP packet.
- specific: Specifies the specific field in the SNMP TRAP packet.
- value_list: Used to pass the name and value pair to be placed into the SNMP packet.

Description: The mkDPItrap() routine creates a TRAP request packet. The information contained in value_list is passed as the set_packet portion of the parse tree.

The length of the remaining packet is stored in the first 2 bytes of the packet.

Note: mkDPItrap() always frees the passed parse tree.

Return Values: If the packet can be created, a pointer to a static buffer containing the packet contents is returned. This is the same buffer that is used by mkDPIregister(). If an error is encountered while creating the packet, a NULL pointer is returned.

Example: The following is an example of the mkDPItrap() call.
```c
struct dpi_set_packet *if_index_value;
unsigned long data;
unsigned char *packet;
int len;

if_index_value = mkDPIset("1.3.6.1.2.1.2.1.1", SNMP_TYPE_NUMBER,
sizeof(unsigned long), &data);
packet = mkDPItrap(2, 0, if_index_value);
len = *packet * 256 + *(packet + 1);
write(fd,packet,len);

**mkDPItrap()**

```c
#include <snmp_dpx.h>
#include <types.h>

unsigned char *mkDPItrap(generic, specific, value_list, enterprise_oid)
    long int generic; /* 4 octet integer */
    long int specific;
    struct dpi_set_packet *value_list;
    char *enterprise_oid;

Parameters:

generic     The generic field for the SNMP TRAP packet.
specific    The specific field for the SNMP TRAP packet.
value_list  A pointer to a structure dpi_set_packet, which contains one or more variables to be sent with the SNMP TRAP packet. Or NULL if no variables are to be sent.
enterprise_oid  A pointer to a character string representing the enterprise object ID (in ASN.1 notation, for example, 1.3.6.1.4.1.2.2.1.4). Or NULL if you want the SNMP agent to use its own enterprise object ID.

Description:  The mkDPItrap() routine can be used to create an extended trap. It is basically the same as the mkDPItrap() routine, but allows you to pass a list of variables, and also an enterprise object ID.
```

**pDPIpacket()**

```c
#include <snmp_dpx.h>
#include <bsdtypes.h>

struct snmp_dpi_hdr *pDPIpacket(packet)
    unsigned char *packet;

Parameters:

packet     Specifies the DPI packet to be parsed.

Description:  The pDPIpacket() routine parses a DPI packet and returns a parse tree representing its contents. The parse tree is dynamically allocated and contains copies of the information within the DPI packet. After a successful call to pDPIpacket(), the packet can be disposed of in any manner the application chooses, without affecting the contents of the parse tree.

Return Values:  If pDPIpacket() is successful, a parse tree is returned. If an error is encountered during the parse, a NULL pointer is returned.

Note:  The parse tree structures are defined in the snmp_dpi.h header file.

Example:  The following is an example of the mkDPIpacket() call.

The root of the parse tree is represented by an snmp_dpi_hdr structure.
struct snmp_dpi_hdr {
    unsigned char proto_major;
    unsigned char proto_minor;
    unsigned char proto_release;

    unsigned char packet_type;
    union {
        struct dpi_get_packet  *dpi_get;
        struct dpi_next_packet *dpi_next;
        struct dpi_set_packet  *dpi_set;
        struct dpi_resp_packet *dpi_response;
        struct dpi_trap_packet *dpi_trap;
    } packet_body;
};

The packet_type field can have one of the following values, which are defined in
the snmp_dpi.h header file:
- SNMP_DPI_GET
- SNMP_DPI_GET_NEXT
- SNMP_DPI_SET

The packet_type field indicates the request that is made of the DPI client. For each
of these requests, the remainder of the packet_body is different. If a GET request is
indicated, the object ID of the desired variable is passed in a dpi_get_packet
structure.

struct dpi_get_packet {
    char *object_id;
};

A GET-NEXT request is similar, but the dpi_next_packet structure also contains the
object ID prefix of the group that is currently being traversed.

struct dpi_next_packet {
    char *object_id;
    char *group_id;
};

If the next object, whose object ID lexicographically follows the object ID indicated
by object_id, does not begin with the suffix indicated by the group_id, the DPI client
must return an error indication of SNMP_NO_SUCH_NAME.

A SET request has the most data associated with it, and this is contained in a
dpi_set_packet structure.

struct dpi_set_packet {
    char     *object_id;
    unsigned char  type;
    unsigned short value_len;
    char      *value;
    struct dpi_set_packet *next;
};

The object ID of the variable to be modified is indicated by object_id. The type of
the variable is provided in type and can have one of the following values:
- SNMP_TYPE_COUNTER
- SNMP_TYPE_EMPTY
- SNMP_TYPE_GAUGE
- SNMP_TYPE_INET
- SNMP_TYPE_NUMBER
- SNMP_TYPE_OBJECT
• SNMP_TYPE_STRING
• SNMP_TYPE_TICKS

The length of the value to be set is stored in value_len and value contains a pointer to the value.

**Note:** The storage pointed to by value is reclaimed when the parse tree is freed.
The DPI client must make provision for copying the value contents.

**query_DPI_port()**

```c
#include <snmp_dpx.h>
#include <bsdtypes.h>

int query_DPI_port (host_name, community_name)
char *host_name;
char *community_name;

Parameters:

host_name Specifies a pointer to the SNMP agent host name or internet address.

community_name Specifies a pointer to the community name to be used when making a request. The community_name constant must be specified in ASCII.

Description: The query_DPI_port() routine is used by a DPI client to determine the TCP port number that is associated with the DPI. This port number is needed to connect() to the SNMP agent. The port number is obtained through an SNMP GET request.

Return Values: An integer representing the TCP port number is returned if successful; a -1 is returned if the port cannot be determined.
```

**Sample SNMP DPI client program for C sockets for version 1.1**

This topic contains an example of an SNMP DPI client program. The DPISAMPL program can be run using the SNMP agents that support the SNMP-DPI interface as described in RFC 1228. See Appendix E, “Related protocol specifications,” on page 743 for information about accessing RFCs.

It can be used to test agent DPI implementations because it provides variables of all types and allows you to generate traps of all types.

DPISAMPL implements a set of variables in the dpiSample table, which consists of a set of objects in the IBM Research tree (1.3.6.1.2.2.1.4). See \[dpiSample table MIB descriptions\] on page 16 for the objectID and type of each object.

**Using the DPISAMPL program**

The DPISAMPL program accepts the following arguments:

? Explains the usage.

-d n Sets the debug at level n. The range is from 0 (for no messages) to 4 (for the most verbose). The default is 0. If a number greater than 4 is specified, tracing is set to level 4.
**-trap gtype stype data**
Generates a trap of the generic type *gtype*, of the specific type *stype*,
and pass *data* as an additional value for the variable
dpiSample.*stype*.0. The values for *gtype* are from 0 through 5. The
values for *stype* indicate how *data* is interpreted. The following
values are valid for *stype*:

1. number
2. octet string
3. object ID
4. empty (ignored)
5. internet address
6. counter
7. gauge
8. time ticks
9. display string
10. octet string

**-std_traps**
Generates or simulates the standard SNMP traps, which are the
generic types 0 through 5. This includes a link down trap.

**-ent_traps**
Generates extended enterprise-specific traps, which are specific
types 1 through 9, using the internal dpiSample variables.

**ent_trapse**
Generates extended enterprise-specific traps, which are specific
types 11 through 19.

**-all_traps**
Generates std_traps, ent_traps, and ent_trapse.

**-iucv**
Uses an AF_IUCV socket to connect to the SNMP agent. This is the
default.

*Note:* Although the IUCV API is no longer supported, use of the
IUCV interaddress space communication mechanism is
supported.

**-u agent_userid**
Specifies the user ID where the SNMP agent is running. The
default is SNMPD.

**-inet**
Uses an AF_INET socket to connect to the SNMP agent.

**agent_hostname**
Specifies the host name of the system where an SNMP DPI-capable
agent is running. The default is localhost.

*Note:* The localhost value is not defined by default on z/OS.
Ensure localhost is defined to the name server or in the host
name resolution file as the local IP address if the
agent_hostname parameter is not explicitly specified.

**community_name**
Specifies the community name, which is required to get the
dpiPort. The default is public.
DPISAMPN NCCFLST for the SNMP manager

The DPISAMPN NCCFLST allows you to exercise the DPISAMPL subagent from a Tivoli® NetView® SNMP management station. The DPISAMPL subagent must be running. This sample allows you to specify which test function you want to run.

You can specify the following on Tivoli NetView:

**agent_host name**
Specifies the host name or IP address of the system where the SNMP agent is running.

**community_name**
Specifies the community name. The CLIST makes the community name uppercases so the SNMP agent must be configured to accept the community name in uppercase.

**function**
Specifies the test function to be performed. Valid test functions are:

- **ALL** Runs all of the tests. This is the default.
- **GET** Retrieves the dpiSample variables one at a time.
- **GETNEXT** Retrieves all the dpiSample variables.
- **ONEGET** Retrieves all the dpiSample variables with one GET.
- **ONESET** Sets all the dpiSample variables at once.
- **QUIT** Causes the DPISAMPLE subagent to terminate.
- **SET** Sets the dpiSample variables one at a time with one SET.
- **TRAPS** Instructs the DPISAMPLE subagent to generate nine enterprise-specific traps.

The NCCFLST assumes that the definitions for the dpiSample table (see [“dpiSample table MIB descriptions” on page 16](#)) have been added to the MIBDESC.DATA file. You can also `GET`, `GETNEXT`, or `SET` dpiSample variables with regular SNMP `GET/GETNEXT/SET` commands.

The DPISAMPL subagent recognizes a few special values in the variable dpiSampleCommand. The following are the special values and their associated subagent actions.

**all_traps**
Generates `std_traps`, `ent_traps`, and `ent_trapse`.

**ent_traps**
Generates extended enterprise-specific traps, which are specific types 1 through 9, using the internal dpiSample variables.

**ent_trapse**
Generates extended enterprise-specific traps, which are specific types 11 through 19.

**quit**
Causes the subagent to terminate.
std_traps
Generates or simulates the standard SNMP traps, which are the generic
types 0 through 5. This includes a link down trap.

Compiling and linking the DPISAMPL.C source code
The source code for the sample DPI program can be found in the SEZAINST data
set, member DPISAMPL.

You can specify the following compile time flags:

_NO_PROTO
The DPISAMPL.C code assumes that it is compiled with an ANSI-C
compliant compiler. It can be compiled without ANSI-C by defining this
flag.

MVS Indicates that compilation is for MVS and uses MVS-specific includes.
Some MVS/VM-specific code is compiled.

When linking the DPISAMPL code, you must use the SEZADPII data set. It
contains the SNMP-DPI interface routines as described in RFC 1228. See
Appendix E, “Related protocol specifications,” on page 743 for information about
accessing RFCs.

dpiSample table MIB descriptions
The following shows the MIB descriptions for the dpiSample table.

# DPISAMPLE.C supports these variables as an SNMP DPI sample sub-agent
# it also generates enterprise specific traps via DPI with these objects
dpiSample 1.3.6.1.4.1.2.2.1.4.  table 0
dpiSampleNumber 1.3.6.1.4.1.2.2.1.4.1.  number 10
# next one is to be able to send a badValue with a SET request
dpiSampleNumberString 1.3.6.1.4.1.2.2.1.4.1.1.  string 10
dpiSampleOctetString 1.3.6.1.4.1.2.2.1.4.2.  string 10
dpiSampleObjectID 1.3.6.1.4.1.2.2.1.4.3.  object 10
# XGMON/SQESERV does not allow to specify empty (so use empty string)
dpiSampleEmpty 1.3.6.1.4.1.2.2.1.4.4.  string 10
dpiSampleNetAddress 1.3.6.1.4.1.2.2.1.4.5.  internet 10
dpiSampleCounter 1.3.6.1.4.1.2.2.1.4.6.  counter 10
dpiSampleGauge 1.3.6.1.4.1.2.2.1.4.7.  gauge 10
dpiSampleTimeTicks 1.3.6.1.4.1.2.2.1.4.8.  ticks 10
dpiSampleDisplayString 1.3.6.1.4.1.2.2.1.4.9.  display 10
dpiSampleCommand 1.3.6.1.4.1.2.2.1.4.10.  display 1

Notes:
1. dpiSample object is not accessible.
2. dpiSampleNumber object is only accessible for the SNMP GET command.
3. dpiSampleNumberString object is only accessible for the SNMP GET command.
4. dpiSampleEmpty object is not accessible for the SNMP SET command.

The DPISAMPL.C source code
The following is the source code for the DPISAMPL.C program.

Note: The characters shown below might vary due to differences in character sets.
This code is included as an example only.
Figure 1. SNMP Dist Prog Interface subagent sample (Part 1 of 23)
```c
#define SNMPAGENTUSERID "SNMPD"
#define SNMPIUCVNAME "SNMP_DPI"
#pragma csect(CODE, "$DPISAMP")
#pragma csect(STATIC,"#DPISAMP")
#include <manifest.h> /* VM specific things */
#include "snmpnms.h" /* short external names for VM/MVS */
#include "snmp@vm.h" /* more of those short names */
#include <saiucv.h>
#include <bsdtime.h>
#include <bsdtypes.h>
#include <socket.h>
#include <in.h>
#include <string.h> /*@Q1C*/
#include <inet.h>
#define asciitoebcdic asciitoe
#define ebcdictoascii ebcdicto
extern char ebcdicto[], asciitoe[];
#pragma linkage(cmxlate,OS)
#define DO_ETOA(a) cmxlate((a),ebcdictoascii,strlen((a)))
#define DO_ATOE(a) cmxlate((a),asciitoebcdic,strlen((a)))
#define DO_ERROR(a) tcperror((a))
#define LOOPBACK "loopback"
#define IUCV TRUE
#define max(a,b) (((a) > (b)) ? (a) : (b))
#define min(a,b) (((a) < (b)) ? (a) : (b))
#else /* we are not on VM or MVS */
#endif OS2
#include <stdlib.h>
#include <types.h>
#include <doscalls.h>
#ifndef sleep
#define sleep(a) DOSSLEEP(1000 * (a))
#endif
#define close soclose

#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/in.h>
//@include <arpa/inet.h>
#define DO_ETOA(a) ; /* no need for this */
#define DO_ATOE(a) ; /* no need for this */
#define DO_ERROR(a) perror((a))
#define LOOPBACK "localhost"
#define IUCV FALSE
#endif AIXX21
#define isdigit(c) (((c) >= '0') && ((c) <= '9'))
#else
//@include <sys/select.h>
@endif /* AIXX21 */
```

Figure 1. SNMP Dist Prog Interface subagent sample (Part 2 of 23)
#ifdef /* defined(VM) || defined(MVS) */

#include <stdio.h>
#include "snmp_dpx.h" /*@E1C*/

#define WAIT_FOR_AGENT 3 /* time to wait before closing agent fd */

#ifndef TRUE
#define TRUE 1
#define FALSE 0
#endif

#ifdef _NO_PROTO /* for classic K&R C */
static void check_arguments();
static void send_packet();
static void print_val();
static void usage();
static void init_connection();
static void init_variables();
static void await_and_read_packet();
static void handle_packet();
static void do_get();
static void do_set();
static void issue_traps();
static void issue_one_trap();
static void issue_one_trape();
static void issue_std_traps();
static void issue_ent_traps();
static void issue_ent_trapse();
static void do_register();
static void dump_bfr();
static struct dpi_set_packet *addtoset();
extern unsigned long lookup_host();
#else /* _NO_PROTO */ /* for ANSI-C compiler */

static void check_arguments(const int argc, char *argv[]);
static void send_packet(const char * packet);
static void print_val(const int index);
static void usage(const char *progname, const int exit_rc);
static void init_connection(void);
static void init_variables(void);
static void await_and_read_packet(void);
static void handle_packet(void);
static void do_get(void);
static void do_set(void);
static void issue_traps(void);
static void issue_one_trap(void);
static void issue_one_trape(void);
static void issue_std_traps(void);
static void issue_ent_traps(void);
static void issue_ent_trapse(void);
static void do_register(void);
static void dump_bfr(const char *buf, const int len);

Figure 1. SNMP Dist Prog Interface subagent sample (Part 3 of 23)
static struct dpi_set_packet *addtoset(struct dpi_set_packet *data,
    int stype);

extern unsigned long lookup_host(const char *hostname);

#define OSTRING "hex01-04:
#define DSTRING "Initial Display String"
#define COMMAND "None"
#define BUFSIZE 4096
#define TIMEOUT 3
#define PACKET_LEN(packet) (((unsigned char)(packet)) * 256 +
    ((unsigned char)*((packet) + 1)) + 2)

/* We have the following instances for OID.x variables */
    /* 0 - table */
    /* 1 - a number */
    static long number = 0;
    static unsigned char *ostring = 0; /* 2 - octet string */
    static int ostring_len = 0; /* and its length */
    static unsigned char *objectID = 0; /* 3 - objectID */
    static int objectID_len = 0; /* and its length */
    static unsigned long ipaddr = 0; /* 4 - some empty variable */
    static unsigned long counter = 1; /* 5 - ipaddress */
    static unsigned long gauge = 1; /* 6 - a counter */
    static unsigned long ticks = 1; /* 7 - a gauge */
    static unsigned long display = 0; /* 8 - time ticks */
    static unsigned char *dstring = 0; /* 9 - display string */
    static unsigned char *command = 0; /* 10 - command */

static char *DPI_var[] = {
    "dpiSample",  /* 11 - a number */
    "dpiSampleNumber", /* 12 - octet string */
    "dpiSampleOctetString", /* 13 - objectID */
    "dpiSampleObjectID", /* 14 - some empty variable */
    "dpiSampleEmpty", /* 15 - ipaddress */
    "dpiSampleInetAddress", /* 16 - a counter */
    "dpiSampleCounter", /* 17 - a gauge */
    "dpiSampleGauge", /* 18 - time ticks */
    "dpiSampleTimeTicks", /* 19 - display string */
    "dpiSampleDisplayString",  /* 20 - command */
    "dpiSampleCommand"  /* 21 - command */
};

static short int valid_types[] = {  /* SNMP_TYPEs accepted on SET */
    -1,  /* 0 do not check type */
    SNMP_TYPE_NUMBER, /* 1 number */
    SNMP_TYPE_STRING, /* 2 octet string */
    SNMP_TYPE_OBJECT, /* 3 object identifier */
    -1, /* SNMP_TYPE_EMPTY */ /* 4 do not check type */
    SNMP_TYPE_INET, /* 5 internet address */
    SNMP_TYPE_COUNTER, /* 6 counter */
    SNMP_TYPE_GAUGE, /* 7 gauge */
    SNMP_TYPE_TICKS, /* 8 time ticks */
    SNMP_TYPE_STRING, /* 9 display string */
    SNMP_TYPE_STRING /* 10 command (display string) */

Figure 1. SNMP Dist Prog Interface subagent sample (Part 4 of 23)
```c
#define OID_COUNT_FOR_TRAPS 9
#define OID_COUNT 10

static char *packet = NULL; /* ptr to send packet. */
static char inbuf[BUFSIZE]; /* buffer for receive packets */
static int dpi_fd; /* fd for socket to DPI agent */
static short int dpi_port; /* DPI_port at agent */
static unsigned long dpi_ipaddress; /* IP address of DPI agent */
static char *dpi_hostname; /* hostname of DPI agent */
static char *dpi_userid; /* userid of DPI agent VM/MVS */
static char *var_gid; /* groupID received */
static char *var_oid; /* objectID received */
static int var_index; /* OID variable index */
static unsigned char var_type; /* SET value type */
static char *var_value; /* SET value */
static short int var_value_len; /* SET value length */
static int debug_lvl = 0; /* current debug level */
static int use_iucv = IUCV; /* optional use of AF_IUCV */
static int do_quit = FALSE; /* Quit in await loop */
static int trap_gtype = 0; /* trap generic type */
static int trap_stype = 0; /* trap specific type */
static char *trap_data = NULL; /* trap data */
static int do_trap = 0; /* switch for traps */
#define ONE_TRAP 1
#define ONE_TRAPE 2
#define STD_TRAPS 3
#define ENT_TRAPS 4
#define ENT_TRAPSE 5
#define ALL_TRAPS 6
#define MAX_TRAPE_DATA 10 /* data for extended trap */
static long trape_gtype = 6; /* trap generic type */
static long trape_stype = 11; /* trap specific type */
static char *trape_eprise = NULL; /* enterprise id */
static char *trape_data[MAX_TRAPE_DATA]; /* pointers to data values */
static int trape_datacnt; /* actual number of values */

#ifdef _NO_PROTO /* for classic K&R C */
main(argc, argv) /* main line */
int argc;
char *argv[];
#else /* _NO_PROTO */ /* for ANSI-C compiler */
main(const int argc, char *argv[]) /* main line */
#endif /* _NO_PROTO */
{
    check_arguments(argc, argv); /* check callers arguments */
    dpi_ipaddress = lookup_host(dpi_hostname); /* get ip address */
    init_connection(); /* connect to specified agent */
    init_variables(); /* initialize our variables */
    if (do_trap) {
        /* we just need to do traps */
        issue_traps(); /* issue the trap(s) */
        sleep(WAIT_FOR_AGENT); /* sleep a bit, so agent can */
        close(dpi_fd); /* read data before we close */
        exit(0); /* and that's it */
    } /* end if (do_trap) */
}
```

Figure 1. SNMP Dist Prog Interface subagent sample (Part 5 of 23)
do_register(); /* register our objectIDs */
printf("$s ready and awaiting queries from agent\n",argv[0]);
while (do_quit == FALSE) { /* forever until quit or error */
    await_and_read_packet(); /* wait for next packet */
    handle_packet(); /* handle it */
    if (do_trap) issue_traps(); /* request to issue traps */
} /* while loop */
sleep(WAIT_FOR_AGENT); /* allow agent to read response */
printf("Quitting, %s set to: quit\n",DPI_var[10]);
exit(2); /* sampleDisplayString == quit */
#endif /* _NO_PROTO */ /* for classic K&R C */
static void issue_traps
#else /* _NO_PROTO */ /* for ANSI-C compiler */
static void issue_traps(void)
#endif /* _NO_PROTO */
{
    switch (do_trap) { /* let's see which one(s) */
        case ONE_TRAP: /* only need to issue one trap */
            issue_one_trap(); /* go issue the one trap */
            break;
        case ONE_TRAPE: /* only need to issue one trape */
            issue_one_trape(); /* go issue the one trape */
            break;
        case STD_TRAPS: /* only need to issue std traps */
            issue_std_traps(); /* standard traps gtypes 0-5 */
            break;
        case ENT.TRAPS: /* only need to issue ent traps */
            issue_ent_traps(); /* enterprise specific traps */
            break;
        case ENT.TRAPSE: /* only need to issue ent trapse */
            issue_ent_trapse(); /* enterprise specific trapse */
            break;
        case ALL_TRAPS: /* only need to issue std traps */
            issue_std_traps(); /* standard traps gtypes 0-5 */
            issue_ent_traps(); /* enterprise specific traps */
            issue_ent_trapse(); /* enterprise specific trapse */
            break;
        default:
            break;
    } /* end switch (do_trap) */
do_trap = 0; /* reset do_trap switch */
#endif /* _NO_PROTO */ /* for classic K&R C */
static void await_and_read_packet() /* await packet from DPI agent */
#else /* _NO_PROTO */ /* for ANSI-C compiler */
static void await_and_read_packet(void) /* await packet from DPI agent */
#endif /* _NO_PROTO */
{
    int len, rc, bytes_to_read, bytes_read = 0;
    #ifdef OS2
        int socks[5];
    #else
        int socks[5];
    #endif

Figure 1. SNMP Dist Prog Interface subagent sample (Part 6 of 23)
```c
fd_set read_mask;
#endif
struct timeval timeout;

#ifdef OS2
socks[0] = dpi_fd;
rc = select(socks, 1, 0, 0, -1L);
#else
FD_ZERO(&read_mask);
FD_SET(dpi_fd, &read_mask); /* wait for data */
rc = select(dpi_fd+1, &read_mask, NULL, NULL, NULL);
#endif
if (rc != 1) { /* exit on error */
    DO_ERROR("await_and_read_packet: select");
    close(dpi_fd);
    exit(1);
}

#ifdef OS2
len = recv(dpi_fd, inbuf, 2, 0); /* read 2 bytes first */
#else
len = read(dpi_fd, inbuf, 2); /* read 2 bytes first */
#endif
if (len <= 0) { /* exit on error or EOF */
    if (len < 0) DO_ERROR("await_and_read_packet: read");
    else printf("Quitting, EOF received from DPI-agent\n");
    close(dpi_fd);
    exit(1);
}
bytes_to_read = (inbuf[0] << 8) + inbuf[1]; /* bytes to follow */
if (BUFSIZE < (bytes_to_read + 2)) { /* exit if too much */
    printf("Quitting, packet larger than %d byte buffer\n",BUFSIZE);
    close(dpi_fd);
    exit(1);
}
while (bytes_to_read > 0) { /* while bytes to read */
#ifdef OS2
    socks[0] = dpi_fd;
    len = select(socks, 1, 0, 3000L);
#else
    timeout.tv_sec = 3; /* wait max 3 seconds */
    timeout.tv_usec = 0;
    FD_SET(dpi_fd, &read_mask); /* check for data */
    len = select(dpi_fd+1, &read_mask, NULL, NULL, &timeout);
#endif
    if (len == 1) { /* select returned OK */
#ifdef OS2
    len = recv(dpi_fd, &inbuf[2] + bytes_read, bytes_to_read, 0);
#else
    len = read(dpi_fd, &inbuf[2] + bytes_read, bytes_to_read);
#endif
    } /* end if (len == 1) */
    if (len <= 0) { /* exit on error or EOF */
        if (len < 0) DO_ERROR("await_and_read_packet: read");
        printf("Can't read remainder of packet\n");
        close(dpi_fd);
    }
}
```

Figure 1. SNMP Dist Prog Interface subagent sample (Part 7 of 23)
```c
exit(1);
} else { /* count bytes_read */
    bytes_read += len;
    bytes_to_read -= len;
}
} /* while (bytes_to_read > 0) */

#else /* _NO_PROTO */ /* for classic K&R C */
static void handle_packet() /* handle DPI packet from agent */
#else /* _NO_PROTO */ /* for ANSI-C compiler */
static void handle_packet(void) /* handle DPI packet from agent */
#endif /* _NO_PROTO */
{
    struct snmp_dpi_hdr *hdr;
    if (debug_lvl > 2) {
        printf("Received following SNMP-DPI packet:\n");
        dump_bfr(inbuf, PACKET_LEN(inbuf));
    }
    hdr = pDPIpacket(inbuf); /* parse received packet */
    if (hdr == 0) { /* ignore if can't parse */
        printf("Ignore received packet, could not parse it!\n");
        return;
    }
    packet = NULL;
    var_type = 0;
    var_oid = "";
    var_gid = "";
    switch (hdr->packet_type) {
        /* extract pointers and/or data from specific packet types, */
        /* such that we can use them independent of packet type. */
        case SNMP_DPI_GET:
            if (debug_lvl > 0) printf("SNMP_DPI_GET for ");
            var_oid = hdr->packet_body.dpi_get->object_id;
            break;
        case SNMP_DPI_GET_NEXT:
            if (debug_lvl > 0) printf("SNMP_DPI_GET_NEXT for ");
            var_oid = hdr->packet_body.dpi_next->object_id;
            var_gid = hdr->packet_body.dpi_next->group_id;
            break;
        case SNMP_DPI_SET:
            if (debug_lvl > 0) printf("SNMP_DPI_SET for ");
            var_value_len = hdr->packet_body.dpi_set->value_len;
            var_value = hdr->packet_body.dpi_set->value;
            var_oid = hdr->packet_body.dpi_set->object_id;
            var_type = hdr->packet_body.dpi_set->type;
            break;
        default: /* Return a GEN_ERROR */
            if (debug_lvl > 0) printf("Unexpected packet_type %d, genErr\n",
                hdr->packet_type);
            packet = mkDPIresponse(SNMP_GEN_ERR, NULL);
            fDPIparse(hdr); /* return storage allocated by pDPIpacket() */
            send_packet(packet);
            return;
    }
}

Figure 1. SNMP Dist Prog Interface subagent sample (Part 8 of 23)
break;
} /* end switch(hdr->packet_type) */
if (debug_lvl > 0) printf("objectID: %s \n", var_oid);

if (strlen(var_oid) <= strlen(OID)) { /* not in our tree */
if (hdr->packet_type == SNMP_DPI_GET_NEXT) var_index = 0; /* OK */
else { /* cannot handle */
    if (debug_lvl>0) printf("...Ignored %s, noSuchName\n", var_oid);
    packet = mkDPIresponse(SNMP_NO_SUCH_NAME, NULL);
    fDPIparse(hdr); /* return storage allocated by pDPIpacket() */
    send_packet(packet);
    return;
}
else { /* Extract our variable index (from OID.index.instance) */
    /* We handle any instance the same (we only have one instance) */
    var_index = atoi(&var_oid[strlen(OID)]);
}
if (debug_lvl > 1) {
    printf("...The groupID=%s\n", var_gid);
    printf("...Handle as if objectID=%s%d\n", OID, var_index);
}
switch (hdr->packet_type) {
    case SNMP_DPI_GET:
        do_get(); /* do a get to return response */
        break;
    case SNMP_DPI_GET_NEXT:
        { char toid[256]; /* space for temporary objectID */
          var_index++; /* do a get for the next variable */
          sprintf(toid,"%s%d", OID, var_index); /* construct objectID */
          var_oid = toid; /* point to it */
          do_get(); /* do a get to return response */
        } break;
    case SNMP_DPI_SET:
        if (debug_lvl > 1) printf("...value_type=%d\n", var_type);
        do_set(); /* set new value first */
        if (packet) break; /* some error response was generated */
        do_get(); /* do a get to return response */
        break;
}
    fDPIparse(hdr); /* return storage allocated by pDPIpacket() */
}

#ifndef _NO_PROTO /* for classic K&R C */
static void do_get() /* handle SNMP_GET request */
#else /* _NO_PROTO */ /* for ANSI-C compiler */
static void do_get(void) /* handle SNMP_GET request */
#endif /* _NO_PROTO */
{
    struct dpi_set_packet *data = NULL;

    switch (var_index) {
        case 0: /* table, cannot be queried by itself */
            printf("...Should not issue GET for table %s.0\n", OID);
            break;
        case 1: /* a number */
Figure 1. SNMP Dist Prog Interface subagent sample (Part 9 of 23)
data = mkDPIset(var_oid,SNMP_TYPE_NUMBER,sizeof(number),&number);
break;
case 2: /* an octet_string (can have binary data) */
data = mkDPIset(var_oid,SNMP_TYPE_STRING,ostring_len,ostring);
break;
case 3: /* object id */
data = mkDPIset(var_oid,SNMP_TYPE_OBJECT,objectId_len,objectId);
break;
case 4: /* some empty variable */
data = mkDPIset(var_oid,SNMP_TYPE_EMPTY,0,NULL);
break;
case 5: /* internet address */
data = mkDPIset(var_oid,SNMP_TYPE_INET,sizeof(ipaddr),&ipaddr);
break;
case 6: /* counter (unsigned) */
data = mkDPIset(var_oid,SNMP_TYPE_COUNTER,sizeof(counter),&counter);
break;
case 7: /* gauge (unsigned) */
data = mkDPIset(var_oid,SNMP_TYPE_GAUGE,sizeof(gauge),&gauge);
break;
case 8: /* time ticks (unsigned) */
data = mkDPIset(var_oid,SNMP_TYPE_TICKS,sizeof(ticks),&ticks);
break;
case 9: /* a display_string (printable ascii only) */
    DO_ETOA(dstring);
data = mkDPIset(var_oid,SNMP_TYPE_STRING,strlen(dstring),dstring);
    DO_ATOE(dstring);
break;
case 10: /* a command request (command is a display string) */
    DO_ETOA(command);
data = mkDPIset(var_oid,SNMP_TYPE_STRING,strlen(command),command);
    DO_ATOE(command);
break;
default: /* Return a NoSuchName */
    if (debug_lvl > 1)
    printf("...GET\[NEXT\] for %s, not found\n", var_oid);
break;
} /* end switch (var_index) */

if (data) {
    if (debug_lvl > 0) {
        printf("...Sending response oid: %s type: %d\n", var_oid, data->type);
        printf("......Current value: ");
        print_val(var_index); /* prints \n at end */
    }
    packet = mkDPIresponse(SNMP_NO_ERROR,data);
} else { /* Could have been an error in mkDPIset though */
    if (debug_lvl > 0) printf("...Sending response noSuchName\n");
    packet = mkDPIresponse(SNMP_NO_SUCH_NAME,NULL);
} /* end if (data) */
if (packet) send_packet(packet);

#ifdef _NO_PROTO
    /* for classic K&R C */
#endif

Figure 1. SNMP Dist Prog Interface subagent sample (Part 10 of 23)
static void do_set() /* handle SNMP_SET request */
#else /* _NO_PROTO */ /* for ANSI-C compiler */
static void do_set(void) /* handle SNMP_SET request */
#endif /* _NO_PROTO */
{
    unsigned long *ulp;
    long *lp;

    if (valid_types[var_index] != var_type &&
        valid_types[var_index] != -1) {
        printf("...Ignored set request with type %d, expect type %d,",
            var_type, valid_types[var_index]);
        printf(" Returning badValue
");
        packet = mkDPIresponse(SNMP_BAD_VALUE, NULL);
        if (packet) send_packet(packet);
        return;
    }

    switch (var_index) {
    case 0: /* table, cannot set table. */
        if (debug_lvl > 0) printf("...Ignored set TABLE, noSuchName
");
        packet = mkDPIresponse(SNMP_NO_SUCH_NAME,NULL);
        break;
    case 1: /* a number */
        lp = (long *)var_value;
        number = *lp;
        break;
    case 2: /* an octet_string (can have binary data) */
        free(ostring);
        ostring = (char *)malloc(var_value_len + 1);
        bcopy(var_value, ostring, var_value_len);
        ostring_len = var_value_len;
        ostring[var_value_len] = '\0'; /* so we can use it as a string */
        break;
    case 3: /* object id */
        free(objectID);
        objectID = (char *)malloc(var_value_len + 1);
        bcopy(var_value, objectID, var_value_len);
        objectID_len = var_value_len;
        if (objectID[objectID_len -1]) {
            objectID[objectID_len] = '\0'; /* a valid one needs a null */
            if (debug_lvl > 0)
                printf("...added a terminating null to objectID\n");
        }
        break;
    case 4: /* an empty variable, cannot set */
        if (debug_lvl > 0) printf("...Ignored set EMPTY, readOnly\n");
        packet = mkDPIresponse(SNMP_READ_ONLY,NULL);
        break;
    case 5: /* Internet address */
        ulp = (unsigned long *)var_value;
        ipaddr = *ulp;
        break;
    case 6: /* counter (unsigned) */
        ulp = (unsigned long *)var_value;
        counter = *ulp;
    }

Figure 1. SNMP Dist Prog Interface subagent sample (Part 11 of 23)
break;
case 7: /* gauge (unsigned) */
    ulp = (unsigned long *)var_value;
gauge = *ulp;
    break;
case 8: /* time ticks (unsigned) */
    ulp = (unsigned long *)var_value;
ticks = *ulp;
    break;
case 9: /* a display_string (printable ascii only) */
    free(dstring);
dstring = (char *)malloc(var_value_len + 1);
bcopy(var_value, dstring, var_value_len);
dstring[var_value_len] = '\0'; /* so we can use it as a string */
    DO_ATOE(dstring);
    break;
case 10: /* a request to execute a command */
    free(command);
    command = (char *)malloc(var_value_len + 1);
bcopy(var_value, command, var_value_len);
    command[var_value_len] = '\0'; /* so we can use it as a string */
    DO_ATOE(command);
    if (strcmp("all_traps", command) == 0) do_trap = ALL_TRAPS;
    else if (strcmp("std_traps", command) == 0) do_trap = STD_TRAPS;
    else if (strcmp("ent_traps", command) == 0) do_trap = ENT_TRAPS;
    else if (strcmp("ent_trapse", command) == 0) do_trap = ENT_TRAPSE;
    else if (strcmp("all_traps", command) == 0) do_trap = ALL_TRAPS;
    else if (strcmp("quit", command) == 0) do_quit = TRUE;
    else break;
    if (debug_lvl > 0)
        printf("...Action requested: %s set to: %s\n", 
            DPI_var[10], command);
    break;
default: /* NoSuchName */
    if (debug_lvl > 0)
        printf("...Ignored set for %s, noSuchName\n", var_oid);
    packet = mkDPIresponse(SNMP_NO_SUCH_NAME,NULL);
    break;
} /* end switch (var_index) */

#endif _NO_PROTO /* for classic K&R C */
#else /* _NO_PROTO */ /* for ANSI-C compiler */
#endif /* _NO_PROTO */
{
    trap_stype = 0;
    trap_data = dpi_hostname;
    for (trap_gtype=0; trap_gtype<6; trap_gtype++) {
        issue_one_trap();
        if (trap_gtype == 0) sleep(10); /* some managers purge cache */
    }
}

Figure 1. SNMP Dist Prog Interface subagent sample (Part 12 of 23)
```c
#ifndef _NO_PROTO
static void issue_ent_traps()
#else /* _NO_PROTO */
static void issue_ent_traps(void)
#endif /* _NO_PROTO */
{
    char temp_string[256];

    trap_gtype = 6;
    for (trap_stype = 1; trap_stype < 10; trap_stype++) {
        trap_data = temp_string;
        switch (trap_stype) {
            case 1:
                sprintf(temp_string, "%ld", number);
                break;
            case 2:
                sprintf(temp_string, "%s", ostring);
                break;
            case 3:
                trap_data = objectID;
                break;
            case 4:
                trap_data = "";
                break;
            case 5:
                trap_data = dpi_hostname;
                break;
            case 6:
                sleep(1); /* give manager a break */
                sprintf(temp_string, "%lu", counter);
                break;
            case 7:
                sprintf(temp_string, "%lu", gauge);
                break;
            case 8:
                sprintf(temp_string, "%lu", ticks);
                break;
            case 9:
                trap_data = dstring;
                break;
        } /* end switch (trap_stype) */
        issue_one_trap();
    }
}

/* issue a set of extended traps, pass enterprise ID and multiple
 * variable (assume octect string) as passed by caller
 */
#ifndef _NO_PROTO
static void issue_ent_trapse()
#else /* _NO_PROTO */
static void issue_ent_trapse(void)
#endif /* _NO_PROTO */
{

Figure 1. SNMP Dist Prog Interface subagent sample (Part 13 of 23)
```
int i, n;
struct dpi_set_packet *data = NULL;
unsigned char *packet = NULL;
unsigned long ipaddr, ulnum;
char oid[256];
char *cp;

trape_gtype = 6;
trape_eprise = ENTERPRISE_OID;
for (n=11; n < (11+OID_COUNT_FOR_TRAPS); n++) {
    data = 0;
    trape_stype = n;
    for (i=1; i<=(n-10); i++)
        data = addtoset(data, i);
    if (data == 0) {
        printf("Could not make dpi_set_packet\n");
        return;
    }
    packet = mkDPItrape(trape_gtype, trape_stype, data, trape_eprise);
    if ((debug_lvl > 0) && (packet)) {
        printf("sending trape packet: %lu %lu enterprise=%s\n",
               trape_gtype, trape_stype, trape_eprise);
    }
    if (packet) send_packet(packet);
    else printf("Could not make trape packet\n");
}

/* issue one extended trap, pass enterprise ID and multiple
 * variable (assume octet string) as passed by caller
 */
#ifdef _NO_PROTO
/* for classic K&R C */
static void issue_one_trape()
#else /* _NO_PROTO */
/* for ANSI-C compiler */
#endif /* _NO_PROTO */
{
    struct dpi_set_packet *data = NULL;
    unsigned char *packet = NULL;
    char oid[256];
    char *cp;
    int i;

    for (i=0; i<trape_datacnt; i++) {
        sprintf(oid,"%s2.%d",OID,i);
        /* assume an octet_string (could have hex data) */
        data = mkDPIlist(data, oid, SNMP_TYPE_STRING,
                         strlen(trape_data[i]), trape_data[i]);
        if (data == 0) {
            printf("Could not make dpiset_packet\n");
        } else if (debug_lvl > 0) {
            printf("Preparing: [oid=%s] value: ", oid);
            printf(""");
            for (cp = trape_data[i]; *cp; cp++) /* loop through data */
                printf("%2.2x",*cp); /* hex print one byte */
    
Figure 1. SNMP Dist Prog Interface subagent sample (Part 14 of 23)
printf("\n\n");

if (debug_lvl > 0) && (packet)) {
  printf("sending trape packet: %lu %lu enterprise=%s
",
    trape_gtype, trape_stype, trape_eprise);
}

if (packet) send_packet(packet);
else printf("Could not make trape packet\n");

#endif /* _NO_PROTO */ /* for ANSI-C compiler */
#endif /* _NO_PROTO */ /* for classic K&R C */
static void issue_one_trap()
{
  long int num; /* must be 4 bytes */
  struct dpi_set_packet *data = NULL;
  unsigned char *packet = NULL;
  unsigned long ipaddr, ulnum;
  char oid[256];
  char *cp;

  switch (trap_gtype) {
  /* all traps are handled more or less the same sofar. */
  /* could put specific handling here if needed/wanted. */
  case 0: /* simulate cold start */
    strcpy(oid,"none");
    break;
  case 1: /* simulate warm start */
    num = 1;
    data = mkDPIset(oid, SNMP_TYPE_NUMBER, sizeof(num), &num);
    break;
  case 4: /* simulate authentication failure */
    strcpy(oid,"none");
    break;
  case 2: /* simulate link down */
    num = 1;
    data = mkDPIset(oid, SNMP_TYPE_NUMBER, sizeof(num), &num);
    break;
  case 3: /* simulate link up */
    strcpy(oid,ifIndex);
    num = 1;
    data = mkDPIset(oid, SNMP_TYPE_NUMBER, sizeof(num), &num);
    break;
  case 5: /* simulate EGP neighbor loss */
    strcpy(oid,egpNeighAddr);
    ipaddr = lookup_host(trap_data);
    data = mkDPIset(oid, SNMP_TYPE_STRING, sizeof(ipaddr), &ipaddr);
    break;
  case 6: /* simulate enterprise specific trap */
    sprintf(oid,"%s%d.0",OID, trap_stype);
    switch (trap_stype) {
    case 1: /* a number */
      num = strtol(trap_data,(char **)0,10);
      data = mkDPIset(oid, SNMP_TYPE_NUMBER, sizeof(num), &num);
      break;
    case 2: /* an octet_string (could have hex data) */
      data = mkDPIset(oid,SNMP_TYPE_STRING,strlen(trap_data),trap_data);
      break;
    case 3: /* object id */
  }

Figure 1. SNMP Dist Prog Interface subagent sample (Part 15 of 23)
data = mkDPIset(oid, SNMP_TYPE_OBJECT, strlen(trap_data) + 1, trap_data);
break;
case 4: /* an empty variable value */
data = mkDPIset(oid, SNMP_TYPE_EMPTY, 0, 0);
break;
case 5: /* internet address */
ipaddr = lookup_host(trap_data);
data = mkDPIset(oid, SNMP_TYPE_INTERNET, sizeof(ipaddr), &ipaddr);
break;
case 6: /* counter (unsigned) */
ulnum = strtoul(trap_data, (char **)0, 10);
data = mkDPIset(oid, SNMP_TYPE_COUNTER, sizeof(ulnum), &ulnum);
break;
case 7: /* gauge (unsigned) */
ulnum = strtoul(trap_data, (char **)0, 10);
data = mkDPIset(oid, SNMP_TYPE_GAUGE, sizeof(ulnum), &ulnum);
break;
case 8: /* time ticks (unsigned) */
ulnum = strtoul(trap_data, (char **)0, 10);
data = mkDPIset(oid, SNMP_TYPE_TICKS, sizeof(num), &ulnum);
break;
case 9: /* a display_string (ascii only) */
DO_ETOA(trap_data);
data = mkDPIset(oid, SNMP_TYPE_STRING, strlen(trap_data), trap_data);
DO_ATOE(trap_data);
break;
default: /* handle as string */
printf("Unknown specific trap type: %s, assume octet_string\n", trap_stype);
data = mkDPIset(oid, SNMP_TYPE_STRING, strlen(trap_data), trap_data);
break;
} /* end switch (trap_stype) */
break;
default: /* unknown trap */
printf("Unknown general trap type: %s\n", trap_gtype);
return;
break;
} /* end switch (trap_gtype) */

packet = mkDPItrap(trap_gtype, trap_stype, data);
if ((debug_lvl > 0) & (packet)) {
printf("sending trap packet: %u %u [oid=%s] value: ",
trap_gtype, trap_stype, oid);
if (trap_stype == 2) {
printf("\n");
for (cp = trap_data; *cp; cp++) /* loop through data */
printf("%2.2x",*cp); /* hex print one byte */
printf("\n");
} else printf("%s\n", trap_data);
}
if (packet) send_packet(packet);
else printf("Could not make trap packet\n");
}

Figure 1. SNMP Dist Proj Interface subagent sample (Part 16 of 23)
#ifdef _NO_PROTO /* for classic K&R C */
static void send_packet(packet) /* DPI packet to agent */
char *packet;
#else /* _NO_PROTO */ /* for ANSI-C compiler */
static void send_packet(const char *packet) /* DPI packet to agent */
#endif /* _NO_PROTO */
{
    int rc;

    if (debug_lvl > 2) {
        printf("...Sending DPI packet:
");
        dump_bfr(packet, PACKET_LEN(packet));
    }
# ifdef OS2
    rc = send(dpi_fd,packet,PACKET_LEN(packet),0);
#else
    rc = write(dpi_fd,(unsigned char *)packet,PACKET_LEN(packet));
#endif
    if (rc != PACKET_LEN(packet)) DO_ERROR("send_packet: write");
    /* no need to free packet (static buffer in mkDPI.... routine) */
}
#endif /* _NO_PROTO */
static void do_register() /* register our objectIDs with agent */
#if define OS2 /* for classic K&R C */
static void do_register() /* register our objectIDs with agent */
#else /* _NO_PROTO */ /* for ANSI-C compiler */
static void do_register(void) /* register our objectIDs with agent */
#endif /* _NO_PROTO */
{
    int i, rc;
    char toid[256];

    if (debug_lvl > 0) printf("Registering variables:
");
    for (i=1; i<=OID_COUNT; i++) {
        sprintf(toid,"%s%d.",OID,i);
        packet = mkDPIregister(toid);
    }
# ifdef OS2
    rc = send(dpi_fd, packet, PACKET_LEN(packet),0);
#else
    rc = write(dpi_fd, packet, PACKET_LEN(packet));
#endif
    if (rc <= 0) {
        DO_ERROR("do_register: write");
        printf("Quitting, unsuccessful register for %s\n",toid);
        close(dpi_fd);
        exit(1);
    }
    if (debug_lvl > 0) {
        printf("...Registered: %-25s oid: %s\n",DPI_var[i],toid);
        printf("......Initial value: ");
        print_val(i); /* prints \n at end */
    }
}
/* add specified variable to list of variable in the dpi_set_packet */

Figure 1. SNMP Dist Prog Interface subagent sample (Part 17 of 23)
```c
/*
#ifdef _NO_PROTO /* for classic K&R C */
struct dpi_set_packet *addtoset(data, stype)
struct dpi_set_packet *data;
int stype;
#else /* _NO_PROTO */ /* for ANSI-C compiler */
struct dpi_set_packet *addtoset(struct dpi_set_packet *data, int stype)
#endif /* _NO_PROTO */
{
  char var_oid[256];
  sprintf(var_oid,"%s%d.0",OID, stype);
  switch (stype) {
  case 1: /* a number */
    data = mkDPIlist(data, var_oid, SNMP_TYPE_NUMBER,
                    sizeof(number), &number);
    break;
  case 2: /* an octet_string (can have binary data) */
    data = mkDPIlist(data, var_oid, SNMP_TYPE_STRING,
                   ostring_len, ostring);
    break;
  case 3: /* object id */
    data = mkDPIlist(data, var_oid, SNMP_TYPE_OBJECT,
                   objectID_len, objectID);
    break;
  case 4: /* some empty variable */
    data = mkDPIlist(data, var_oid, SNMP_TYPE_EMPTY, 0, NULL);
    break;
  case 5: /* internet address */
    data = mkDPIlist(data, var_oid, SNMP_TYPE_INTERNET,
                   sizeof(ipaddr), &ipaddr);
    break;
  case 6: /* counter (unsigned) */
    data = mkDPIlist(data, var_oid, SNMP_TYPE_COUNTER,
                   sizeof(counter), &counter);
    break;
  case 7: /* gauge (unsigned) */
    data = mkDPIlist(data, var_oid, SNMP_TYPE_GAUGE,
                   sizeof(gauge), &gauge);
    break;
  case 8: /* time ticks (unsigned) */
    data = mkDPIlist(data, var_oid, SNMP_TYPE_TICKS,
                   sizeof(ticks), &ticks);
    break;
  case 9: /* a display_string (printable ascii only) */
    DO_ETOA(dstring);
    data = mkDPIlist(data, var_oid, SNMP_TYPE_STRING,
                   strlen(dstring), dstring);
    DO_ATOE(dstring);
    break;
  } /* end switch (stype) */
  return(data);
}

#ifdef _NO_PROTO /* for classic K&R C */

Figure 1. SNMP Dist Prog Interface subagent sample (Part 18 of 23)
```
static void print_val(index)
int index;
#else /* _NO_PROTO */ /* for ANSI-C compiler */
static void print_val(const int index)
#endif /* _NO_PROTO */
{
    char *cp;
    struct in_addr temp_ipaddr;

    switch (index) {
    case 1 :
        printf("%ld\n",number);
        break;
    case 2 :
        printf("\"

        for (cp = ostring; cp < ostring + ostring_len; cp++)
            printf("%2.2x",*cp);
        printf("H\n")
        break;
    case 3 :
        printf("%s\n",objectID_len, objectID);
        break;
    case 4 :
        printf("no value (EMPTY)\n");
        break;
    case 5 :
        temp_ipaddr.s_addr = ipaddr;
        printf("%s\n",inet_ntoa(temp_ipaddr));
        break;
    case 6 :
        printf("%lu\n",counter);
        break;
    case 7 :
        printf("%lu\n",gauge);
        break;
    case 8 :
        printf("%lu\n",ticks);
        break;
    case 9 :
        printf("%s\n",dstring);
        break;
    case 10 :
        printf("%s\n",command);
        break;
    } /* end switch(index) */
} /* end switch(index) */
#endif /* _NO_PROTO */ /* for classic K&R C */
static void check_arguments(argc, argv) /* check arguments */
int argc;
char *argv[];

Figure 1. SNMP Dist Prog Interface subagent sample (Part 19 of 23)
static void check_arguments(const int argc, char *argv[])
{
    char *hname, *cname;
    int i, j;

dpi_userid = hname = cname = NULL;
for(i=1; argc > i; i++) {
    if (strcmp(argv[i],"-d") == 0) {
        i++;
        if (argc > i) {
            debug_lvl = atoi(argv[i]);
            if (debug_lvl >= 5) {
                DPIdebug(1);
            }
        }
    } else if (strcmp(argv[i],"-trap") == 0) {
        if (argc > i+3) {
            trap_gtype = atoi(argv[i+1]);
            trap_stype = atoi(argv[i+2]);
            trap_data = argv[i+3];
            i = i + 3;
            do_trap = ONE_TRAP;
        } else usage(argv[0], 1);
    } else if (strcmp(argv[i],"-trape") == 0) {
        if (argc > i+4) {
            trape_gtype = strtoul(argv[i+1],(char**)0,10);
            trape_stype = strtoul(argv[i+2],(char**)0,10);
            trape_eprise = argv[i+3];
            for (i = i + 4, j = 0;
                 argc > i && (j < MAX_TRAPE_DATA);
                 i++, j++) {
                trape_data[j] = argv[i];
            }
            trape_datacnt = j;
            do_trap = ONE_TRAPE;
            break; /* -trape must be last option */
        } else usage(argv[0], 1);
    } else if (strcmp(argv[i],"-all_traps") == 0) {
        do_trap = ALL_TRAPS;
    } else if (strcmp(argv[i],"-std_traps") == 0) {
        do_trap = STD_TRAPS;
    } else if (strcmp(argv[i],"-ent_traps") == 0) {
        do_trap = ENT_TRAPS;
    } else if (strcmp(argv[i],"-ent_trapse") == 0) {
        do_trap = ENT_TRAPSE;
    } else if (strcmp(argv[i],"-inet") == 0) {
        use_iucv = 0;
    } else if (strcmp(argv[i],"-iucv") == 0) {
        use_iucv = TRUE;
    } else if (strcmp(argv[i],"-u") == 0) {
        use_iucv = TRUE; /* -u implies -iucv */
    }++;

Figure 1. SNMP Dist Prog Interface subagent sample (Part 20 of 23)
if (argc > i) {
    dpi_userid = argv[i];
}
#endif
} else if (strcmp(argv[i], "?") == 0) {
    usage(argv[0], 0);
} else {
    if (hname == NULL) hname = argv[i];
    else if (cname == NULL) cname = argv[i];
    else usage(argv[0], 1);
}

if (hname == NULL) hname = LOOPBACK; /* use default */
if (cname == NULL) cname = PUBLIC_COMMUNITY_NAME; /* use default */
#if defined(VM) || defined(MVS)
    if (dpi_userid == NULL) dpi_userid = SNMPAGENTUSERID;
    if (debug_lvl > 2)
        printf("hname=%s, cname=%s, userid=%s\n", hname, cname, dpi_userid);
#else
    if (debug_lvl > 2)
        printf("hname=%s, cname=%s\n", hname, cname);
#endif
if (use_iucv != TRUE) {
    if (dpi_port == -1) {
        printf("No response from agent at %s(%s)\n", hname, cname);
        exit(1);
    } else dpi_port = -1;
    dpi_hostname = hname;
}
#endif /* _NO_PROTO */ /* for classic K&R C */
static void usage(char *pname, int exit_rc)
{
    printf("Usage: %s [-d debug_lvl] [-std_traps] [-ent_traps] [-all_traps] [-trap g_type s_type data]", pname);
    printf(" [-all_traps]\n"");
    printf("%-s[-trap g_type s_type enterprise data1 data2 .. datan]\n",
            strlen(pname)+8,"");
    printf("%-s[-std_traps] [-ent_traps] [-ent_trapse]\n",
            strlen(pname)+8,"");
    #if defined(VM) || defined(MVS)
        printf("%-s[-iucv] [-u agent_userid]\n", strlen(pname)+8,"");
        printf("%-s", strlen(pname)+8,"");
        printf("[-iucv] agent_username [community_name]\n");
        printf("default: -d 0 -iucv -u %s, SNMPAGENTUSERID]
",")
        printf("-all %s %s\n", LOOPBACK, PUBLIC_COMMUNITY_NAME);
    #else
        printf("",")
    #endif

Figure 1. SNMP Dist Prog Interface subagent sample (Part 21 of 23)
```c
printf("%s[agent_hostname [community_name]]\n", strlen(pname)+8,"");
printf("default: -d 0 %s\n", LOOPBACK, PUBLIC_COMMUNITY_NAME);
#endif
exit(exit_rc);
}
#endif /* _NO_PROTO */ /* for classic K&R C */
static void init_variables() /* initialize our variables */
#else /* _NO_PROTO */ /* for ANSI-C compiler */
static void init_variables(void) /* initialize our variables */
#endif /* _NO_PROTO */
{
    char ch, *cp;
    ostring = (char *)malloc(strlen(OSTRING) + 4 + 1);
    bcopy(OSTRING, ostring, strlen(OSTRING));
    ostring_len = strlen(OSTRING);
    for (ch=1; ch<5; ch++) /* add hex data 0x01020304 */
        ostring[ostring_len++] = ch;
    ostring[ostring_len] = '\0'; /* so we can use it as a string */
    objectID = (char *)malloc(strlen(OID));
    objectID_len = strlen(OID);
    bcopy(OID, objectID, strlen(OID));
    if (objectID[objectID_len - 1] == '.') /* if trailing dot, */
        objectID[objectID_len - 1] = '\0'; /* remove it */
    else objectID_len++;
    dstring = (char *)malloc(strlen(DSTRING) + 1);
    bcopy(DSTRING, dstring, strlen(DSTRING) + 1);
    command = (char *)malloc(strlen(COMMAND) + 1);
    bcopy(COMMAND, command, strlen(COMMAND) + 1);
    ipaddr = dpi_ipaddress;
}
#endif /* _NO_PROTO */ /* for classic K&R C */
static void init_connection() /* connect to the DPI agent */
#else /* _NO_PROTO */ /* for ANSI-C compiler */
static void init_connection(void) /* connect to the DPI agent */
#endif /* _NO_PROTO */
{
    int rc;
    int sasize; /* size of socket structure */
    struct sockaddr_in sin; /* socket address AF_INET */
    struct sockaddr *sa; /* socket address general */
    if (use_iucv == TRUE) {
        printf("Connecting to %s userid %s (TCP, AF_IUCV)\n",
            dpi_hostname, dpi_userid); /* @P1C*/
        bzero(&siu, sizeof(siu));
        siu.siucv_family = AF_IUCV;
        siu.siucv_port = 0; /* @P1C*/
        memset(siu.siucv_nodeid, '\0', sizeof(siu.siucv_nodeid));
        Figure 1. SNMP Dist Prog Interface subagent sample (Part 22 of 23)
```
memset(siu.siucv_userid, ' ', sizeof(siu.siucv_userid));
memset(siu.siucv_name, ' ', sizeof(siu.siucv_name));
bcopy(dpi_userid, siu.siucv_userid, min(8,strlen(dpi_userid)));
bcopy(SNMPIUCVNAME, siu.siucv_name, min(8,strlen(SNMPIUCVNAME)));
dpi_fd = socket(AF_IUCV, SOCK_STREAM, 0);
sa = (struct sockaddr *) &siu;
sasize = sizeof(struct sockaddr_iucv);
} else {
    #endif
    printf("Connecting to %s DPI_port %d (TCP, AF_INET)\n", 
            dpi_hostname,dpi_port);
bzero(&sin,sizeof(sin));
sin.sin_family = AF_INET;
sin.sin_port = htons(dpi_port);
sin.sin_addr.s_addr = dpi_ipaddress;
dpi_fd = socket(AF_INET, SOCK_STREAM, 0);
sa = (struct sockaddr *) &sin;
sasize = sizeof(struct sockaddr_in);
#if defined(VM) || defined(MVS)
    } else {
    #endif
    if (dpi_fd < 0) { /* exit on error */
        DO_ERROR("init_connection: socket");
        exit(1);
    }
    rc = connect(dpi_fd, sa, sasize); /* connect to agent */
    if (rc != 0) { /* exit on error */
        DO_ERROR("init_connection: connect");
        close(dpi_fd);
        exit(1);
    }
#endif
#if _NO_PROTO /* for classic K&R C */
static void dump_bfr(buf, len) /* hex dump buffer */
    char *buf;
    int len;
#else /* _NO_PROTO */ /* for ANSI-C compiler */
static void dump_bfr(const char *buf, const int len)
#endif /* _NO_PROTO */
{ /* _NO_PROTO */
    register int i;
    if (len == 0) printf(" empty buffer\n"); /* buffer is empty */
    for (i=0; i<len; i++) { /* loop through buffer */
        if (((i&15) == 0) printf(" "); /* indent new line */
            printf("%2.2x",(unsigned char)buf[i]);/* hex print one byte */
        if (((i&15) == 15) printf("\n"); /* nl every 16 bytes */
            else if (((i&3) == 3) printf(" "); /* space every 4 bytes */
    }
    if (i&15) printf("\n"); /* always end with nl */
}

Figure 1. SNMP Dist Prog Interface subagent sample (Part 23 of 23)
Chapter 3. SNMP agent Distributed Protocol Interface version 2.0

The simple network management protocol (SNMP) agent Distributed Protocol Interface (DPI) permits you to dynamically add, delete, or replace management variables in the local management information base (MIB). The SNMP DPI protocol is also supported with the SNMP agent on OS/2®, VM, and AIX®. This makes it easier to port subagents between those platforms and z/OS, as well as connect agents and subagents across these platforms.

The SNMP agent DPI Application Programming Interface (API) is for the DPI subagent programmer.

The following RFCs are related to SNMP and will be helpful when you are programming an SNMP API (see Appendix E, “Related protocol specifications,” on page 743 for information about accessing RFCs):

- RFC 1592 is the SNMP DPI 2.0 RFC.
- RFC 1901 through RFC 1908 are the SNMP Version 2 RFCs.

The primary goal of RFC 1592 is to specify the SNMP DPI. This is a protocol by which subagents can exchange SNMP related information with an agent.

To provide an environment that is generally platform independent, RFC 1592 strongly suggests that you also define a DPI API. There is a sample DPI API available in the RFC. The document describes the same sample API as the IBM supported DPI Version 2.0 API. See "DPI subagent example" on page 107.

SNMP agents and subagents

SNMP agents are primarily responsible for responding to SNMP operation requests. An operation request can originate from any entity that supports the management portion of the SNMP protocol. An example of this is z/OS UNIX SNMP command, osnmp, included with this version of TCP/IP. Examples of SNMP operations are GET, GETNEXT, and SET. An operation is performed on an MIB object.

A subagent extends the set of MIB objects provided by the SNMP agent. With the subagent, you define MIB objects useful in your own environment and register them with the SNMP agent.

When the agent receives a request for an MIB object, it passes the request to the subagent. The subagent then returns a response to the agent. The agent creates an SNMP response packet and sends the response to the remote network management station that initiated the request. The existence of the subagent is transparent to the network management station.

To allow the subagents to perform these functions, the agent provides for subagent connections through:

- A TCP connection
- An AF_UNIX streams connection
For the TCP connections, the agent binds to an arbitrarily chosen TCP port and listens for connection requests. A well-known port is not used. Every invocation of the SNMP agent could potentially use a different TCP port.

For UNIX streams connections, the agent is within the same machine. AF_UNIX connections should be used if possible, because they do not pass into TCP/IP, but flow only within UNIX System Services and hence require fewer system resources.

A DPI SNMP Subagent does not have to directly retrieve a dpiMIB object or objects, but instead uses either DPIconnect_to_agent_TCP() or DPIconnect_to_agent_UNIXstream(). DPIconnect_to_agent_TCP automatically retrieves the object dpiPortForTCP from the dpiMIB through an SNMP agent. DPIconnect_to_agent_TCP then establishes an AF_INET6 or AF_INET TCP socket connection with the SNMP agent.

The query_DPI_port() function issued in Version 1.1 is implicitly run by the DPIconnect_to_agent_TCP() function. The DPI subagent programmer would normally use the DPIconnect_to_agent_TCP() function to connect to the agent, and hence does not need to explicitly retrieve the value of the DPI TCP port.

Conversely, DPIconnect_to_agent_UNIXstream retrieves the value of the object dpiPathNameForUnixStream from the dpiMIB to establish an AF_UNIX connection with the SNMP agent.

After a successful connection to the SNMP agent the subagent registers the MIB trees for the set of variables it supports with the SNMP agent. When all variable classes are registered, the subagent waits for requests from the SNMP agent.

If connections to the SNMP agent are restricted by the security product, then the security product user ID associated with the subagent must be permitted to the agent’s security product resource name for the connection to be accepted. See the Simple Network Management Protocol (SNMP) information in the information in the z/OS Communications Server: IP Configuration Guide for more information about security product access between subagents and the z/OS Communications Server SNMP agent.

DPI agent requests

The SNMP agent can initiate several DPI requests:

- CLOSE
- COMMIT
- GET
- GETBULK
- GETNEXT
- SET
- UNDO
- UNREGISTER

The GET, GETNEXT, and SET requests correspond to the SNMP requests that a network management station can make. The subagent responds to a request with a response packet. The response packet can be created using the mkDPIresponse() library routine, which is part of the DPI API library.

The GETBULK requests are translated into multiple GETNEXT requests by the agent. According to RFC 1592, a subagent can request that the GETBULK be
passed to it, but the z/OS version of DPI does not yet support that request. (See Appendix E, “Related protocol specifications,” on page 743 for information about accessing RFCs.)

The COMMIT, UNDO, UNREGISTER, and CLOSE are specific SNMP DPI requests.

The subagent normally responds to a request with a RESPONSE packet. For the CLOSE and UNREGISTER request, the subagent does not need to send a RESPONSE.

See the following related information.

- “DPI subagent GETNEXT processing” on page 53
- “DPI subagent UNREGISTER request” on page 55
- “DPI subagent TRAP request” on page 54
- “DPI subagent CLOSE request” on page 56
- “Overview of subagent processing” on page 107
- “SNMP DPI: Connecting to the agent” on page 109
- “SNMP DPI: Registering a subtree with the agent” on page 111
- “SNMP DPI: Processing requests from the agent” on page 113
- “SNMP DPI: Processing a GET request” on page 116
- “SNMP DPI: Processing a SET/COMMIT/UNDO request” on page 122

### SNMP DPI version 2.0 library

z/OS Communications Server provides the following DPI library routines:

<table>
<thead>
<tr>
<th>Name</th>
<th>Contents</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>snmp_dpi.h</td>
<td>header file</td>
<td>/usr/lpp/tcpip/snmp/include</td>
</tr>
<tr>
<td>snmp_IDPI.o</td>
<td>• z/OS UNIX System Services</td>
<td>/usr/lpp/tcpip/snmp/build/libdpi20</td>
</tr>
<tr>
<td></td>
<td>object files</td>
<td></td>
</tr>
<tr>
<td>snmp_mDPI.o</td>
<td>• DPI Version 2.0 library</td>
<td></td>
</tr>
<tr>
<td></td>
<td>functions</td>
<td></td>
</tr>
<tr>
<td>snmp_qDPI.o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dpi_mvs_sample.c</td>
<td>SNMP DPI Version 2.0 C sample</td>
<td>/usr/lpp/tcpip/samples</td>
</tr>
<tr>
<td></td>
<td>source</td>
<td></td>
</tr>
<tr>
<td>dpiSimpl.mi2</td>
<td>SNMP DPI Version 2.0 sample</td>
<td>/usr/lpp/tcpip/samples</td>
</tr>
<tr>
<td></td>
<td>MIB definitions</td>
<td></td>
</tr>
</tbody>
</table>

### SNMP DPI Version 2.0 API

DPI Version 2.0 is intended for use with UNIX System Services sockets and is not for use with other socket libraries. A DPI subagent must include the snmp_dpi.h header in any C part that intends to use DPI. The path for snmp_dpi.h is /usr/lpp/tcpip/snmp/include. By default, when you include the snmp_dpi.h include file, you will be exposed to the DPI Version 2.0 API. For a list of the functions provided, read more about the “The snmp_dpi.h include file” on page 106. This is the recommended use of the SNMP DPI API.

When you prelink your object code into an executable file, you must use the DPI Version 2.0 functions as provided in the snmp_IDPI.o, snmp_mDPI.o, and snmp_qDPI.o object files in /usr/lpp/tcpip/snmp/build/libdpi20.
Notes:
1. The object files are located only in a z/OS UNIX file system. Files in a z/OS UNIX file system can be accessed from JCL using the path parameter on an explicit DD definition.
2. Together the snmp_dpi.h include file and the dpi_mvs_sample.c file comprise an example of the DPI Version 2.0 API.
3. Debugging information (resulting from the DPIdebug function) is routed to SYSLOGD. Ensure the SYSLOG daemon is active.
4. Compile your subagent code using the DEF(MVS) compiler option.
5. Waiting for a DPI packet depends on the platform and how the chosen transport protocol is implemented. In addition, some subagents want to control the sending of and waiting for packets themselves, because they might need to be driven by other interrupts as well.
6. There is a set of DPI transport-related functions that are implemented on all platforms to hide the platform-dependent issues for those subagents that do not need detailed control for the transport themselves.

For more information about SNMP, see the Simple Network Management Protocol (SNMP) information in the z/OS Communications Server: IP Configuration Reference or the Managing TCP/IP network resources with SNMP information in the z/OS Communications Server: IP System Administrator’s Commands.

Compiling and linking DPI Version 2.0

DPI Version 2.0 is installed in a z/OS UNIX file system only. You can build a subagent for either the UNIX System Services shell (using a z/OS UNIX file system and c89) or MVS (using JCL).

See the documentation provided by your C compiler for exact details of building a C application. The information provided in the following topics is intended as general guidance.

Compiling and linking DPI Version 2.0: UNIX System Services environment

Use c89 to compile a DPI subagent under the UNIX System Services shell. Every C file using DPI functions must include the DPI header file (snmp_dpi.h) from /usr/lpp/tcpip/snmp/include. Also include the three DPI library object files (snmp_qDPI.o, snmp_lDPI.o, and snmp_mDPI.o) from /usr/lpp/tcpip/snmp/build/libdpi20.

The following is an example of how c89 is called to compile and build dpi_mvs_sample.c:

c89 -o dpi_mvs_sample -I /usr/lpp/tcpip/snmp/include \ 
/usr/lpp/tcpip/samples/dpi_mvs_sample.c \ 
/usr/lpp/tcpip/snmp/build/libdpi20/snmp_lDPI.o \ 
/usr/lpp/tcpip/snmp/build/libdpi20/snmp_mDPI.o \ 
/usr/lpp/tcpip/snmp/build/libdpi20/snmp_qDPI.o

Use the -I option to add the z/OS UNIX file system directory where snmp_dpi.h resides to the compiler include search path.
Compiling and linking DPI Version 2.0: MVS environment

C programs that use DPI must:

- Compile with the longname compiler option
- Include snmp_dpi.h from /usr/lpp/tcpip/snmp/include

Add #include to the source code. You must inform the compiler that
/usr/lpp/tcpip/snmp/include should be searched for include files. Use either a
SYSLIB DD with a PATH parameter pointing to the z/OS UNIX file system
directory, or use the SEARCH compiler parameter.

Prelink DPI subagent to resolve longnames. In the prelink JCL, define three DDs
pointing to each DPI object file, and then include each, such as:

```
DPI1 DD PATH='/usr/lpp/tcpip/snmp/build/libdpi20/snmp_lDPI.o'
DPI2 DD PATH='/usr/lpp/tcpip/snmp/build/libdpi20/snmp_mDPI.o'
DPI3 DD PATH='/usr/lpp/tcpip/snmp/build/libdpi20/snmp_qDPI.o'
```

```
INCLUDE DPI1
INCLUDE DPI2
INCLUDE DPI3
```

Then, link edit the prelink output as usual.

DPI Version 1.x base code considerations

Use the DPI Version 1.1 API as described in [Chapter 2, “SNMP agent Distributed
Protocol Interface version 1.1,” on page 3](#).

The DPI Version 2.0 API provided with z/OS is for UNIX System Services sockets
use only. Earlier versions of DPI were supported on C sockets.

See "[Migrating your SNMP DPI subagent to Version 2.0](#)" for more detail about the
changes that you must make to your DPI Version 1.x source.

If you want to convert to DPI Version 2.0, which prepares you also for SNMP
Version 2, you must make changes to your code.

You can keep your existing DPI Version 1.1 subagent and communicate with a
DPI-capable agent that supports DPI Version 1.1 in addition to DPI Version 2.0. For
example, the z/OS SNMP agent provides support for multiple versions of DPI,
including Version 1.0, Version 1.1, and Version 2.0.

Migrating your SNMP DPI subagent to Version 2.0

The information presented in this topic are guidelines and are not exact procedures.
Your specific implementation will vary from the guidelines presented.

When you want to change your DPI Version 1.x-based subagent code to the DPI
Version 2.0 level, use these guidelines for the required actions and the
recommended actions.

Required actions for migrating your SNMP DPI subagent to
Version 2.0

The following actions are required to migrate SNMP DPI subagent to Version 2.0:
Add an mkDPIopen() call and send the created packet to the agent. This opens your DPI connection with the agent. Wait for the response and ensure that the open is accepted. You need to pass a subagent ID (object identifier), which must be a unique ASN.1 OID.

See “The mkDPIopen() function” on page 65 for more information.

Change your mkDPIregister() calls and pass the parameters according to the new function prototype. You must also expect a RESPONSE to the REGISTER request.

See “The mkDPIregister() function” on page 67 for more information.

Change mkDPIset() and mkDPIlist() calls to the new mkDPIset() call. Basically all mkDPIset() calls are now of the DPI Version 1.1 mkDPIlist() form.

See “The mkDPIset() function” on page 71 for more information.

Change mkDPItrap() and mkDPItrape() calls to the new mkDPItrap() call. Basically all mkDPItrap() calls are now of the DPI Version 1.1 mkDPItrape() form.

See “The mkDPItrap() function” on page 73 for more information.

Add code to recognize DPI RESPONSE packets, which should be expected as a result of OPEN, REGISTER, and UNREGISTER requests.

Add code to expect and handle the DPI UNREGISTER packet from the agent. It might send such packets if an error occurs or if a higher priority subagent registers the same subtree as you have registered.

Add code to unregister your subtrees and close the DPI connection when you want to terminate the subagent.

See “The mkDPIunregister() function” on page 75 and “The mkDPIclose() function” on page 64 for more information.

Add code to use the new SNMP Version 2 error codes as defined in the snmp_dpi.h include file.

When migrating DPI Version 1.1 subagents to DPI Version 2.0, remove the include for manifest.h.

Change your code that handles a GET request. It should return a varBind with SNMP_TYPE_noSuchObject value or SNMP_TYPE_noSuchInstance value instead of an error SNMP_ERROR_noSuchName if the object or the instance do not exist. This is not considered an error any more. Therefore, you should return an SNMP_ERROR_noError with an error index of 0.

Note: A varBind (variable binding) is the group ID, instance ID, type, length, and value that completely describes a variable in the MIB.

Change your code that handles a GETNEXT request. It should return a varBind with SNMP_TYPE_endOfMibView value instead of an error SNMP_ERROR_noSuchName if you reach the end of your MIB or subtree. This is not considered an error any more. Therefore, you should return an SNMP_ERROR_noError with an error index of 0.

Change your code that handles SET requests to follow the two-phase SET/COMMIT scheme as described in “DPI subagent SET processing” on page 52.

See the sample handling of SET/COMMIT/UNDO in “SNMP DPI: Processing a SET/COMMIT/UNDO request” on page 122.

Recommended actions for migrating your SNMP DPI subagent to Version 2.0

The following actions are recommended:
• Do not refer to the object ID pointer (object_p) in the snmp_dpi_xxxx_packet structures any more. Instead start using the group_p and instance_p pointers. The object_p pointer might be removed in a future version of the DPI API.

• Check “Transport-related DPI API functions” on page 77 to see if you want to use those functions instead of using your own code for those functions.

• Consider using more than one varBind per DPI packet. You can specify this on the REGISTER request. You must then be prepared to handle multiple varBinds per DPI packet. The varBinds are chained through the various snmp_dpi_xxxx_packet structures.

  See “The mkDPIopen() function” on page 65 for more information.

• Consider specifying a timeout when you issue a DPI OPEN or DPI REGISTER.

  See “The mkDPIopen() function” on page 65 and “The mkDPIregister() function” on page 67 for more information.

• Ensure SYSLOGD is active. The result of using DPIdebug is routed to SYSLOGD. For information on how to configure SYSLOGD, see the Syslog daemon information in the z/OS Communications Server: IP Configuration Reference.

DPI Version 2.0 recognizes mkDPIlist; however, Version 2.0 subagents should use mkDPIset instead.

**snmp_dpi_xxxx_packet structures name changes**

A number of field names in the snmp_dpi_xxxx_packet structures have changed so that the names are now more consistent throughout the DPI code.

The new names indicate if the value is a pointer (_p) or a union (_u). The names that have changed and that affect the subagent code are listed in the table below.

<table>
<thead>
<tr>
<th>Old name</th>
<th>New name</th>
<th>Data structure (XXXX)</th>
</tr>
</thead>
<tbody>
<tr>
<td>group_id</td>
<td>group_p</td>
<td>getnext</td>
</tr>
<tr>
<td>object_id</td>
<td>object_p</td>
<td>get, getnext, set</td>
</tr>
<tr>
<td>value</td>
<td>value_p</td>
<td>set</td>
</tr>
<tr>
<td>type</td>
<td>value_type</td>
<td>set</td>
</tr>
<tr>
<td>next</td>
<td>next_p</td>
<td>set</td>
</tr>
<tr>
<td>enterprise</td>
<td>enterprise_p</td>
<td>trap</td>
</tr>
<tr>
<td>packet_body</td>
<td>data_u</td>
<td>dpi_hdr</td>
</tr>
<tr>
<td>dpi_get</td>
<td>get_p</td>
<td>hdr (packet_body)</td>
</tr>
<tr>
<td>dpi_getnext</td>
<td>next_p</td>
<td>hdr (packet_body)</td>
</tr>
<tr>
<td>dpi_set</td>
<td>set_p</td>
<td>hdr (packet_body)</td>
</tr>
<tr>
<td>dpi_trap</td>
<td>trap_p</td>
<td>hdr (packet_body)</td>
</tr>
</tbody>
</table>

There is no clean approach to make this change transparent. You probably will need to change the names in your code. You could try a simple set of defines like:

```c
#define packet_body     data_u
#define dpi_get          get_p
#define dpi_set          set_p
#define dpi_next         next_p
#define dpi_response     resp_p
#define dpi_trap         trap_p
#define group_id         group_p
#define object_id        object_p
#define value            value_p
#define type             value_type
#define next             next_p
#define enterprise_p     enterprise_p
```

Chapter 3. SNMP agent Distributed Protocol Interface version 2.0  47
If the names conflict with other definitions, change your code.

**SNMP DPI environment variables**

Table 2 provides a list of environment variables for the SNMP DPI.

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMP_PORT</td>
<td>Specifies the port to which a DPI subagent will direct a connection query. This variable defaults to 161, which is the default port on which the SNMP agent listens for queries.</td>
</tr>
</tbody>
</table>

**SNMP DPI subagent programming concepts**

When implementing a subagent, use the DPI Version 2 approach and keep the following in mind:

- Use the SNMP Version 2 error codes only, even though there are definitions for the SNMP Version 1 error codes.
- Implement the SET, COMMIT, UNDO processing properly.
- Use the SNMP Version 2 approach for GET requests, and pass back noSuchInstance value or noSuchObject value if appropriate. Continue to process all remaining varBinds.

More than one varBind can be specified in the SNMP PDU for the requested operation. For example, using the SNMP network manager, a user can request the retrieval of multiple objects in the same request (GET or GETNEXT). The varBind portion of the PDU sent would include multiple object identifiers (OIDs). The subagent limitations are passed to the agent through the max_varBinds parameter on the mkDPIopen call. When the subagent receives a request from the agent, it needs to handle multiple OIDs per request if it specified a max_varBinds value other than 1.

- Use the SNMP Version 2 approach for GETNEXT, and pass back endOfMibView value if appropriate. Continue to process all remaining varBinds.
- Specify the timeout period in the OPEN and REGISTER packets, when you are processing a request from the agent (GET, GETNEXT, SET, COMMIT, or UNDO). If you fail to respond within the timeout period, the agent will probably close your DPI connection and discard your RESPONSE packet if it comes in later. If you can detect that the response is not going to be received in the time period, then you might decide to stop the request and return an SNMP_ERROR_genErr in the RESPONSE.
- Issue an SNMP DPI ARE_YOU_THERE request periodically to ensure that the agent is still connected and still knows about you.
- OS/2 runs on an ASCII based machine. However, when you are running a subagent on an EBCDIC based machine and you use the (default) native character set, all OID strings and all variable values of type OBJECT_IDENTIFIER or DisplayString objects that are known by the agent (in its compiled MIB) will be passed to you in EBCDIC format. OID strings include the group ID, instance ID, enterprise ID, and subagent ID. You should structure your response with the EBCDIC format.
- If you receive an error RESPONSE on the OPEN packet, you will also receive a DPI CLOSE packet with an SNMP_CLOSE_openError code. In this situation, the agent closes the connection.
- The DisplayString is only a textual convention. In the SNMP PDU (SNMP packet), the type is an OCTET_STRING.

  When the type is OCTET_STRING, it is not clear if this is a DisplayString or any arbitrary data. This means that the agent can only know about an object being a DisplayString if the object is included in some sort of a compiled MIB. If it is, the agent will use SNMP_TYPE_DisplayString in the type field of the varBind in a DPI SET packet. When you send a DisplayString in a RESPONSE packet, the agent will handle it as such.

  See the following related information.
  "DPI subagent example" on page 107

**Specifying the SNMP DPI API**

The following topics describe each type of DPI processing in this order:
- Connect processing
- OPEN request
- REGISTER request
- GET, SET, GETNEXT, GETBULK, TRAP, and ARE_YOU_THERE processing
- UNREGISTER request
- CLOSE request

**DPI subagent connect processing**

There are various connect functions that allow connections through either TCP or UNIXstream. Determine which is appropriate for you by evaluating whether you are connecting to the same machine or a different machine. If the agent and the subagent are using the same machine, use the UNIXstream connection for better performance. If the agent and the subagent are using different machines, you must use the TCP connection. There are two connect processing parameters:
- Hostname—name or the IP address of the agent
- Community name—password that allows the DPI connect function to obtain the port (for TCP) or path name (for UNIX) that allows the socket connect to occur.

  See the following related information.
  "SNMP DPI: Connecting to the agent" on page 109

**DPI subagent OPEN request**

Next, the DPI subagent must open a connection with the agent. To do so, it must send a DPI OPEN packet in which these parameters must be specified:

- The maximum timeout value in seconds. The agent is requested to wait this long for a response to any request for an object that is being handled by this subagent.

  The agent can have an absolute maximum timeout value which is used if the subagent asks for too large a timeout value. The value 0 can be used to indicate that the agent default timeout value should be used. A subagent is advised to use a reasonably short interval of a few seconds. If a specific subtree needs more time, a specific REGISTER can be done for that subtree with a longer timeout value.

- The maximum number of varBinds that the subagent is prepared to handle per DPI packet. Specifying 1 would result in DPI Version 1 behavior of one varBind...
per DPI packet that the agent sends to the subagent. The value 0 means that the
agent will try to combine up to as many varBinds as are present in the SNMP
packet that belongs to the same subtree.

- The character set you want to use. The default value 0 is the native character set
of the machine platform where the agent runs. Because the subagent and agent
normally run on the same system or platform, use the native EBCDIC character
set on MVS.

If your platform is EBCDIC-based, using the native EBCDIC character set makes
it easy to recognize the string representations of the fields, such as the group ID
and instance ID. At the same time, the agent translates the value from ASCII
NTV to EBCDIC and from EBCDIC to ASCII NTV for objects that it knows from
a compiled MIB to have a textual convention of DisplayString. This fact cannot
be determined from the SNMP PDU encoding because, in the PDU, the object is
only known to be an OCTET STRING.

If your subagent runs on an ASCII-based platform and the agent runs on an
EBCDIC-based platform (or the other way around), you can specify that you
want to use the ASCII character set. The agent and subagent programmers know
how to handle the string-based data in this situation.

- The subagent ID. This is an ASN.1 object identifier that uniquely identifies the
subagent. This OID is represented as a null-terminated string using the selected
character set.

For example: 1.3.5.1.2.3.4.5

- The subagent description. This is a DisplayString describing the subagent. This
is a character string using the selected character set.

For an example see “DPI subagent example” on page 107

After a subagent has sent a DPI OPEN packet to an agent, it should expect a DPI
RESPONSE packet that informs the subagent about the result of the request. The
packet ID of the RESPONSE packet should be the same as that of the OPEN
request to which the RESPONSE packet is the response. See “DPI RESPONSE error
codes” on page 102 for a list of valid codes that can be expected.

If you receive an error RESPONSE on the OPEN packet, you also receive a DPI
CLOSE packet with an SNMP_CLOSE_openError code. In this situation, the agent
closes the connection.

If the OPEN is accepted, the next step is to REGISTER one or more MIB subtrees.

See the following related information.

“SNMP DPI: Connecting to the agent” on page 109

**DPI subagent REGISTER request**

Before a subagent receives any requests for MIB objects, it must first register with
the SNMP agent the variables or subtree that it supports. The subagent must
specify the following parameters in the REGISTER request:

- The subtree to be registered.

**Object level registration:** This is a null-terminated string in the selected
character set that specifies the subtree to be registered. Object level registration
requires a trailing period following the object number, indicating a register
request to support all instances of an object (for example, ifDescr). Object level
registration requires that the subtree must have a trailing period. For example:
1.3.6.1.2.1.2.2.1.2.
**Instance level registration:** Instance level registration does not require a trailing period for the subtree. Instance level registration can be used to allow different subagents to support separate instances of a particular MIB object. Registration by subagents at the instance level rather than the object level is accomplished by simply adding the instance number after the object number when building the registration packet using the mkDPIregister call. For example, passing the object number `1.3.6.1.2.1.2.2.1.2.` (note the ending period) would support all instances of ifDescr. However, a subagent could pass an object or instance number like `1.3.6.1.2.1.2.2.1.2.8` (note the addition of the 8 after the period) to support only ifDescr.8 (instance 8).

- The requested priority for the registration. The values are:
  - `-1` Request for the best available priority
  - `0` Request for the next best available priority than the highest (best) priority currently registered for this subtree
  - `NNN` Any other positive value requests a specific priority, if available, or the next best priority that is available.

- The maximum timeout value in seconds. The agent is requested to wait this long for a response to any request for an object in this subtree. The agent can have an absolute maximum timeout value that is used if the subagents ask for too large a timeout value. The value `0` can be used to indicate that the DPI OPEN value should be used for timeout.

After a subagent has sent a DPI REGISTER packet to the agent, it should expect a DPI RESPONSE packet that informs the subagent about the result of the request. The packet ID of the RESPONSE packet should be the same as that of the REGISTER packet to which the RESPONSE packet is the response.

If the response is successful, the error_index field in the RESPONSE packet contains the priority that the agent assigned to the subtree registration. See “DPI RESPONSE error codes” on page 102 for a list of valid codes that can be expected.

**Error Code: higherPriorityRegistered:** The response to a REGISTER request might return the error code higherPriorityRegistered. This might be caused by one of the following:

- Another subagent already registered the same subtree at a better priority than what you are requesting.
- Another subagent already registered a subtree at a higher level (at any priority).

For instance, if a registration already exists for subtree `1.2.3.4.5.6` and you try to register for subtree `1.2.3.4.5.6.<anything>` then you will receive the higherPriorityRegistered error code.

If you receive this error code, your subtree will be registered, but you will not see any requests for the subtree. These requests are passed to the subagent that registered with a better priority. If you remain connected and the other subagent goes away, you will get control over the subtree at that point in time.

See the following related information.

“SNMP DPI: Registering a subtree with the agent” on page 111

**DPI subagent GET processing**

The DPI GET packet holds one or more varBinds that the subagent has taken responsibility for.
If the subagent encounters an error while processing the request, it creates a DPI RESPONSE packet with an appropriate error indication in the error_code field and sets the error_index to the position of the varBind at which the error occurs. The first varBind is index 1, the second varBind is index 2, and so on. No name, type, length, or value information needs to be provided in the packet because, by definition, the varBind information is the same as in the request to which this is a response and the agent still has that information.

If there are no errors, the subagent creates a DPI RESPONSE packet in which the error_code is set to SNMP_ERROR_noError (0) and error_index is set to 0. The packet must also include the name, type, length, and value of each varBind requested.

When you get a request for a nonexisting object or a nonexisting instance of an object, you must return a NULL value with a type of SNMP_TYPE_noSuchObject or SNMP_TYPE_noSuchInstance respectively. These two values are not considered errors, so the error_code and error_index values should be 0.

The DPI RESPONSE packet is then sent back to the agent.

See the following related information.

- "SNMP DPI: Processing a GET request" on page 116
- "The mkDPIresponse() function" on page 69

**DPI subagent SET processing**

A DPI SET packet contains the name, type, length, and value of each requested varBind, plus the value type, value length, and value to be set.

If the subagent encounters an error while processing the request, it creates a DPI RESPONSE packet with an appropriate error indication in the error_code field and an error_index listing the position of the varBind at which the error occurs. The first varBind is index 1, the second varBind is index 2, and so on. No name, type, length, or value information needs to be provided in the packet because, by definition, the varBind information is the same as in the request to which this is a response and the agent still has that information.

If there are no errors, the subagent creates a DPI RESPONSE packet in which the error_code is set to SNMP_ERROR_noError (0) and error_index is set to 0. No name, type, length, or value information is needed because the RESPONSE to a SET should contain exactly the same varBind data as the data present in the request. The agent can use the values it already has.

This suggests that the agent must keep state information, and that is the case. It needs to do that anyway to be able to later pass the data with a DPI COMMIT or DPI UNDO packet. Because there are no errors, the subagent must have allocated the required resources and prepared itself for the SET. It does not yet carry out the SET, which will be done at COMMIT time.

The subagent sends a DPI RESPONSE packet, indicating success or failure for the preparation phase, back to the agent. The agent will issue a SET request for all other varBinds in the same original SNMP request it received. This can be to the same subagent or to one or more different subagents.

After all SET requests have returned a "no error" condition, the agent starts sending DPI COMMIT packets to the subagents. If any SET request returns an
error, the agent sends DPI UNDO packets to those subagents that indicated successful processing of the SET preparation phase.

When the subagent receives the DPI COMMIT packet, all the varBind information will again be available in the packet. The subagent can now carry out the SET request.

If the subagent encounters an error while processing the COMMIT request, it creates a DPI RESPONSE packet with value SNMP_ERROR_commitFailed in the error_code field and an error_index that lists at which varBind the error occurs. The first varBind is index 1, the second varBind is 2, and so on. No name, type, length, or value information is needed. The fact that a commitFailed error exists does not mean that this error should be returned easily. A subagent should do all that is possible to make a COMMIT succeed.

If there are no errors and the SET and COMMIT have been carried out with success, the subagent creates a DPI RESPONSE packet in which the error_code is set to SNMP_ERROR_noError (0) and error_index is set to 0. No name, type, length, or value information is needed.

So far discussion has focused on successful SET and COMMIT sequences. However, after a successful SET, the subagent might receive a DPI UNDO packet. The subagent must now undo any preparations it made during the SET processing, such as free allocated memory.

Even after a COMMIT, a subagent might still receive a DPI UNDO packet. This occurs if some other subagent could not complete a COMMIT request. Because of the SNMP requirement that all varBinds in a single SNMP SET request must be changed as if simultaneous, all committed changes must be undone if any of the COMMIT requests fail. In this case the subagent must try and undo the committed SET operation.

If the subagent encounters an error while processing the UNDO request, it creates a DPI RESPONSE packet with value SNMP_ERROR_undoFailed in the error_code field and an error_index that lists at which varBind the error occurs. The first varBind is index 1, the second varBind is 2, and so on. No name, type, length, or value information is needed. The fact that an undoFailed error exists does not mean that this error should be returned easily. A subagent should do all that is possible to make an UNDO succeed.

If there are no errors and the UNDO has been successful, the subagent creates a DPI RESPONSE packet in which the error_code is set to SNMP_ERROR_noError (0) and error_index is set to 0. No name, type, length, or value information is needed.

**DPI subagent GETNEXT processing**

The DPI GETNEXT packet contains the objects on which the GETNEXT operation must be performed. For this operation, the subagent is to return the name, type, length, and value of the next variable it supports whose (ASN.1) name lexicographically follows the one passed in the group ID (subtree) and instance ID.

In this case, the instance ID might not be present (NULL) in the incoming DPI packet, implying that the NEXT object must be the first instance of the first object in the subtree that was registered.
It is important to realize that a given subagent might support several discontinuous sections of the MIB tree. In that situation, it would be incorrect to jump from one section to another. This problem is correctly handled by examining the group ID in the DPI packet. This group ID represents the reason why the subagent is being called. It holds the prefix of the tree that the subagent had indicated it supported (registered).

If the next variable supported by the subagent does not begin with that prefix, the subagent must return the same object instance as in the request, for example the group ID and instance ID with a value of SNMP_TYPE_endOfMibView (implied NULL value). This endOfMibView is not considered an error, so the error_code and error_index should be 0. If required, the SNMP agent will call upon the subagent again, but pass it a different group ID (prefix). This is illustrated in the discussion below.

Assume there are two subagents. The first subagent registers two distinct sections of the tree: A and C. In reality, the subagent supports variables A.1 and A.2, but it correctly registers the minimal prefix required to uniquely identify the variable class it supports.

The second subagent registers section B, which appears between the two sections registered by the first agent.

If a management station begins browsing the MIB, starting from A, the following sequence of queries of the form GET-NEXT (group ID, instance ID) would be performed:

Subagent 1 gets called:
   get-next(A,none) = A.1
   get-next(A,1) = A.2
   get-next(A,2) = endOfMibView

Subagent 2 is then called:
   get-next(B,none) = B.1
   get-next(B,1) = endOfMibView

Subagent 1 gets called again:
   get-next(C,none) = C.1

**DPI subagent GETBULK processing request**

You must ask the agent to translate GETBULK requests into multiple GETNEXT requests. This is basically the default and is specified in the DPI REGISTER packet. The majority of DPI subagents will run on the same machine as the agent, or on the same physical network. Therefore, repetitive GETNEXT requests remain local, and, in general, should not be a problem.

**Note:** Currently, z/OS SNMP does not support GETBULK protocol between agent and subagent. These requests are translated into multiple GETNEXT requests.

See the following related information.

“DPI subagent GETNEXT processing” on page 53

**DPI subagent TRAP request**

A subagent can request that the SNMP agent generates a trap. The subagent must provide the desired values for the generic and specific parameters of the trap. It
can optionally provide a set of one or more name, type, length, or value parameters that will be included in the trap packet.

It can optionally specify an enterprise ID (object identifier) for the trap to be generated. If a NULL value is specified for the enterprise ID, the agent will use the subagent identifier from the DPI OPEN packet as the enterprise ID to be sent with the trap.

See the following related information.

“SNMP DPI: Generating a TRAP” on page 126

DPI subagent ARE_YOU_THERE request

A subagent can send an ARE_YOU_THERE packet to the agent. If the connection is in a healthy state, the agent responds with a RESPONSE packet with SNMP_ERROR_DPI_noError. If the connection is not in a healthy state, the agent might respond with a RESPONSE packet with an error indication, but the agent might not react at all. In this situation, you would time out while waiting for a response.

DPI subagent UNREGISTER request

A subagent can unregister a previously registered subtree. The subagent must specify the following parameters in the UNREGISTER request:

- The subtree to be unregistered.

  Object level unregistration: This is a null-terminated string in the selected character set specifying the subtree that is to be unregistered. Object level unregistration requires a trailing period, which is following the object number, indicating an unregister request to all supported instances of an object (for example, ifDescr). Object level unregistration requires that the subtree must have a trailing period. For example: 1.3.6.1.2.1.2.2.1.2.

  Instance level unregistration: Instance level unregistration does not require a trailing period for the subtree.

Note: Unregistration at the instance level can be done only if the original registration was done using instance level registration.

Unregistration by subagent at the instance level, rather than the object level, is accomplished by adding the instance number after the object number when building the unregistration packet using the mkDPIunregister call. For example, passing the object number 1.3.6.1.2.1.2.2.1.2. (note the ending period) would support all instances of ifDescr. However, a subagent could pass an object or instance number 1.3.6.1.2.1.2.2.1.2.8 (note the addition of the 8 after the period) to support only ifDescr.8 (instance 8).

- The reason for the unregister. See “DPI UNREGISTER reason codes” on page 103 for a list of valid reason codes.

After a subagent has sent a DPI UNREGISTER packet to the agent, it should expect a DPI RESPONSE packet that informs the subagent about the result of the request. The packet ID of the RESPONSE packet should be the same as that of the REGISTER packet to which the RESPONSE packet is the response. See “DPI RESPONSE error codes” on page 102 for a list of valid codes that can be expected.

A subagent should also be prepared to handle incoming DPI UNREGISTER packets from the agent. In this situation, the DPI packet contains a reason code for the UNREGISTER request. A subagent does not have to send a response to an
UNREGISTER request. The agent assumes that the subagent will handle it appropriately. The registration is removed regardless of what the subagent returns.

See the following related information.

“SNMP DPI: Processing an UNREGISTER request” on page 125

DPI subagent CLOSE request

When a subagent is finished and wants to end processing, it should first UNREGISTER its subtrees and then close the connection with the agent. To do so, the subagent must send a DPI CLOSE packet, which specifies a reason for the closing. See “DPI CLOSE reason codes” on page 102 for a list of valid codes. You should not expect a response to the CLOSE request.

A subagent should also be prepared to handle an incoming DPI CLOSE packet from the agent. In this case, the packet contains a reason code for the CLOSE request. A subagent does not have to send a response to a CLOSE request. The agent assumes that the subagent will handle it appropriately. The close takes place regardless of what the subagent does with it.

See the following related information.

“SNMP DPI: Processing a CLOSE request” on page 126

Multithreading programming considerations

The DPI Version 2.0 program does not support multithreaded subagents.

There are several static buffers in the DPI code. For compatibility reasons, that cannot be changed. Real multithread support will probably mean several potentially incompatible changes to the DPI Version 2.0 API.

Use a locking mechanism: Because the DPI API is not reentrant, to use your subagent in a multithreaded process you should use some locking mechanism of your own around the static buffers. Otherwise, one thread might be writing into the static buffer while another is writing into the same buffer at the same time. There are two static buffers. One buffer is for building the serialized DPI packet before sending it out and the other buffer is for receiving incoming DPI packets.

Basically, all DPI functions that return a pointer to an unsigned character are the DPI functions that write into the static buffer to create a serialized DPI packet:

```
mkDPIAreYouThere()
mk DPIopen ()
mk D P Iregister ()
mkDPIunregister ()
mkDPItrap ()
mkDPIresponse ()
mk DPIpacket ()
mk DPIclose ()
```

After you have called the DPI send packet to agent() function for the buffer, which is pointed to by the pointer returned by one of the preceding functions, the buffer is free to use again.

There is one function that reads the static input buffer:

```
pDPIpacket ()
```
The input buffer gets filled by the DPI\texttt{await\_packet\_from\_agent()} function. Upon return from the await, you receive a pointer to the static input buffer. The \texttt{pDPIpacket()} function parses the static input buffer and returns a pointer to dynamically allocated memory. Therefore, after the \texttt{pDPIpacket()} call the buffer is available for use again.

The DPI internal handle structures and control blocks used by the underlying code to send and receive data to and from the agent are also static data areas. Ensure that you use your own locking mechanism around the functions that add, change, or delete data in those static structures. The functions that change those internal static structures are:

\begin{verbatim}
DPIconnect\_to\_agent\_TCP() /* everyone has this one */
DPIconnect\_to\_agent\_UNIXstream() /* supported */
DPIdisconnect\_from\_agent() /* everyone has this one */
\end{verbatim}

Other functions will access the static structures. These other functions must be assured that the structure is not being changed while they are referencing it during their execution. The other functions are:

\begin{verbatim}
DPI\texttt{await\_packet\_from\_agent()}
DPI\texttt{send\_packet\_to\_agent()}
DPI\texttt{get\_fd\_for\_handle()}
\end{verbatim}

While the last three functions can be executed concurrently in different threads, you must ensure that no other thread is adding or deleting handles in these static structures during this process.

\section*{Functions, data structures, and constants}

Use these lists to locate the descriptions for the functions, data structures, and constants.

\subsection*{Basic DPI Functions:}

\begin{verbatim}
"The DPI\texttt{debug()} function\" on page 59
"The DPI\texttt{PACKET\_LEN()} macro\" on page 60
"The \texttt{fDPIparse()} function\" on page 61
"The \texttt{fDPIset()} function\" on page 62
"The \texttt{mkDPIAreYouThere()} function\" on page 63
"The \texttt{mkDPIclose()} function\" on page 64
"The \texttt{mkDPIopen()} function\" on page 65
"The \texttt{mkDPIregister()} function\" on page 67
"The \texttt{mkDPIresponse()} function\" on page 69
"The \texttt{mkDPIset()} function\" on page 71
"The \texttt{mkDPItrap()} function\" on page 73
"The \texttt{mkDPIunregister()} function\" on page 75
"The \texttt{pDPIpacket()} function\" on page 76
\end{verbatim}

\subsection*{DPI Transport-Related Functions:}

\begin{verbatim}
"The DPI\texttt{await\_packet\_from\_agent()} function\" on page 78
"The DPI\texttt{connect\_to\_agent\_TCP()} function\" on page 80
"The DPI\texttt{connect\_to\_agent\_UNIXstream()} function\" on page 82
"The DPI\texttt{disconnect\_from\_agent()} function\" on page 84
"The DPI\texttt{get\_fd\_for\_handle()} function\" on page 85
"The DPI\texttt{send\_packet\_to\_agent()} function\" on page 86
"The \texttt{lookup\_host()} function\" on page 88
"The lookup\_host6() function\" on page 89
\end{verbatim}
Data Structures:
- “The snmp_dpi_close_packet structure” on page 91
- “The snmp_dpi_get_packet structure” on page 92
- “The snmp_dpi_hdr structure” on page 93
- “The snmp_dpi_next_packet structure” on page 95
- “The snmp_dpi_resp_packet structure” on page 96
- “The snmp_dpi_set_packet structure” on page 97
- “The snmp_dpi_ureg_packet structure” on page 99
- “The snmp_dpi_u64 structure” on page 100

Constants and Values:
- “DPI CLOSE reason codes” on page 102
- “DPI packet types” on page 102
- “DPI RESPONSE error codes” on page 102
- “DPI UNREGISTER reason codes” on page 103
- “DPI SNMP value types” on page 103
- “Value representation of DPI SNMP value types” on page 104

Related Information:
- “DPI OPEN character set selection” on page 101
- “The snmp_dpi.h include file” on page 106

**Basic DPI API functions**

This topic describes each of the basic DPI functions that are available to the DPI subagent programmer.

**The Basic DPI Functions are:**
- “The DPIdebug() function” on page 59
- “The DPI_PACKET_LEN() macro” on page 60
- “The fDPIparse() function” on page 61
- “The fDPIset() function” on page 62
- “The mkDPIAreYouThere() function” on page 63
- “The mkDPIclose() function” on page 64
- “The mkDPIopen() function” on page 65
- “The mkDPIregister() function” on page 67
- “The mkDPIresponse() function” on page 69
- “The mkDPIset() function” on page 71
- “The mkDPItrap() function” on page 73
- “The mkDPIunregister() function” on page 75
- “The pDPIpacket() function” on page 76
The DPIdebug() function

Format

`#include <snmp_dpi.h>

void DPIdebug(int level);`

Parameters

`level` If this value is 0, tracing is turned off. If it has any other value, tracing is turned on at the specified level. The higher the value, the more detail. A higher level includes all lower levels of tracing. Currently there are two levels of detail:

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Display packet creation and parsing.</td>
</tr>
<tr>
<td>2</td>
<td>Display hex dump of incoming and outgoing DPI packets.</td>
</tr>
</tbody>
</table>

Usage

The DPIdebug() function turns DPI internal debugging or tracing on or off.

The trace output is sent to the SYSLOG Daemon. See the IBM 3172 Enterprise-specific MIB variables information in z/OS Communications Server: IP System Administrator's Commands for more information.

Examples

```c
#include <snmp_dpi.h>

DPIdebug(2);
```

Context

“The snmp_dpi.h include file” on page 106
The DPI_PACKET_LEN() macro

Format

#include <snmp_dpi.h>

int DPI_PACKET_LEN(unsigned char *packet_p)

Parameters

packet_p

A pointer to a serialized DPI packet

Return codes

An integer representing the total DPI packet length

Usage

The DPI_PACKET_LEN macro generates C code that returns an integer representing the length of a DPI packet. It uses the first two octets in network byte order of the packet to calculate the length.

Examples

#include <snmp_dpi.h>
unsigned char *pack_p;
int length;

pack_p = mkDPIclose(SNMP_CLOSE_goingDown);
if (pack_p) {
    length = DPI_PACKET_LEN(pack_p);
    /* send packet to agent */
} /* endif */
The fDPIparse() function

Format

```c
#include <snmp_dpi.h>

void fDPIparse(snmp_dpi_hdr *hdr_p);
```

Parameters

hdr_p A pointer to the parse tree. The parse tree is represented by an
snmp_dpi_hdr structure.

Usage

The fDPIparse() function frees a parse tree that was previously created by a call to
pDPIpacket(). The parse tree might have been created in other ways too. After
calling fDPIparse(), no further references to the parse tree can be made.

A complete or partial DPI parse tree is also implicitly freed by a call to a DPI
function that serializes a parse tree into a DPI packet. The section that describes
each function tells you if this is the case. An example of such a function is
mkDPIresponse().

Examples

```c
#include <snmp_dpi.h>
snmp_dpi_hdr *hdr_p;
unsigned char *pack_p; /* assume pack_p points to */
   /* incoming DPI packet */
 hdr_p = pDPIpacket(pack_p);

/* handle the packet and when done do the following */
if (hdr_p) fDPIparse(hdr_p);
```

Context

"The snmp_dpi_hdr structure” on page 93
"The pDPIpacket() function” on page 76
"The snmp_dpi.h include file” on page 106"
The **fDPIset()** function

**Format**

```c
#include <snmp_dpi.h>

void fDPIset(snmp_dpi_set_packet *packet_p);
```

**Parameters**

`packet_p`

A pointer to the first `snmp_dpi_set_packet` structure in a chain of such structures.

**Usage**

The `fDPIset()` function is typically used if you must free a chain of one or more `snmp_dpi_set_packet` structures. This might be the case if you are in the middle of preparing a chain of such structures for a DPI RESPONSE packet, but then run into an error before you can actually make the response.

If you get to the point where you make a DPI response packet to which you pass the chain of `snmp_dpi_set_packet` structures, the `mkDPIresponse()` function will free the chain of `snmp_dpi_set_packet` structures.

**Examples**

```c
#include <snmp_dpi.h>
unsigned char *pack_p;
snmp_dpi_hdr *hdr_p;
snmp_dpi_set_packet *set_p, *first_p;
long int num1 = 0, num2 = 0;

hdr_p = pDPIpacket(pack_p); /* assume pack_p */
/* analyze packet and assume all OK */ /* points to the */
/* now prepare response; 2 varBinds */ /* incoming packet */

set_p = mkDPIset(snmp_dpi_NULL_p, /* create first one */
                "1.3.6.1.2.3.4.5." , "1.0", /* OID=1, instance=0 */
                SNMP_TYPE_Integer32,
                sizeof(num1), &num1);

if (set_p) { /* if success, then */
    first_p = set_p; /* save ptr to first */
    set_p = mkDPIset(set_p, /* chain next one */
                     "1.3.6.1.2.3.4.5." , "1.1", /* OID=1, instance=1 */
                     SNMP_TYPE_Integer32,
                     sizeof(num2), &num2);
    if (set_p) { /* success 2nd one */
        pack_p = mkDPIresponse(hdr_p, /* make response */
                               SNMP_ERROR_noError, /* It will also free */
                               0L, first_p); /* the set_p tree */
        /* send DPI response to agent */
    } else { /* 2nd mkDPIset fail */
        fDPIset(first_p); /* must free chain */
    }
} /* endif */
```

**Context**

- "The **fDPIparse()** function” on page 61
- "The `snmp_dpi_set_packet` structure” on page 97
- "The **mkDPIresponse()** function” on page 69
The mkDPIAreYouThere() function

Format

```c
#include <snmp_dpi.h>

unsigned char *mkDPIAreYouThere(void);
```

Parameters

None

Return codes

If successful, a pointer to a static DPI packet buffer is returned. The first 2 bytes of the buffer in network byte order contain the length of the remaining packet. The macro DPI_PACKET_LEN can be used to calculate the total length of the DPI packet.

If not successful, a NULL pointer is returned.

Note: The static buffer for the DPI packet is shared by other mkDPIxxxx() functions that create a serialized DPI packet.

Usage

The mkDPIAreYouThere() function creates a serialized DPI ARE_YOU_HERE packet that can be sent to the DPI peer, which is normally the agent.

A subagent connected through TCP or UNIXstream probably does not need this function because, normally when the agent breaks the connection, you will receive an EOF on the file descriptor.

If your connection to the agent is still healthy, the agent will send a DPI RESPONSE with SNMP_ERROR_DPI_noError in the error code field and 0 in the error index field. The RESPONSE will have no varBind data. If your connection is not healthy, the agent might send a response with an error indication, or might not send a response at all.

Examples

```c
#include <snmp_dpi.h>

unsigned char *pack_p;

pack_p = mkDPIAreYouThere();
if (pack_p) {
    /* send the packet to the agent */
} /* endif */
/* wait for response with DPIawait_packet_from_agent() */
/* normally the response should come back pretty quickly, */
/* but it depends on the load of the agent */
```

Context

"The snmp_dpi_resp_packet structure" on page 96
"The DPIawait_packet_from_agent() function" on page 78
The mkDPIclose() function

Format

```
#include <snmp_dpi.h>

unsigned char *mkDPIclose(char reason_code);
```

Parameters

reason_code

The reason for closing the DPI connection. See “DPI CLOSE reason codes” on page 102 for a list of valid reason codes.

Return codes

If successful, a pointer to a static DPI packet buffer is returned. The first 2 bytes of the buffer in network byte order contain the length of the remaining packet. The macro DPI_PACKET_LEN can be used to calculate the total length of the DPI packet.

If not successful, a NULL pointer is returned.

Note: The static buffer for the DPI packet is shared by other mkDPIxxxx() functions that create a serialized DPI packet.

Usage

The mkDPIclose() function creates a serialized DPI CLOSE packet that can be sent to the DPI peer. As a result of sending the packet, the DPI connection will be closed.

Sending a DPI CLOSE packet to the agent implies an automatic DPI UNREGISTER for all registered subtrees on the connection being closed.

Examples

```
#include <snmp_dpi.h>
unsigned char *pack_p;

pack_p = mkDPIclose(SNMP_CLOSE_goingDown);
if (pack_p) {
    /* send the packet to the agent */
} /* endif */
```

Context

“The snmp_dpi_close_packet structure” on page 91
“DPI CLOSE reason codes” on page 102
The mkDPIopen() function

**Format**

```c
#include <snmp_dpi.h>

unsigned char *mkDPIopen( /* Make a DPI open packet */
  char *oid_p, /* subagent Identifier (OID) */
  char *description_p, /* subagent descriptive name */
  unsigned long timeout, /* requested default timeout */
  unsigned long max_varBinds, /* max varBinds per DPI packet*/
  char character_set, /* selected character set */
  #define DPI_NATIVE_CSET 0 /* 0 = native character set */
  #define DPI_ASCII_CSET 1 /* 1 = ASCII character set */
  unsigned long password_len, /* length of password (if any)*/
  unsigned char *password_p); /* ptr to password (if any) */
```

**Parameters**

- **oid_p** A pointer to a null-terminated character string representing the object identifier which uniquely identifies the subagent. The OID valued pointed to by oid_p must be in the EBCDIC character set when communicating with a TCP/IP UNIX System Services SNMP agent. The agent will add the OID passed in the mkDPIopen call to the sysORTable as sysORID in a corresponding new entry. By convention, sysORID should match a capabilities statement OID to refer to the MIBs supported by the subagent. For a list of MIB variables, see the z/OS Communications Server: IP System Administrator’s Commands.

- **description_p** A pointer to a null-terminated character string, which is a descriptive name for the subagent. This can be any DisplayString.

- **timeout** The requested timeout for this subagent. An agent often has a limit for this value and it will use that limit if this value is larger. A timeout of 0 has a special meaning in the sense that the agent will use its own default timeout value.

- **max_varBinds** The maximum number of varBinds per DPI packet that the subagent is prepared to handle. It must be a positive number or 0.
  - If a value greater than 1 is specified, the agent will try to combine as many varBinds that belong to the same subtree per DPI packet as possible up to this value.
  - If a value of 0 is specified, the agent will try to combine up to as many varBinds as are present in the SNMP packet and belong to the same subtree; there is no limit on the number of varBinds present in the DPI packet.

- **character_set** The character set that you want to use for string-based data fields in the DPI packets and structures. See “DPI OPEN character set selection” on page 101 for more information.

**DPI_NATIVE_CSET** Specifies that you want to use the native character set of the platform on which the agent that you connect to is running.
password_len
The length in octets of an optional password. It depends on the agent implementation if a password is needed.
If coded, this parameter is ignored with the z/OS SNMP agent.

password_p
A pointer to an octet string representing the password for this subagent. A password might include any character value, including the NULL character. If the password_len is 0, this can be a NULL pointer.
If coded, this parameter is ignored with the SNMP agent.

Return codes
If successful, a pointer to a static DPI packet buffer is returned. The first 2 bytes of the buffer in network byte order contain the length of the remaining packet. The macro DPI_PACKET_LEN can be used to calculate the total length of the DPI packet.
If not successful, a NULL pointer is returned.

Note: The static buffer for the DPI packet is shared by other mkDPIxxxx() functions that create a serialized DPI packet.

Usage
The mkDPIopen() function creates a serialized DPI OPEN packet that can then be sent to the DPI peer that is a DPI-capable SNMP agent.

Normally you will want to use the native character set, which is the easiest for the subagent programmer. However, if the agent and subagent each run on their own platforms and those platforms use different native character sets, you must select the ASCII character set, so that you both know exactly how to represent string-based data that is being sent back and forth.

Currently, if you specify a password parameter, it will be ignored. You do not need to specify a password to connect to the SNMP agent; you can pass a length of 0 and a NULL pointer for the password.

Examples
#include <snmp_dpi.h>
unsigned char *pack_p;

pack_p = mkDPIopen("1.3.6.1.2.3.4.5",
    "Sample DPI subagent"
    0L, 2L, DPI_NATIVE_CSET, /* max 2 varBinds */
    0L, (char *)0);
if (pack_p) {
    /* send packet to the agent */
} /* endif */

Context
[DPI OPEN character set selection” on page 101]
The mkDPIregister() function

Format

```c
#include <snmp_dpi.h>

unsigned char *mkDPIregister( /* Make a DPI register packet */
    unsigned short timeout, /* in seconds (16-bit) */
    long int priority, /* requested priority */
    char *group_p, /* ptr to group ID (subtree) */
    char bulk_select);/* Bulk selection (GETBULK) */
#define DPI_BULK_NO 0 /* map GETBULK into GETNEXTs */
```

Parameters

**timeout**

The requested timeout in seconds. An agent often has a limit for this value and it will use that limit if this value is larger. The value 0 has special meaning in the sense that it tells the agent to use the timeout value that was specified in the DPI OPEN packet.

**priority**

The requested priority. This field can contain any of these values:

- `-1` Requests the best available priority.
- `0` Requests a better priority than the highest priority currently registered. Use this value to obtain the SNMP DPI Version 1 behavior.
- `nnn` Any positive value. You will receive that priority if available; otherwise, you will receive the next best priority that is available.

**group_p**

A pointer to a null-terminated character string that represents the subtree to be registered. For object level registration, this group ID must have a trailing period. For instance level registration, this group ID would simply have the instance number follow the object number subtree.

**bulk_select**

Specifies if you want the agent to pass GETBULK on to the subagent or to map them into multiple GETNEXT requests. The choices are:

- `DPI_BULK_NO` Do not pass any GETBULK requests, but instead map a GETBULK request into multiple GETNEXT requests.

Return codes

If successful, a pointer to a static DPI packet buffer is returned. The first 2 bytes of the buffer in network byte order contain the length of the remaining packet. The macro DPI_PACKET_LEN can be used to calculate the total length of the DPI packet.

If not failure, a NULL pointer is returned.

**Note:** The static buffer for the DPI packet is shared by other mkDPIxxxx() functions that create a serialized DPI packet.

Usage

The mkDPIregister() function creates a serialized DPI REGISTER packet that can then be sent to the DPI peer that is a DPI-capable SNMP agent.
Normally, the SNMP agent sends a DPI RESPONSE packet back. This packet identifies if the register was successful or not.

The agent returns the assigned priority in the error index field of the response packet.

**Examples**

```c
#include <snmp_dpi.h>
unsigned char *pack_p;

pack_p = mkDPIregister(0,0L,"1.3.6.1.2.3.4.5." DPI_BULK_NO);
if (pack_p) {
    /* send packet to agent and await response */
} /* endif */
```

**Context**

"The snmp_dpi_resp_packet structure" on page 96
The mkDPIresponse() function

Format

```c
#include <snmp_dpi.h>

unsigned char *mkDPIresponse(/* Make a DPI response packet*/
    snmp_dpi_hdr *hdr_p, /* ptr to packet to respnd to*/
    long int error_code, /* error code: SNMP_ERROR_xxx*/
    long int error_index, /* index to varBind in error */
    snmp_dpi_set_packet *packet_p); /* ptr to varBinds, a chain */
    /* of dpi_set_packets */
```

Parameters

hdr_p A pointer to the parse tree of the DPI request to which this DPI packet will be the response. The function uses this parse tree to copy the packet_id and the DPI version and release, so that the DPI packet is correctly formatted as a response.

error_code The error code.

See "DPI RESPONSE error codes" on page 102 for a list of valid codes.

error_index Specifies the first varBind in error. Counting starts at 1 for the first varBind. This field should be 0 if there is no error.

packet_p A pointer to a chain of snmp_dpi_set_packet structures. This partial parse tree will be freed by the mkDPIresponse() function, so upon return you cannot refer to it anymore. Pass a NULL pointer if there are no varBinds to be returned.

Return codes

If successful, a pointer to a static DPI packet buffer is returned. The first 2 bytes of the buffer in network byte order contain the length of the remaining packet. The macro DPI_PACKET_LEN can be used to calculate the total length of the DPI packet.

If not successful, a NULL pointer is returned.

Note: The static buffer for the DPI packet is shared by other mkDPIxxxx() functions that create a serialized DPI packet.

Usage

The mkDPIresponse() function is used at the subagent side to prepare a DPI RESPONSE packet to a GET, GETNEXT, SET, COMMIT, or UNDO request. The resulting packet can be sent to the DPI peer, which is normally a DPI-capable SNMP agent.

Examples

```c
#include <snmp_dpi.h>
unsigned char pack_p;
unsigned char *pack_p;
snmp_dpi_hdr *hdr_p;
long int num;

hdr_p = pDPIpacket(pack_p); /* parse incoming packet */
    /* assume it's in pack_p */
if (hdr_p) {
```
/* analyze packet, assume GET, no error */
set_p = mkDPIset(snmp_dpi_set_packet_NULL_p,
"1.3.6.1.2.3.4.5.","1.0".
SNMP_TYPE_Integer32,
sizeof(num), &num);

if (set_p) {
    pack_p = mkDPIresponse(hdr_p,
    SNMP_ERROR_noError, 0L, set_p);
    if (pack_p) {
        /* send packet to agent */
    } /* endif */
} /* endif */
} /* endif */

Context
"The pDPIpacket() function" on page 76
"The snmp_dpi_hdr structure" on page 93
"The snmp_dpi_set_packet structure" on page 97
The mkDPIset() function

Format

```c
#include <snmp_dpi.h>

snmp_dpi_set_packet *mkDPIset( /* Make DPI set packet tree */
    snmp_dpi_set_packet *packet_p, /* ptr to SET structure */
    char *group_p, /* ptr to group ID(subtree)*/
    char *instance_p, /* ptr to instance OIDstring*/
    int value_type, /* value type: SNMP_TYPE_xxx*/
    int value_len, /* length of value */
    void *value_p); /* ptr to value */
```

Parameters

`packet_p`
A pointer to a chain of snmp_dpi_set_packet structures. Pass a NULL pointer if this is the first structure to be created.

`group_p`
A pointer to a null-terminated character string that represents the registered subtree that caused this GET request to be passed to this DPI subagent. The subtree must have a trailing period.

`instance_p`
A pointer to a null-terminated character string that represents the rest, which is the piece following the subtree part, of the object identifier of the variable instance being accessed. Use of the term `instance_p` here should not be confused with an OBJECT instance because this string can consist of a piece of the object identifier plus the INSTANCE IDENTIFIER.

`value_type`
The type of the value.

See "DPI SNMP value types" on page 103 for a list of currently defined value types.

`value_len`
This is the value that specifies the length in octets of the value pointed to by the `value` field. The length can be 0 if the value is of type SNMP_TYPE_NULL.

The maximum value is 64KB minus 1. However, the implementation often makes the length significantly less.

`value_p`
A pointer to the actual value. This field can contain a NULL pointer if the value is of implicit or explicit type SNMP_TYPE_NULL.

Return codes
If successful and a chain of one or more packets was passed in the `packet_p` parameter, the same pointer that was passed in `packet_p` is returned. A new dynamically allocated structure has been added to the end of that chain of snmp_dpi_get_packet structures.

If successful and a NULL pointer was passed in the `packet_p` parameter, a pointer to a new dynamically allocated structure is returned.

If not successful, a NULL pointer is returned.

Usage
The mkDPIset() function is used at the subagent side to prepare a chain of one or more snmp_dpi_set_packet structures. This chain is used to create a DPI
RESPONSE packet by a call to mkDPIresponse() that can be sent to the DPI peer, which is normally a DPI-capable SNMP agent.

The chain of snmp_dpi_set_packet structures can also be used to create a DPI TRAP packet that includes varBinds as explained in “The mkDPItrap() function” on page 73.

For the value_len, the maximum value is 64KB minus 1. However, the implementation often makes the length significantly less. For example, the SNMP PDU size might be limited to 484 bytes at the SNMP manager or agent side. In this case, the total response packet cannot exceed 484 bytes, so a value_len is limited to 484 bytes. You can send the DPI packet to the agent, but the manager will never see it.

Examples

```c
#include <snmp_dpi.h>
unsigned char *pack_p;
snmp_dpi_hdr *hdr_p;
snmp_dpi_set_packet *set_p;
long int num;

hdr_p = pDPIpacket(pack_p) /* parse incoming packet */
    /* assume it's in pack_p */
if (hdr_p) {
    /* analyze packet, assume GET, no error */
    set_p = mkDPIset(snmp_dpi_set_packet_NULL_p,
                     "1.3.6.1.2.3.4.5.", "1.0",
                     SNMP_TYPE_Integer32,
                     sizeof(num), &num);
    if (set_p) {
        pack_p = mkDPIresponse(hdr_p,
                               SNMP_ERROR_noError,
                               0L, set_p);
        if (pack_p)
            /* send packet to agent */
        } /* endif */
    } /* endif */
} /* endif */
```

If you must chain many snmp_dpi_set_packet structures, be sure to note that the packets are chained only by forward pointers. It is recommended that you use the last structure in the existing chain as the packet_p parameter. Then, the underlying code does not have to scan through a possibly long chain of structures to chain the new structure at the end.

Context

“The pDPIpacket() function” on page 76
“The mkDPIresponse() function” on page 69
“The mkDPItrap() function” on page 73
“The snmp_dpi_hdr structure” on page 93
“The snmp_dpi_set_packet structure” on page 97
“DPI SNMP value types” on page 103
“Value representation of DPI SNMP value types” on page 104
The mkDPItrap() function

Format

```c
#include <snmp_dpi.h>

unsigned char *mkDPItrap( /* Make a DPI trap packet */
    long int generic, /* generic traptype (32 bit)*/
    long int specific, /* specific traptype (32 bit)*/
    snmp_dpi_set_packet *packet_p, /* ptr to varBinds, a chain */
    char *enterprise_p); /* ptr to enterprise OID */
```

Parameters

generic
The generic trap type. The range of this value is 0-6, where 6, which is enterprise specific, is the type that is probably used most by DPI subagent programmers. The values in the range 0-5 are well defined standard SNMP traps.

specific
The enterprise specific trap type. This can be any value that is valid for the MIB subtrees that the subagent implements.

packet_p
A pointer to a chain of snmp_dpi_set_structures, representing the varBinds to be passed with the trap. This partial parse tree will be freed by the mkDPItrap() function so you cannot refer to it anymore upon completion of the call. A NULL pointer means that there are no varBinds to be included in the trap.

enterprise_p
A pointer to a null-terminated character string representing the enterprise ID (object identifier) for which this trap is defined. A NULL pointer can be used. In this case, the subagent identifier, as passed in the DPI OPEN packet, will be used when the agent receives the DPI TRAP packet.

Return codes
If successful, a pointer to a static DPI packet buffer is returned. The first 2 bytes of the buffer in network byte order contain the length of the remaining packet. The macro DPI_PACKET_LEN can be used to calculate the total length of the DPI packet.
If not successful, a NULL pointer is returned.

Note: The static buffer for the DPI packet is shared by other mkDPIxxxx() functions that create a serialized DPI packet.

Usage
The mkDPItrap() function is used at the subagent side to prepare a DPI TRAP packet. The resulting packet can be sent to the DPI peer, which is normally a DPI-capable SNMP agent.

Examples
```c
#include <snmp_dpi.h>
unsigned char *pack_p;
snmp_dpi_set_packet *set_p;
long int num;

set_p = mkDPIset(snmp_dpi_set_packet_NULL_p,
```
"1.3.6.1.2.3.4.5.", "1.0",
SNMP_TYPE_Integer32,
sizeof(num), &num);
if (set_p) {
    pack_p = mkDPItrap(6,1,set_p, (char *)0);
    if (pack_p) {
        /* send packet to agent */
    } /* endif */
} /* endif */

**Context**

"The mkDPIset() function" on page 71
The mkDPIunregister() function

Format

```
#include <snmp_dpi.h>

unsigned char *mkDPIunregister( /* Make DPI unregister packet */
    char reason_code; /* unregister reason code */
    char *group_p); /* ptr to group ID (subtree) */
```

Parameters

**reason_code**

The reason for the unregister.
See “DPI UNREGISTER reason codes” on page 103 for a list of the currently defined reason codes.

**group_p**

A pointer to a null-terminated character string that represents the subtree to be unregistered. For object level registration, this group ID must have a trailing period. For instance level registration, this group ID would simply have the instance number follow the object number subtree.

Return codes

If successful, a pointer to a static DPI packet buffer is returned. The first 2 bytes of the buffer in network byte order contain the length of the remaining packet. The macro DPI_PACKET_LEN can be used to calculate the total length of the DPI packet.
If not successful, a NULL pointer is returned.

Note: The static buffer for the DPI packet is shared by other mkDPIxxxx() functions that create a serialized DPI packet.

Usage

The mkDPIunregister() function creates a serialized DPI UNREGISTER packet that can be sent to the DPI peer, which is a DPI-capable SNMP agent.

Normally, the SNMP peer then sends a DPI RESPONSE packet back. This packet identifies if the unregister was successful or not.

Examples

```
#include <snmp_dpi.h>
unsigned char *pack_p;

pack_p = mkDPIunregister( SNMP_UNREGISTER_goingDown,
    "1.3.6.1.2.3.4.5.");
if (pack_p) {
    /* send packet to agent and await response */
} /* endif */
```

Context

“The snmp_dpi_ureg_packet structure” on page 99
The pDPIpacket() function

Format

#include <snmp_dpi.h>

snmp_dpi_hdr *pDPIpacket(unsigned char *packet_p);

Parameters

packet_p

A pointer to a serialized DPI packet.

Return codes

If successful, a pointer to a DPI parse tree (snmp_dpi_hdr) is returned. Memory for the parse tree has been dynamically allocated, and it is the callers responsibility to free it when no longer needed. You can use the fDPIparse() function to free the parse tree.

If not successful, a NULL pointer is returned.

Usage

The pDPIpacket() function parses the buffer pointed to by the packet_p parameter. It ensures that the buffer contains a valid DPI packet and that the packet is for a DPI version and release that is supported by the DPI functions in use.

Examples

#include <snmp_dpi.h>

unsigned char *pack_p;

snmp_dpi_hdr *hdr_p;

dpi_hdr_p = pDPIpacket(pack_p); /* parse incoming packet */
/* assume it's in pack_p */
if (hdr_p) {
  /* analyze packet, and handle it */
}

Context

"The snmp_dpi_hdr structure" on page 93
"The snmp_dpi.h include file" on page 106
"The fDPIparse() function" on page 61
Transport-related DPI API functions

This topic describes each of the DPI transport-related functions that are available to the DPI subagent programmer. These functions try to hide any platform specific issues for the DPI subagent programmer so that the subagent can be made as portable as possible. If you need detailed control for sending and awaiting DPI packets, you might have to do some of the transport-related code yourself.

The transport-related functions are basically the same for any platform, except for the initial call to set up a connection. SNMP currently supports the TCP transport type, as well as UNIXstream.

The Transport-Related DPI API Functions are:
- "The DPIawait_packet_from_agent() function" on page 78
- "The DPIconnect_to_agent_TCP() function" on page 80
- "The DPIconnect_to_agent_UNIXstream() function" on page 82
- "The DPIdisconnect_from_agent() function" on page 84
- "The DPIget_fd_for_handle() function" on page 85
- "The DPIsend_packet_to_agent() function" on page 86
- "The lookup_host() function" on page 88
- "The lookup_host6() function" on page 89
The DPIawait_packet_from_agent() function

Format

#include <snmp_dpi.h>

int DPIawait_packet_from_agent( /* await a DPI packet */
  int handle, /* on this connection */
  int timeout, /* timeout in seconds */
  unsigned char **message_p, /* receives ptr to data */
  unsigned long *length); /* receives length of data */

Parameters

handle
A handle as obtained with a DPIconnect_to_agent_xxxx() call.

timeout
A timeout value in seconds. There are two special values:
-1 Causes the function to wait forever until a packet arrives.
0 Means that the function will only check if a packet is waiting. If not, an immediate return is made. If there is a packet, it will be returned.

message_p
The address of a pointer that will receive the address of a static DPI packet buffer or, if there is no packet, a NULL pointer.

length The address of an unsigned long integer that will receive the length of the received DPI packet or, if there is no packet, a 0 value.

Return codes
If successful, a 0 (DPI_RC_OK) is returned. The buffer pointer and length of the caller will be set to point to the received DPI packet and to the length of that packet.
If not successful, a negative integer is returned, which indicates the kind of error that occurred. See "Return codes from DPI transport-related functions" on page 105 for a list of possible error codes.

DPI_RC_NOK
This is a return code indicating the DPI code is out of sync or has a bug.

DPI_RC_EOF
End of file on the connection. The connection has been closed.

DPI_RC_IO_ERROR
An error occurred with an underlying select() or recvfrom() call, or a DPI packet was read that was less than 2 bytes. DPI uses the first 2 bytes to get the packet length.

DPI_RC_INVALID_HANDLE
A bad handle was passed as input. Either the handle is not valid, or it describes a connection that has been disconnected.

DPI_RC_TIMEOUT
No packet was received during the specified timeout period.

DPI_RC_PACKET_TOO_LARGE
The packet received was too large.
Usage
The DPIawait_packet_from_agent() function is used at the subagent side to await a DPI packet from the DPI-capable SNMP agent. The programmer can specify how long to wait.

Examples
#include <snmp_dpi.h>
int handle;
unsigned char *pack_p;
unsigned long length;

handle = DPIconnect_to_agent_TCP("127.0.0.1", "public");
if (handle < 0) {
    printf("Error %d from connect\n",handle);
    exit(1);
} /* endif */
/* do useful stuff */
rc = DPIawait_packet_from_agent(handle, -1,
    &pack_p, &length);
if (rc) {
    printf("Error %d from await packet\n");
    exit(1);
} /* endif */
/* handle the packet */

Context
"The DPIconnect_to_agent_TCP() function" on page 80
"The DPIconnect_to_agent_UNIXstream() function" on page 82
The DPIconnect_to_agent_TCP() function

Format

```
#include <snmp_dpi.h>

int DPIconnect_to_agent_TCP(   /* Connect to DPI TCP port */
    char *hostname_p, /* target hostname/IP address */
    char *community_p); /* community name */
```

Parameters

hostname_p
A pointer to a null-terminated character string representing the host name or IP address in IPv4 dotted-decimal or IPv6 colon-hexadecimal notation of the host where the DPI-capable SNMP agent is running.

community_p
A pointer to a null-terminated character string representing the community name that is required to obtain the dpiPort from the SNMP agent through an SNMP GET request.

Note: For z/OS Communications Server, the SNMP community passed by the subagent must be in ASCII only.

Return codes
If successful, a nonnegative integer that represents the connection is returned. It is to be used as a handle in subsequent calls to DPI transport-related functions.

If not successful, a negative integer is returned, which indicates the kind of error that occurred. See "Return codes from DPI transport-related functions" on page 105 for a list of possible error codes.

DPI_RC_NO_PORT
Unable to obtain the dpiPort number. There are many reasons for this, for example: bad host name, bad community name, or default timeout (9 seconds) before a response from the agent.

DPI_RC_IO_ERROR
An error occurred with an underlying select(), or DPI was not able to set up a socket (could be due to an error on a socket(), bind(), connect() call, or other internal errors).

Usage
The DPIconnect_to_agent_TCP() function is used at the subagent side to set up a TCP connection to the DPI-capable SNMP agent.

As part of the connection processing, the DPIconnect_to_agent_TCP() function sends an SNMP GET request to the SNMP agent to retrieve the port number of the DPI port to be used for the TCP connection. By default, this SNMP GET request is sent to the well-known SNMP port 161. If the SNMP agent is listening on a port other than well-known port 161, the SNMP_PORT environment variable can be set to the port number of the SNMP agent prior to issuing the DPIconnect_to_agent_TCP(). Use setenv() to override port 161 before using this function.
Examples
#include <snmp_dpi.h>
int handle;

handle = DPIconnect_to_agent_TCP("127.0.0.1", "public");
if (handle < 0) {
    printf("Error %d from connect\n",handle);
    exit(1);
} /* endif */

Context
"Return codes from DPI transport-related functions" on page 105
"The DPIconnect_to_agent_UNIXstream() function" on page 82
The DPIconnect_to_agent_UNIXstream() function

Format

```
#include <snmp_dpi.h>

int DPIconnect_to_agent_UNIXstream(    /* Connect to DPI UNIXstream */
    char   *hostname_p,    /* target hostname/IP address */
    char   *community_p);  /* community name */
```

Parameters

**hostname_p**
A pointer to a null-terminated character string representing the local host
name or IP address in IPv4 dotted-decimal or IPv6 colon-hexadecimal
notation of the local host where the DPI-capable SNMP agent is running.

**community_p**
A pointer to a null-terminated character string representing the community
name that is required to obtain the UNIX® path name from the SNMP
agent through an SNMP GET request.

**Note:** For z/OS Communications Server, the SNMP community passed by
the subagent must be in ASCII only.

Return codes

If successful, a nonnegative integer that represents the connection is returned. It
is to be used as a handle in subsequent calls to DPI transport-related functions.
If not successful, a negative integer is returned, which indicates the kind of
error that occurred. See “Return codes from DPI transport-related functions” on
page 105 for a list of possible error codes.

**DPI_RC_NO_PORT**
Unable to obtain the UNIX path name. There are many reasons for this, for
example: bad host name, bad community name, or default timeout (9
seconds) before a response from the agent.

**DPI_RC_IO_ERROR**
An error occurred with an underlying select(), or DPI was not able to set
up a socket (could be due to an error on a socket(), bind(), connect() call,
or other internal errors).

Usage

The DPIconnect_to_agent_UNIXstream() function is used at the subagent side to
set up an AF_UNIX connection to the DPI-capable SNMP agent.

As part of the connection processing, the DPIconnect_to_agent_UNIXstream() function
sends an SNMP GET request to the SNMP agent to retrieve the path name
for the UNIX streams connection. By default, this SNMP GET request is sent to the
well-known SNMP port 161. If the SNMP agent is listening on a port other than
well-known port 161, the SNMP_PORT environment variable can be set to the port
number of the SNMP agent prior to issuing the
DPIconnect_to_agent_UNIXstream(). Use setenv() to override port 161 before using
this function.

The DPIconnect_to_agent_UNIXstream() function uses a path name in the z/OS
UNIX file system as the name of the socket for the connect. This path name is
available at the SNMP agent through the MIB object 1.3.6.1.4.1.2.2.1.1.3, which has
the name dpiPathNameForUnixStream. The SNMP agent uses the default name
/var/dpi_socket if you do not supply another name in the agent startup parameter (-s) or in the OSNMPD.DATA file. Whichever name is used, the SNMP agent creates the path name as a character special file during initialization.

You must either define the subagents with superuser authority or set the read and write file access permission bits for the path name for the class associated with the user ID of the subagent before subagents can successfully connect to the agent using the path name. You can use the agent -C startup parameter to specify which permission bits should be set.

To run a user-written subagent from a non-privileged user ID, set the permission bits for the character special file to write access. Otherwise, a subagent using this function must be run from a superuser or other user with appropriate privileges.

**Examples**

```c
#include <snmp_dpi.h>
int handle;

handle = DPIconnect_to_agent_UNIXstream("127.0.0.1", "public");
if (handle < 0) {
   printf("Error %d from connect\n", handle);
   exit(1);
} /* endif */
```

**Context**

- "Return codes from DPI transport-related functions" on page 105
- "The DPIconnect_to_agent_TCP() function" on page 80
The DPIdisconnect_from_agent() function

Format

```
#include <snmp_dpi.h>

void DPIdisconnect_from_agent( /* disconnect from DPI (agent)*/
   int handle); /* close this connection */
```

Parameters

handle
A handle as obtained with a DPIconnect_to_agent_xxxx() call.

Usage

The DPIdisconnect_from_agent() function is used at the subagent side to terminate
a connection to the DPI-capable SNMP agent.

Examples

```
#include <snmp_dpi.h>
int handle;

handle = DPIconnect_to_agent_TCP("127.0.0.1", "public");
if (handle < 0) {
   printf("Error %d from connect\n",handle);
   exit(1);
} /* endif */
/* do useful stuff */
DPIdisconnect_from_agent(handle);
```

Context

"The DPIconnect_to_agent_TCP() function" on page 80
"The DPIconnect_to_agent_UNIXstream() function" on page 82
The DPIget_fd_for_handle() function

Format

#include <snmp_dpi.h>

int DPIget_fd_for_handle( /* get the file descriptor */ int handle); /* for this handle */

Parameters

handle
A handle that was obtained with a DPIconnect_to_agent_xxxx() call.

Return codes
If successful, a positive integer representing the file descriptor associated with the specified handle.

If not successful, a negative integer is returned, which indicates the error that occurred. See “Return codes from DPI transport-related functions” on page 105 for a list of possible error codes.

DPI_RC_INVALID_HANDLE
A bad handle was passed as input. Either the handle is not valid, or it describes a connection that has been disconnected.

Usage
The DPIget_fd_for_handle function is used to obtain the file descriptor for the handle, which was obtained with a DPIconnect_to_agent_TCP() call or a DPIconnect_to_agent_UNIXstream() call.

Using this function to retrieve the file descriptor associated with your DPI connections enables you to use either the select or selectex socket calls. Using selectex enables your program to wait for event control blocks (ECBs), in addition to a read condition. This is one example of how an MVS application can wait for notification of the receipt of a modify command (through an ECB post) or DPI packet at the same time.

Examples

#include <snmp_dpi.h>
#include /* other include files for BSD sockets and such */
int handle;
int fd;

handle = DPIconnect_to_agent_TCP("127.0.0.1","public");
if (handle < 0) {
   printf("Error %d from connect\n",handle);
   exit(1);
}
fd = DPIget_fd_for_handle(handle);
if (fd <0) {
   printf("Error %d from get_fd\n",fd);
   exit(1);
}

Context

“The DPIconnect_to_agent_TCP() function” on page 80
“The DPIconnect_to_agent_UNIXstream() function” on page 82
The DPIsend_packet_to_agent() function

Format

#include <snmp_dpi.h>

int DPIsend_packet_to_agent( /* send a DPI packet */
    int handle, /* on this connection */
    unsigned char *message_p, /* ptr to the packet data */
    unsigned long length); /* length of the packet */

Parameters

handle
A handle as obtained with a DPIconnect_to_agent_xxxx() call.

message_p
A pointer to the buffer containing the DPI packet to be sent.

length
The length of the DPI packet to be sent. The DPI_PACKET_LEN macro is a useful macro to calculate the length.

Return codes
If successful, a 0 (DPI_RC_OK) is returned.

If not successful, a negative integer is returned, which indicates the kind of error that occurred. See "Return codes from DPI transport-related functions" on page 105 for a list of possible error codes.

DPI_RC_NOK
This is a return code, but it really means the DPI code is out of sync or has a bug.

DPI_RC_IO_ERROR
An error occurred with an underlying send(), or the send() failed to send all of the data on the socket (incomplete send).

DPI_RC_INVALID_ARGUMENT
The message_p parameter is NULL or the length parameter has a value of 0.

DPI_RC_INVALID_HANDLE
A bad handle was passed as input. Either the handle is not valid, or it describes a connection that has been disconnected.

Usage
The DPIsend_packet_to_agent() function is used at the subagent side to send a DPI packet to the DPI-capable SNMP agent.

Examples

#include <snmp_dpi.h>
int handle;
unsigned char *pack_p;

handle = DPIconnect_to_agent_TCP("127.0.0.1", "public");
if (handle < 0) {
    printf("Error %d from connect\n",handle);
    exit(1);
} /* endif */
pack_p = mkDPIopen("1.3.6.1.2.3.4.5",
    "Sample DPI subagent",
    0L,2L,DPI_NATIVE_CSET,
0,(char *)0);
if (pack_p) {
    rc = DPIsend_packet_to_agent(handle, pack_p,
        DPI_PACKET_LEN(pack_p));
    if (rc) {
        printf("Error %d from send packet\n");
        exit(1);
    } /* endif */
} else {
    printf("Can't make DPI OPEN packet\n");
    exit(1);
} /* endif */
/* await the response */

Context
"The DPIconnect_to_agent_TCP() function" on page 80
"The DPIconnect_to_agent_UNIXstream() function" on page 82
"The DPI_PACKET_LEN() macro" on page 60
The lookup_host() function

Format

```c
#include <snmp_dpi.h>

unsigned long lookup_host( /* find IP address in network */
    char *hostname_p); /* byte order for this host */
```

Parameters

hostname_p
A pointer to a null-terminated character string representing the host name or IP address in dotted-decimal notation of the host where the DPI-capable SNMP agent is running.

Return codes
If successful, the IP address is returned in network byte order, so it is ready to be used in a sockaddr_in structure.

If not successful, a value of 0 is returned.

Usage
The lookup_host() function is used to obtain the IP address in network byte order of a host or IP address in dotted decimal notation. This function is implicitly executed by both DPIconnect_to_agent_TCP and DPIconnect_to_agent_UNIXstream.

Context
“The DPIconnect_to_agent_TCP() function” on page 80
The lookup_host6() function

Format

```c
#include <snmp_dpi.h>

struct sockaddr_in6 *lookup_host6( /* find IPv6 address in network */
    char *hostname_p); /* byte order for this host */
```

Parameters

`hostname_p`
A pointer to a null-terminated character string representing the host name or IPv6 address in colon-hexadecimal notation of the host where the DPI-capable SNMP agent is running.

Return codes
If successful, a pointer to a sockaddr_in6 structure is returned. The structure is filled in with the IPv6 address of the specified host in network byte order.

If not successful, a NULL pointer is returned.

Usage
The lookup_host6() function is used to obtain an IPv6 address in network byte order of a host specified by host name or IPv6 address in colon-hexadecimal notation. This function can be implicitly executed by DPIconnect_to_agent_TCP and DPIconnect_to_agent_UNIXstream.

If the function is successful, the caller does not own the sockaddr_in6 structure pointed to by the return value. If the caller needs to store the IPv6 address or the entire structure, it should do so immediately after lookup_host6() returns, because subsequent calls to lookup_host6() will cause the contents of the sockaddr_in6 to be overwritten.

Context

“The DPIconnect_to_agent_TCP() function” on page 80
DPI structures

This topic describes each data structure that is used in the SNMP DPI API:

- “The snmp_dpi_close_packet structure” on page 91
- “The snmp_dpi_get_packet structure” on page 92
- “The snmp_dpi_hdr structure” on page 93
- “The snmp_dpi_next_packet structure” on page 95
- “The snmp_dpi_resp_packet structure” on page 96
- “The snmp_dpi_set_packet structure” on page 97
- “The snmp_dpi_ureg_packet structure” on page 99
- “The snmp_dpi_u64 structure” on page 100
The `snmp_dpi_close_packet` structure

**Format**

```c
struct dpi_close_packet {
    char reason_code; /* reason for closing */
};
typedef struct dpi_close_packet snmp_dpi_close_packet;
#define snmp_dpi_close_packet_NULL_p ((snmp_dpi_close_packet*)0)
```

**Parameters**

**reason_code**

The reason for the close.

See "DPI CLOSE reason codes” on page 102 for a list of valid reason codes.

**Usage**

The `snmp_dpi_close_packet` structure represents a parse tree for a DPI CLOSE packet.

The `snmp_dpi_close_packet` structure might be created as a result of a call to `pDPIpacket()`. This is the case if the DPI packet is of type SNMP_DPI_CLOSE. The `snmp_dpi_hdr` structure then contains a pointer to an `snmp_dpi_close_packet` structure.

An `snmp_dpi_close_packet` structure is also created as a result of an `mkDPIclose()` call, but the programmer never sees the structure because `mkDPIclose()` immediately creates a serialized DPI packet from it and then frees the structure.

It is recommended that DPI subagent programmer uses `mkDPIclose()` to create a DPI CLOSE packet.

**Context**

- “The `pDPIpacket()` function” on page 76
- “The `mkDPIclose()` function” on page 64
- “The `snmp_dpi_hdr` structure” on page 93
The `snmp_dpi_get_packet` structure

Format

```c
struct dpi_get_packet {
    char    *object_p; /* ptr to OID string */
    char    *group_p; /* ptr to subtree(group) */
    char    *instance_p; /* ptr to rest of OID */
    struct dpi_get_packet *next_p; /* ptr to next in chain */
};
typedef struct dpi_get_packet snmp_dpi_get_packet;
#define snmp_dpi_get_packet_NULL_p ((snmp_dpi_get_packet *)0)
```

Parameters

**object_p**

A pointer to a null-terminated character string that represents the full object identifier of the variable instance that is being accessed. It basically is a concatenation of the fields `group_p` and `instance_p`. Using this field is not recommended because it is only included for DPI Version 1 compatibility and it might be withdrawn in a later version.

**group_p**

A pointer to a null-terminated character string that represents the registered subtree that caused this SET request to be passed to this DPI subagent. The subtree must have a trailing period.

**instance_p**

A pointer to a null-terminated character string that represents the rest, which is the piece following the subtree part, of the object identifier of the variable instance being accessed.

Use of the term `instance_p` here should not be confused with an OBJECT instance because this string might consist of a piece of the object identifier plus the INSTANCE IDENTIFIER.

**next_p**

A pointer to a possible next `snmp_dpi_get_packet` structure. If this next field contains the NULL pointer, this is the end of the chain.

Usage

The `snmp_dpi_get_packet` structure represents a parse tree for a DPI GET packet.

At the subagent side, the `snmp_dpi_get_packet` structure is normally created as a result of a call to `pDPIpacket()`. This is the case if the DPI packet is of type `SNMP_DPI_GET`. The `snmp_dpi_hdr` structure then contains a pointer to a chain of one or more `snmp_dpi_get_packet` structures.

The DPI subagent programmer uses this structure to find out which variable instances are to be returned in a DPI RESPONSE.

Context

- “The `pDPIpacket()` function” on page 76
- “The `snmp_dpi_hdr` structure” on page 93
The `snmp_dpi_hdr` structure

**Format**

```c
struct snmp_dpi_hdr {
    unsigned char proto_major; /* always 2: SNMP_DPI_PROTOCOL*/
    unsigned char proto_version; /* DPI version */
    unsigned char proto_release; /* DPI release */
    unsigned short packet_id; /* 16-bit, DPI packet ID */
    unsigned char packet_type; /* DPI packet type */
    union {
        snmp_dpi_reg_packet  *reg_p;
        snmp_dpi_ureg_packet *ureg_p;
        snmp_dpi_get_packet  *get_p;
        snmp_dpi_next_packet *next_p;
        snmp_dpi_next_packet *bulk_p;
        snmp_dpi_set_packet  *set_p;
        snmp_dpi_resp_packet *resp_p;
        snmp_dpi_trap_packet *trap_p;
        snmp_dpi_open_packet *open_p;
        snmp_dpi_close_packet *close_p;
        unsigned char       *any_p;
    } data_u;
};
typedef struct snmp_dpi_hdr snmp_dpi_hdr;
#define snmp_dpi_hdr_NULL_p ((snmp_dpi_hdr *)0)
```

**Parameters**

- **proto_major**
  - The major protocol. For SNMP DPI, it is always 2.

- **proto_version**
  - The DPI version.

- **proto_release**
  - The DPI release.

- **packet_id**
  - This field contains the packet ID of the DPI packet. When you create a response to a request, the packet ID must be the same as that of the request. This is taken care of if you use the `mkDPIresponse()` function.

- **packet_type**
  - The type of DPI packet (parse tree) that you are dealing with.

  See “DPI packet types” on page 102 for a list of currently defined DPI packet types.

- **data_u**
  - A union of pointers to the different types of data structures that are created based on the `packet_type` field. The pointers themselves have names that are self-explanatory.

The fields `proto_major`, `proto_version`, `proto_release`, and `packet_id` are basically for DPI internal use, so the DPI programmer normally does not need to be concerned about them.

**Usage**

The `snmp_dpi_hdr` structure is the anchor of a DPI parse tree. At the subagent side, the `snmp_dpi_hdr` structure is normally created as a result of a call to `pDPIpacket()`.
The DPI subagent programmer uses this structure to interrogate packets. Depending on the `packet_type`, the pointer to the chain of one or more `packet_type` specific structures that contain the actual packet data can be picked.

The storage for a DPI parse tree is always dynamically allocated. It is the responsibility of the caller to free this parse tree when it is no longer needed. You can use the `fDPIparse()` function to do that.

**Note:** Some `mkDPIxxxx` functions do free the parse tree that is passed to them. An example is the `mkDPIresponse()` function.

**Context**
- “The `fDPIparse()` function” on page 61
- “The `pDPIpacket()` function” on page 76
- “The `snmp_dpi_close_packet structure`” on page 91
- “The `snmp_dpi_get_packet structure`” on page 92
- “The `snmp_dpi_next_packet structure`” on page 95
- “The `snmp_dpi_resp_packet structure`” on page 96
- “The `snmp_dpi_set_packet structure`” on page 97
- “The `snmp_dpi_ureg_packet structure`” on page 99
The `snmp_dpi_next_packet` structure

**Format**

```c
struct dpi_next_packet {
    char *object_p; /* ptr to OID (string) */
    char *group_p; /* ptr to subtree(group) */
    char *instance_p; /* ptr to rest of OID */
    struct dpi_next_packet *next_p; /* ptr to next in chain */
};
typedef struct dpi_next_packet snmp_dpi_next_packet;
#define snmp_dpi_next_packet_NULL_p ((snmp_dpi_next_packet *)0)
```

**Parameters**

- **object_p**
  A pointer to a null-terminated character string that represents the full object identifier of the variable instance that is being accessed. It basically is a concatenation of the fields `group_p` and `instance_p`. Using this field is not recommended because it is only included for DPI Version 1 compatibility and it might be withdrawn in a later version.

- **group_p**
  A pointer to a null-terminated character string that represents the registered subtree that caused this GETNEXT request to be passed to this DPI subagent. This subtree must have a trailing period.

- **instance_p**
  A pointer to a null-terminated character string that represents the rest, which is the piece following the subtree part, of the object identifier of the variable instance being accessed.

  Use of the term `instance_p` here should not be confused with an OBJECT instance because this string might consist of a piece of the object identifier plus the INSTANCE IDENTIFIER.

- **next_p**
  A pointer to a possible next `snmp_dpi_next_packet` structure. If this next field contains the NULL pointer, this is the end of the chain.

**Usage**

The `snmp_dpi_next_packet` structure represents a parse tree for a DPI GETNEXT packet.

At the subagent side, the `snmp_dpi_next_packet` structure is normally created as a result of a call to `pDPIpacket()`. This is the case if the DPI packet is of type `SNMP_DPI_GETNEXT`. The `snmp_dpi_hdr` structure then contains a pointer to a chain of one or more `snmp_dpi_next_packet` structures.

The DPI subagent programmer uses this structure to find out which variables instances are to be returned in a DPI RESPONSE.

**Context**

- "The pDPIpacket() function" on page 76
- "The snmp_dpi_hdr structure" on page 93
The `snmp_dpi_resp_packet` structure

**Format**

```c
struct dpi_resp_packet {
    char error_code; /* like: SNMP_ERROR_xxx */
    unsigned long int error_index; /* 1st varBind in error */
    typedef dpi_set_packet *varBind_p; /* ptr to varBind, chain */
    /* of dpi_set_packets */
};
typedef struct dpi_resp_packet snmp_dpi_resp_packet;
#define snmp_dpi_resp_packet_NULL_p ((snmp_dpi_resp_packet *)0)
```

**Parameters**

**error_code**

The return code or the error code.

See “DPI RESPONSE error codes” on page 102 for a list of valid codes.

**error_index**

Specifies the first varBind in error. Counting starts at 1 for the first varBind. This field should be 0 if there is no error.

**resp_priority**

This is a redefinition of the error_index field. If the response is a response to a DPI REGISTER request and the error_code is equal to SNMP_ERROR_DPI_noError or SNMP_ERROR_DPI_higherPriorityRegistered, then this field contains the priority that was actually assigned. Otherwise, this field is set to 0 for responses to a DPI REGISTER.

**varBind_p**

A pointer to the chain of one or more snmp_dpi_set_structures, representing varBinds of the response. This field contains a NULL pointer if there are no varBinds in the response.

**Usage**

The `snmp_dpi_resp_packet` structure represents a parse tree for a DPI RESPONSE packet.

The `snmp_dpi_resp_packet` structure is normally created as a result of a call to `pDPIpacket()`. This is the case if the DPI packet is of type SNMP_DPI_RESPONSE. The `snmp_dpi_hdr` structure then contains a pointer to an `snmp_dpi_resp_packet` structure.

At the DPI subagent side, a DPI RESPONSE should only be expected at initialization and termination time when the subagent has issued a DPI OPEN, DPI REGISTER, or DPI UNREGISTER request.

The DPI programmer is advised to use the `mkDPIresponse()` function to prepare a DPI RESPONSE packet.

**Context**

- “The `pDPIpacket()` function” on page 76
- “The `mkDPIresponse()` function” on page 69
- “The `snmp_dpi_set_packet` structure” on page 97
- “The `snmp_dpi_hdr` structure” on page 93

z/OS V1R11.0 Comm Svr: IP Programmer’s Guide and Reference
The `snmp_dpi_set_packet` structure

**Format**

```c
struct dpi_set_packet {
    char *object_p; /* ptr to Object ID (string) */
    char *group_p; /* ptr to subtree (group) */
    char *instance_p; /* ptr to rest of OID */
    unsigned char value_type; /* value type: SNMP_TYPE_xxx */
    unsigned short value_len; /* value length */
    char *value_p; /* ptr to the value itself */
    struct dpi_set_packet *next_p; /* ptr to next in chain */
};
```

```c
typedef struct dpi_set_packet snmp_dpi_set_packet;
#define snmp_dpi_set_packet_NULL_p ((snmp_dpi_set_packet *)0)
```

**Parameters**

- **object_p**
  
  A pointer to a null-terminated character string that represents the full
  object identifier of the variable instance that is being accessed. It basically
  is a concatenation of the fields `group_p` and `instance_p`. Using this field is
  not recommended because it is only included for DPI Version 1
  compatibility and it might be withdrawn in a later version.

- **group_p**
  
  A pointer to a null-terminated character string that represents the
  registered subtree that caused this SET, COMMIT, or UNDO request to be
  passed to this DPI subagent. The subtree must have a trailing period.

- **instance_p**
  
  A pointer to a null-terminated character string that represents the rest,
  which is the piece following the subtree part, of the object identifier of
  the variable instance being accessed.

  Use of the term `instance_p` here should not be confused with an OBJECT
  instance because this string might consist of a piece of the object identifier
  plus the INSTANCE IDENTIFIER.

- **value_type**
  
  The type of the value.

  See ["DPI SNMP value types" on page 103](#) for a list of currently defined
  value types.

- **value_len**
  
  This is an unsigned 16-bit integer that specifies the length in octets of the
  value pointed to by the `value` field. The length can be 0 if the value is of
  type SNMP_TYPE_NULL.

- **value_p**
  
  A pointer to the actual value. This field can contain a NULL pointer if the
  value is of type SNMP_TYPE_NULL.

  See ["Value representation of DPI SNMP value types" on page 104](#) for
  information on how the data is represented for the various value types.

- **next_p**
  
  A pointer to a possible next `snmp_dpi_set_packet` structure. If this next
  field contains the NULL pointer, this is the end of the chain.

**Usage**

The `snmp_dpi_set_packet` structure represents a parse tree for a DPI SET request.
The `snmp_dpi_set_packet` structure might be created as a result of a call to `pDPIpacket()`. This is the case if the DPI packet is of type SNMP_DPI_SET, SNMP_DPI_COMMIT, or SNMP_DPI_UNDO. The `snmp_dpi_hdr` structure then contains a pointer to a chain of one or more `snmp_dpi_set_packet` structures.

This structure can also be created with an `mkDPIset()` call, which is typically used when preparing varBinds for a DPI RESPONSE packet.

**Context**
- "The `pDPIpacket()` function" on page 76
- "The `mkDPIset()` function" on page 71
- "DPI SNMP value types" on page 103
- "Value representation of DPI SNMP value types" on page 104
- "The `snmp_dpi_hdr` structure" on page 93
The `snmp_dpi_ureg_packet` structure

**Format**

```c
struct dpi_ureg_packet {
    char reason_code;/* reason for unregister */
    char *group_p; /* ptr to subtree(group)*/
    struct dpi_ureg_packet *next_p; /* ptr to next in chain */
};
typedef struct dpi_ureg_packet snmp_dpi_ureg_packet;
#define snmp_dpi_ureg_packet_NULL_p ((snmp_dpi_ureg_packet *)0)
```

**Parameters**

*reason_code*

The reason for the unregister. See "DPI UNREGISTER reason codes" on page 103 for reason codes.

*group_p*

A pointer to a null-terminated character string that represents the subtree to be unregistered. This subtree must have a trailing period.

*next_p*

A pointer to a possible next `snmp_dpi_ureg_packet` structure. If this next field contains the NULL pointer, this is the end of the chain. Currently, multiple unregister requests are not supported in one DPI packet, so this field should always be 0.

**Usage**

The `snmp_dpi_ureg_packet` structure represents a parse tree for a DPI UNREGISTER request.

The `snmp_dpi_ureg_packet` structure is normally created as a result of a call to `pDPIpacket()`. This is the case if the DPI packet is of type `SNMP_DPI_UNREGISTER`. The `snmp_dpi_hdr` structure then contains a pointer to an `snmp_dpi_ureg_packet` structure.

The DPI programmer is advised to use the `mkDPIunregister()` function to create a DPI UNREGISTER packet.

**Context**

- "The `pDPIpacket()` function" on page 76
- "The `mkDPIunregister()` function" on page 75
- "The `snmp_dpi_hdr` structure" on page 93
The `snmp_dpi_u64` structure

**Format**

```c
struct snmp_dpi_u64 {
    unsigned long high; /* - high order 32 bits */
    unsigned long low;  /* - low order 32 bits */
};
typedef struct snmp_dpi_u64 snmp_dpi_u64;
```

**Note:** This structure is supported only in SNMP Version 2.

**Parameters**

- **high**  The high order, most significant, 32 bits.
- **low**   The low order, least significant, 32 bits.

**Usage**

The `snmp_dpi_u64` structure represents an unsigned 64-bit integer as needed for values with a type of SNMP_TYPE_Counter64.

The `snmp_dpi_u64` structure might be created as a result of a call to `pDPIpacket()`. This is the case if the DPI packet is of type SNMP_DPI_SET and one of the values has a type of SNMP_TYPE.Counter64. The value_p pointer of the `snmp_dpi_set_packet` structure will then point to an `snmp_dpi_u64` structure.

The DPI programmer must also use an `snmp_dpi_u64` structure as the parameter to an `mkDPIset()` call if you want to create a value of type SNMP_TYPE.Counter64.

**Context**

- "The `pDPIpacket()` function" on page 76
- "The `snmp_dpi_set_packet` structure" on page 97
- "DPI SNMP value types" on page 103
- "Value representation of DPI SNMP value types" on page 104
DPI OPEN character set selection

The version of DPI Version 2.0 included with SNMP requires use of the EBCDIC character set. Any DisplayString MIB objects known to the agent (in its compiled MIB) supplied with SNMP will have ASCII conversion handled by the agent. The subagent will always deal with the values of these objects in EBCDIC. Any portion of an instance identifier that is a DisplayString must be in ASCII. The agent does not handle instance IDs.

When the DPI subagent sends a DPI OPEN packet, it must specify the character set that it wants to use. The subagent here needs to know or determine in an implementation dependent manner if the agent is running on a system with the same character set as the subagent. If you connect to the agent at loopback or your own machine, you might assume that you are using the same character set.

The DPI subagent has two choices:

DPI_NATIVE_CSET

Specifies that you want to use the native character set of the platform on which the agent that you connect to is running.

DPI_ASCII_CSET

Specifies that you want to use the ASCII character set. The agent will not translate between ASCII and the native character set.

Although you can specify ASCII, the SNMP agent does not support it.

The DPI packets have a number of fields that are represented as strings. The fields that must be represented in the selected character set are:

- The null-terminated string pointed to by the description_p, enterprise_p, group_p, instance_p, and oid_p parameters in the various mkDPIxxx(...) functions.
- The string pointed to by the value_p parameter in the mkDPIset(...) function, that is if the value_type parameter specifies that the value is an SNMP_TYPE_DisplayString or an SNMP_TYPE_OBJECT_IDENTIFIER.
- The null-terminated string pointed to by the description_p, enterprise_p, group_p, instance_p, and oid_p pointers in the various snmp_dpi_xxxx_packet structures.
- The string pointed to by the value_p pointer in the snmp_dpi_set_packet structure, that is if the value_type field specifies that the value is an SNMP_TYPE_DisplayString or an SNMP_TYPE_OBJECT_IDENTIFIER.

See the following related information.

"The mkDPIopen() function" on page 65

SNMP DPI constants, values, return codes, and include file

This topic describes all the constants and names for values as they are defined in the snmp_dpi.h include file (see "The snmp_dpi.h include file" on page 106):

"DPI close reason codes" on page 102
"DPI packet types" on page 102
"DPI RESPONSE error codes" on page 102
"DPI UNREGISTER reason codes" on page 103
"DPI SNMP value types" on page 103
"Value representation of DPI SNMP value types" on page 104
"Value ranges and limits for DPI SNMP value types" on page 105
"Return codes from DPI transport-related functions" on page 105
**DPI CLOSE reason codes**

The currently defined DPI CLOSE reason codes as defined in the snmp_dpi.h include file are:

```c
#define SNMP_CLOSE_otherReason 1
#define SNMP_CLOSE_goingDown 2
#define SNMP_CLOSE_unsupportedVersion 3
#define SNMP_CLOSE_protocolError 4
#define SNMP_CLOSE_authenticationFailure 5
#define SNMP_CLOSE_byManager 6
#define SNMP_CLOSE_timeout 7
#define SNMP_CLOSE_openError 8
```

These codes are used in the `reason_code` parameter for the `mkDPIclose()` function and in the `reason_code` field in the `snmp_dpi_close_packet` structure.

See the following related information.

- "The `snmp_dpi_close_packet` structure" on page 91
- "The `mkDPIclose()` function" on page 64

---

**DPI packet types**

The currently defined DPI packet types as defined in the snmp_dpi.h include file are:

```c
#define SNMP_DPI_GET 1
#define SNMP_DPI_GET_NEXT 2 /* old DPI Version 1.x style */
#define SNMP_DPI_GETNEXT 2
#define SNMP_DPI_SET 3
#define SNMP_DPI_TRAP 4
#define SNMP_DPI_RESPONSE 5
#define SNMP_DPI_REGISTER 6
#define SNMP_DPI_UNREGISTER 7
#define SNMP_DPI_OPEN 8
#define SNMP_DPI_CLOSE 9
#define SNMP_DPI_COMMIT 10
#define SNMP_DPI_UNDO 11
#define SNMP_DPI_GETBULK 12
#define SNMP_DPI_TRAPV2 13 /* reserved, not implemented */
#define SNMP_DPI_INFORM 14 /* reserved, not implemented */
#define SNMP_DPI_ARE_YOU_THERE 15
```

These packet types are used in the `type` parameter for the `packet_type` field in the `snmp_dpi_hdr` structure.

See the following related information.

- "The `snmp_dpi_hdr` structure" on page 93

---

**DPI RESPONSE error codes**

In case of an error on an SNMP request like GET, GETNEXT, SET, COMMIT, or UNDO, the RESPONSE can have one of these currently defined error codes. They are defined in the snmp_dpi.h include file:

```c
#define SNMP_ERROR_noError 0
#define SNMP_ERROR_tooBig 1
#define SNMP_ERROR_noSuchName 2
#define SNMP_ERROR_badValue 3
#define SNMP_ERROR_readOnly 4
#define SNMP_ERROR_genErr 5
#define SNMP_ERROR_noAccess 6
```

---
In case of an error on a DPI only request (OPEN, REGISTER, UNREGISTER, ARE_YOU_THERE), the RESPONSE can have one of these currently defined error codes. They are defined in the snmp_dpi.h include file:

```c
#define SNMP_ERROR_DPI_noError 0
#define SNMP_ERROR_DPI_otherError 101
#define SNMP_ERROR_DPI_notFound 102
#define SNMP_ERROR_DPI_alreadyRegistered 103
#define SNMP_ERROR_DPI_higherPriorityRegistered 104
#define SNMP_ERROR_DPI_mustOpenFirst 105
#define SNMP_ERROR_DPI_notAuthorized 106
#define SNMP_ERROR_DPI_viewSelectionNotSupported 107
#define SNMP_ERROR_DPI_getBulkSelectionNotSupported 108
#define SNMP_ERROR_DPI_duplicateSubAgentIdentifier 109
#define SNMP_ERROR_DPI_invalidDisplayString 110
#define SNMP_ERROR_DPI_characterSetSelectionNotSupported 111
```

These codes are used in the `error_code` parameter for the `mkDPIresponse()` function and in the `error_code` field in the `snmp_dpi_resp_packet` structure.

See the following related information.

- "The snmp_dpi_resp_packet structure" on page 96
- "The mkDPIresponse() function" on page 69

### DPI UNREGISTER reason codes

These are the currently defined DPI UNREGISTER reason codes. They are defined in the snmp_dpi.h include file:

```c
#define SNMP_UNREGISTER_otherReason 1
#define SNMP_UNREGISTER_goingDown 2
#define SNMP_UNREGISTER_justUnregister 3
#define SNMP_UNREGISTER_newRegistration 4
#define SNMP_UNREGISTER_higherPriorityRegistered 5
#define SNMP_UNREGISTER_byManager 6
#define SNMP_UNREGISTER_timeout 7
```

These codes are used in the `reason_code` parameter for the `mkDPIunregister()` function and in the `reason_code` field in the `snmp_dpi_ureg_packet` structure.

See the following related information.

- "The snmp_dpi_ureg_packet structure" on page 99
- "The mkDPIunregister() function" on page 75

### DPI SNMP value types

These are the currently defined value types as defined in the snmp_dpi.h include file:
#define SNMP_TYPE_MASK 0x7f /* mask to isolate type*/
#define SNMP_TYPE_Integer32 (128|1) /* 32-bit INTEGER */
#define SNMP_TYPE_OCTET_STRING 2 /* OCTET STRING */
#define SNMP_TYPE_OBJECT_IDENTIFIER 3 /* OBJECT IDENTIFIER */
#define SNMP_TYPE_NULL 4 /* NULL, no value */
#define SNMP_TYPE_IpAddress 5 /* IMPLICIT OCTETSTRING*/
#define SNMP_TYPE_Counter32 (128|6) /* 32-bit Counter */
#define SNMP_TYPE_Gauge32 (128|7) /* 32-bit Gauge */
#define SNMP_TYPE_TimeTicks (128|8) /* 32-bit TimeTicks in */
/* hundredths of a sec */
#define SNMP_TYPE_DisplayString 9 /* DisplayString (TC) */
#define SNMP_TYPE_BIT_STRING 10 /* BIT STRING */
#define SNMP_TYPE_NsapAddress 11 /* IMPLICIT OCTETSTRING*/
#define SNMP_TYPE_UInteger32 (128|12) /* 32-bit INTEGER */
#define SNMP_TYPE_Counter64 13 /* 64-bit Counter */
#define SNMP_TYPE_Opaque 14 /* IMPLICIT OCTETSTRING*/
#define SNMP_TYPE_noSuchObject 15 /* IMPLICIT NULL */
#define SNMP_TYPE_noSuchInstance 16 /* IMPLICIT NULL */
#define SNMP_TYPE_endOfMibView 17 /* IMPLICIT NULL */

These value types are used in the value_type parameter for the mkDPIset() function and in the value_type field in the snmp_dpi_set_packet structure.

See the following related information.

- "The snmp_dpi_set_packet structure" on page 97
- "The mkDPIset() function" on page 71
- "Value representation of DPI SNMP value types" on page 104
- "Value ranges and limits for DPI SNMP value types" on page 105

## Value representation of DPI SNMP value types

Values in the snmp_dpi_set_packet structure are represented as follows:

- 32-bit integers are defined as long int or unsigned long int. A long int is assumed to be 4 bytes.
- 64-bit integers are represented as an snmp_dpi_u64.
  
  Unsigned 64 bit integers are only dealt with in SNMP. In a structure that has two fields, the high order piece and the low order piece, each is of type unsigned long int. These are assumed to be 4 bytes.
- Object identifiers are null-terminated strings in the selected character set, representing the OID in ASN.1 dotted-decimal notation. The length includes the terminating NULL.
  
  An ASCII example:
  
  '312e332e362e312e322e312e312e3000'h
  
  represents "1.3.6.1.2.1.1.1.0" which is sysDescr0.
  
  An EBCDIC example:
  
  'f14bf34bf64bf14bf24bf14bf14bf000'h
  
  represents "1.3.6.1.2.1.1.1.0" which is sysDescr0.
- DisplayStrings are in the selected character set. The length specifies the length of the string.
  
  An ASCII example:
  
  '6162630d0a'h
  
  represents "abc\r\n", no NULL.
  
  An EBCDIC example:
  
  '8182830d25'h
  
  represents "abc\r\n", no NULL.
IpAddress and Opaque are implicit OCTET_STRING, so they are a sequence of octets or bytes. This means, for instance, that the IP address is in network byte order.

NULL has a 0 length for the value, no value data, so a NULL pointer is returned in the value_p field.

noSuchObject, noSuchInstance, and endOfMibView are implicit NULL and are represented as such.

BIT_STRING is an OCTET_STRING of the form uubbbb...bb, where the first octet (uu) is 0x00-0x07 and indicates the number of unused bits in the last octet (bb). The bb octets represent the bit string itself, where bit 0 comes first and so on.

See the following related information.

“Value ranges and limits for DPI SNMP value types” on page 105

---

### Value ranges and limits for DPI SNMP value types

The following rules apply to object IDs in ASN.1 notation:

- The object ID consists of 1 to 128 subIDs, which are separated by periods.
- Each subID is a positive number. No negative numbers are allowed.
- The value of each number cannot exceed 4294967295. This value is 2 to the power of 32 minus 1.
- The valid values of the first subID are 0, 1, or 2.
- If the first subID has a value of 0 or 1, the second subID can only have a value of 0 through 39.

The following rules apply to DisplayString:

- A DisplayString (Textual Convention) is basically an OCTET STRING in SNMP terms.
- The maximum size of a DisplayString is 255 octets or bytes.

More information on the DPI SNMP value types can be found in the SNMP Structure of Management Information (SMI) and SNMP Textual Conventions (TC) RFCs. These two RFCs are RFC 1902 and RFC 1903. See Appendix E, “Related protocol specifications,” on page 743 for information about accessing RFCs.

---

### Return codes from DPI transport-related functions

These are the currently defined values for the return codes from DPI transport-related functions. They are defined in the snmp_dpi.h include file:

```c
#define DPI_RC_OK 0 /* all OK, no error */
#define DPI_RC_NOK -1 /* some other error */
#define DPI_RC_NO_PORT -2 /* can't determine DPIport */
#define DPI_RC_NO_CONNECTION -3 /* no connection to DPIagent */
#define DPI_RC_EOF -4 /* EOF received on connection */
#define DPI_RC_IO_ERROR -5 /* Some I/O error on connect */
#define DPI_RC_INVALID_HANDLE -6 /* unknown/invalid handle */
#define DPI_RC_TIMEOUT -7 /* timeout occurred */
#define DPI_RC_PACKET_TOO_LARGE -8 /* packed too large, dropped */
#define DPI_RC_UNSUPPORTED_DOMAIN -9 /* unsupported domain for connect */
#define DPI_RC_INVALID_ARGUMENT -10 /* invalid argument passed */
```

These values are used as return codes for the transport-related DPI functions.
See the following related information.

- “The DPIconnect_to_agent_TCP() function” on page 80
- “The DPIconnect_to_agent_UNIXstream() function” on page 82
- “The DPIawait_packet_from_agent() function” on page 78
- “The DPIsend_packet_to_agent() function” on page 86

The snmp_dpi.h include file

```c
#include <snmp_dpi.h>
```

**snmp_dpi.h include parameters**

None

**snmp_dpi.h include description**

The snmp_dpi.h include file defines the SNMP DPI API to the DPI subagent programmer. It has all the function prototype statements, and it also has the definitions for the snmp_dpi structures.

The same include file is used at the agent side, so you will see some definitions that are unique to the agent side. Also, other functions or prototypes of functions not implemented on SNMP might exist. Therefore, only use the API as it is documented in this manual.

Macros, functions, structures, constants, and values defined in the snmp_dpi.h include file are:

- “The DPIawait_packet_from_agent() function” on page 78
- “The DPIconnect_to_agent_TCP() function” on page 80
- “The DPIconnect_to_agent_UNIXstream() function” on page 82
- “The DPIdebug() function” on page 59
- “The DPIdisconnect_from_agent() function” on page 84
- “The DPI_PACKET_LEN() macro” on page 60
- “The DPIsend_packet_to_agent() function” on page 86
- “The fDPIparse() function” on page 61
- “The fDPIset() function” on page 62
- “The mkDPIAreYouThere() function” on page 63
- “The mkDPIclose() function” on page 64
- “The mkDPIopen() function” on page 65
- “The mkDPIregister() function” on page 67
- “The mkDPIresponse() function” on page 69
- “The mkDPIset() function” on page 71
- “The mkDPItrap() function” on page 73
- “The mkDPIunregister() function” on page 75
- “The pDPIpacket() function” on page 76
- “The snmp_dpi_close_packet structure” on page 91
- “The snmp_dpi_get_packet structure” on page 92
- “The snmp_dpi_next_packet structure” on page 95
- “The snmp_dpi_hdr structure” on page 93
- “The lookup_host() function” on page 88
DPI subagent example

This is an example of a DPI version 2.0 subagent. The code is called dpi_mvs_sample.c in the /usr/lpp/tcpip/samples directory.

Note: The example code in this document was copied from the sample file at the time of the publication. There might be differences in the code presented and the code that is included with the product. Always use the code provided in the /usr/lpp/tcpip/samples directory as the authoritative sample code.

The DPI subagent example includes:
- “Overview of subagent processing” on page 107
- “SNMP DPI: Connecting to the agent” on page 109
- “SNMP DPI: Registering a subtree with the agent” on page 111
- “SNMP DPI: Processing requests from the agent” on page 113
- “SNMP DPI: Processing a GET request” on page 116
- “SNMP DPI: Processing a GETNEXT request” on page 119
- “SNMP DPI: Processing a SET/COMMIT/UNDO request” on page 122
- “SNMP DPI: Processing an UNREGISTER request” on page 125
- “SNMP DPI: Processing a CLOSE request” on page 126
- “SNMP DPI: Generating a TRAP” on page 126

See the following related information.
- “SNMP DPI subagent programming concepts” on page 48

Overview of subagent processing

This overview assumes that the subagent communicates with the agent over a TCP connection. Other connection implementations are possible and, in that case, the processing approach might be a bit different.

In this overview, the agent will be requested to send at most one varBind per DPI packet, so there will be no need to loop through a list of varBinds. Potentially, you might gain performance improvements if you allow for multiple varBinds per DPI packet on GET, GETNEXT, SET requests, but to do so, your code will have to loop through the varBind list and so it becomes more complicated. The DPI subagent programmer can handle that once you understand the basics of the DPI API.

The following are the supported MIB variable definitions for DPI_SIMPLE:

```
DPISimple-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE, snmpModules, enterprises
```

Chapter 3. SNMP agent Distributed Protocol Interface version 2.0 107
FROM SNMPv2-SMI
DisplayString
FROM SNMPv2-TC

ibm OBJECT IDENTIFIER ::= { enterprises 2 }
ibmDPI OBJECT IDENTIFIER ::= { ibm 2 }
dpi20MIB OBJECT IDENTIFIER ::= { ibmDPI 1 }

-- dpiSimpleMIB MODULE-IDENTITY
-- LAST-UPDATED "9401310000Z"
-- DESCRIPTION
-- "The MIB module describing DPI Simple Objects for
-- the dpi_samp.c program"
-- ::= { snmpModules x }

dpiSimpleMIB OBJECT IDENTIFIER ::= { dpi20MIB 5 }

dpiSimpleInteger OBJECT-TYPE
SYNTAX INTEGER
ACCESS read-only
STATUS mandatory
DESCRIPTION
"A sample integer32 value"
::= { dpiSimpleMIB 1 }

dpiSimpleString OBJECT-TYPE
SYNTAX DisplayString
ACCESS read-write
STATUS mandatory
DESCRIPTION
"A sample Display String"
::= { dpiSimpleMIB 2 }

dpiSimpleCounter32 OBJECT-TYPE
SYNTAX Counter -- Counter32 is SNMPv2
ACCESS read-only
STATUS mandatory
DESCRIPTION
"A sample 32-bit counter"
::= { dpiSimpleMIB 3 }

dpiSimpleCounter64 OBJECT-TYPE
SYNTAX Counter -- Counter64 is SNMPv2,
-- No SMI support for it yet
ACCESS read-only
STATUS mandatory
DESCRIPTION
"A sample 64-bit counter"
::= { dpiSimpleMIB 4 }
END

To make the code more readable, the following names have been defined in our
dpi_mvs_sample.c source file.
#define DPI_SIMPLE_SUBAGENT "1.3.6.1.4.1.2.2.1.5"
#define DPI_SIMPLE_MIB "1.3.6.1.4.1.2.2.1.5."
#define DPI_SIMPLE_INTEGER "1.0" /* dpiSimpleInteger.0 */
#define DPI_SIMPLE_STRING "2.0" /* dpiSimpleString.0 */
#define DPI_SIMPLE_COUNTER32 "3.0" /* dpiSimpleCounter32.0 */
#define DPI_SIMPLE_COUNTER64 "4.0" /* dpiSimpleCounter64.0 */

In addition, the following variables have been defined as global variables in our
dpi_mvs_sample.c source file.
SNMP DPI: Connecting to the agent

Before a subagent can receive or send any DPI packets from or to the SNMP DPI-capable agent, it must connect to the agent and identify itself to the agent.

The following example code returns a response. It is assumed that there are no errors in the request, but proper code should do the checking for that. Proper checking is done for lexicographic next object, but no checking is done for ULONG_MAX, or making sure that the instance ID is indeed valid (digits and periods). If the code gets to the end of our dpiSimpleMIB, an endOfMibView must be returned as defined by the SNMP Version 2 rules. You will need to specify:

- A host name or IP address in dotted decimal notation that specifies where the agent is running. Often the name loopback can be used if the subagent runs on the same system as the agent.
- A community name that is used to obtain the dpi TCP port from the agent. Internally that is done by sending a regular SNMP GET request to the agent. In an open environment, the well-known community name public can probably be used.

The function returns a negative error code if an error occurs. If the connection setup is successful, it returns a handle that represents the connection and that must be used on subsequent calls to send or await DPI packets.

The second step is to identify the subagent to the agent. This is done by making a DPI-OPEN packet, sending it to the agent, and then awaiting the response from the agent. The agent can accept or deny the OPEN request. Making a DPI-OPEN packet is done by calling mkDPIopen(), which expects the following parameters:

- A unique subagent identification (an object identifier).
- A description, which can be the NULL string (""").
- Overall subagent timeout in seconds. The agent uses this value as a timeout value for a response when it sends a request to the subagent. The agent may have a maximum value for this timeout that will be used if you exceed it.
- The maximum number of varBinds per DPI packet that the subagent is willing or is able to handle.
- The desired character set. In most cases you want to use the native character set.
- Length of a password. A 0 means no password.
- Pointer to the password or NULL if no password. It depends on the agent if subagents must specify a password to open up a connection.
The function returns a pointer to a static buffer holding the DPI packet if successful. If it fails, it returns a NULL pointer.

When the DPI-OPEN packet has been created, you must send it to the agent. You can use the DPIsend_packet_to_agent() function, which expects the following parameters:
- The handle of a connection from DPIconnect_to_agent_TCP.
- A pointer to the DPI packet from mkDPIopen.
- The length of the packet. The snmp_dpi.h include file provides a macro DPI_PACKET_LEN that calculates the packet length of a DPI packet.

This function returns DPI_RC_OK (value 0) if successful. Otherwise, an appropriate DPI_RC_xxxx error code as defined in snmp_dpi.h is returned.

Now wait for a response to the DPI-OPEN. To await such a response, you call the DPIawait_packet_from_agent() function, which expects the following parameters:
- The handle of a connection from DPIconnect_to_agent_TCP.
- A timeout in seconds, which is the maximum time to wait for response.
- A pointer to a pointer, which will receive a pointer to a static buffer containing the awaited DPI packet. If the system fails to receive a packet, a NULL pointer is stored.
- A pointer to a long integer (32-bit), which will receive the length of the awaited packet. If it fails, it will be set to 0.

This function returns DPI_RC_OK (value 0) if successful. Otherwise, an appropriate DPI_RC_xxxx error code as defined in snmp_dpi.h is returned.

The last step is to ensure that you received a DPI-RESPONSE back from the agent. If so, ensure that the agent accepted you as a valid subagent. This will be shown by the error_code field in the DPI response packet.

The following example code establishes a connection and opens it by identifying you to the agent.

```c
static void do_connect_and_open(char *hostname_p, char *community_p)
{
    unsigned char *packet_p;
    int rc;
    unsigned long length;
    snmp_dpi_hdr *hdr_p;

    #ifdef MVS
    __etoa(community_p); /* Translate to ASCII */
    #endif /* MVS */

    #ifndef DPI_MINIMAL_SUBAGENT
    #ifdef INCLUDE_UNIX_DOMAIN_FOR_DPI
    if (unix_sock) {
        handle =
            DPIconnect_to_agent_UNIXstream( /* (UNIX) connect to */
                hostname_p, /* agent on this host */
                community_p); /* snmp community name */
    } else
    #endif /* def INCLUDE_UNIX_DOMAIN_FOR_DPI */
    #endif /* ndef DPI_MINIMAL_SUBAGENT */

    handle =
        DPIconnect_to_agent_TCP( /* (TCP) connect to agent */
            hostname_p, /* on this host */
            community_p); /* snmp community name */
```
SNMP DPI: Registering a subtree with the agent

After setting up a connection to the agent and identifying yourself, register one or more MIB subtrees or instances for which you want to be responsible to handle SNMP requests.

To do so, the subagent must create a DPI-REGISTER packet and send it to the agent. The agent will then send a response to indicate success or failure of the register request.

To create a DPI-REGISTER packet, the subagent uses a call to the mkDPIregister() function, which expects these parameters:

- A timeout value in seconds for this subtree. If you specify 0, your overall timeout value that was specified in DPI-OPEN is used. You can specify a different value if you expect longer processing time for a specific subtree.
- A requested priority. Multiple subagents may register the same subtree at different priorities. For example, 0 is better than 1 and so on. The agent considers the subagent with the best priority to be the active subagent for the subtree. If you specify -1, you are asking for the best priority available. If you specify 0, you are asking for a better priority than any existing subagent may already have.
The MIB subtree or instance that you want to control. For object level registration, this group ID must have a trailing dot. For instance level registration, this group ID would simply have the instance number follow the object number subtree.

You have no choice in GETBULK processing. You must ask the agent to map a GETBULK into multiple GETNEXT packets.

The function returns a pointer to a static buffer holding the DPI packet if successful. If it fails, it returns a NULL pointer.

Now send this DPI-REGISTER packet to the agent with the DPIsend_packet_to_agent() function. This is similar to sending the DPI_OPEN packet. Then wait for a response from the agent. Again, use the DPIawait_packet_from_agent() function in the same way as you awaited a response on the DPI-OPEN request. Once you have received the response, check the return code to ensure that registration was successful.

The following code example demonstrates how to register one MIB subtree with the agent.

```c
static void do_register(void)
{
    unsigned char *packet_p;
    int rc;
    unsigned long length;
    snmp_dpi_hdr *hdr_p;
    int i;
    char buf[512];
    for (i=0; i<4; i++) {
        strcpy(buf,DPI_SIMPLE_MIB);
        if (instance_level) {
            switch (i) {
            case 0:
                strcat(buf,DPI_SIMPLE_INTEGER);
                break;
            case 1:
                strcat(buf,DPI_SIMPLE_STRING);
                break;
            case 2:
                strcat(buf,DPI_SIMPLE_COUNTER32);
                break;
            case 3:
                strcat(buf,DPI_SIMPLE_COUNTER64);
                break;
            } /* endswitch */
        } /* endif */
        packet_p = mkDPIregister( /* Make DPIregister packet */
                                timeout, /* timeout in seconds */
                                0, /* requested priority */
                                buf, /* ptr to the subtree */
                                DPI_BULK_NO); /* Map GetBulk into GetNext */
        if (!packet_p) exit(1); /* If it failed, exit */
        rc = DPIsend_packet_to_agent( /* send REGISTER packet */
                                    handle, /* on this connection */
                                    packet_p, /* this is the packet */
                                    DPI_PACKET_LEN(packet_p)); /* and this is its length */
        if (rc != DPI_RC_OK) exit(1); /* If it failed, exit */
        rc = DPIawait_packet_from_agent( /* wait for response */
                                             0); /* timeout in seconds */
    } /* endfor */
}
```

112  z/OS V1R11.0 Comm Svr: IP Programmer’s Guide and Reference
SNMP DPI: Processing requests from the agent

After registering your sample MIB subtree with the agent, expect that SNMP requests for that subtree are passed back to you for processing. Since the requests arrive in the form of DPI packets on the connection that you have established, go into a While loop to await DPI packets from the agent.

Because the subagent cannot know in advance which kind of packet arrives from the agent, await a DPI packet (forever), then parse the packet, check the packet type, and process the request based on the DPI packet type. A call to pDPIpacket, which expects as parameter a pointer to the encoded or serialized DPI packet, returns a pointer to a DPI parse tree. The pointer points to an snmp_dpi_hdr structure which looks as follows:

```c
struct snmp_dpi_hdr {
    unsigned char proto_major;
    unsigned char proto_version;
    unsigned char proto_release;
    unsigned short packet_id;
    unsigned char packet_type;
    union {
        snmp_dpi_reg_packet *reg_p;
        snmp_dpi_ureg_packet *ureg_p;
        snmp_dpi_get_packet *get_p;
        snmp_dpi_next_packet *next_p;
        snmp_dpi_next_packet *bulk_p;
        snmp_dpi_set_packet *set_p;
        snmp_dpi_resp_packet *resp_p;
        snmp_dpi_trap_packet *trap_p;
        snmp_dpi_open_packet *open_p;
        snmp_dpi_close_packet *close_p;
        unsigned char *any_p;
    } data_u;
};
typedef struct snmp_dpi_hdr snmp_dpi_hdr;
#define snmp_dpi_hdr_NULL_p ((snmp_dpi_hdr *)0)
```

With the DPI parse tree, you decide how to process the DPI packet. The following code example demonstrates the high-level process of a DPI subagent.

```c
main(int argc, char *argv[], char *envp[])
{
    unsigned char *packet_p;
```
int i = 0;
int rc = 0;
#ifndef DPI_VERY_MINIMAL_SUBAGENT /* with VERY minimal agent */
int debug = 0;
#endif /* ndef DPI_VERY_MINIMAL_SUBAGENT */
unsigned long length;
snmp_dpi_hdr *hdr_p;
char *hostname_p = NULL; /* @L1C*/
char *community_p = SNMP_COMMUNITY;
char *cmd_p = "";
char hostname[MAX_HOSTNAME_LEN+1]; /* @L1A*/
if (argc >= 1) cmd_p = argv[0];
for (i=1; i < argc; i++) {
    if (strcmp(argv[i],"-h") == 0) {
        if (i+1 >= argc) {
            printf("Need hostname\n\n");
            usage(cmd_p);
        } /* endif */
        hostname_p = argv[++i];
    } /* endif */
    else if (strcmp(argv[i],"-c") == 0) {
        if (i+1 >= argc) {
            printf("Need community name\n\n");
            usage(cmd_p);
        } /* endif */
        community_p = argv[++i];
    } /* endif */
    else if (strcmp(argv[i],"-unix") == 0) {
        unix_sock = 1;
    } /* endif */
    else if (strcmp(argv[i],"-ireg") == 0) {
        instance_level = 1;
    } /* endif */
    else if (strcmp(argv[i],"-d") == 0) {
        if (i+1 >= argc) {
            debug = 1;
            continue;
        }
        if (strlen(argv[i+1]) == 1 && isdigit(*argv[i+1])) {
            i++;
            debug = atoi(argv[i+1]);
        } else {
            debug = 1;
        } /* endif */
    } /* endif */
    else { /* define DPI_VERY_MINIMAL_SUBAGENT */
        usage(cmd_p);
    } /* endif */
} /* endfor */
#endif /* ndef DPI_VERY_MINIMAL_SUBAGENT */
if (debug) {
    printf("%s - %s\n", __FILE__, VERSION);
    DPIdebug(debug); /* turn on DPI debugging */
    timeout += 6; /* longer timeout please */
} /* endif */
#endif /* ndef DPI_VERY_MINIMAL_SUBAGENT */
if (hostname_p == NULL) { /* -h not specified. Try to
    obtain local host name
    if (gethostname(hostname, MAX_HOSTNAME_LEN) != 0) { /* @L1A*/
        printf("gethostname failed. *
    ");
    exit(1); /* @L1A*/
}
else {
    hostname_p = hostname;
}

/* first init value2_p, our dpiSimpleString (DisplayString) */
/* since we treat it as display string keep terminating NULL */
value2_p = (char *) malloc(strlen("Initial String")+1);
if (value2_p) {
    memcpy(value2_p,"Initial String",strlen("Initial String")+1);
    value2_len = strlen("Initial String")+1;
} /* endif */

do_connect_and_open(hostname_p, /* connect and DPI-OPEN */
    community_p);

do_register(); /* register our subtree */
do_trap(); /* issue a trap as sample */

while (rc == 0) { /* do forever */
    rc = DPIawait_packet_from_agent( /* wait for a DPI packet */
        handle, /* on this connection */
        -1, /* wait forever */
        &packet_p, /* receives ptr to packet */
        &length); /* receives packet length */

    if (rc != DPI_RC_OK) exit(1); /* If it failed, exit */
    hdr_p = pDPIpacket(packet_p); /* parse DPI packet */
    if (hdr_p == snmp_dpi_hdr_NULL_p)/* If we fail to parse it */
        exit(1); /* then exit */

    switch(hdr_p->packet_type) { /* handle by DPI type */
    case SNMP_DPI_GET:
        rc = do_get(hdr_p,
            hdr_p->data_u.get_p);
        break;
    case SNMP_DPI_GETNEXT:
        rc = do_next(hdr_p,
            hdr_p->data_u.next_p);
        break;
    case SNMP_DPI_SET:
    case SNMP_DPI_COMMIT:
    case SNMP_DPI_UNDO:
        rc = do_set(hdr_p,
            hdr_p->data_u.set_p);
        break;
    case SNMP_DPI_CLOSE:
        rc = do_close(hdr_p,
            hdr_p->data_u.close_p);
        break;
    case SNMP_DPI_UNREGISTER:
        rc = do_unreg(hdr_p,
            hdr_p->data_u.ureg_p);
        break;
    default:
        printf("Unexpected DPI packet type %d\n",
            hdr_p->packet_type);
        rc = -1;
    } /* endswitch */
    if (rc) exit(1);
} /* endwhile */

return(0); /* end of main() */
SNMP DPI: Processing a GET request

When the DPI packet is parsed, the snmp_dpi_hdr structure will show in the
packet_type that this is an SNMP_DPI_GET packet. In that case, the packet_body
contains a pointer to a GET-varBind, which is represented in an
snmp_dpi_get_packet structure:

```
struct dpi_get_packet {
  char *object_p; /* ptr to OIDstring */
  char *group_p; /* ptr to sub-tree */
  char *instance_p; /* ptr to rest of OID */
  struct dpi_get_packet *next_p; /* ptr to next in chain */
};
typedef struct dpi_get_packet snmp_dpi_get_packet;
#define snmp_dpi_get_packet_NULL_p ((snmp_dpi_get_packet *)0)
```

Assuming you have registered subtree 1.3.6.1.4.1.2.2.1.5 and a GET request comes
in for one variable (1.3.6.1.4.1.2.2.1.5.1.0) that is object 1 instance 0 in the subtree,
the fields in the snmp_dpi_get_packet would have pointers to:

- `object_p` -> "1.3.6.1.4.1.2.2.1.5.1.0"
- `group_p` -> "1.3.6.1.4.1.2.2.1.5."
- `instance_p` -> "1.0"
- `next_p` -> `snmp_dpi_get_packet_NULL_p`

If there are multiple varBinds in a GET request, each one is represented in an
snmp_dpi_get_packet structure and all the snmp_dpi_get_packet structures are
chained using the next pointer. As long as the next pointer is not the
snmp_dpi_get_packet_NULL_p pointer, there are more varBinds in the list.

Now you can analyze the varBind structure for whatever checking you want to do.
When you are ready to make a response that contains the value of the variable,
you prepare a SET-varBind, which is represented in an snmp_dpi_set_packet
structure:

```
struct dpi_set_packet {
  char *object_p; /* ptr to OIDstring */
  char *group_p; /* ptr to sub-tree */
  char *instance_p; /* ptr to rest of OID */
  unsigned char value_type; /* SNMP_TYPE_xxxx */
  unsigned short value_len; /* value length */
  char *value_p; /* ptr to value itself */
  struct dpi_set_packet *next_p; /* ptr to next in chain */
};
typedef struct dpi_set_packet snmp_dpi_set_packet;
#define snmp_dpi_set_packet_NULL_p ((snmp_dpi_set_packet *)0)
```

You can use the mkDPIset() function to prepare such a structure. This function
expects the following parameters:

- A pointer to an existing snmp_dpi_set_packet structure if the new varBind must
  be added to an existing chain of varBinds. If this is the first or the only varBind
  in the chain, pass the snmp_dpi_set_packet_NULL_p pointer to indicate this.
- A pointer to the subtree that you registered.
- A pointer to the rest of the OID; in other words, the piece that follows the
  subtree.
- The value type of the value to be bound to the variable name. This must be one
  of the SNMP_TYPE_xxxx values as defined in the snmp_dpi.h include file.
The length of the value. For integer type values, this must be a length of 4. Work with 32-bit signed or unsigned integers except for the Counter64 type. For the Counter64 type, point to an snmp_dpi_u64 structure and pass the length of that structure.

A pointer to the value.

Memory for the varBind is dynamically allocated and the data itself is copied. So upon return you can dispose of our own pointers and allocated memory as you please. If the call is successful, a pointer is returned as follows:

- To a new snmp_dpi_set_packet if it is the first or only varBind.
- To the existing snmp_dpi_set_packet that you passed on the call. In this case, the new packet has been chained to the end of the varBind list.

If the mkDPIset() call fails, a NULL pointer is returned.

When you have prepared the SET-varBind data, you can create a DPI RESPONSE packet using the mkDPIresponse() function that expects these parameters:

- A pointer to an snmp_dpi_hdr. You should use the header of the parsed incoming packet. It is used to copy the packet_id from the request into the response, such that the agent can correlate the response to a request.
- A return code which is an SNMP error code. If successful, this should be SNMP_ERROR_noError (value 0). If failure, it must be one of the SNMP_ERROR_xxxx values as defined in the snmp_dpi.h include file.

A request for a nonexisting object or instance is not considered an error. Instead, you must pass a value type of SNMP_TYPE_noSuchObject or SNMP_TYPE_noSuchInstance respectively. These two value types have an implicit value of NULL, so you can pass a 0 length and a NULL pointer for the value in this case.

- The index of the varBind in error starts counting at 1. Pass 0 if no error occurred, or pass the proper index of the first varBind for which an error was detected.
- A pointer to a chain of snmp_dpi_set_packets (varBinds) to be returned as response to the GET request. If an error was detected, an snmp_dpi_set_packet_NULL_p pointer may be passed.

The following code example returns a response. You assume that there are no errors in the request, but proper code should do the checking for that. For instance, you return a noSuchInstance if the instance is not exactly what you expect and a noSuchObject if the object instance_ID is greater than 3. However, there might be no instance_ID at all and you should check for that, too.

```c
static int do_get(snmp_dpi_hdr *hdr_p, snmp_dpi_get_packet *pack_p)
{
    unsigned char  *packet_p;
    int rc;
    snmp_dpi_set_packet *varBind_p;
    char  *i_p;

    varBind_p = (snmp_dpi_set_packet_NULL_p); /* init the varBind chain */
    if (instance_level) {
        if (pack_p->instance_p) {
            printf("unexpected INSTANCE ptr \
");
            return(-1);
        }
    }
    i_p = pack_p->group_p + strlen(DPI_SIMPLE_MIB);

    return(0);
}
```
i_p = pack_p->instance_p;

if ((i_p && (strcmp(i_p,"1.0") == 0)) {  
    varBind_p = mkDPIset( /* Make DPI set packet */
        varBind_p, /* ptr to varBind chain */
        pack_p->group_p, /* ptr to subtree */
        pack_p->instance_p, /* ptr to rest of OID */
        SNMP_TYPE_Integer32, /* value type Integer 32 */
        sizeof(value1), /* length of value */
        value1); /* ptr to value */
} else if (i_p && (strcmp(i_p,"2.0") == 0)) {  
    varBind_p = mkDPIset( /* Make DPI set packet */
        varBind_p, /* ptr to varBind chain */
        pack_p->group_p, /* ptr to subtree */
        pack_p->instance_p, /* ptr to rest of OID */
        SNMP_TYPE_DisplayString,/* value type */
        value2_len, /* length of value */
        value2_p); /* ptr to value */
} else if (i_p && (strcmp(i_p,"3.0") == 0)) {  
    varBind_p = mkDPIset( /* Make DPI set packet */
        varBind_p, /* ptr to varBind chain */
        pack_p->group_p, /* ptr to subtree */
        pack_p->instance_p, /* ptr to rest of OID */
        SNMP_TYPE_Counter32, /* value type */
        sizeof(value3), /* length of value */
        value3); /* ptr to value */
} else if (i_p && (strcmp(i_p,"4.0") == 0)) {  
    varBind_p = mkDPIset( /* Make DPI set packet */
        varBind_p, /* ptr to varBind chain */
        pack_p->group_p, /* ptr to subtree */
        pack_p->instance_p, /* ptr to rest of OID */
        SNMP_TYPE_Counter64, /* value type */
        sizeof(value4), /* length of value */
        value4); /* ptr to value */
} else if (i_p && (strcmp(i_p,"4") > 0)) {  
    varBind_p = mkDPIset( /* Make DPI set packet */
        varBind_p, /* ptr to varBind chain */
        pack_p->group_p, /* ptr to subtree */
        pack_p->instance_p, /* ptr to rest of OID */
        SNMP_TYPE_noSuchObject, /* value type */
        0L, /* length of value */
        (unsigned char *)0); /* ptr to value */
} else {  
    varBind_p = mkDPIset( /* Make DPI set packet */
        varBind_p, /* ptr to varBind chain */
        pack_p->group_p, /* ptr to subtree */
        pack_p->instance_p, /* ptr to rest of OID */
        SNMP_TYPE_noSuchInstance,/* value type */
        0L, /* length of value */
        (unsigned char *)0); /* ptr to value */
} /* endif */ /* ndef EXCLUDE_SNMP_SMIv2_SUPPORT */

if (!varBind_p) return(-1); /* If it failed, return */

packet_p = mkDPIresponse( /* Make DPI response packet */
    hdr_p, /* ptr parsed request */
    SNMP_ERROR_noError, /* all is OK, no error */
    0L, /* index is zero, no error */
    varBind_p); /* varBind response data */
if (!packet_p) return(-1); /* If it failed, return */

rc = DPIsend_packet_to_agent( /* send RESPONSE packet */
    handle, /* on this connection */
    packet_p, /* this is the packet */
    DPI_PACKET_LEN(packet_p)); /* and this is its length */

return(rc); /* return retcode */
} /* end of do_get() */

---

**SNMP DPI: Processing a GETNEXT request**

When a DPI packet is parsed, the snmp_dpi_hdr structure shows in the
packet_type that this is an SNMP_DPI_GETNEXT packet, and so the packet_body
contains a pointer to a GETNEXT-varBind, which is represented in an
snmp_dpi_next_packet structure:

```
struct dpi_next_packet {
    char *object_p; /* ptr to OIDstring */
    char *group_p; /* ptr to sub-tree */
    char *instance_p; /* ptr to rest of OID */
    struct dpi_next_packet *next_p; /* ptr to next in chain */
};
```

typedef struct dpi_next_packet snmp_dpi_next_packet;
#define snmp_dpi_next_packet_NULL_p ((snmp_dpi_next_packet *)0)

Assuming you have registered subtree dpiSimpleMIB and a GETNEXT arrives for
one variable (dpiSimpleInteger.0) that is object 1 instance 0 in the subtree, the fields
in the snmp_dpi_get_packet structure would have pointers to:

- `object_p` -> "1.3.6.1.4.1.2.2.1.5.0"
- `group_p` -> "1.3.6.1.4.1.2.2.1.5."
- `instance_p` -> "1.0"
- `next_p` -> snmp_dpi_next_packet_NULL_p

If there are multiple varBinds in a GETNEXT request, each one is represented in an
snmp_dpi_next_packet structure and all the snmp_dpi_next_packet structures are
chained by the next pointer. As long as the next pointer is not the
snmp_dpi_next_packet_NULL_p pointer, there are more varBinds in the list.

Now you can analyze the varBind structure for whatever checking you want to do.
You must find out which OID is the one that lexicographically follows the one in
the request. It is that OID with its value that you must return as a response.
Therefore, you must now also set the proper OID in the response. When you are
ready to make a response that contains the new OID and the value of that variable,
you must prepare a SET-varBind which is represented in an snmp_dpi_set_packet:

```
struct dpi_set_packet {
    char *object_p; /* ptr to OIDstring */
    char *group_p; /* ptr to sub-tree */
    char *instance_p; /* ptr to rest of OID */
    unsigned char value_type; /* SNMP_TYPE_xxxx */
    unsigned short value_len; /* value length */
    char *value_p; /* ptr to value itself */
    struct dpi_set_packet *next_p; /* ptr to next in chain */
};
```

typedef struct dpi_set_packet snmp_dpi_set_packet;
#define snmp_dpi_set_packet_NULL_p ((snmp_dpi_set_packet *)0)

You can use the mkDPIset() function to prepare such a structure. This function
expects the following parameters:
- A pointer to an existing snmp_dpi_set_packet structure if the new varBind must be added to an existing chain of varBinds. If this is the first or only varBind in the chain, pass the snmp_dpi_set_packet_NULL_p pointer to indicate this.
- A pointer to the desired subtree.
- A pointer to the rest of the OID, in other words the piece that follows the subtree.
- The value type of the value to be bound to the variable name. This must be one of the SNMP_TYPE_xxxx values as defined in the snmp_dpi.h include file.
- The length of the value. For integer type values, this must be a length of 4. Work with 32-bit signed or unsigned integers except for the Counter64 type. For Counter 64 type, point to an snmp_dpi_u64 structure and pass the length of that structure.
- A pointer to the value.

Memory for the varBind is dynamically allocated and the data itself is copied. Upon return, you can dispose of your own pointers and allocated memory as you please. If the call is successful, a pointer is returned as follows:
- A new snmp_dpi_set_packet if it is the first or only varBind.
- The existing snmp_dpi_set_packet that you passed on the call. In this case, the new packet has been chained to the end of the varBind list.

If the mkDPIset() call fails, a NULL pointer is returned.

When you have prepared the SET-varBind data, create a DPI RESPONSE packet using the mkDPIresponse() function, which expects these parameters:
- A pointer to an snmp_dpi_hdr. Use the header of the parsed incoming packet. It is used to copy the packet_id from the request into the response, such that the agent can correlate the response to a request.
- A return code that is an SNMP error code. If successful, this should be SNMP_ERROR_noError (value 0). If failure, it must be one of the SNMP_ERROR_xxxx values as defined in the snmp_dpi.h include file.

A request for a nonexisting object or instance is not considered an error. Instead, pass the OID and value of the first OID that lexicographically follows the nonexisting object or instance.

Reaching the end of the subtree is not considered an error. For example, if there is no NEXT OID, this is not an error. In this situation, return the original OID as received in the request and a value_type of SNMP_TYPE_endOfMibView. This value_type has an implicit value of NULL, so you can pass a 0 length and a NULL pointer for the value.
- The index of the first varBind in error starts counting at 1. Pass 0 if no error occurred, or pass the proper index of the first varBind for which an error was detected.
- A pointer to a chain of snmp_dpi_set_packets (varBinds) to be returned as response to the GETNEXT request. If an error was detected, an snmp_dpi_set_packet_NULL_p pointer may be passed.

The following code example returns a response. It is assumed that there are no errors in the request, but proper code should do the checking for that. Proper checking is done for lexicographic next object, but no checking is done for ULONG_MAX, or making sure that the instance ID is indeed valid (digits and periods). If the code gets to the end of our dpiSimpleMIB, an endOfMibView is returned as defined by the SNMP Version 2 rules.
static int do_next(snmp_dpi_hdr *hdr_p, snmp_dpi_next_packet *pack_p)
{
  unsigned char *packet_p;
  int rc;
  unsigned long subid;  /* subid is unsigned */
  unsigned long instance; /* same with instance */
  char *cp;
  snmp_dpi_set_packet *varBind_p;

  varBind_p = /* init the varBind chain */
              snmp_dpi_set_packet_NULL_p; /* to a NULL pointer */

  /* If we have done instance level registration, then we should */
  /* never get a getNext. Anyway, if we do, then we skip this and */
  /* return an endOfMibView. */
  if (instance_level) {
    varBind_p = mkDPIset( /* Make DPI set packet */
                           varBind_p, /* ptr to varBind chain */
                           pack_p->group_p, /* ptr to subtree */
                           pack_p->instance_p, /* ptr to rest of OID */
                           SNMP_TYPE_endOfMibView, /* value type */
                           0L, /* length of value */
                           (unsigned char *)0); /* ptr to value */
  }
  else {
    if (pack_p->instance_p) { /* we have an instance ID */
      cp = pack_p->instance_p; /* pick up ptr */
      subid = strtoul(cp, cp, 10); /* convert subid (object) */
      if (*cp == '.') { /* followed by a dot ? */
        cp++; /* point after it if yes */
      }
      instance=strtoul(cp,cp,10); /* convert real instance */
      /* not that we need it, we */
      subid++; /* only have instance 0, */
      /* so NEXT is next object */
      instance = 0; /* and always instance 0 */
    } else { /* no real instance passed */
      instance = 0; /* so we can use 0 */
      if (subid == 0) subid++; /* if object 0, start at 1 */
    } /* endif */
  } else { /* no instance ID passed */
    subid = 1; /* so do first object */
    instance = 0; /* instance 0 (all we have) */
  } /* endif */

  /* we have set subid and instance such that we can basically */
  /* process the request as a GET now. Actually, we don't even */
  /* need instance, because all out object instances are zero. */
  if (instance != 0) printf("Strange instance: %lu\n",instance);

  switch (subid) {
    case 1: /* Make DPI set packet */
      varBind_p = mkDPIset( /* ptr to varBind chain */
                            pack_p->group_p, /* ptr to subtree */
                            DPI_SIMPLE_INTEGER, /* ptr to rest of OID */
                            SNMP_TYPE_Integer32, /* value type Integer 32 */
                            sizeof(value1), /* length of value */
                            value1); /* ptr to value */
      break;
    case 2: /* Make DPI set packet */
      varBind_p = mkDPIset( /* ptr to varBind chain */
                            pack_p->group_p, /* ptr to subtree */
                            DPI_SIMPLE_STRING, /* ptr to rest of OID */
                            value2); /* ptr to value */
      break;
  }
}

Chapter 3. SNMP agent Distributed Protocol Interface version 2.0 121
SNMP DPI: Processing a SET/COMMIT/UNDO request

These three requests can come in one of these sequences:
- SET, COMMIT
- SET, UNDO
- SET, COMMIT, UNDO

The normal sequence is SET and then COMMIT. When a SET request is received, preparations must be made to accept the new value. For example, check that
request is for an existing object and instance, check the value type and contents to be valid, and allocate memory, but do not yet make the change.

If there are no SET errors, the next received request will be a COMMIT request. It is then that the change must be made, but keep enough information such that you can UNDO the change later if you get a subsequent UNDO request. The latter may happen if the agent discovers any errors with other subagents while processing requests that belong to the same original SNMP SET packet. All the varBinds in the same SNMP request PDU must be processed as if atomic.

When the DPI packet is parsed, the snmp_dpi_hdr structure shows in the packet_type that this is an SNMP_DPI_SET, SNMP_DPI_COMMIT, or SNMP_DPI_UNDO packet. In that case, the packet_body contains a pointer to a SET-varBind, represented in an snmp_dpi_set_packet structure. COMMIT and UNDO have same varBind data as SET upon which they follow:

```c
struct dpi_set_packet {
    char    *object_p; /* ptr to OIDstring */
    char    *group_p;  /* ptr to sub-tree */
    char    *instance_p; /* ptr to rest of OID */
    unsigned char value_type; /* SNMP_TYPE_xxxx */
    unsigned short value_len; /* value length */
    char    *value_p; /* ptr to value itself */
    struct dpi_set_packet *next_p; /* ptr to next in chain */
};
typedef struct dpi_set_packet snmp_dpi_set_packet;
#define snmp_dpi_set_packet_NULL_p ((snmp_dpi_set_packet *)0)
```

Assuming we have a registered subtree dpiSimpleMIB and a SET request comes in for one variable (dpiSimpleString.0) that is object 1 instance 0 in the subtree, and also assuming that the agent knows about our compiled dpiSimpleMIB so that it knows this is a DisplayString (as opposed to just an arbitrary OCTET_STRING), the pointers in the snmp_dpi_set_packet structure would have pointers and values, such as:

- `object_p` -> "1.3.6.1.4.1.2.2.1.5.2.0"
- `group_p` -> "1.3.6.1.4.1.2.2.1.5."
- `instance_p` -> "2.0"
- `value_type` -> SNMP_TYPE_DisplayString
- `value_len` -> 8
- `value_p` -> pointer to the value to be set
- `next_p` -> snmp_dpi_set_packet_NULL_p

If there are multiple varBinds in a SET request, each one is represented in an snmp_dpi_set_packet structure and all the snmp_dpi_set_packet structures are chained by the next pointer. As long as the next pointer is not the snmp_dpi_set_packet_NULL_p pointer, there are more varBinds in the list.

Now you can analyze the varBind structure for whatever checking you want to do. When you are ready to make a response that contains the value of the variable, you can prepare a new SET-varBind. However, by definition, the response to a successful SET is exactly the same as the SET request. So there is no need to return any varBinds. A response with SNMP_ERROR_noError and an index of zero will do. If there is an error, a response with the SNMP_ERROR_xxxx error code and an index pointing to the varBind in error (counting starts at 1) will do.

The following code example returns a response. It is assumed that there are no errors in the request, but proper code should do the checking for that. The code also does not check if the varBind in the COMMIT or UNDO is the same as that in the SET request. A proper agent would make sure that that is the case, but a
proper subagent may want to verify that for itself. Only one check is done that this is dpiSimpleString,0, and if it is not, a noCreation is returned.

static int do_set(snmp_dpi_hdr *hdr_p, snmp_dpi_set_packet *pack_p)
{
    unsigned char *packet_p;
    int rc;
    int index = 0;
    int error = SNMP_ERROR_noError;
    snmp_dpi_set_packet *varBind_p;
    char *i_p;

    varBind_p = snmp_dpi_set_packet_NULL_p; /* init the varBind chain */
    if (instance_level) {
        i_p = pack_p->group_p + strlen(DPI_SIMPLE_MIB);
    } else {
        i_p = pack_p->instance_p;
    }

    if (!i_p ││ (strcmp(i_p,"2.0") != 0))
    {
        if (i_p && (strncmp(i_p,"1." ,2) == 0))
        {
            error = SNMP_ERROR_notWritable;
        } else if (i_p && (strncmp(i_p,"2." ,2) == 0))
        {
            error = SNMP_ERROR_noCreation;
        } else if (i_p && (strncmp(i_p,"3." ,2) == 0))
        {
            error = SNMP_ERROR_notWritable;
        } else {
            error = SNMP_ERROR_noCreation;
        } /* endif */

        packet_p = mkDPIresponse( /* Make DPIresponse packet */
            hdr_p, /* ptr parsed request */
            error, /* all is OK, no error */
            index, /* index is 1, 1st varBind */
            varBind_p); /* varBind response data */

        if (!packet_p) return(-1); /* If it failed, return */

        rc = DPIsend_packet_to_agent( /* send RESPONSE packet */
            handle, /* on this connection */
            packet_p, /* this is the packet */
            DPI_PACKET_LEN(packet_p));/* and this is its length */

        return(rc); /* return retcode */
    }

    switch (hdr_p->packet_type) {
    case SNMP_DPI_SET:
        if (!pack_p->value_type || pack_p->value_type != SNMP_TYPE_OCTET_STRING)
        {
            error = SNMP_ERROR_wrongType;
            break; /* from switch */
        } /* endif */

        new_val_p = (char *)0;
        if (new_val_p) free(new_val_p); /* free these memory areas */
        if (old_val_p) free(old_val_p); /* if we allocated any */
        new_val_p = (char *)0;

        break; /* from switch */
    } /* switch */
}
old_val_p = (char *)0;
new_val_len = 0;
old_val_len = 0;

new_val_p = malloc(pack_p->value_len); /* allocate memory for new value to set */
if (new_val_p) /* If success, then also */
    memcpy(new_val_p, pack_p->value_p, pack_p->value_len); /* copy new value to our own and newly allocated memory area. */
new_val_len = pack_p->value_len;
} else { /* Else failed to malloc, */
    error = SNMP_ERROR_genErr; /* so that is a genErr */
    index = 1; /* at first varBind */
} /* endif */
break;

SNMP DPI: Processing an UNREGISTER request

An agent can send an UNREGISTER packet if some other subagent does a register for the same subtree at a higher priority. An agent can also send an UNREGISTER if, for example, an SNMP manager tells the agent to make the subagent connection or the registered subtree not valid.

Here is an example of how to handle such a packet.
SNMP DPI: Processing a CLOSE request

An agent can send a CLOSE packet if it encounters an error or for some other reason. It can also do so if an SNMP MANAGER tells it to make the subagent connection not valid.

Here is an example of how to handle such a packet.

static int do_close(snmp_dpi_hdr *hdr_p, snmp_dpi_close_packet *pack_p)
{
    printf("DPI CLOSE received from agent, reason=%d\n", pack_p->reason_code);
    DPIdisconnect_from_agent(handle);
    return(-1); /* causes exit in main loop */
} /* end of do_close() */

SNMP DPI: Generating a TRAP

Issue a trap any time after a DPI OPEN was successful. To do so, you must create a trap packet and send it to the agent. With the TRAP, you can pass different kinds of varBinds, if you want. In this example, three varBinds are passed; one with integer data, one with an octet string, and one with a counter. You can also pass an Enterprise ID, but with DPI 2.0, the agent will use your subagent ID as the enterprise ID if you do not pass one with the trap. In most cases, that will probably not cause problems.

You must first prepare a varBind list chain that contains the three variables that you want to pass along with the trap. To do so, prepare a chain of three snmp_dpi_set_packet structures, which looks like:

```c
struct dpi_set_packet {
    char *object_p; /* ptr to OIDstring */
    char *group_p; /* ptr to sub-tree */
    char *instance_p; /* ptr to rest of OID */
    unsigned char value_type; /* SNMP_TYPE_xxxx */
    unsigned short value_len; /* value length */
    char *value_p; /* ptr to value itself */
    struct dpi_set_packet *next_p; /* ptr to next in chain */
};
typedef struct dpi_set_packet snmp_dpi_set_packet;
#define snmp_dpi_set_packet_NULL_p ((snmp_dpi_set_packet *)0)
```

You can use the mkDPIset() function to prepare such a structure. This function expects the following parameters:
A pointer to an existing snmp_dpi_set_packet structure if the new varBind must be added to an existing chain of varBinds. If this is the first or the only varBind in the chain, pass the snmp_dpi_set_packet_NULL_p pointer to indicate this.

A pointer to the desired subtree.

A pointer to the rest of the OID, in other words, the piece that follows the subtree.

The value type of the value to be bound to the variable name. This must be one of the SNMP_TYPE_xxxx values as defined in the snmp_dpi.h include file.

The length of the value. For integer type values, this must be a length of 4. Always work with 32-bit signed or unsigned integers except for the Counter64 type. For the Counter64 type, point to an snmp_dpi_u64 structure and pass the length of that structure.

A pointer to the value.

Memory for the varBind is dynamically allocated and the data itself is copied. Upon return, you can dispose of your own pointers and allocated memory as you please. If the call is successful, a pointer is returned as follows:

- To a new snmp_dpi_set_packet if it is the first or only varBind.
- To the existing snmp_dpi_set_packet that you passed on the call. In this case, the new packet has been chained to the end of the varBind list.

If the mkDPIset() call fails, a NULL pointer is returned.

When you have prepared the SET-varBind data, create a DPI TRAP packet. To do so, use the mkDPItrap() function, which expects these parameters:

- The generic trap code. Use 6 for enterprise specific trap type.
- The specific trap type. This is a type that is defined by the MIB that you are implementing. In our example you just use a 1.
- A pointer to a chain of varBinds or the NULL pointer if no varBinds need to be passed with the trap.
- A pointer to the enterprise OID if you want to use a different enterprise ID than the OID you used to identify yourself as a subagent at DPI-OPEN time.

The following code creates an enterprise-specific trap with specific type 1 and passes 3 varBinds. The first varBind with object 1, instance 0, Integer32 value; the second varBind with object 2, instance 0, Octet String; the third with Counter32. You pass no enterprise ID.

```c
static int do_trap(void)
{
    unsigned char *packet_p;
    int rc;
    snmp_dpi_set_packet *varBind_p, *set_p;

    varBind_p = snmp_dpi_set_packet_NULL_p; /* init the varBind chain */
    varBind_p = mkDPIset( /* Make DPI set packet */
        varBind_p, /* ptr to varBind chain */
        DPI_SIMPLE_MIB, /* ptr to subtree */
        DPI_SIMPLE_INTEGER, /* ptr to rest of OID */
        SNMP_TYPE_Integer32, /* value type Integer 32 */
        sizeof(value1), /* length of value */
        value1); /* ptr to value */

    if (!varBind_p) return(-1); /* If it failed, return */
```
set_p = mkDPIset( /* Make DPI set packet */
    varBind_p, /* ptr to varBind chain */
    DPI_SIMPLE_MIB, /* ptr to subtree */
    DPI_SIMPLE_STRING, /* ptr to rest of OID */
    SNMP_TYPE_DisplayString, /* value type */
    value2_len, /* length of value */
    value2_p); /* ptr to value */

if (!set_p) { /* if we failed... then */
    fDPIset(varBind_p); /* free earlier varBinds */
    return(-1); /* If it failed, return */
}

set_p = mkDPIset( /* Make DPI set packet */
    varBind_p, /* ptr to varBind chain */
    DPI_SIMPLE_MIB, /* ptr to subtree */
    DPI_SIMPLE_COUNTER32, /* ptr to rest of OID */
    SNMP_TYPE_Counter32, /* value type */
    sizeof(value3), /* length of value */
    value3); /* ptr to value */

if (!set_p) { /* if we failed... then */
    fDPIset(varBind_p); /* free earlier varBinds */
    return(-1); /* If it failed, return */
}

#ifndef EXCLUDE_SNMP_SMIv2_SUPPORT
    set_p = mkDPIset( /* Make DPI set packet */
        varBind_p, /* ptr to varBind chain */
        DPI_SIMPLE_MIB, /* ptr to subtree */
        DPI_SIMPLE_COUNTER64, /* ptr to rest of OID */
        SNMP_TYPE_Counter64, /* value type */
        sizeof(value4), /* length of value */
        value4); /* ptr to value */

if (!set_p) { /* if we failed... then */
    fDPIset(varBind_p); /* free earlier varBinds */
    return(-1); /* If it failed, return */
}
#endif /* ndef EXCLUDE_SNMP_SMIv2_SUPPORT */

packet_p = mkDPItrap( /* Make DPItrap packet */
    6, /* enterpriseSpecific */
    varBind_p, /* ptr to varBind chain */
    1, /* specific type = 1 */
    (char *)0); /* default enterpriseID */

if (!packet_p) return(-1); /* If it failed, return */

rc = DPIsend_packet_to_agent( /* send TRAP packet */
    handle, /* on this connection */
    packet_p, /* this is the packet */
    DPI_PACKET_LEN(packet_p)); /* and this is its length */

return(rc); /* return retcode */
} /* end of do_trap() */
Chapter 4. Running the sample SNMP DPI client program for version 2.0

This topic explains how to run the sample SNMP DPI client program, dpi_mvs_sample.c, installed in /usr/lpp/tcpip/samples. It can be run using the SNMP agents that support the SNMP-DPI interface as described in RFC 1592. (See Appendix E, “Related protocol specifications,” on page 743 for information about accessing RFCs.)

The sample implements a set of variables described by the DPISimple-MIB, a set of objects in the IBM Research tree (under the 1.3.6.1.4.1.2.2.1.5 object ID prefix). See “DPISimple-MIB descriptions” on page 130 for the object ID and type of each object.

Using the sample SNMP DPI client program

The dpi_mvs_sample.c program accepts the following arguments:

- ? Explains the usage
- -d n Sets the debug at level n. For levels that cause DPI API debug messages to be created, the messages are written to the syslog daemon under the daemon facility.
  
  The range is 0 (for no messages) to 2 (for the most verbose). The default value is 1 if you specify -d with no value.
  
  0 No debug messages
  1 Informational processing debug messages are written to stdout; DPI packet creation debug messages are written to the syslog daemon by the DPI API.
  2 Informational processing debug messages are written to stdout; DPI packet creation debug messages and traces of packets sent and received are written to the syslog daemon by the DPI API.
- -h hostname Specifies the host name or IP address where an SNMP DPI-capable agent is running; the default is the local host.
- -c community_name Specifies the community name for the SNMP agent that is required to get the dpiPort; the default is public.
- -ireg Specifies that the subagent should do instance-level registration of MIB objects.
- -unix Specifies that the subagent should connect to the SNMP agent using a UNIX stream socket instead of a TCP socket. You must also define INCLUDE_UNIX_DOMAIN_FOR_DPI when compiling the subagent.

Compiling and linking the dpi_mvs_sample.c source code

The dpi_mvs_sample.c program is located in /usr/lpp/tcpip/samples.

You can specify the following compile-time flags:
INCLUDE_UNIX_DOMAIN_FOR_DPI
Indicates that the sample subagent should be compiled to connect to the
agent using a UNIX Stream socket instead of a TCP connection.

MVS  Indicates that compilation is for MVS, and uses MVS-specific includes.
Some MVS/VM-specific code is compiled.

DPISimple-MIB descriptions

The following shows the MIB descriptions for DPISimple-MIB implemented by the
sample subagent.

# dpi_mvs_sample.c supports these variables as an SNMP DPI
# sample sub-agent
# it also generates enterprise specific traps via DPI with these objects

<table>
<thead>
<tr>
<th>Name</th>
<th>OID</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpiSimpleInteger</td>
<td>1.3.6.1.4.1.2.2.1.5.1.0</td>
<td>integer</td>
<td>5</td>
</tr>
<tr>
<td>dpiSimpleString</td>
<td>1.3.6.1.4.1.2.2.1.5.2.0</td>
<td>string</td>
<td>&quot;Initial String&quot;</td>
</tr>
<tr>
<td>dpiSimpleCounter32</td>
<td>1.3.6.1.4.1.2.2.1.5.3.0</td>
<td>counter32</td>
<td>1</td>
</tr>
<tr>
<td>dpiSimpleCounter64</td>
<td>1.3.6.1.4.1.2.2.1.5.4.0</td>
<td>counter64</td>
<td></td>
</tr>
<tr>
<td>X'8000000000000001'</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Of the above, only dpiSimpleString can be changed with an SNMP SET request.
Chapter 5. SNMP manager API

z/OS Communications Server provides the SNMP manager application programming interface (API) for writing SNMP managers. Application developers can use this API to build SNMP management applications that can be used to retrieve SNMP management data.

SNMP protocol

SNMP is a set of Internet Engineering Task Force (IETF) standards for network management, including a protocol, a database structure specification, a set of data objects, and controls for using the protocol. The SNMP protocol is based on the TCP/IP protocol. SNMP has evolved over many years, which has resulted in three major versions of the protocol: SNMPv1, SNMPv2c, and SNMPv3.

Elements of an SNMP model for a managed network are as follows:

Agent  This entity implements the SNMP protocol stack (sometimes called the engine). The agent's role is to receive and respond to requests using the SNMP protocol. It routes requests from managers to the appropriate subagents. It communicates with managers using the SNMP protocol. For z/OS Communications Server, the agent is the osnmpd daemon.

SubAgents  These entities are sometimes called the monitoring agents. Subagents provide the data that represents the managed objects. They communicate with the agents. An example in z/OS Communications Server is the TCP/IP subagent.

Manager  The role of the manager is to generate requests to retrieve and modify management information. The manager uses the SNMP protocol stack to receive responses from these requests and can also receive notifications, which are unsolicited events. The manager uses the SNMP protocol to communicate with the agent.

Management Information Base (MIB)  The MIB defines a set of managed objects. Each managed object has a unique identifier, which is sometimes referred to as an object identifier (OID).

SNMP Messages  Messages are exchanged between the manager and agent entities over the UDP transport of TCP/IP. This facilitates the exchange of SNMP operations. The messages, called PDUs, have formats that are defined by the SNMPv1, SNMPv2c, and SNMPv3 protocols; the types are not interoperable. The messages that are sent and received depend on the role of the entity.
The SNMP manager API overview

The SNMP manager API simplifies management application development by hiding SNMP protocol stack complexities, which enables an application to focus on management.

The SNMP manager API provides the following:

- A set of C functions and a header file that your application can use to build an SNMP manager. These functions are 31-bit DLLs and a 64-bit DLL. See “Steps for compiling and linking SNMP manager API applications” on page 159 for more information about compiling and linking the SNMP manager API.
- The ability to build, send, and receive messages for SNMPv1, SNMPv2c, and SNMPv3 using the functions provided by the API.
- Helper functions to perform operations on the decoded PDU response.

The SNMP notification API overview

The SNMP notification API is an extension of the SNMP manager API. It leverages the SNMP manager API’s functionality to send notifications to SNMP agents, the SNMP notification receivers, or both. Available notifications include informs and both Version 1 and 2 traps.

The SNMP notification API provides the following:

- A set of C functions and header files that your application can use to build an SNMP notification originator. These functions are included within the SNMP manager API libraries.
- The ability to build and send notifications for SNMPv1, SNMPv2c, and SNMPv3 using the functions provided by the API.

SNMP manager API functions

Several functions, data structures, and constants are defined in the snmpmgr.h file in the /usr/include directory. To build, send, and receive an SNMP message, your SNMP manager needs to call certain functions. After each call to one of these functions, your SNMP manager should verify that a successful return code, SNMP_MGR_RC_OK, was passed back. If an error occurs during the function call, an invalid return code is passed back. In addition to the error code, your SNMP manager API log file contains helpful information about the specific cause of the error. See “Debugging the SNMP manager API” on page 159 for information about using the debugging features.

By calling the following functions from your SNMP manager, you build the data structures necessary to send an SNMP message to an SNMP agent or subagent. The packet your SNMP manager receives as a response from the target agent can then be decoded and parsed with the defined set of helper functions. These helper functions provide your SNMP manager the ability to extract those same data structures that were used to create the outgoing packet.

A sample manager implementation, snmpSMgr.c, is found in the/usr/lpp/tcpip/samples directory.
**snmpAddVarBind** – Adds a VarBind to the SnmpVarBinds structure

```c
#include <snmpmgr.h>
int snmpAddVarBind(SnmpVarBinds **varbinds, const char *oid, const smiValue *value)
```

### snmpAddVarBind description

This function adds a VarBind structure (OID and value) to the input SnmpVarBinds structure. If space is available in the SnmpVarBinds structure, that space is used; if space is not available in the SnmpVarBinds structure, storage space is reallocated and the SnmpVarbinds structure is updated.

### snmpAddVarBind parameters

- **varbinds**
  
  This input and output parameter is a pointer to the address of the SnmpVarBinds structure into which the new VarBind structure needs to be created. If there is sufficient storage for this new VarBind structure in the array of VarBind structures pointed to from this SnmpVarBinds structure, then the output SnmpVarBinds parameter is not changed. Otherwise, storage is reallocated for this new VarBind structure and the new pointer to the beginning of the VarBind array is stored in this output parameter. This parameter is required.

- **oid**
  
  This input parameter is the string representation of the OID of the VarBind structure that is to be created. This parameter is required.

- **value**
  
  This input parameter is the address of the smiValue structure of the VarBind structure that is to be created. If this address is NULL, the created VarBind structure will contain an empty smiValue structure.

### snmpAddVarBind result

- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if either the `varbinds` parameter or the `oid` parameter is NULL
- SNMP_MGR_RC_OUT_OF_MEMORY if sufficient storage could not be allocated for the new VarBind, the OID container, or the value container
- SNMP_MGR_RC_INVALID_OID if an OID container cannot be created using this function’s `oid` parameter
- SNMP_MGR_RC_INVALID_VALUE if a value container cannot be created using this function’s `value` parameter

**Rule:** You must call the `snmpCreateVarBinds()` function before you call this function, because the `varbinds` parameter that is used on this function is returned from the `snmpCreateVarBinds()` function.

**snmpBuildPDU** – Builds an SNMP PDU

```c
#include <snmpmgr.h>
int snmpBuildPDU(SnmpPDU **pdu, const SnmpSession *snmpSession, const int pduType, const SnmpVarBinds *varbinds, const smiUINT32 non_repeaters, const smiUINT32 max_repetitions, smiUINT32 *req_id)
```
**snmpBuildPDU description**

This function creates and initializes an SNMP PDU. The PDU is built using the input parameters. The security-related information stored in the PDU is obtained from the session parameter. This function encodes the PDU using Basic Encoding Rules (BER), which are used by SNMP.

**snmpBuildPDU parameters**

*pdu*

This output parameter is a pointer to the variable where the address of the SnmpPDU structure that is created is stored. This parameter is required.

*snmpSession*

This input parameter is the address of the SNMP session for which the SNMP PDU needs to be built. This parameter is the output parameter of the snmpBuildSession() function and is required.

*pduType*

This input parameter specifies the type of the PDU. This parameter is required. The valid values are:

- **SNMP_PDU_NULL**
  - If used, the RFC-defined default PDU type SNMP_PDU_GETNEXT is built and returned
- **SNMP_PDU_GET**
- **SNMP_PDU_GETNEXT**
- **SNMP_PDU_SET**
- **SNMP_PDU_GETBULK**

*varbinds*

This input parameter is a pointer to the array of VarBind structures built using the snmpCreateVarBinds() and snmpAddVarBind() functions. This parameter is required.

*non_repeaters*

This input parameter is the number of non-repeaters for an SNMP_PDU_GETBULK request. This parameter is required, but is ignored if the *pduType* value is not SNMP_PDU_GETBULK.

*max_repetitions*

This input parameter specifies the maximum number of repetitions for an SNMP_PDU_GETBULK request. This parameter is required, but is ignored if the *pduType* value is not SNMP_PDU_GETBULK.

*req_id*

This input parameter is a pointer to the request ID to be stored in the SnmpPDU field. If the value of this parameter is greater than or equal to 0, then this value is used. Otherwise, a random request ID is generated and stored in the SnmpPDU field.

**snmpBuildPDU result**

- **SNMP_MGR_RC_OK** if successful
- **SNMP_MGR_RC_NULL_PTR** if this function's *pdu*, *snmpSession*, or *varbinds* parameter is NULL
- **SNMP_MGR_RC_INVALID_PDU_TYPE** if this function's *pduType* parameter is not valid.
• SNMP_MGR_RC_INVALID_PARAMETERS if this function’s `non_repeaters` or `max_repetitions` parameter is not valid, or if this function’s `req_id` parameter value is less than 0
• SNMP_MGR_RC_OUT_OF_MEMORY if storage could not be allocated for either the SnmpPDU structure or the encoded PDU string
• SNMP_MGR_RC_ENCODE_ERROR if an error was encountered while encoding the PDU
• SNMP_MGR_RC_INTERNAL_ERROR if an error occurred in an internal function. See your SNMP manager API’s log file for more information about the error, including the internal return code value. Your IBM service representative uses this internal return code to help solve your error.

**Rule:** You must call the `snmpBuildSession()` and `snmpCreateVarBinds()` functions before you call this function because the outputs from these functions are required as input for the `snmpBuildPDU()` function. After you are finished using the SnmpPDU created by this function, you must free the storage allocated by calling the `snmpFreePDU()` function.

**snmpBuildSession – Creates a session**

```c
#include <snmpmgr.h>
int snmpBuildSession(const SnmpConfigEntry *configEntry, SnmpSession **snmpSession, 
                     const snmpSockAddr *localAddr)
```

**snmpBuildSession description**

This function creates an SNMP session for a particular configuration entry, which represents a session with a target agent. Sessions are supported for communication using SNMPv1, SNMPv2c, and SNMPv3. This function opens a socket and returns the handle to the session, which is used as input to the `snmpBuildPDU()`, `snmpBuildV1TrapPDU()`, `snmpBuildV2TrapOrInformPDU()` and `snmpSendRequest()` functions.

**snmpBuildSession parameters**

- `configEntry`
  
  This input parameter specifies the configuration information that is to be used to create this session. The format of the SnmpConfigEntry structure is specified in the `snmpmgr.h` file. This parameter is typically obtained from the `snmpInitialize()` call, but alternatively you can manually create an SnmpConfigEntry structure. This parameter is required.

- `snmpSession`
  
  This output parameter is a pointer to the variable where the address of the SnmpSession parameter is stored. This parameter is required.

- `localAddr`
  
  This input parameter specifies the local address and port that are to be used for sending the SNMP message. If this parameter is set to NULL, the TCP/IP stack selects the source address and port.

**snmpBuildSession result**

- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function’s `configEntry` or `snmpSession` parameter is NULL
- SNMP_MGR_RC.SOCK_ERROR if there was an error encountered when opening the socket, setting the socket to be non-blocking, or attempting to bind the socket to a local address or port
SNMP_MGR_RC_OUT_OF_MEMORY if storage for the session could not be allocated.

Rule: Typically, you should call the snmpInitialize() function before you call this function, because the SnmpConfigEntry input parameter is returned from the snmpInitialize() function. However, alternatively you can manually create an SnmpConfigEntry structure. After you are finished with the SNMP session returned by this function, you must close the socket and free the storage that was allocated by calling the snmpTerminateSession() function.

**snmpCreateVarBinds – Creates a VarBind structure**

```c
#include <snmpmgr.h>
int snmpCreateVarBinds(SnmpVarBinds **varbinds, const int numVarbinds,
                       const char *oid, const smiValue *value)
```

**snmpCreateVarBinds description**
This function creates an SnmpVarBinds structure, which is used as input to the snmpBuildPDU() function. Optionally, the number of VarBind structures that are added to this VarBinds structure can be specified along with the first OID, value pair.

**snmpCreateVarBinds parameters**

`varbinds`
This output parameter is a pointer to the address of the SnmpVarBinds structure that is created and returned. This parameter is required.

`numVarbinds`
This input parameter specifies the number of variable bindings that will be part of the created SnmpVarBinds structure. This value must be greater than or equal to 0. If this value is 0, the oid and value parameters are ignored.

`oid`
This input parameter specifies the OID value for the first VarBind structure. If this parameter set to NULL, no VarBind structure is created within the SnmpVarBinds structure.

`value`
This input parameter is the address of the smiValue value of the VarBind structure that is to be created. This parameter is valid only if the oid parameter is specified. If an oid value is specified and the smiValue address is NULL, the created VarBind structure will contain an empty smiValue.

**snmpCreateVarBinds result**
- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function's `varbinds` parameter is NULL
- SNMP_MGR_RC_INVALID_PARAMETERS if this function's `numVarBinds` parameter is less than 0 or if the `oid` parameter is NULL but the `value` parameter is not NULL
- SNMP_MGR_RC_OUT_OF_MEMORY if storage cannot be allocated for the SnmpVarBinds structure

**Tip:** Use the snmpAddVarBind() function to add more VarBind structures to the SnmpVarBinds structure.
Rule: After you are finished using the SnmpVarBinds structure and the array of VarBind structures it contains, you must free the storage that was allocated for these structures by calling the snmpFreeVarBinds() function.

**snmpFreeDecodedPDU - Free the decoded PDU**

```c
#include <snmpmgr.h>
int snmpFreeDecodedPDU(SnmpDecodedPDU *decodedPDU)
```

**snmpFreeDecodedPDU description**
This function frees the storage that was allocated for the decoded PDU by the snmpSendRequest() function.

**snmpFreeDecodedPDU parameters**

*decodedPDU*
This input parameter is a pointer to the decoded PDU that is returned by the snmpSendRequest() function. This parameter is required.

**snmpFreeDecodedPDU result**
- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function’s `decodedPDU` parameter is NULL

**snmpFreeOID - Free an OID string**

```c
#include <snmpmgr.h>
int snmpFreeOID(char *oidString)
```

**snmpFreeOID description**
This function frees the storage that was allocated for the OID string by the snmpGetOID() function.

**snmpFreeOID parameters**

*oidString*
This input parameter is a pointer to the OID string that is returned by the snmpGetOID() function. This parameter is required.

**snmpFreeOID result**
- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function’s `oidString` parameter is NULL

**snmpFreePDU – Frees the resources of a PDU**

```c
#include <snmpmgr.h>
int snmpFreePDU(SnmpPDU *pdu)
```

**snmpFreePDU description**
Frees the storage for the SNMP PDU, which was allocated by the snmpBuildPDU() function.

**snmpFreePDU parameters**

*pdu*
This input parameter is the address of the PDU that is to be freed. This parameter is required.

**snmpFreePDU result**
- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function’s `pdu` parameter is NULL
snmpFreeVarBinds – Frees the VarBinds structure

```c
#include <snmpmgr.h>
int snmpFreeVarBinds(SnmpVarBinds *varbinds)
```

**snmpFreeVarBinds description**
This function frees the storage for the SnmpVarBinds structure and its array of VarBind structures that was created by the snmpCreateVarBinds() and snmpAddVarBind() functions.

**snmpFreeVarBinds parameters**

- `varbinds`
  
  This input parameter specifies the address of the SnmpVarBinds structure that is to be freed. This parameter is required.

**snmpFreeVarBinds result**

- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function’s `varbinds` parameter is NULL

---

snmpGetErrorInfo - Get the error information from the PDU response

```c
#include <snmpmgr.h>
int snmpGetErrorInfo(const SnmpDecodedPDU *decodedPDU, int *error_index, int *error_status)
```

**snmpGetErrorInfo description**
This function is used to retrieve the `error_index` and `error_status` values from a decoded response PDU.

**snmpGetErrorInfo parameters**

- `decodedPDU`
  
  This input parameter is a pointer to the decoded PDU that the error information is retrieved from. This parameter is required and is returned from a call to the snmpSendRequest() function.

- `error_index`
  
  This output parameter is a pointer to an integer where the error index from the PDU is stored. This parameter is required.

- `error_status`
  
  This output parameter is a pointer to an integer where the error status from the PDU is stored. This parameter is required.

**snmpGetErrorInfo result**

- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function’s `pdu`, `error_index`, or `error_status` parameter is NULL

**Rule:** You must call the snmpSendRequest() function before you call this function because the decodedPDU value used as an input parameter with this function is returned from the snmpSendRequest() function.
snmpGetNumberOfVarBinds – Get the number of VarBinds attached to the PDU

```c
#include <snmpmgr.h>
int snmpGetNumberOfVarBinds(const SnmpDecodedPDU *decodedPDU,
                             int *numVarbinds)
```

**snmpGetNumberOfVarBinds description**
This function retrieves the number of VarBind structures that are attached to the decoded PDU that is returned from the snmpSendRequest() function. Your SNMP manager can use this number to loop through the array of VarBind structures in the PDU, retrieving each of them by calling the snmpGetVarBind() function.

**snmpGetNumberOfVarBinds parameters**

- `decodedPDU`  
  This input parameter is a pointer to the decoded PDU from which the number of VarBind structures is retrieved. This parameter is required and is returned from a call to the snmpSendRequest() function.

- `numVarbinds`  
  This output parameter is a pointer to an integer where the number of VarBind structures is stored. This parameter is required.

**snmpGetNumberOfVarBinds result**

- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function's `decodedPDU` or `numVarbinds` parameter is NULL

**Rule:** You must call the snmpSendRequest() function before you call this function, because the `decodedPDU` input parameter used by this function is returned from the snmpSendRequest() function.

---

snmpGetOID – Get the OID from the VarBind structure

```c
#include <snmpmgr.h>
int snmpGetOid(const VarBind *varbind, char **oidString)
```

**snmpGetOID description**
This function retrieves the OID value from a VarBind structure and returns the OID value as a string.

**snmpGetOID parameters**

- `varbind`  
  This input parameter is a pointer to the VarBind structure from which the OID value is retrieved. This parameter is required and is returned from a call to the snmpGetVarBind() function.

- `oidString`  
  This output parameter is a pointer to the address of the OID value, in string format, that is to be returned. This parameter is required.

**snmpGetOID result**

- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function's `varbind` or `oidString` parameter is NULL
- SNMP_MGR_RC_OUT_OF_MEMORY if storage could not be allocated for the OID string
Rule: Your SNMP manager must call the snmpGetVarBind() function before calling this function, because the required \textit{varbind} input parameter is returned from a call to the snmpGetVarBind() function. After you finish using the \textit{oidString} value that is returned from this function, you must free the storage that was allocated for this string by calling the snmpFreeOID() function.

\textbf{snmpGetRequestId – Get the PDU’s requestId value}

\verbatim
#include <snmpmgr.h>
int snmpGetRequestId(const SnmpDecodedPDU *decodedPDU, int *req_id)
\endverbatim

\textbf{snmpGetRequestId description}
This function is used to retrieve the \textit{requestId} field from a decoded response PDU.

\textbf{snmpGetRequestId parameters}

\textit{decodedPDU}

This input parameter is a pointer to the decoded PDU that the \textit{requestId} value will be retrieved from. This parameter is required and is returned from a call to the snmpSendRequest() function.

\textit{req_id}

This output parameter is the address where the retrieved \textit{requestId} value is stored.

\textbf{snmpGetRequestId result}

- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function’s \textit{decodedPDU} or \textit{req_id} parameter is NULL

Rule: You must call the snmpSendRequest() function before you call this function because the required \textit{decodedPDU} input parameter is returned from the snmpSendRequest() function.

\textbf{snmpGetSockFd – Get the socket’s file descriptor}

\verbatim
#include <snmpmgr.h>
int snmpGetSockFd(const SnmpSession *snmpSession)
\endverbatim

\textbf{snmpGetSockFd description}
This function retrieves the socket’s file descriptor, which is stored in the session.

\textbf{snmpGetSockFd parameters}

\textit{snmpSession}

This input parameter is a pointer to the session that the socket’s file descriptor is retrieved from. This parameter is required and is returned from a call to the snmpBuildSession() function.

\textbf{snmpGetSockFd result}

Return the integer value of the socket’s file descriptor.

- SNMP_MGR_RC_NULL_PTR if the parameter is NULL

Rule: You must call the snmpBuildSession() function before you call this function because the required \textit{snmpSession} input parameter is returned by the snmpBuildSession() function.

\textbf{snmpGetValue – Get the value from the VarBind structure}

\verbatim
#include <snmpmgr.h>
int snmpGetValue(const VarBind *varbind, smiValue *value)
\endverbatim

140  z/OS V1R11.0 Comm Svr: IP Programmer’s Guide and Reference
**snmpGetValue description**
This function retrieves the value from a VarBind structure and returns the value in an smiValue structure.

**snmpGetValue parameters**
- **varbind**
  This input parameter is a pointer to the VarBind structure that the value is to be retrieved from. This parameter is required and is returned from a call to the snmpGetVarBind() function.

- **value**
  This output parameter is a pointer to the completed smiValue structure.

**snmpGetValue result**
- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function's varbind or value parameter is NULL
- SNMP_MGR_RC_INVALID_VALUE if the smi_type value of the varbind parameter is not valid

**Rule:** Your SNMP manager must call the snmpGetVarBind() function before calling this function because the required varbind input parameter is returned by a call to the snmpGetVarBind() function.

**snmpGetVarbind – Get a VarBind attached to the PDU**

```c
#include <snmpmgr.h>
int snmpGetVarBind(const SnmpDecodedPDU *decodedPDU, const int varbindNum, VarBind *varbind)
```

**snmpGetVarbind description**
Given an index into a PDU's array of VarBind structures, this function completes the contents of the input VarBind structure.

**snmpGetVarbind parameters**
- **decodedPDU**
  This input parameter is a pointer to the decoded PDU that the VarBind structure is retrieved from. This parameter is required and is returned by a call to the snmpSendRequest() function.

- **varbindNum**
  This input parameter is an index into the array of VarBind structures in the decoded response PDU.

- **varbind**
  This output parameter is a pointer to the VarBind structure that is to be completed.

**snmpGetVarbind result**
- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_INVALID_PARAMETERS if the varbindNum value is less than 0
- SNMP_MGR_RC_NULL_PTR if this function's pdu or varbind parameter is NULL

**Rule:** You must call the snmpSendRequest() function before you call this function because the required decodedPDU input parameter is returned by the snmpSendRequest() function.
### snmpInitialize – Initialize the manager environment

```c
#include <snmpmgr.h>
int snmpInitialize(const int functionsRequested,
                   const char *configFileName,
                   SnmpConfigEntry **configList)
```

#### snmpInitialize description
This function initializes the SNMP manager API. Optionally, your SNMP manager can pass a configuration file to this call. The format and syntax of this configuration file is described in “SNMP manager API configuration file” on page 156. Your SNMP manager passes a single SnmpConfigEntry structure as input to the snmpBuildSession() function.

#### snmpInitialize parameters

- **functionsRequested**
  This input parameter specifies the functions that might have to be performed by this SNMP engine. This parameter helps the SNMP engine to initialize the required components. Currently the only supported value is 0.

- **configFileName**
  This input parameter is the name of the configuration file that contains the configuration entries. A configuration entry is used on the snmpBuildSession() call. The file name can be a z/OS UNIX file name or an MVS data set name. If this parameter is set to NULL, no configuration file processing is performed. You must manually create the SnmpConfigEntry structures that are required by the other SNMP manager API functions.

- **configList**
  This output parameter is a pointer that is used to return the configuration list that has been read from the configuration file during the initialization process. This list is a linked list of SnmpConfigEntry structures, as defined in the snmpmgr.h file. Each SnmpConfigEntry value represents a target agent defined in the configuration file. This parameter is used only if the configFileName parameter is provided. Otherwise, the value NULL should be specified.

#### snmpInitialize result

- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_CONFIG_ERROR if there was an error in the configuration file
- SNMP_MGR_RC_FILE_ERROR if the configuration file could not be opened.
- SNMP_MGR_RC_NULL_PTR if this function’s configList parameter is NULL and the configFileName parameter is not NULL

- SNMP_MGR_RC_INTERNAL_ERROR if an error occurred in an internal function. See your SNMP manager API’s log file for more information about the error, including the internal return code value. Your IBM service representative uses this internal return code to help solve your error.

**Rule:** Because an SnmpConfigEntry structure is required to create an SNMP session, your SNMP manager must call this function before calling the snmpBuildSession() function. After you are finished using the list of SnmpConfigEntry structures, you must free their storage by calling the snmpTerminate() function.
snmpSendRequest – Send the snmpPDU request to an agent

```c
#include <snmpmgr.h>
int snmpSendRequest(const SnmpSession *snmpSession, const SnmpPDU *pdu,
                    const int waitInterval, SnmpDecodedPDU **decodedResponse,
                    const int receiveOnly)
```

**snmpSendRequest description**
This function sends the request PDU to the agent and waits for a response PDU. This function is a blocking function and after the function sends the request, it waits for the length of time specified by the `waitInterval` value, until it receives a response from the agent.

**snmpSendRequest parameters**

- **snmpSession**
  This input parameter is a pointer to the SNMP session that was created by the `snmpBuildSession()` function. This parameter is required.

- **pdu**
  This input parameter is a pointer to the BER-encoded PDU value that is to be sent to the agent. This parameter is required.

- **waitInterval**
  This input parameter specifies the number of seconds that this call waits to receive a return. If the response does not return within this specified time, this function returns an error code that specifies the timeout period. If this parameter's value is 0, this call waits until a response is received. This parameter's value must be non-negative.

- **decodedResponse**
  This output parameter is a pointer to the address of the decoded response PDU that is returned from the agent. The value might be NULL if the PDU is not received before the `waitInterval` period expires. This decoded PDU is used as input to several of the helper functions. This parameter is required.

- **receiveOnly**
  This input parameter specifies whether or not to retry receiving the response PDU if the first attempt timed out. The assumption is that the request has been successfully sent but that there is no response from the SNMP agent. This parameter can be set to 0 if your SNMP manager is sending a PDU, or it can be set to 1 if you want to retry receiving a response. If you specify a value of 1, a PDU is not sent to the target agent.

**snmpSendRequest result**
- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function’s `snmpSession`, `pdu`, or `decodedResponse` parameter value is NULL
- SNMP_MGR_RC_TIMEOUT if there is a timeout while waiting for a response from the target agent
- SNMP_MGR_RC_IO_ERROR if there is an error while waiting for a response from the target agent that is not the result of a timeout
- SNMP_MGR_RC_INVALID_PARAMETERS if the `waitInterval` parameter value is less than 0 or if the `receiveOnly` parameter value is not 0 or 1
- SNMP_MGR_RC_ENCODE_ERROR if an error was encountered while encoding a PDU. If your SNMP manager sends an SNMPv3 request to a target agent without the agent’s `engineID` value, this function handles the receipt of the agent’s report PDU. In doing so, a new request PDU is built and is sent to the target agent, using the agent’s `engineID` value.
- SNMP_MGR_RC_INTERNAL_ERROR if an error occurred in an internal function. See your SNMP manager API log file for more information about the error, including the internal return code value. Your IBM service representative uses this internal return code to help solve your error.

- SNMP_MGR_RC_USM_UNKNOWN_USERNAME if the target agent responded with a report PDU, indicating that a user name that was not valid was sent in your request.

- SNMP_MGR_RC_USM_UNSUPPORTED_SECLEVEL if the target agent responded with a report PDU, indicating that a security level that was not valid was sent in your request.

- SNMP_MGR_RC_USM_WRONG_DIGEST if the target agent responded with a report PDU, indicating that the message digest (created with the authentication key you defined) is not valid.

- SNMP_MGR_RC_USM_NOT_IN_WINDOW if the target agent responded with a report PDU, indicating that your request did not fall within the target agent’s accepted time range.

- SNMP_MGR_RC_USM_DECRYPTION_ERROR if the target agent responded with a report PDU, indicating that the target agent could not successfully decrypt your encrypted request.

**Rule:** Since this function requires an SNMP session and an SNMP PDU as input parameters, your SNMP manager must call the `snmpBuildSession()` and `snmpBuildPDU()` functions before calling this function. Your SNMP manager is responsible for allocating storage for the response PDU. After you are finished using this PDU, ensure that you free its storage.

**snmpSetLogFunction – Set the logging level**

```c
#include <snmpmgr.h>
int snmpSetLogFunction(SnmpLogFunc funcName)
```

**snmpSetLogFunction description**

Use this function to define an external function to be used for logging SNMP manager API messages. You should define such a function if you want the SNMP manager API log messages to be logged to the same location as other applications on your system. Your logging function must have only two parameters: an integer to define the level of the log message, and a string to define the log message itself. After you have called the `snmpSetLogFunction()` function from your SNMP manager, all of the SNMP manager API log messages are sent to your defined logging function. An example definition for your logging function is as follows:

```c
void myLogger(int logLevel, char *logMsg);
```

Based on that example, you would then need to set the parameter for this function to point to your logging function as follows:

```c
SnmpLogFunc funcName = myLogger;
rc = snmpSetLogFunc(funcName);
```

If you choose not to define your own logging function, you can log messages to a file defined by the SNMP_MGR_LOG_FILE environment variable, or have messages logged to syslog (the default). For more information about your logging options, see “Debugging the SNMP manager API” on page 159.
snmpSetLogFunction parameters

*funcName*

This input parameter specifies the SNMP manager’s preferred logging function.

snmpSetLogFunction result

- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if the *funcName* parameter is NULL

snmpSetLogLevel – Set the logging level

```c
#include <snmpmgr.h>
int snmpSetLogLevel(const int logLevel)
```

snmpSetLogLevel description

This function sets the log level for messages. See “Debugging the SNMP manager API” on page 159 for more information about logging.

snmpSetLogLevel parameters

*logLevel*

This input parameter specifies the preferred message logging level. Valid values for the *logLevel* parameter, as defined in snmpmgr.h, are defined as follows:

- SNMP_LOG_NONE (0)
- SNMP_LOG_ERROR (1)
- SNMP_LOG_TRACE (2)
- SNMP_LOG_DUMP (4)
- SNMP_LOG_ALL (7)

snmpSetLogLevel result

- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_INVALID_PARAMETERS if the *logLevel* parameter is not valid

Guideline: Because the SNMP_MGR_LOG_LEVEL environment variable is not read until the snmpInitialize() function is called, you should call this function prior to calling the snmpInitialize() function, to make sure that all possible messages are successfully logged.

Rule: The value of the SNMP_MGR_LOG_LEVEL environment variable, when set, cannot be changed by calling this function with a new log level. You must unset the environment variable for this function to operate correctly.

snmpSetRequestId – Set the PDU’s requestId value

```c
#include <snmpmgr.h>
int snmpSetRequestId(const SnmpSession *snmpSession, SnmpPDU *pdu, const int req_id)
```

snmpSetRequestId description

Use this function to set the requestId field in an SNMP PDU. This function rebuilds the encoded PDU that is returned by the snmpBuildPDU() function.

snmpSetRequestId parameters

*snmpSession*

This input parameter is a pointer to the SNMP session information
required to reconstruct a PDU with the new request ID. This input is required and is returned by the snmpBuildSession() function.

*pdu*  This input parameter is a pointer to the encoded PDU where the request ID is to be set. This parameter is required and is returned by the snmpBuildPDU() function.

*req_id*  This input parameter specifies the integer requestId value to set in the PDU. The value of this parameter must be greater than or equal to 0.

**snmpSetRequestId result**
- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if either the snmpSession or pdu pointer is NULL
- SNMP_MGR_RC_INVALID_PARAMETERS if the *req_id* parameter is less than 0
- SNMP_MGR_RC_ENCODE_ERROR if an error was encountered while encoding the new PDU
- SNMP_MGR_RC_INTERNAL_ERROR if an error occurred in an internal function. See your SNMP manager API log file for more information about the error, including the internal return code value. Your IBM service representative uses this internal return code to help solve your error.

**Rules:**
- Because this function requires an SNMP session and an SnmpPDU value as input, your SNMP manager must call the snmpBuildSession() and snmpBuildPDU() functions before calling this function.
- SNMPv1 traps do not use a request ID. Therefore, your SNMP manager must not call this function for a PDU type SNMP_PDU_TRAPV1

**snmpTerminate – Release the resources**

```c
#include <snmpmgr.h>
int snmpTerminate(SnmpConfigEntry *headEntry)
```

**snmpTerminate description**
This function releases the resources that were allocated by the snmpInitialize() function.

**snmpTerminate parameters**

- **headEntry**
  
  This input parameter points to the first entry in the linked list of SnmpConfigEntry objects returned by the snmpInitialize() function.

**snmpTerminate result**
- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function’s *headEntry* parameter is NULL

**snmpTerminateSession – Terminate a session**

```c
#include <snmpmgr.h>
int snmpTerminateSession(SnmpSession *snmpSession)
```

**snmpTerminateSession description**
This function terminates an SNMP session and releases all of the resources held by that session.
**snmpTerminateSession parameters**

*snmpSession*

This input parameter is the address of the session that was created by the snmpBuildSession() call.

**snmpTerminateSession result**

- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function’s *snmpSession* parameter is NULL
- SNMP_MGR_RC_SOCK_ERROR if the socket that was created by the snmpBuildSession() function cannot be closed

**Tip:** After a session is terminated, it cannot be used again.

**snmpValueCreateBits – Create an smiValue of type Bits**

```c
#include <snmpmgr.h>
int snmpValueCreateBits(smiValue *value, char *inBits, smiUINT32 inLen)
```

**snmpValueCreateBits description**

This function completes an smiValue structure of type SNMP_SYNTAX_BITS, based on the input parameter values. This smiValue structure can then be used as input to the snmpCreateVarBinds() or snmpAddVarBind() function.

**snmpValueCreateBits parameters**

- **value** This output parameter is the address of the smiValue structure to be completed. This parameter is required.
- **inBits** This input parameter is a pointer to the value to be stored in the smiValue structure. This parameter is required.
- **inLen** This input parameter is the length, in bytes, of the *inBits* parameter value. This parameter is required.

**snmpValueCreateBits result**

- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function’s *value* or *inBits* parameter is NULL
- SNMP_MGR_RC_OUT_OF_MEMORY if storage could not be allocated for the string representation of the value in the smiValue structure

**snmpValueCreateCounter32 – Create an smiValue of type Counter32**

```c
#include <snmpmgr.h>
int snmpValueCreateCounter32(smiValue *value, smiUINT32 inInt)
```

**snmpValueCreateCounter32 description**

This function completes an smiValue structure of type SNMP_SYNTAX_CNTR32, based on the input integer. This smiValue structure can then be used as input to the snmpCreateVarBinds() or snmpAddVarBind() function.

**snmpValueCreateCounter32 parameters**

- **value** This output parameter is the address of the smiValue structure to be completed. This parameter is required.
- **inInt** This input parameter is the unsigned integer to be stored in the smiValue structure. This parameter is required.
snmpValueCreateCounter32 result
- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function's value parameter is NULL

snmpValueCreateCounter64 – Create an smiValue of type Counter64

#include <snmpmgr.h>
int snmpValueCreateCounter64(smiValue *value, smiUINT32 hiPart, smiUINT32 loPart)

snmpValueCreateCounter64 description
This function completes an smiValue structure of type SNMP_SYNTAX_CNTR64, based on the input integers. This smiValue structure can then be used as input to the snmpCreateVarBinds() or snmpAddVarBind() function.

snmpValueCreateCounter64 parameters
value  This output parameter is the address of the smiValue structure to be completed. This parameter is required.
hiPart This input parameter is the high-order 32 bits to be stored in the smiValue structure. This parameter is required.
loPart This input parameter is the low-order 32 bits to be stored in the smiValue structure. This parameter is required.

snmpValueCreateCounter64 result
- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function's value parameter is NULL

snmpValueCreateGauge32 – Create an smiValue of type Gauge32

#include <snmpmgr.h>
int snmpValueCreateGauge32(smiValue *value, smiUINT32 inInt)

snmpValueCreateGauge32 description
This function completes an smiValue structure of type SNMP_SYNTAX_Gauge32, based on the input integer. This smiValue structure can then be used as input to the snmpCreateVarBinds() or snmpAddVarBind() function.

snmpValueCreateGauge32 parameters
value  This output parameter is the address of the smiValue structure to be completed. This parameter is required.
inInt  This input parameter is the unsigned integer to be stored in the smiValue structure. This parameter is required.

snmpValueCreateGauge32 result
- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function's value parameter is NULL

snmpValueCreateInteger – Create an smiValue of type Integer

#include <snmpmgr.h>
int snmpValueCreateInteger(smiValue *value, smiINT32 inInt)
**snmpValueCreateInteger**

**description**
This function completes an smiValue structure of type SNMP_SYNTAX_INT, based on the input integer. This smiValue structure can then be used as input to the snmpCreateVarBinds() or snmpAddVarBind() function.

**parameters**
- **value**  This output parameter is the address of the smiValue structure to be completed. This parameter is required.
- **inInt**  This input parameter is the integer to be stored in the smiValue structure. This parameter is required.

**result**
- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function’s **value** parameter is NULL
- SNMP_MGR_RC_INVALID_PARAMETERS if this function’s **inInt** parameter value is less than 0

---

**snmpValueCreateInteger32** – Create an smiValue of type Integer32

```c
#include <snmpmgr.h>
int snmpValueCreateInteger32(smiValue *value, smiINT32 inInt)
```

**description**
This function completes an smiValue structure of type SNMP_SYNTAX_INT32, based on the input integer. This smiValue structure can then be used as input to the snmpCreateVarBinds() or snmpAddVarBind() function.

**parameters**
- **value**  This output parameter is the address of the smiValue structure to be completed. This parameter is required.
- **inInt**  This input parameter is the integer to be stored in the smiValue structure. This parameter is required.

**result**
- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function’s **value** parameter is NULL
- SNMP_MGR_RC_INVALID_PARAMETERS if this function’s **inInt** parameter value is less than 0

---

**snmpValueCreateIPAddr** – Create an smiValue of type IPAddr

```c
#include <snmpmgr.h>
int snmpValueCreateIPAddr(smiValue *value, char *inAddr, smiUINT32 inLen)
```

**description**
This function completes an smiValue structure of type SNMP_SYNTAX_IPADDR, based on the input parameters. This smiValue structure can then be used as input to the snmpCreateVarBinds() or snmpAddVarBind() function.

**parameters**
- **value**  This output parameter is the address of the smiValue structure to be completed. This parameter is required.
- **inAddr**  This input parameter is the ASCII string representation of the IP address.
- **inLen**  This input parameter is the length of the IP address in bytes.

---

Chapter 5. SNMP manager API 149
inAddr  This input parameter is a pointer to the value to be stored in the smiValue structure. This parameter is required.

inLen   This input parameter is the length, in bytes, of the inAddr parameter value. This parameter is required.

snmpValueCreateIPAddr result
- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function's value parameter or inAddr parameter is NULL
- SNMP_MGR_RC_OUT_OF_MEMORY if storage could not be allocated for the string representation of the value in the smiValue structure

snmpValueCreateNSApAddr – Create an smiValue of type NSApAddr

#include <snmpmgr.h>
int snmpValueCreateNSApAddr(smiValue *value, char *inAddr, smiUINT32 inLen)

snmpValueCreateNSApAddr description
This function completes an smiValue structure of type SNMP_SYNTAX_NSAPADDR, based on the input parameters. This smiValue structure can then be used as input to the snmpCreateVarBinds() or snmpAddVarBind() function.

snmpValueCreateNSApAddr parameters
value   This output parameter is the address of the smiValue structure to be completed. This parameter is required.

inAddr  This input parameter is a pointer to the value to be stored in the smiValue structure, as a binary network address. This parameter is required.

inLen   This input parameter is the length, in bytes, of the inAddr parameter value. This parameter is required.

snmpValueCreateNSApAddr result
- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function's value parameter or inAddr parameter is NULL
- SNMP_MGR_RC_OUT_OF_MEMORY if storage could not be allocated for the string representation of the value in the smiValue structure

snmpValueCreateNull – Create an smiValue of type Null

#include <snmpmgr.h>
int snmpValueCreateNull(smiValue *value)

snmpValueCreateNull description
This function completes an smiValue structure of type SNMP_SYNTAX_NULL, based on the input parameters. This smiValue structure can then be used as input to the snmpCreateVarBinds() or snmpAddVarBind() function.

snmpValueCreateNull parameters
value   This output parameter is the address of the smiValue structure to be completed. This parameter is required.


**snmpValueCreateNull result**
- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function’s `value` parameter is NULL

**snmpValueCreateOctet – Create an smiValue of type Octet**

```c
#include <snmpmgr.h>
int snmpValueCreateOctet(smiValue *value, char *inOctet, smiUINT32 inLen)
```

**snmpValueCreateOctet description**
This function completes an smiValue structure of type SNMP_SYNTAX_OCTET, based on the input parameters. This smiValue structure can then be used as input to the `snmpCreateVarBinds()` or `snmpAddVarBind()` function.

**snmpValueCreateOctet parameters**
- `value` This output parameter is the address of the smiValue structure to be completed. This parameter is required.
- `inOctet` This input parameter is a pointer to the value to be stored in the smiValue structure. This parameter is required.
- `inLen` This input parameter is the length, in bytes, of the `inOctet` parameter value. This parameter is required.

**snmpValueCreateOctet result**
- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function’s `value` parameter or `inOctet` parameter is NULL
- SNMP_MGR_RC_OUT_OF_MEMORY if storage could not be allocated for the string representation of the value in the smiValue structure

**snmpValueCreateOID – Create an smiValue of type OID**

```c
#include <snmpmgr.h>
int snmpValueCreateOID(smiValue *value, char *inOID, smiUINT32 inLen)
```

**snmpValueCreateOID description**
This function completes an smiValue structure of type SNMP_SYNTAX_OID, based on the input parameters. This smiValue structure can then be used as input to the `snmpCreateVarBinds()` or `snmpAddVarBind()` function.

**snmpValueCreateOID parameters**
- `value` This output parameter is the address of the smiValue structure to be completed. This parameter is required.
- `inOID` This input parameter is a pointer to the value to be stored in the smiValue structure. This parameter is required.
- `inLen` This input parameter is the length, in bytes, of the `inOID` parameter value. This parameter is required.

**snmpValueCreateOID result**
- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function’s `value` parameter or `inOID` parameter is NULL
snmpValueCreateOpaque – Create an smiValue of type Opaque

```c
#include <snmpmgr.h>
int snmpValueCreateOpaque(smiValue *value, char *inOpaque, smiUINT32 inLen)
```

**snmpValueCreateOpaque description**
This function completes an smiValue structure of type SNMP_SYNTAX_OPAQUE, based on the input parameters. This smiValue structure can then be used as input to the snmpCreateVarBinds() or snmpAddVarBind() function.

**snmpValueCreateOpaque parameters**
- **value**
  This output parameter is the address of the smiValue structure to be completed. This parameter is required.
- **inOpaque**
  This input parameter is a pointer to the value to be stored in the smiValue structure. This parameter is required.
- **inLen**
  This input parameter is the length, in bytes, of the `inOpaque` parameter value. This parameter is required.

**snmpValueCreateOpaque result**
- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function’s `value` parameter or `inOpaque` parameter is NULL
- SNMP_MGR_RC_OUT_OF_MEMORY if storage could not be allocated in the smiValue structure for the string representation of the value

snmpValueCreateTimerTicks – Create an smiValue of type TimerTicks

```c
#include <snmpmgr.h>
int snmpValueCreateTimerTicks(smiValue *value, smiUINT32 inInt)
```

**snmpValueCreateTimerTicks description**
This function completes an smiValue structure of type SNMP_SYNTAX_TIMETICKS, based on the input integer. This smiValue structure can then be used as input to the snmpCreateVarBinds() or snmpAddVarBind() function.

**snmpValueCreateTimerTicks parameters**
- **value**
  This output parameter is the address of the smiValue structure to be completed. This parameter is required.
- **inInt**
  This input parameter is the unsigned integer to be stored in the smiValue structure. This parameter is required.

**snmpValueCreateTimerTicks result**
- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function’s `value` parameter is NULL
snmpValueCreateUnsignedInteger32 – Create an smiValue of type UnsignedInteger32

```c
#include <snmpmgr.h>
int snmpValueCreateUnsignedInteger32(smiValue *value, smiUINT32 inInt)
```

**snmpValueCreateUnsignedInteger32 description**
This function completes an smiValue structure of type SNMP_SYNTAX_UINT32, based on the input integer. This smiValue structure can then be used as input to the snmpCreateVarBinds() or snmpAddVarBind() function.

**snmpValueCreateUnsignedInteger32 parameters**
- `value` This output parameter is the address of the smiValue structure to be completed. This parameter is required.
- `inInt` This input parameter is the unsigned integer to be stored in the smiValue structure. This parameter is required.

**snmpValueCreateUnsignedInteger32 result**
- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function’s `value` parameter is NULL

**SNMP notification API functions**

The SNMP manager API can also be used to send notifications. When sending a notification, you can build the PDU using either the snmpBuildV1TrapPDU() or snmpBuildV2TrapOrInformPDU() function, which are defined in the snmpntfy.h file. Use the following SNMP manager API functions to use the SNMP notification API. See “SNMP manager API functions” on page 132 for descriptions of these functions.

- snmpAddVarBind
- snmpBuildSession
- snmpCreateVarBinds
- snmpFreePDU
- snmpFreeVarBinds
- snmpInitialize
- snmpSendRequest
- snmpSetLogFunction
- snmpSetLogLevel
- snmpSetRequestId
- snmpTerminate
- snmpTerminateSession

After each call to one of these functions, your SNMP manager should verify that a successful return code, SNMP_MGR_RC_OK, was returned. If an error occurred during the function call, an invalid return code is returned. In addition to the error code, your SNMP manager API log file contains information about the specific cause of the error. See “Debugging the SNMP manager API” on page 159 for more information about how to use the debugging features.

By calling the following functions from your SNMP manager, you build the data structures necessary to send an SNMP notification to an SNMP agent or subagent.
snmpBuildV1TrapPDU – Builds an SNMP V1 trap PDU

```c
#include <snmpntfy.h>
int snmpBuildV1TrapPDU(SnmpPDU **pdu, const SnmpSession *snmpSession,
    const SnmpVarBinds *varbinds, char *ent_p,
    char *local_ip, int generic, int specific,
    unsigned int timestamp);
```

**snmpBuildV1TrapPDU description**

This function creates and initializes an SNMP PDU of type SNMP_PDU_TRAPV1. The PDU is built using the input parameters. The security-related information stored in the PDU is obtained from the `session` parameter. This function encodes the PDU using Basic Encoding Rules (BER), which are used by SNMP.

**snmpBuildV1TrapPDU parameters**

- **pdu**
  - This output parameter is a pointer to the variable into which to store the address of the SnmpPDU structure that is created. This parameter is required.

- **snmpSession**
  - This input parameter is the address of the SNMP session for which the SNMP PDU needs to be built. This parameter is the output parameter of the `snmpBuildSession()` function and is required.

- **varbinds**
  - This input parameter is a pointer to the array of VarBind structures that was built using the `snmpCreateVarBinds()` and `snmpAddVarBind()` functions. This parameter is required.

- **ent_p**
  - This input parameter is a pointer to the enterprise OID that generates the trap. This parameter is required.

- **local_ip**
  - This input parameter is the address of the system that generates the trap (a character string). This parameter is required.

- **generic**
  - This input parameter indicates the generic trap type. This parameter is required.

- **specific**
  - This input parameter indicates the specific trap type. This parameter is required.

- **timestamp**
  - This input parameter specifies the amount of time that has elapsed between the last network re-initialization and generation of the trap. This parameter is required.

**snmpBuildV1TrapPDU result**

- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function's `pdu`, `snmpSession`, `varbinds`, `ent_p`, or `local_ip` parameter is NULL
- SNMP_MGR_RC_INVALID_PARAMETERS if one of the following is true for this function:
  - The `generic` parameter value is not in the range 0 - 6
  - The `generic` parameter value is in the range 0 - 5 and the `specific` parameter value is not 0
  - The `specific` parameter value is less than 0
- SNMP_MGR_RC_OUT_OF_MEMORY if storage could not be allocated for either the SnmpPDU structure or the encoded PDU string
- SNMP_MGR_RC_ENCODE_ERROR if an error was encountered while encoding the PDU
- SNMP_MGR_RC_INTERNAL_ERROR if an error occurred in an internal function. See your SNMP manager API log file for more information about the error, including the internal return code value. Your IBM service representative uses this internal return code to help solve your error

**Rule:** The snmpBuildSession() and snmpCreateVarBinds() functions must have been successfully called before calling this function, because output from those functions is required as input for this function. After you are finished using the SnmpPDU structure created by this function, you must free the storage that was allocated by calling the snmpFreePDU() function.

**snmpBuildV2TrapOrInformPDU – Builds a SNMP V2 trap or inform PDU**

```c
#include <snmpntfy.h>

int snmpBuildPDU(SnmpPDU **pdu, const SnmpSession *snmpSession ,
              const int pduType, const SnmpVarBinds *varbinds,
              smiUINT32 *req_id)
```

**snmpBuildV2TrapOrInformPDU description**

This function creates and initializes either a v2 trap or an inform SNMP PDU. The PDU is built using the input parameters. The security-related information stored in the PDU is obtained from the session parameter. This function encodes the PDU using Basic Encoding Rules (BER), which are used by SNMP.

**snmpBuildV2TrapOrInformPDU parameters**

- **pdu** This output parameter is a pointer to the variable into which to store the address of the SnmpPDU structure that is created. This parameter is required.

- **snmpSession** This input parameter is the address of the SNMP session for which the SNMP PDU needs to be built. This parameter is the output parameter of the snmpBuildSession() function and is required.

- **pduType** This input parameter specifies the type of the PDU. This parameter is required. The valid values are:
  - SNMP_PDU_TRAPV2
  - SNMP_PDU_INFORM

- **varbinds** This input parameter is a pointer to the array of VarBind structures built using the snmpCreateVarBinds() and snmpAddVarBind() functions. This parameter is required.

- **req_id** This input parameter is a pointer to the request ID to be stored in the SnmpPDU structure. If the value of this parameter is greater than or equal to 0, this value is used. Otherwise, a random request ID is generated and stored in the SnmpPDU structure.

**snmpBuildV2TrapOrInformPDU result**

- SNMP_MGR_RC_OK if successful
- SNMP_MGR_RC_NULL_PTR if this function's pdu, snmpSession, or varbinds parameter is NULL
- SNMP_MGR_RC_INVALID_PDU_TYPE if this function’s pduType parameter is not valid
- SNMP_MGR_RC_OUT_OF_MEMORY if storage could not be allocated for either the SnmpPDU structure or for the encoded PDU string
- SNMP_MGR_RC_ENCODE_ERROR if an error was encountered while encoding the PDU
- SNMP_MGR_RC_INTERNAL_ERROR if an error occurred in an internal function. See your SNMP manager API log file for more information about the error, including the internal return code value. Your IBM service representative uses this internal return code to help solve your error.

Rules:
- The snmpBuildSession() and snmpCreateVarBinds() functions must have been successfully called before calling this function, because the output from the snmpBuildSession() and snmpCreateVarBinds() functions is required as input for this function. After you are finished using the SnmpPDU structure created by this function, you must free the storage allocated by calling the snmpFreePDU() function.
- SNMPv2 traps and informs require the following two VarBind structures in the VarBind array:
  - sysUpTime
  - snmpTrapOid

Your SNMP manager must call the snmpCreateVarBinds() and snmpAddVarBind() functions to create these VarBind structures, in this order, before calling this function.

SNMP manager API configuration file

You can create a configuration information file for use with the snmpInitialize() function of the SNMP manager API. The configuration statements can be defined and stored in a z/OS UNIX file or an MVS data set. See "snmpInitialize – Initialize the manager environment" on page 142 for more information about using this file. Following is a sample configuration file.

```plaintext
# Format of entries (SNMPv1 and SNMPv2c):
#
# targetAddr targetPort version communityName
#
# Community-based security (SNMPv1 and SNMPv2c)
#---------------------------------------------------------------
9.1.1.1 161 snmpv1 -
9.1.1.2 - snmpv2c public
#

# Format of entries (SNMPv3):
#
# targetAddr targetPort version userName password secLevel authProto authKey privProto privKey
#
# User-based security (SNMPv3)
#---------------------------------------------------------------
9.8.0.1 1161 snmpv3 userid - AuthPriv HMAC-SHA 7236726349623ADCDEFF32223ACC DES 32472653666765764674EEAEED0DB
```
SNMP manager API statement syntax

**targetAddr**
IP address (IPv4 dotted decimal format or IPv6 colon hexadecimal format) of the node of the target agent (maximum 19 characters). There is no default value.

**targetPort**
Port number of the target agent, in the range 1 - 65535. Use a dash (-) to indicate the default value (161).

**version**
Specifies the administrative model that is supported by the target agent. Valid values are:
- `snmpv1`
  Community-based SNMPV1 security
- `snmpv2c`
  Community-based SNMPV2 security
- `snmpv3`
  User-based SNMPV3 security

There is no default value.

**communityName**
Specifies the community name for community-based security (SNMPV1 or SNMPV2c). A dash (-) can be used to indicate the default value (public).

**userName**
Specifies the security name of the principal using this configuration file entry. For user-based security, this is the user name. The user must be defined at the target agent. This field is ignored unless SNMPv3 is specified for the version keyword. A valid value is a user name that is 1 - 32 characters in length. There is no default value.

**password**
Specifies the password that is to be used in generating the authentication and privacy keys for this user. If a password is specified, it is used to automatically generate any needed keys and the authKey and privKey fields are ignored. This field is ignored unless SNMPv3 is specified for the version keyword. If you do not want to specify a password, set the field to a single dash (-). (The minimum number of characters that you can specify is eight, and the maximum number is 64 characters.)

*Rule:* If you define a password in your configuration entry, the authKey and privKey fields must be set to a dash (-), which specifies no key.

*Guideline:* You should not use the password instead of keys in this configuration file, because using keys is more secure than storing passwords in this file.

*Tip:* To use a different password for authentication and privacy, you can overwrite the authPassword or privPassword field in the SnmpConfigEntry structure. By default, both of these password fields contain the value defined in the configuration file.

**secLevel**
Specifies the security level to be used when communicating with the target SNMP agent when this entry is used. This field is ignored unless SNMPv3 is specified for the version keyword. Valid values are:
- noAuthNoPriv or none to indicate that no authentication or privacy is requested
- AuthNoPriv or auth to indicate that authentication is requested but privacy is not requested
- AuthPriv or priv to indicate that both authentication and privacy are requested
- Dash (-) to indicate the default value (noAuthNoPriv)

**authProto**

SNMP authentication protocol to be used when communicating with the target SNMP agent when this entry is used. This field is ignored unless SNMPv3 is specified for the version keyword. Valid values are:
- HMAC-MD5
- HMAC-SHA
- A single dash (-) for no authentication

**authKey**

Specifies the SNMP authentication key to be used when communicating with the target SNMP agent when this entry is used. This key must be the non-localized key. This field is ignored if the password keyword is used. This field is ignored unless SNMPv3 is specified for the version keyword and a non-default value is specified for the authProto parameter. Valid values are:
- 16 bytes (32 hexadecimal digits) when the authProto value is HMAC-MD5
- 20 bytes (40 hexadecimal digits) when the authProto value is HMAC-SHA
- A dash (-) indicates the default value, which is no key

**privProto**

Specifies the SNMP privacy protocol to be used when communicating with the target SNMP agent when this entry is used. This field is ignored unless SNMPv3 is specified for the version keyword. Valid values are:
- DES for CBC-DES
- A dash (-) to indicate the default value, which is no privacy

**privKey**

Specifies the SNMP privacy key to be used when communicating with the target SNMP agent when this entry is used. This key must be the non-localized key. This field is ignored if the password keyword is used. The privacy and authentication keys are assumed to have been generated using the same authentication protocol (for example, both with HMAC-MD5 or both with HMAC-SHA). This field is ignored unless the value snmpv3 is specified for the admin keyword and a non-default value is specified for the privProto parameter. Valid values are:
- 16 bytes (32 hexadecimal digits) when the authProto value is HMAC-MD5
- 20 bytes (40 hexadecimal digits) when the authProto value is HMAC-SHA
- A dash (-) to indicate the default value (no key)

**SNMP manager API general rules**

- All parameters for an entry must be contained on one line in the configuration file.
Steps for compiling and linking SNMP manager API applications

To use the SNMP manager API or SNMP notification API, perform the following steps:

1. Write your SNMP manager source application.
   Make sure to include the `<snmpmgr.h>` header file, which is available in the /
   /usr/include directory. If your SNMP manager will be sending notifications,
   make sure to include the SNMP notification header, `<snmpntfy.h>`, which is
   also available in the /usr/include directory.

2. Compile your application using the DLL compiler option. See the z/OS XL
   C++ User's Guide for more information about how to specify compiler
   options.

3. Include the SNMP manager API definition side deck (/usr/lib/EZBSNMPA.x,
   /usr/lib/EZBSNMPX.x, or /usr/lib/EZBSNMP6.x) when prelinking or
   binding the application.

Running your SNMP manager API application

The SNMP API provides the following DLLs for running your application:

- 31-bit DLL EZBSNMPA
- 31-bit DLL EZBSNMPX for applications compiled with XPLINK
- 64-bit DLL EZBSNMP6 for applications compiled with XPLINK

These DLLs are included in the SYS1.SIEALNKE data set and in z/OS UNIX, in
the /usr/lib directory. Ensure that the SYS1.SIEALNKE file is in your LNKLST
statement before running your application.

Debugging the SNMP manager API

You can debug problems with the SNMP manager API in two ways:

For manager applications, call the snmpSetLogLevel() routine using the following
debug levels:

<table>
<thead>
<tr>
<th>Debug level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMP_LOG_NONE (0)</td>
<td>No logging</td>
</tr>
<tr>
<td>SNMP_LOG_ERROR (1)</td>
<td>Log only errors</td>
</tr>
<tr>
<td>SNMP_LOG_TRACE (2)</td>
<td>Trace function upon entry and exit</td>
</tr>
<tr>
<td>SNMP_LOG_DUMP(4)</td>
<td>Dump the session object</td>
</tr>
<tr>
<td>SNMP_LOG_ALL (7)</td>
<td>Log all levels</td>
</tr>
</tbody>
</table>

When calling the snmpSetLogLevel() routine, you can use multiple trace levels by
specifying either the numerical value of each desired item, or the logical name. For example, for both SNMP_LOG_ERROR (1) and SNMP_LOG_TRACE (2), you could issue one of the following:

- `snmpSetLogLevel(SNMP_LOG_ERROR + SNMP_LOG_TRACE)`
- `snmpSetLogLevel(3)`

Set the SNMP_MGR_LOG_LEVEL debuglevel environment variable to turn on debugging. This environment variable is read when the snmpInitialize() function is called. You can set multiple trace levels by adding the levels that you want to trace.

The SNMP manager API attempts to read the SNMP_MGR_LOG_LEVEL environment variable in the snmpInitialize() function. If your SNMP manager calls the snmpSetLogLevel() function before calling the snmpInitialize() function, all API-generated trace messages in the snmpInitialize() function are logged. If not, logging begins inside the snmpInitialize() function after the environment variable is validated. After the environment variable has been read in the snmpInitialize() function, the value of the environment variable SNMP_MGR_LOG_LEVEL (if set) is used as the log level for your SNMP manager application. After this point, calls to the snmpSetLogLevel() function do not change the log level. You must unset the environment variable for this function to operate correctly.

The SNMP manager API, by default, uses the SYSLOG daemon, and uses the current SYSLOG configuration for the output location. However, by declaring the SNMP_MGR_LOG_FILE environment variable, the SNMP manager can also send the output stream to an individual file (in addition to SYSLOG or another logging function). Use the SNMP_MGR_LOG_FILE environment variable to test your application. If the environment variable is declared, the log messages generated by the SNMP manager API are sent to both the file specified by the environment variable and either SYSLOG or your own logging function.

You can use your own logging function, rather than the SYSLOG default. Using your own logging function has the benefit of providing the log messages from the SNMP manager API, and your application logging. Your logging function overrides the default, SYSLOG logging function, which means that after you have enabled your logging function in the SNMP manager API, log messages are no longer sent to SYSLOG. If you have declared the SNMP_MGR_LOG_FILE environment variable, log messages are sent to both your logging function and to the file specified by the environment variable.

To use your own logging function, your SNMP manager needs to pass the name of the function as a parameter to the SNMP manager API's snmpSetLogFunction() routine. Every message produced by the SNMP manager API is then sent to your logging function, along with the integer that specifies the level of the log message (for example, SNMP_LOG TRACE). Your function definition must be defined as follows so that the SNMP manager API calls it correctly:

```c
void myLogger(int logLevel, char *logMsg);
```

**Sample SNMP manager API source code**

You can find a sample SNMP manager implementation, snmpSMgr.c, in the `usr/lpp/tcpip/samples` directory.
Chapter 6. Resource Reservation Setup Protocol API (RAPI)

The z/OS UNIX RSVP agent includes an application programming interface (API) for the Resource ReSerVation Protocol (RSVP), known as RAPI.

The RAPI interface is one realization of the generic API contained in the RSVP functional specification (see RFC 2205; see Appendix E, “Related protocol specifications,” on page 743 for information about accessing RFCs). RSVP describes a resource reservation setup protocol designed for an integrated services internet. RSVP provides receiver-initiated setup of resource reservations for multicast or unicast data flows. See the RSVP applicability statement in reference RFC 2210 for more information.

The RAPI interface is a set of C language bindings whose calls are defined in this topic. Applications use RAPI to request enhanced Quality of Service (QoS). The RSVP agent then uses the RSVP protocol to propagate the QoS request through the routers along the paths for the data flow. Each router can accept or deny the request, depending upon the availability of resources. In the case of failure, the RSVP agent returns the decision to the requesting application by way of RAPI.

RSVP is a receiver-oriented signaling protocol that enables applications to request Quality of Service on an IP network. The types of Quality of Service requested by those applications are defined by Integrated Services. RSVP signaling applies to simplex unicast or multicast data flows. Although RSVP distinguishes senders from receivers, the same application can act in both roles.

RSVP assigns QoS to specific IP data flows that can be either multipoint-to-multipoint or point-to-point data flows, known as sessions. A session is defined by a particular transport protocol, IP destination address, and destination port. To receive data packets for a particular multicast session, an application must join the corresponding IP multicast group.

A data source, or sender, is defined by an IP source address and a source port. A given session can have multiple senders (S1, S2, ... Sn), and if the destination is a multicast address, multiple receivers (R1, R2, ... Rn).

Under RSVP, QoS requests are made by the data receivers. A QoS request contains a flowspec, together with a filter spec. The flowspec includes an Rspec, which defines the desired QoS and is used to control the packet scheduling mechanism in the router or host, and also a Tspec, which defines the traffic expected by the receiver. The filter spec controls packet classification to determine which sender data packets receive the corresponding QoS.

The detailed manner in which reservations from different receivers are shared in the internet is controlled by a reservation parameter known as the reservation style. The RSVP Functional Specification (see RFC 2205; see Appendix E, “Related protocol specifications,” on page 743 for information about accessing RFCs) contains a definition and explanation of the different reservation styles. Also see the z/OS Communications Server: IP Configuration Guide and z/OS Communications Server: IP Diagnosis Guide for more information about the RSVP agent.
API outline

Using the RAPI interface, an application uses the rapi_session() call to define an API session for sending a single simplex data flow or receiving such a data flow. The rapi_sender() call can then be used to register as a data sender, and the rapi_reserve() call can be used to make a QoS reservation as a data receiver.

The rapi_sender() or rapi_reserve() calls can be repeated with different parameters to dynamically modify the state at any time or they can be issued in null forms that retract the corresponding registration. The application can call rapi_release() to close the session and delete all of its resource reservations.

A single API session, defined by a single rapi_session() call, can define only one sender at a time. More than one API session can be established for the same RSVP session. For example, if an application sends multiple UDP data flows that are distinguished by source port, it will call rapi_session() and rapi_sender() separately for each of these flows.

The rapi_session() call allows the application to specify an upcall (or callback) routine that is invoked to signal RSVP state change and error events. There are five types of events:

- RAPI_PATH_EVENT signals the arrival or change of path state.
- RAPI_RESV_EVENT signals the arrival or change of reservation state.
- RAPI_PATH_ERROR signals the corresponding path error.
- RAPI_RESV_CONFIRM signals the arrival of a CONFIRM message.
- RAPI_RESV_ERROR signals the corresponding reservation error.

A synchronous error in a RAPI routine returns an appropriate error code. Asynchronous RSVP errors are delivered to the application by way of the RAPI upcall routine.

Compiling and linking RAPI applications

To use the RAPI interface, an application must perform the following steps:

1. Include the <rapi.h> header file, which is available in the /usr/include directory.
2. Compile the application with the DLL compiler option. See the z/OS XL C/C++ User's Guide for more information about how to specify compiler options.
3. Include the RAPI definition side deck (rapi.x), which is available in the /usr/lib directory, when prelinking or binding the application.
4. If the Binder is used instead of the C Prelinker, specify the Binder DYNAM=DLL option. See the z/OS MVS Program Management: User's Guide and Reference for information about specifying Binder options.

Running RAPI applications

At execution time, the RAPI application must have access to the RAPI DLL (rapi.dll), which is available in the /usr/lib directory. Ensure that the LIBPATH environment variable includes this directory when running the application. The RAPI application must run with superuser authority to use RAPI.
Event upcall

An upcall is invoked by the asynchronous event mechanism. It executes the function whose address was specified in the event_rtn parameter in the rapi_session() call.

The event upcall function template is defined as follows:

```c
typedef int rapi_event_rtn_t(
    rapi_sid_t Sid,  /* Session ID */
    rapi_eventinfo_t EventType, /* Event type */
    rapi_styleid_t Style, /* Reservation style */
    int ErrorCode, /* Error event: code */
    int ErrorValue, /* Error event: value */
    rapi_addr_t *ErrorNode, /* Node detecting error */
    unsigned int ErrorFlags, /* Error flags */
    int FilterspecNo, /* number of filterspecs */
    rapi_filter_t *Filterspec_list,
    int FlowspecNo, /* number of flowspecs */
    rapi_flowspec_t *Flowspec_list,
    int AdspecNo, /* number of adspecs */
    rapi_adspec_t *Adspec_list,
    void *Event_arg /* application argument */
);
```

**rapi_event_rtn_t description**

This is the template for the function address that is supplied on the rapi_session call. The event upcall function is invoked from the asynchronous event mechanism when an event occurs.

**rapi_event_rtn_t parameters**

- **Sid**  
  This parameter is the session ID for the session initiated by a successful rapi_session() call.

- **EventType**  
  This parameter contains the upcall event type. See the description of this parameter in "rapi_event_rtn_t result" on page 164.

- **Style**  
  This parameter contains the style of the reservation; it is nonzero only for a RAPI_RESV_EVENT or RAPI_RESV_ERROR event.

- **ErrorCode, ErrorValue**  
  These values encode the error cause, and they are set only for a RAPI_PATH_ERROR or RAPI_RESV_ERROR event. See "RAPI error handling" on page 177 for interpretation of these values.

- **ErrorNode**  
  This is the IP address of the node that detected the error, and it is set only for a RAPI_PATH_ERROR or RAPI_RESV_ERROR event.

- **ErrorFlags**  
  These error flags are set only for a RAPI_PATH_ERROR or RAPI_RESV_ERROR event.
RAPI_ERRF_InPlace
The reservation failed, but another (presumably smaller) reservation is still in place on the same interface.

RAPI_ERRF_NotGuilty
The reservation failed, but the request from this client was merged with a larger reservation upstream, so this client reservation might not have caused the failure.

FilterSpec_list, FilterSpecNo
The FilterSpec_list parameter is a pointer to an area that contains a sequential vector of RAPI filter spec or sender template objects. The number of objects in this vector is specified in FilterSpecNo. If the FilterSpecNo value is 0, the FilterSpec_list parameter value is NULL.

Flowspec_list, FlowspecNo
The Flowspec_list parameter is a pointer to an area that contains a sequential vector of RAPI flowspec or Tspec objects. The number of objects in this vector is specified in FlowspecNo. If the FlowspecNo value is 0, the Flowspec_list parameter value is NULL.

Adspec_list, AdspecNo
The Adspec_list parameter is a pointer to an area that contains a sequential vector of RAPI adspec objects. The number of objects in this vector is specified in AdspecNo. If the AdspecNo value is 0, the Adspec_list parameter value is NULL.

Event_arg
This is the value that is supplied in the rapi_session() call.

rapi_event_rtn_t result
When the application upcall function returns, any areas pointed to by Flowspec_list, FilterSpec_list, or Adspec_list become not valid for further reference. The upcall function must copy any values it wants to save.

The specific parameters depend upon EventType, which can have one of the following values:

RAPI_PATH_EVENT
A path event indicates that RSVP sender (Path) state from a remote node has arrived or changed at the local node. A RAPI_PATH_EVENT event containing the complete current list of senders (or possibly no senders, after a path teardown) in the path state for the specified session is triggered whenever the path state changes.

FilterSpec_list, Flowspec_list, and Adspec_list are of equal length, and corresponding entries contain sender templates, sender Tspecs, and Adspects, respectively, for all senders known at this node. A missing object is generally indicated by an empty RAPI object.

RAPI_PATH_EVENT events are enabled by the initial rapi_session() call.

RAPI_RESV_EVENT
A reservation event indicates that reservation state has arrived or changed at the node, implying (but not assuring) that reservations have been established or deleted along the entire data path to one or more receivers. RAPI_RESV_EVENT upcalls containing the current reservation state for the API session are triggered whenever the reservation state changes.
Flowspec_list will either contain one flowspec object or be empty (if the state has been torn down), and FilterSpec_list contain zero or more corresponding filter spec objects. Adspec_list is empty.

RAPI_RESV_EVENT upcalls are enabled by a rapi_sender() call; the sender template from the latter call matches the filter spec returned in the upcall triggered by a reservation event.

RAPI_PATH_ERROR
A path error event indicates that an asynchronous error has been found in the sender information specified in a rapi_sender() call.

The ErrorCode and ErrorValue parameters specify the error. FilterSpec_list and Flowspec_list each contain one object, the sender template and corresponding sender Tspec (if any) in error, while Adspec_list is empty. If there is no sender Tspec, the object in Flowspec_list is an empty RAPI object. The Adspec_list is empty.

RAPI_PATH_ERROR events are enabled by a rapi_sender() call, and the sender Tspec in that call matches the sender Tspec returned in a subsequent upcall triggered by a RAPI_PATH_ERROR event.

RAPI_RESV_ERROR
A reservation error upcall indicates that an asynchronous reservation error has occurred.

The ErrorCode and ErrorValue parameters specify the error. Flowspec_list contains one flowspec, while FilterSpec_list can contain zero or more corresponding filter specs. Adspec_list is empty.

RAPI_RESV_ERROR events are enabled by a rapi_reserve() call.

RAPI_RESV_CONFIRM
A RAPI_RESV_CONFIRM event indicates that a reservation has been made at least up to an intermediate merge point, and probably (but not necessarily) all the way to at least one sender.

The parameters of a RAPI_RESV_CONFIRM event are the same as those for a RAPI_RESV_EVENT event upcall.

The accompanying table summarizes the upcalls. n is a nonnegative integer.

<table>
<thead>
<tr>
<th>Upcall</th>
<th>Enabled by</th>
<th>FilterSpecNo</th>
<th>FlowspecNo</th>
<th>AdspecNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAPI_PATH_EVENT</td>
<td>rapi_session</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>RAPI_PATH_ERROR</td>
<td>rapi_sender</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>RAPI_RESV_EVENT</td>
<td>rapi_sender</td>
<td>n</td>
<td>1 or 0</td>
<td>0</td>
</tr>
<tr>
<td>RAPI_RESV_ERROR</td>
<td>rapi_reserve</td>
<td>n</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>RAPI_RESV_CONFIRM</td>
<td>rapi_reserve</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The RSVP API provides the following client library calls:
- rapi_release()
- rapi_reserve()
- rapi_sender()
- rapi_session()
- rapi_version()
To use these calls, the application must include the file <rapi.h>. See “RAPI header files” on page 179 for more information on header files.

rapi_release - Remove a session

#include <rapi.h>

int rapi_release (rapi_sid_t Sid)

rapi_release description
The rapi_release() call removes the reservation, if any, and the state corresponding to a given session handle. This call will be made implicitly if the application terminates without closing its RSVP sessions.

rapi_release parameters
Sid This parameter is the session ID for the session initiated by a successful rapi_session() call.

rapi_release result
If the session handle is not valid, the call returns a corresponding RAPI error code; otherwise, it returns 0.

rapi_reserve - Make, modify, or delete a reservation

#include <rapi.h>

int rapi_reserve(
    rapi_sid_t Sid, /* Session ID */
    int Flags, /* Flags */
    rapi_addr_t *RHost, /* Receive host addr */
    rapi_styleid_t StyleId, /* Style ID */
    rapi_stylex_t *Style_Ext, /* Style extension */
    rapi_policy_t *Rcvr_Policy, /* Receiver policy */
    int FilterSpecNo, /* Number of filter specs */
    rapi_filter_t *FilterSpec_list, /* List of filter specs */
    int FlowspecNo, /* Number of flowspecs */
    rapi_flowspec_t *Flowspec_list /* List of flowspecs */
)

rapi_reserve description
The rapi_reserve() function is called to make, modify, or delete a resource reservation for a session. The call can be repeated with different parameters, allowing the application to modify or remove the reservation; the latest call will take precedence.

rapi_reserve parameters
Sid This parameter is the session ID for the session initiated by a successful rapi_session() call.

Flags No flags are currently defined for this call.

RHost This parameter is used to define the interface address on which data will be received for multicast flows. It is useful for a multihomed host. If it is NULL or the host address is INADDR_ANY, the default interface will be chosen.

StyleId This parameter specifies the reservation style ID (see Flowspec_list, FlowspecNo).

Style_Ext This parameter must be NULL.
Rcvr_Policy: This parameter must be NULL.

FilterSpec_list, FilterSpecNo
The FilterSpec_list parameter is a pointer to an area containing a sequential vector of RAPI filter spec objects. The number of objects in this vector is specified in FilterSpecNo. If FilterSpecNo is 0, the FilterSpec_list parameter is ignored and can be NULL.

Flowspec_list, FlowspecNo
The Flowspec_list parameter is a pointer to an area containing a sequential vector of RAPI flow spec objects. The number of objects in this vector is specified in FlowspecNo. If FlowspecNo is 0, the Flowspec_list parameter is ignored and can be NULL.

If FlowspecNo is 0, the call will remove the current reservations for the specified session, and FilterSpec_list and Flowspec_list will be ignored. Otherwise, the parameters depend upon the style, as follows:

**Wildcard Filter (WF)**
Use StyleId = RAPI_RSTYLE_WILDCARD. The Flowspec_list parameter can be NULL (to delete the reservation) or else point to a single flowspec. The FilterSpec_list parameter should be empty.

**Fixed Filter (FF)**
Use StyleId = RAPI_RSTYLE_FIXED. FilterSpecNo must equal FlowspecNo. Entries in Flowspec_list and FilterSpec_list parameters will correspond in pairs.

**Shared Explicit (SE)**
Use StyleId = RAPI_RSTYLE_SE. The Flowspec_list parameter should point to a single flowspec. The FilterSpec_list parameter can point to a list of any length.

**rapi_reserve result**
Depending upon the parameters, each call might or might not result in new admission control calls, which could fail asynchronously.

If there is a synchronous error in this call, rapi_reserve() returns a RAPI error code; otherwise, it returns 0.

Applications measure success in the form of errors returned when making QoS requests. No final acknowledgment will occur.

An admission control failure (for example, refusal of the QoS request) is reported asynchronously by an upcall of type RAPI_RESV_ERROR. A RSVP_Err_NO_PATH error code indicates that RSVP state from one or more of the senders specified in FilterSpec_list has not (yet) propagated all the way to the receiver; it might also indicate that one or more of the specified senders has closed its API session and that its RSVP state has been deleted from the routers.

**rapi_sender - Specify sender parameters**

```
#include <rapi.h>

int rapi_sender(
    rapi_sid_t    Sid,   /* Session ID */
```
int Flags, /* Flags */
rapi_addr_t *LHost, /* Local Host */
rapi_filter_t *SenderTemplate, /* Sender template */
rapi_tspec_t *SenderTspec, /* Sender Tspec */
rapi_adspec_t *SenderAdspec, /* Sender Adspec */
rapi_policy_t *SenderPolicy, /* Sender policy data */
int TTL /* Multicast data TTL */
)

rapi_sender description
An application must issue a rapi_sender() call if it intends to send a flow of data
for which receivers can make reservations. This call defines, redefines, or deletes
the parameters of that flow. A rapi_sender() call can be issued more than once for
the same API session; the most recent one takes precedence.

Once a successful rapi_sender() call has been made, the application can receive
upcalls of type RAPI_RESV_EVENT or RAPI_PATH_ERROR.

rapi_sender parameters
Sid This parameter is the session ID for the session initiated by a successful
rapi_session() call.
Flags No flags are currently defined for this call.
LHost This parameter can point to a rapi_addr_t structure specifying the IP
source address and, if applicable, the source port from which data is sent,
or it can be NULL.

If the IP source address is INADDR_ANY, the API uses the default IP
address of the local host. This is sufficient unless the host is multihomed.
The port number can be zero if the protocol for the session does not have
ports.

A NULL LHost parameter indicates that the application wishes to
withdraw its registration as a sender. In this case, the following parameters
will all be ignored.

SenderTemplate
This parameter can be a pointer to a RAPI filter specification structure
specifying the format of data packets to be sent, or it can be NULL.

If this parameter is NULL, a sender template will be created internally
from the Dest and LHost parameters. The Dest parameter was supplied in
an earlier rapi_session() call. If a SenderTemplate parameter is present, the
(non-NULL) LHost parameter is ignored.

SenderTspec
This parameter is a pointer to a Tspec that defines the traffic that this
sender will create and must not be NULL.

SenderAdspec
This parameter must be NULL or unpredictable results can occur.

SenderPolicy
This parameter must be NULL.

TTL This parameter specifies the IP TTL (Time-to-Live) value with which
multicast data will be sent. It allows RSVP to send its control messages
with the same TTL scope as the data packets.
**rapi_sender result**
If there is a synchronous error, rapi_sender() returns a RAPI error code; otherwise, it returns 0.

**rapi_session - Create a session**

```c
#include <rapi.h>

rapi_sid_t rapi_session(
   rapi_addr_t *Dest,  /* Session: (Dst addr, port) */
   int Protid,  /* Protocol Id */
   int Flags,  /* Flags */
   rapi_event_rtn_t Event_rtn,  /* Address of upcall routine */
   void *Event_arg,  /* App argument to upcall */
   int *Errnop /* Place to return error code*/
)
```

**rapi_session description**
The rapi_session() call creates an API session.

After a successful rapi_session() call has been made, the application can receive upcalls of type RAPI_PATH_EVENT for the API session.

**rapi_session parameters**
The parameters are as follows:

- **Dest**
  This parameter points to a rapi_addr_t structure defining the destination IP address and a port number to which data will be sent. The Dest and Protid parameters define an RSVP session. If the Protid specifies UDP or TCP transport, the port corresponds to the appropriate transport port number.

- **Protid**
  The IP protocol ID for the session. If it is omitted (that is, zero), 17 (UDP) is assumed.

- **Flags**
  The valid values for Flags are as follows:

  - **RAPI_USE_INTSERV**
    If set, IntServ formats are used in upcalls; otherwise, the Simplified format is used.

- **Event_rtn**
  This parameter is a function typedef for an upcall function that will be invoked to notify the application of RSVP errors and state change events. Pending events cause the invocation of the upcall function. The application must supply an upcall routine for event processing.

- **Event_arg**
  This parameter is an argument that will be passed to any invocation of the upcall routine.

- **Errnop**
  The address of an integer into which a RAPI error code will be returned. If Errnop is NULL, no error code is returned.

**rapi_session result**
If the call succeeds, the rapi_session() call returns a nonzero session handle for use in subsequent calls related to this API session.

If the call fails synchronously, it returns zero (RAPI_NULL_SID) and stores a RAPI error code into an integer variable pointed to by the Errnop parameter.
**rapi_session extended description**
An application can have multiple API sessions registered for the same or different RSVP sessions at the same time. There can be at most one sender associated with each API session; however, an application can announce multiple senders for a given RSVP session by announcing each sender in a separate API session.

Two API sessions for the same RSVP session, if they are receiving data, are assumed to have joined the same multicast group and will receive the same data packets.

**rapi_version - RAPI version**
```
#include <rapi.h>

int rapi_version(void)
```

**rapi_version description**
This call obtains the version of the interface. It can be used by an application to adapt to different versions.

**rapi_version result**
This call returns a single integer that defines the version of the interface. The returned value is composed of a major number and a minor number, encoded as $100 \times \text{major} + \text{minor}$

The API described in this topic has major version number 6.

**RAPI formatting routines**
For convenience of applications, RAPI includes standard routines for displaying the contents of RAPI objects.

These standard formatting routines are:
- `rapi_fmt_adspec`
- `rapi_fmt_filtspec`
- `rapi_fmt_flowspec`
- `rapi_fmt_tspec`

**rapi_fmt_adspec - Format an adspec**
```
#include <rapi.h>

void rapi_fmt_adspec(
    rapi_adspec_t *adspecp, /* Addr of RAPI adspec */
    char *buffer,   /* Addr of buffer */
    int length     /* Length of buffer */
)
```

**rapi_fmt_adspec description**
The `rapi_fmt_adspec()` call formats a given RAPI adspec into a buffer of given address and length. The output is truncated if the length is too small. If it is NULL, this function returns without performing any formatting.
rapi_fmt_adspec parameters

adspecp  This parameter is a pointer to the adspec to be formatted. If it is NULL, this function returns without performing any formatting.

buffer  This is a pointer to the user-supplied buffer into which the formatted output will be placed. If the buffer is too small to contain the output, then the formatted output is truncated. If this parameter is NULL, this function returns without performing any formatting.

length  This is the length of the buffer pointed to with the buffer parameter. If this parameter is 0, this function returns without performing any formatting.

rapi_fmt_adspec result
If possible, the input object is formatted into the user-supplied buffer. There is no return value.

The following example shows possible adspec output:
[GEN AS[brk=y hop=0 BW=0 lat=0 mtu=0] ]

The output reflects the following code:
GEN  Generic adspec

rapi_fmt_filtspec - Format a filtspec

#include <rapi.h>

void rapi_fmt_filtspec(
    rapi_filtspec_t *filtp, /* Addr of RAPI Filtspec */
    char *buffer, /* Addr of buffer */
    int length /* Length of buffer */
)

rapi_fmt_filtspec description
The rapi_fmt_filtspec() call formats a given RAPI filter spec into a buffer of given address and length. The output is truncated if the length is too small. If it is NULL, this function returns without performing any formatting.

rapi_fmt_filtspec parameters

filtp  This parameter is a pointer to the Filtspec to be formatted. If it is NULL, this function returns without performing any formatting.

buffer  This is a pointer to the user-supplied buffer into which the formatted output will be placed. If the buffer is too small to contain the output, then the formatted output is truncated. If this parameter is NULL, this function returns without performing any formatting.

length  This is the length of the buffer pointed to with the buffer parameter. If this parameter is 0, this function returns without performing any formatting.

rapi_fmt_filtspec result
If possible, the input object is formatted into the user-supplied buffer. There is no return value.

The following example shows possible filtspec output:
9.67.200.2/8000
showing the IP address and port.

rapi_fmt_flowspec - Format a flowspec

```c
#include <rapi.h>

void rapi_fmt_flowspec(
    rapi_flowspec_t *specp, /* Addr of RAPI flowspec */
    char *buffer, /* Addr of buffer */
    int length /* Length of buffer */
)
```

**rapi_fmt_flowspec description**
The `rapi_fmt_flowspec()` call formats a given RAPI `flowspec` into a buffer of given address and length. The output is truncated if the length is too small.

**rapi_fmt_flowspec parameters**
- **specp** This parameter is a pointer to the flowspec to be formatted. If it is NULL, this function returns without performing any formatting.
- **buffer** This is a pointer to the user-supplied buffer into which the formatted output will be placed. If the buffer is too small to contain the output, then the formatted output is truncated. If this parameter is NULL, this function returns without performing any formatting.
- **length** This is the length of the buffer pointed to with the buffer parameter. If this parameter is 0, this function returns without performing any formatting.

**rapi_fmt_flowspec result**
If possible, the input object is formatted into the user-supplied buffer. There is no return value.

The following example shows the formatted output for a Controlled Load flowspec.

```
[CL TS[r=90000 b=6000 p=5.5e+06 m=1024 M=2048] ]
```

**Note:** Many of the RAPI object values are floating point numbers. The formatting functions display large floating point values in a user-friendly way, such as that shown for the `Tspec` `p` value. The output reflects the following codes:

- **CL** Controlled load
- **TS** `Tspec`, listing the `Tspec` values

The following example shows the formatted output for a guaranteed flowspec.

```
[GUAR TS[r=90000 b=6000 p=5.5e+06 m=1024 M=2048] RS[R=90000 S=1] ]
```

**Note:** Many of the RAPI object values are floating point numbers. The formatting functions display large floating point values in a user-friendly way, such as that shown for the `Tspec` `p` value. The output reflects the following codes:

- **GUAR** Guaranteed
- **TS** `Tspec`, listing the `Tspec` values
- **RS** `Rspec`, listing the `Rspec` values
rapi_fmt_tspec - Format a tspec

```c
#include <rapi.h>

void rapi_fmt_tspec(
    rapi_tspec_t *tspecp, /* Addr of RAPI Tspec */
    char *buffer, /* Addr of buffer */
    int length /* Length of buffer */
)
```

**rapi_fmt_tspec description**

The `rapi_fmt_tspec()` call formats a given RAPI `Tspec` into a buffer of given address and length. The output is truncated if the length is too small.

**rapi_fmt_tspec parameters**

- **tspecp** This parameter is a pointer to the Tspec to be formatted. If it is NULL, this function returns without performing any formatting.

- **buffer** This is a pointer to the user-supplied buffer into which the formatted output will be placed. If the buffer is too small to contain the output, then the formatted output is truncated. If this parameter is NULL, this function returns without performing any formatting.

- **length** This is the length of the buffer pointed to with the buffer parameter. If this parameter is 0, this function returns without performing any formatting.

**rapi_fmt_tspec result**

If possible, the input object is formatted into the user-supplied buffer. There is no return value.

The following example shows possible Tspec output:
```
[GEN TS[r=55000 b=6000 p=5.5e+06 m=1024 M=2048] ]
```

**Note:** Many of the RAPI object values are floating point numbers. The formatting functions display large floating point values in a user-friendly way, such as that shown for the Tspec p value.

The output reflects the following codes:

- **GEN** Generic Tspec
- **TS** Tspec, listing the Tspec values

**RAPI objects**

*Flowspecs, filter specs, sender templates, and sender Tspecs* are encoded as variable-length RAPI objects.

Every RAPI object begins with a header of type `rapi_hdr_t`, which contains:
- The total length of the object in bytes
- The type

An empty object consists only of a header, with type 0 and length `sizeof (rapi_hdr_t)`.

Integrated services data structures are defined in RFC 2210, which describes the use of the RSVP with the Controlled-Load and Guaranteed services. (See Appendix E, “Related protocol specifications,” on page 743 for information about...
accessing RFCs.) RSVP defines several data objects which carry resource reservation information but are opaque to RSVP itself. The usage and data format of those objects is given in RFC 2210.

**RAPI objects - Flowspecs**

There are two formats for RAPI flowspecs. For further details, see "The <rapi.h> header" on page 180.

**RAPI_FLOWSTYPE_Simplified**

This is a simplified format. It consists of a simple list of parameters needed for either Guaranteed or Controlled Load service, using the service type QOS_GUARANTEED or QOS_CNTR_LOAD, respectively.

The RAPI client library routines map this format to or from an appropriate Integrated Services data structure.

**RAPI_FLOWSTYPE_Intserv**

This flowspec must be a fully formatted Integrated Services flowspec data structure.

**RAPI_FLOWSTYPE_Intserv upcalls**

In an upcall, a flowspec is by default delivered in simplified format. However, if the RAPI_USE_INTSERV flag was set in the rapi_session() call, then the IntServ format is used in upcalls.

**RAPI objects - Sender tspecs**

There are two formats for RAPI Sender Tspecs. For further details, see "The <rapi.h> header" on page 180.

**RAPI_TSPECTYPE_Simplified**

This is a simplified format consisting of a simple list of parameters with the service type QOS_TSPEC. The RAPI client library routines map this format to or from an appropriate Integrated Services data structure.

**RAPI_TSPECTYPE_Intserv**

This Tspec must be a fully formatted Integrated Services Tspec data structure.

**RAPI_TSPECTYPE_Intserv upcalls**

In an upcall, a sender Tspec is by default delivered in simplified format. However, if the RAPI_USE_INTSERV flag was set in the rapi_session() call, then the IntServ format is used in upcalls.

**RAPI objects - Adspecs**

There are two formats for RAPI adspecs. For further details, see "The <rapi.h> header" on page 180.

**RAPI_ADSTYPE_Simplified**

This is a simplified format, consisting of a list of adspec parameters for all possible services. The RAPI client library routines map this format to an appropriate Integrated Services data structure.

**RAPI_ADSTYPE_Intserv**

This adspec must be a fully formatted Integrated Services Adspec data structure.
**RAPI_ADSTYPE_Intserv upcalls**

In an upcall, an *adspec* is by default delivered in *simplified* format. However, if the RAPI_USE_INTSERV flag was set in the rapi_session() call, then the *IntServ* format is used in upcalls.

**RAPI objects - Filter specs and sender templates**

These objects have the following format:

*RAPI_FILTERFORM_BASE*  
This object consists of a socket address structure defining the IP address and port.

**RAPI asynchronous event handling**

The RAPI interface provides an asynchronous upcall mechanism using the select() function. The upcall mechanism is a cooperative effort between RAPI and the using application. The following shows the steps that must be taken by a RAPI application to receive asynchronous upcalls:

1. The upcall function pointer must be specified on the rapi_session() call that initiates the RAPI session. If the upcall function requires an argument, that also must be specified on rapi_session(). The argument is defined as a pointer to void.

2. The application must provide a means to be notified of asynchronous events. The best way to do this is to create a thread using pthread_create().

3. The thread created above must issue the rapi_getfd() call to learn the file descriptor of the socket used by RAPI for asynchronous communication.

4. The thread should then enter an endless loop to detect asynchronous events using the select() call with the file descriptor learned using rapi_getfd(). When an event is detected, the thread should call rapi_dispatch(), which then in turn calls the upcall function synchronously.

The following example illustrates these steps. This example is for illustration purposes only. It is not a complete program.

```c
/* Issue a rapi_session() call to initialize RAPI. */
rapi_sid = rapi_session(&destination, protocol, 0, rapi_async, /* upcall function pointer */ 0, /* no upcall argument */ &rc);
```

```c
/* Create a pthread to handle RAPI upcalls. */
pthread_create(&thread_d, NULL, &rapi_th, NULL);
```

```c
/* Function: rapi_th() */
void *rapi_th(void *arg)
{
    fd_set fds;
    struct timeval tv;
    /* code */
}
```
int rc = SUCCESSFUL;

/*****************************/
/* This is the pthread created to handle RAPI upcalls. First, get */
/* the rapi socket descriptor to use on select(). */
/*****************************/
pthread_mutex_lock(&rapi_lock);
fd = rapi_getfd(rapi_sid);
pthread_mutex_unlock(&rapi_lock);
if (fd > 0) {
    /*****************************/
    /* Loop as long as all is well, waiting via select() for an */
    /* asynchronous RAPI packet to arrive. */
    /*****************************/
    while (rc == SUCCESSFUL) {
        tv.tv_sec = 1;
        tv.tv_usec = 0;
        FD_ZERO(&fds);
        FD_SET(fd, &fds);
        switch(select(FD_SETSIZE, &fds, (fd_set *) NULL,
                    (fd_set *) NULL, &tv)) {
            /*****************************/
            /* Bad return from select(). Get out. */
            /*****************************/
            case -1:
                rc = UNSUCCESSFUL;
                break;
            /*****************************/
            /* Time out on select(). Ignore. */
            /*****************************/
            case 0:
                break;
            /*****************************/
            /* Dispatch data have arrived. Call the upcall function via */
            /* rapi_dispatch(). */
            /*****************************/
            default:
                pthread_mutex_lock(&rapi_lock);
                rc = rapi_dispatch();
                pthread_mutex_unlock(&rapi_lock);
                break;
        }
    }
}
/*****************************/
/* Error on rapi_getfd(). */
/*****************************/
else {
    rc = UNSUCCESSFUL;
}

pthread_exit(NULL);

rapi_dispatch - Dispatch API event

#include <rapi.h>

int rapi_dispatch(void)
rapi_dispatch description
The application should call this routine whenever a read event is signaled on a file descriptor returned by rapi_getfd(). The rapi_dispatch() routine can be called at any time, but it will generally have no effect unless there is a pending event.

rapi_dispatch parameters
There are no parameters to this call.

rapi_dispatch result
Calling this routine can result in one or more upcalls to the application from any of the open API sessions known to this instance of the library.

If this call encounters an error, rapi_dispatch() returns a RAPI error code; otherwise, it returns 0. See “RAPI error codes” on page 178 for a list of error codes.

rapi_getfd - Get file descriptor

#include <rapi.h>

int rapi_getfd (rapi_sid_t Sid)

rapi_getfd description
After a rapi_session() call has completed successfully and before rapi_release() has been called, the application can call rapi_getfd() to obtain the file descriptor associated with that session. When a read event is signaled on this file descriptor, the application should call rapi_dispatch().

rapi_getfd parameters
Sid This parameter is the session ID for the session initiated by a successful rapi_session() call.

rapi_getfd result
If Sid is illegal or undefined, this call returns -1; otherwise, it returns the file descriptor.

RAPI error handling
Errors can be detected synchronously or asynchronously.

When an error is detected synchronously, a RAPI error code is returned in the Errnop argument of rapi_session(), or as the function return value of rapi_sender(), rapi_reserve(), rapi_release(), or rapi_dispatch().

When an error is detected asynchronously, it is indicated by a RAPI_PATH_ERROR or RAPI_RESV_ERROR event. An RSVP error code and error value are then contained in the ErrorCode and ErrorValue arguments of the event_upcall() function. In case of an API error (RSVP error code 20), a RAPI error code is contained in the ErrorValue argument.

A description of RSVP error codes and values can be found in RFC 2205. See Appendix E, “Related protocol specifications,” on page 743 for information about accessing RFCs.
RAPI error codes

*[RAPI_ERR_OK]*  
No error

*[RAPI_ERR_INVAL]*  
Parameter not valid

*[RAPI_ERR_MAXSESS]*  
Too many sessions

*[RAPI_ERR_BADSID]*  
Session identity out of legal range

*[RAPI_ERR_N_FFS]*  
Wrong filter number or flow number for style

*[RAPI_ERR_BADSTYLE]*  
Illegal reservation style

*[RAPI_ERR_SYSCALL]*  
A system error has occurred; its nature can be indicated by *errno*.

*[RAPI_ERR_OVERFLOW]*  
Parameter list overflow

*[RAPI_ERR_MEMFULL]*  
Not enough memory

*[RAPI_ERR_NORSVP]*  
The RSVP agent is not active or is unable to respond.

*[RAPI_ERR_OBJTYPE]*  
Object type not valid

*[RAPI_ERR_OBJLEN]*  
Object length not valid

*[RAPI_ERR_NOTSPEC]*  
No sender Tspec

*[RAPI_ERR_INTSERV]*  
Integrated Services parameter format not valid

*[RAPI_ERR_GPI_CONFLICT]*  
IPSEC: Conflicting C-type

*[RAPI_ERR_BADPROTO]*  
IPSEC: Protocol not AH or ESP

*[RAPI_ERR_BADVDPORT]*  
IPSEC: vDstPort is 0.

*[RAPI_ERR_GPISESS]*  
IPSEC: Parameters for GPI_SESSION flag not valid, or other parameter error

*[RAPI_ERR_BADSEND]*  
Sender address not my interface

*[RAPI_ERR_BADRECV]*  
Receiver address not my interface

*[RAPI_ERR_BADSPORT]*  
Source port not valid: the source port is nonzero when the destination port is 0.
Unsupported feature

Unknown error

[RAPI_ERR_BADSEND], [RAPI_ERR_BADRECV] and [RAPI_ERR_BADSPORT] occur only asynchronously, as the ErrorValue when the ErrorCode is 20 (API error).

RSVP error codes

<table>
<thead>
<tr>
<th>Value</th>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>RSVP_Err_NONE</td>
<td>No error (confirmation)</td>
</tr>
<tr>
<td>1</td>
<td>RSVP_Err_ADMISSION</td>
<td>Admission control failure</td>
</tr>
<tr>
<td>2</td>
<td>RSVP_Err_POLICY</td>
<td>Policy control failure</td>
</tr>
<tr>
<td>3</td>
<td>RSVP_Err_NO_PATH</td>
<td>No path information</td>
</tr>
<tr>
<td>4</td>
<td>RSVP_Err_NO_SENDER</td>
<td>No sender information</td>
</tr>
<tr>
<td>5</td>
<td>RSVP_Err_BAD_STYLE</td>
<td>Conflicting style</td>
</tr>
<tr>
<td>6</td>
<td>RSVP_Err_UNKNOWN_STYLE</td>
<td>Unknown style</td>
</tr>
<tr>
<td>7</td>
<td>RSVP_Err_BAD_DSTPORT</td>
<td>Conflicting destination port in session</td>
</tr>
<tr>
<td>8</td>
<td>RSVP_Err_BAD_SNDPORT</td>
<td>Conflicting source port</td>
</tr>
<tr>
<td>9</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>RSVP_Err_PREEMPTED</td>
<td>Service preempted</td>
</tr>
<tr>
<td>13</td>
<td>RSVP_Err_UNKN_OBJ_CLASS</td>
<td>Unknown object class</td>
</tr>
<tr>
<td>14</td>
<td>RSVP_Err_UNKNOWN_CTYPE</td>
<td>Unknown object C-Type</td>
</tr>
<tr>
<td>15</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>RSVP_Err_API_ERROR</td>
<td>API error</td>
</tr>
<tr>
<td>21</td>
<td>RSVP_Err_TC_ERROR</td>
<td>Traffic control error</td>
</tr>
<tr>
<td>22</td>
<td>RSVP_Err_TC_SYS_ERROR</td>
<td>Traffic control system error</td>
</tr>
<tr>
<td>23</td>
<td>RSVP_Err_RSVP_SYS_ERROR</td>
<td>RSVP system error</td>
</tr>
</tbody>
</table>

RAPI header files

The following topics apply to RAPI header files.

RAPI header files: Integer and floating point types

Types u_int8_t, u_int16_t and u_int32_t, which appear in the <rapi.h> header file, are unsigned integer types of length 8, 16, and 32 bits, respectively.
Type `float32_t` is a floating-point type of length 32 bits. It is defined by including the `<rapi.h>` header file.

The `<rapi.h>` header

This header file contains the definitions of the RSVP API (RAPI) library calls.

Inclusion of this header can make available other symbols in addition to those specified in this topic.

`<rapi.h>` header general definitions

The following general definitions apply to the `<rapi.h>` header:

- Macro `RAPI_VERSION` is defined with value `100 * major + minor`, where `major` is the major version number and `minor` is the minor version number. The value of `RAPI_VERSION` is returned by `rapi_version()`.
- Type `rapi_addr_t` is defined for protocol addresses. It is defined to be `struct sockaddr`.
- Enumeration `qos_service_t` is defined by `typedef` and has at least the following members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>QOS_CNTR_LOAD</td>
<td>Controlled-load service</td>
</tr>
<tr>
<td>QOS_GUARANTEED</td>
<td>Guaranteed service</td>
</tr>
<tr>
<td>QOS_TSPECTYPE</td>
<td>Generic Tspec</td>
</tr>
</tbody>
</table>

- Enumeration `rapi_format_t` is defined by `typedef` and has at least the following members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAPI_ADSTYPE_Intserv</td>
<td>Int-Serv format adspec</td>
</tr>
<tr>
<td>RAPI_ADSTYPE_Simplified</td>
<td>Simplified format adspec</td>
</tr>
<tr>
<td>RAPIEMPTY_OTYPE</td>
<td>Empty object</td>
</tr>
<tr>
<td>RAPIFILTERFORM_BASE</td>
<td>Simple V4: Only sockaddr</td>
</tr>
<tr>
<td>RAPI_FLOWSTYPE_Intserv</td>
<td>Int-Serv format flowspec</td>
</tr>
<tr>
<td>RAPI_FLOWSTYPE_Simplified</td>
<td>Simplified format flowspec</td>
</tr>
<tr>
<td>RAPI_TSPECTYPE_Intserv</td>
<td>Int-Serv format (sndr)Tspec</td>
</tr>
<tr>
<td>RAPI_TSPECTYPE_Simplified</td>
<td>Simplified format (sndr)Tspec</td>
</tr>
</tbody>
</table>

- Type `rapi_hdr_t` is defined by `typedef` as a structure to represent a generic RAPI object header. It has the following members, followed by type-specific contents:

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>form</td>
<td>int</td>
<td>Format</td>
</tr>
<tr>
<td>len</td>
<td>unsigned int</td>
<td>Actual length in bytes</td>
</tr>
</tbody>
</table>

`<rapi.h>` header `tspec` definitions

The following Tspec definitions apply to the `<rapi.h>` header:

- Type `qos_Tspec_body` is defined by `typedef` as a structure with at least the following members:
Type *qos_tspecx_t* is defined by typedef as a structure that contains the generic Tspec parameters, and has at least the following members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>spec_Tspec_r</td>
<td>float32_t</td>
<td>Token bucket average rate in bytes per second</td>
</tr>
<tr>
<td>spec_Tspec_b</td>
<td>float32_t</td>
<td>Token bucket depth in bytes</td>
</tr>
<tr>
<td>spec_Tspec_m</td>
<td>u_int32_t</td>
<td>Minimum policed unit in bytes</td>
</tr>
<tr>
<td>spec_Tspec_M</td>
<td>u_int32_t</td>
<td>Maximum packet size in bytes</td>
</tr>
<tr>
<td>spec_Tspec_p</td>
<td>float32_t</td>
<td>Peak data rate in bytes per second</td>
</tr>
</tbody>
</table>

The following macros are defined with the values given below:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>xtspec_r</td>
<td>xtspec_Tspec.spec_Tspec_r</td>
</tr>
<tr>
<td>xtspec_b</td>
<td>xtspec_Tspec.spec_Tspec_b</td>
</tr>
<tr>
<td>xtspec_m</td>
<td>xtspec_Tspec.spec_Tspec_m</td>
</tr>
<tr>
<td>xtspec_M</td>
<td>xtspec_Tspec.spec_Tspec_M</td>
</tr>
<tr>
<td>xtspec_p</td>
<td>xtspec_Tspec.spec_Tspec_p</td>
</tr>
</tbody>
</table>

Type *rapi_tspec_t* is defined by typedef as a structure to represent a Tspec descriptor, and has at least the following members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>form</td>
<td>rapi_format_t</td>
<td>Tspec format</td>
</tr>
<tr>
<td>IS_t</td>
<td>IS_tsbody_t</td>
<td>Int-serv format Tspec</td>
</tr>
<tr>
<td>len</td>
<td>unsigned int</td>
<td>Actual length in bytes</td>
</tr>
<tr>
<td>qosxt</td>
<td>qos_tspecx_t</td>
<td>Simplified format Tspec</td>
</tr>
<tr>
<td>tspecbody_u</td>
<td>union</td>
<td></td>
</tr>
</tbody>
</table>

The following macros are defined with the values given below:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tspecbody_qosx</td>
<td>tspecbody_u.qosxt</td>
</tr>
<tr>
<td>tspecbody_IS</td>
<td>tspecbody_u.IS</td>
</tr>
</tbody>
</table>

*rapi.h* header flowspec definitions

The following flowspec definitions apply to the *rapi.h* header:

- Type *qos_flowspecx_t* is defined by typedef as a structure that contains the union of the parameters for **controlled-load service** and **guaranteed service** models, and has at least the following members:
<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>spec_type</td>
<td>qos_service_t</td>
<td>QoS service type</td>
</tr>
<tr>
<td>xspec_R</td>
<td>float32_t</td>
<td>Rate in bytes per second</td>
</tr>
<tr>
<td>xspec_S</td>
<td>u_int32_t</td>
<td>Slack term in microseconds</td>
</tr>
<tr>
<td>xspec_Tspec</td>
<td>qos_Tspec_body</td>
<td>Tspec</td>
</tr>
</tbody>
</table>

- The following macros are defined with the values given below:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>xspec_r</td>
<td>xspec_Tspec.spec_Tspec_r</td>
</tr>
<tr>
<td>xspec_b</td>
<td>xspec_Tspec.spec_Tspec_b</td>
</tr>
<tr>
<td>xspec_m</td>
<td>xspec_Tspec.spec_Tspec_m</td>
</tr>
<tr>
<td>xspec_M</td>
<td>xspec_Tspec.spec_Tspec_M</td>
</tr>
<tr>
<td>xspec_p</td>
<td>xspec_Tspec.spec_Tspec_p</td>
</tr>
</tbody>
</table>

- Type `rapi_flowspec_t` is defined by typedef as a structure to represent a flowspec descriptor, and has at least the following members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>len</td>
<td>unsigned int</td>
<td>Actual length in bytes</td>
</tr>
<tr>
<td>form</td>
<td>rapi_format_t</td>
<td>Flowspec format</td>
</tr>
<tr>
<td>IS</td>
<td>IS_specbody_t</td>
<td>Int-serv format flowspec</td>
</tr>
<tr>
<td>specbody_u</td>
<td>union</td>
<td></td>
</tr>
<tr>
<td>qosx</td>
<td>qos_flowspecx_t</td>
<td>Simplified format flowspec</td>
</tr>
</tbody>
</table>

- The following macros are defined with the values given below:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>specbody_qosx</td>
<td>specbody_u.qosx</td>
</tr>
<tr>
<td>specbody_IS</td>
<td>specbody_u.IS</td>
</tr>
</tbody>
</table>

**<rapi.h> header adspec definitions**

The following adspec definitions apply to the `<rapi.h>` header:

- Type `qos_adspecx_t` is defined by typedef as a structure that contains the union of all adspec parameters for controlled-load service and guaranteed service models, and has at least the following members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>General path characterization parameters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xaspec_flags</td>
<td>u_int8_t</td>
<td>Flags(1)</td>
</tr>
<tr>
<td>xaspec_hopcnt</td>
<td>u_int16_t</td>
<td></td>
</tr>
<tr>
<td>xaspec_path_bw</td>
<td>float32_t</td>
<td></td>
</tr>
<tr>
<td>xaspec_min_latency</td>
<td>u_int32_t</td>
<td></td>
</tr>
<tr>
<td>xaspec_composed_MTU</td>
<td>u_int32_t</td>
<td></td>
</tr>
<tr>
<td>Controlled-load service adspec parameters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xClaspec_flags</td>
<td>u_int8_t</td>
<td>Flags</td>
</tr>
</tbody>
</table>

z/OS V1R11.0 Comm Svr: IP Programmer's Guide and Reference
### Member Usage

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>xClaspec_override</td>
<td>u_int8_t</td>
<td>See note (2)</td>
</tr>
<tr>
<td>xClaspec_hopcnt</td>
<td>u_int16_t</td>
<td></td>
</tr>
<tr>
<td>xClaspec_path_bw</td>
<td>float32_t</td>
<td></td>
</tr>
<tr>
<td>xClaspec_min_latency</td>
<td>u_int32_t</td>
<td></td>
</tr>
<tr>
<td>xClaspec_composed_MTU</td>
<td>u_int32_t</td>
<td></td>
</tr>
</tbody>
</table>

#### Guaranteed service adspec parameters

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>xGaspec_flags</td>
<td>u_int8_t</td>
<td>Flags</td>
</tr>
<tr>
<td>xGaspec_Ctot</td>
<td>u_int32_t</td>
<td></td>
</tr>
<tr>
<td>xGaspec_Dtot</td>
<td>u_int32_t</td>
<td></td>
</tr>
<tr>
<td>xGaspec_Csum</td>
<td>u_int32_t</td>
<td></td>
</tr>
<tr>
<td>xGaspec_Dsum</td>
<td>u_int32_t</td>
<td></td>
</tr>
<tr>
<td>xGaspec_override</td>
<td>u_int8_t</td>
<td>See note (2)</td>
</tr>
<tr>
<td>xGaspec_hopcnt</td>
<td>u_int16_t</td>
<td></td>
</tr>
<tr>
<td>xGaspec_path_bw</td>
<td>float32_t</td>
<td></td>
</tr>
<tr>
<td>xGaspec_min_latency</td>
<td>u_int32_t</td>
<td></td>
</tr>
<tr>
<td>xGaspec_composed_MTU</td>
<td>u_int32_t</td>
<td></td>
</tr>
</tbody>
</table>

#### Notes:

1. FLG_IGN is not allowed; FLG_PARM is assumed.
2. A value of 1 means "override all generic parameters."

- The following macros are defined with bitwise-distinct integral values for use in the xaspec_flags xClaspec_flags and xGaspec_flags fields:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>XASPEC_FLG_BRK</td>
<td>Break bit: service unsupported in some node.</td>
</tr>
<tr>
<td>XASPEC_FLG_IGN</td>
<td>Ignore flag: Do not include this service.</td>
</tr>
<tr>
<td>XASPEC_FLG_PARM</td>
<td>Parms-present flag: Include service parameters.</td>
</tr>
</tbody>
</table>

- Type rapi_adspec_t is defined by typedef as a structure to represent an adspec descriptor, and has at least the following members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>adsbody_u</td>
<td>union</td>
<td></td>
</tr>
<tr>
<td>adsx</td>
<td>qos_adspecx_t</td>
<td>Simplified format adspec</td>
</tr>
<tr>
<td>form</td>
<td>rapi_format_t</td>
<td>adspec format</td>
</tr>
<tr>
<td>ISa</td>
<td>IS_adsbody_t</td>
<td>Int-serv format adspec</td>
</tr>
<tr>
<td>len</td>
<td>unsigned int</td>
<td>Actual length in bytes</td>
</tr>
</tbody>
</table>

- The following macros are defined with the values given below:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>adspecbody_IS</td>
<td>adsbody_u.ISa</td>
</tr>
</tbody>
</table>
<rapi.h> header filter spec definitions
The following filter spec definitions apply to the <rapi.h> header:

- Type `rapi_filter_base_t` is defined by typedef as a structure that contains at least the following member:

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>sender</td>
<td>struct sockaddr_in</td>
</tr>
</tbody>
</table>

- Type `rapi_filter_t` is defined by typedef as a structure that contains at least the following members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>rapi_filter_base_t</td>
<td></td>
</tr>
<tr>
<td>filt_u</td>
<td>union</td>
<td></td>
</tr>
<tr>
<td>form</td>
<td>rapi_format_t</td>
<td>Filterspec</td>
</tr>
<tr>
<td>len</td>
<td>u_int32_t</td>
<td>actual length</td>
</tr>
</tbody>
</table>

- The following macros are defined with the values given below:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>rapi_filt4</td>
<td>filt_u.base.sender</td>
</tr>
<tr>
<td>rapi_filtbase4_addr</td>
<td>rapi_filt4.sin_addr</td>
</tr>
<tr>
<td>rapi_filtbase4_port</td>
<td>rapi_filt4.sin_port</td>
</tr>
</tbody>
</table>

<rapi.h> header policy definitions
The following policy definitions apply to the <rapi.h> header:

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>form</td>
<td>rapi_format_t</td>
</tr>
<tr>
<td>len</td>
<td>u_int32_t</td>
</tr>
<tr>
<td>pol_u</td>
<td>union</td>
</tr>
</tbody>
</table>

<rapi.h> header reservation style definitions
The following reservation style definitions apply to the <rapi.h> header:

- Enumeration `rapi_styleid_t` is defined by typedef for reservation style identifiers, and has at least the following members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAPI_RSTYLE_WILDCARD</td>
<td>Reservation will be shared among a wildcard selection of senders.</td>
</tr>
<tr>
<td>RAPI_RSTYLE_FIXED</td>
<td>Reservation will not be shared and will be dedicated to a particular sender.</td>
</tr>
<tr>
<td>RAPI_RSTYLE_SE</td>
<td>Reservation will be shared among an explicit list of senders.</td>
</tr>
</tbody>
</table>

- Type `rapi_stylex_t` is defined by typedef as `void`. 
The following function interface definitions apply to the `<rapi.h>` header:

- Type `rapi_sid_t` is defined by typedef as `unsigned int` for RAPI client handles.
- Macro `NULL_SID` is defined for error returns from `rapi_session()`.
- The following macro is defined and evaluated to a bitwise-distinct integral value:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAPI_USE_INTSERV</td>
<td>Use Int-Serv fmt in upcalls</td>
</tr>
</tbody>
</table>

Enumeration `rapi_eventinfo_t` is defined by typedef for RAPI event types, and has at least the following members:

<table>
<thead>
<tr>
<th>Member</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAPI_PATH_ERROR</td>
</tr>
<tr>
<td>RAPI_PATH_EVENT</td>
</tr>
<tr>
<td>RAPI_RESV_CONFIRM</td>
</tr>
<tr>
<td>RAPI_RESV_ERROR</td>
</tr>
<tr>
<td>RAPI_RESV_EVENT</td>
</tr>
</tbody>
</table>

- The following macros are defined and evaluate to distinct integral values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAPI_ERRF_InPlace</td>
<td>Left reservation in place</td>
</tr>
<tr>
<td>RAPI_ERRF_NotGuilty</td>
<td>This receiver not guilty</td>
</tr>
</tbody>
</table>

- Type `rapi_event_rtn_t` is defined by typedef as a function that conforms to the prototype defined in the definition for event upcall.
- The following macros are defined and evaluate to distinct integral values for use as RAPI error codes. Macro `RAPI_ERR_OK` (which indicates that there is no error) evaluates to 0.

<table>
<thead>
<tr>
<th>Error code</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAPI_ERR_BADPROTO</td>
</tr>
<tr>
<td>RAPI_ERR_BADRECV</td>
</tr>
<tr>
<td>RAPI_ERR_BADSEND</td>
</tr>
<tr>
<td>RAPI_ERR_BADSID</td>
</tr>
<tr>
<td>RAPI_ERR_BADSPORT</td>
</tr>
<tr>
<td>RAPI_ERR_BADSTYLE</td>
</tr>
<tr>
<td>RAPI_ERR_BADVDPOR</td>
</tr>
<tr>
<td>RAPI_ERR_GPI_CONFLICT</td>
</tr>
<tr>
<td>RAPI_ERR_GPISESS</td>
</tr>
<tr>
<td>RAPI_ERR_INTSERV</td>
</tr>
<tr>
<td>RAPI_ERR_INVAL</td>
</tr>
<tr>
<td>RAPI_ERR_MAXSESS</td>
</tr>
<tr>
<td>RAPI_ERR_MEMFULL</td>
</tr>
<tr>
<td>RAPI_ERR_N_FFS</td>
</tr>
</tbody>
</table>
Integrated services data structures and macros

The following defines the integrated services (see RFC 2210) data formats. (See Appendix E, “Related protocol specifications,” on page 743 for information about accessing RFCs.) The RAPI interface was designed to allow an application to specify either the int-serv format of a flowspec, Tspec, or adspec, or a simplified version of each.

The simplified versions allow almost any int-serv version to be generated, but there can be circumstances in which this is not adequate. For example, more general forms of flowspec, containing more than one service, might be defined in the future (so that in case the Resv message reaches a node that does not implement service A, it can drop back to service B). Allowing an application to specify the body of an arbitrary int-serv data object allows for such contingencies.
Future versions of this specification might change the definitions in this topic. Application writers are advised not to use these definitions except when absolutely necessary.

Notes:
1. The values in the data structures defined in this topic are in host byte order.
2. Inclusion of this header can make available other symbols in addition to those specified in this topic.

**Integrated services data structures and macros general definitions**

The following general definitions apply to the integrated services data structures and macros:

- The following macro is defined with the value given below:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordsof(x)</td>
<td>(((x)+3)/4)</td>
<td>number of 32-bit words</td>
</tr>
</tbody>
</table>

- The following macros are defined with the following integer values for service numbers:

  Note: The values are protocol values defined in RFC 2211, RFC 2212, and RFC 2215. See Appendix E, “Related protocol specifications,” on page 743 for information about accessing RFCs.

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENERAL_INFO</td>
<td>1</td>
</tr>
<tr>
<td>GUARANTEED_SERV</td>
<td>2</td>
</tr>
<tr>
<td>CONTROLLED_LOAD_SERV</td>
<td>5</td>
</tr>
</tbody>
</table>

- Enumeration `int_serv_wkp` is defined for well-known parameter identities and has at least the following members with the following integer values:

  Note: The values are protocol values defined in RFC 2215. See Appendix E, “Related protocol specifications,” on page 743 for information about accessing RFCs.

<table>
<thead>
<tr>
<th>Member</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS_WKP_HOP_CNT</td>
<td>4</td>
<td>Number of network nodes supporting Integrated Services along the flow path</td>
</tr>
<tr>
<td>IS_WKP_PATH_BW</td>
<td>6</td>
<td>Available bandwidth in bytes per second throughout the flow path</td>
</tr>
<tr>
<td>IS_WKP_MIN_LATENCY</td>
<td>8</td>
<td>Minimum end-to-end latency in microseconds</td>
</tr>
<tr>
<td>IS_WKP_COMPOSED_MTU</td>
<td>10</td>
<td>Maximum transmission unit without causing IP fragmentation along the flow path</td>
</tr>
<tr>
<td>IS_WKP_TB_TSPEC</td>
<td>127</td>
<td>Token-bucket TSPEC parameter</td>
</tr>
</tbody>
</table>
The following macros are defined with the values given below:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTSERV_VERS_MASK</td>
<td>0xf0</td>
</tr>
<tr>
<td>INTSERV_VERSION0</td>
<td>0</td>
</tr>
<tr>
<td>Intserv_Version(x)</td>
<td>(((x)&amp;ismh_version &amp;INTSERV_VERS_MASK)&gt;&gt;4)</td>
</tr>
<tr>
<td>Intserv_Version_OK(x)</td>
<td>(((x)-&gt;ismh_version &amp;INTSERV_VERS_MASK)==\INTSERV_VERSION0)</td>
</tr>
</tbody>
</table>

Type IS_main_hdr_t is defined by typedef as a structure to represent an Integrated Services main header, and has at least the following members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ismh_len32b</td>
<td>u_int16_t</td>
<td>Number of 32-bit words excluding this header</td>
</tr>
<tr>
<td>ismh_unused</td>
<td>u_int8_t</td>
<td></td>
</tr>
<tr>
<td>ismh_version</td>
<td>u_int8_t</td>
<td>Version</td>
</tr>
</tbody>
</table>

Type IS_serv_hdr_t is defined by typedef as a structure to represent an Integrated Services service element header, and has at least the following members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>issh_flags</td>
<td>u_int8_t</td>
<td>Flag byte</td>
</tr>
<tr>
<td>issh_len32b</td>
<td>u_int16_t</td>
<td>Number of 32-bit words excluding this header</td>
</tr>
<tr>
<td>issh_service</td>
<td>u_int8_t</td>
<td>Service number</td>
</tr>
</tbody>
</table>

The following macro is defined with the value given below to indicate the break bit in the IS_serv_hdr_t flag byte:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISSH_BREAK_BIT</td>
<td>0x80</td>
</tr>
</tbody>
</table>

Type IS_parm_hdr_t is defined by typedef as a structure to represent an Integrated Services parameter element header, and has at least the following members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>isph_flags</td>
<td>u_int8_t</td>
<td>Flags</td>
</tr>
<tr>
<td>isph_len32b</td>
<td>u_int16_t</td>
<td>Number of 32-bit words excluding this header</td>
</tr>
<tr>
<td>isph_parm_num</td>
<td>u_int8_t</td>
<td>Parameter number</td>
</tr>
</tbody>
</table>

The following macro is defined with the value given below to indicate the not valid bit in the IS_parm_hdr_t flag byte:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISPH_FLG_INV</td>
<td>0x80</td>
</tr>
</tbody>
</table>
The following macros are defined with the values given below:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Next_Main_Hdr(p)</td>
<td>((IS_main_hdr_t *)((u_int32_t *)((u_int32_t *)(p)+1+(p)-&gt;ismh_len32b))</td>
</tr>
<tr>
<td>Next_Parm_Hdr(p)</td>
<td>((IS_parm_hdr_t *)((u_int32_t *)((u_int32_t *)(p)+1+(p)-&gt;isph_len32b))</td>
</tr>
<tr>
<td>Next_Serv_Hdr(p)</td>
<td>((IS_serv_hdr_t *)((u_int32_t *)(p)+1+(p)-&gt;issh_len32b))</td>
</tr>
<tr>
<td>Non_Is_Hop</td>
<td>((IS_serv_hdr_t *)p)-&gt;issh_flags &amp; ISSH_BREAK_BIT</td>
</tr>
<tr>
<td>Set_Break_Bit(p)</td>
<td>((IS_serv_hdr_t *)p)-&gt;issh_flags</td>
</tr>
<tr>
<td>Set_Main_Hdr(p, len)</td>
<td>((p)-&gt;ismh_version = INTSERV_VERSION0; \</td>
</tr>
<tr>
<td></td>
<td>\ (p)-&gt;ismh_unused = 0; \ \ (p)-&gt;ismh_len32b = wordsof(len); )</td>
</tr>
<tr>
<td>Set_Parm_Hdr(p, id, len)</td>
<td>((p)-&gt;isph_parm_num = (id); \ \</td>
</tr>
<tr>
<td></td>
<td>\ (p)-&gt;isph_flags = 0; \ \ (p)-&gt;isph_len32b = wordsof(len); )</td>
</tr>
<tr>
<td>Set_Serv_Hdr(p, s, len)</td>
<td>((p)-&gt;issh_service = (s); \ \ \ \</td>
</tr>
<tr>
<td></td>
<td>\ \ (p)-&gt;issh_flags = 0; \ \ \ \ \ \ (p)-&gt;issh_len32b = wordsof(len); )</td>
</tr>
</tbody>
</table>

**Integrated services data structures and macros generic tspec format**

The following generic tspec formats apply to the integrated services data structures and macros:

- The following macros define constraints on the *token bucket* parameters for both the controlled-load and guaranteed service. These constraints are imposed by the respective service specifications and are not an indication of what minimum or maximum values a RAPI implementation will accept.

The following macros are defined with values of type `float32_t`:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Usage</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TB_MIN_RATE</td>
<td>Minimum token bucket rate</td>
<td>1 byte per second</td>
</tr>
<tr>
<td>TB_MAX_RATE</td>
<td>Maximum token bucket rate</td>
<td>40 terabytes per second</td>
</tr>
<tr>
<td>TB_MIN_DEPTH</td>
<td>Minimum token bucket depth</td>
<td>1 byte</td>
</tr>
<tr>
<td>TB_MAX_DEPTH</td>
<td>Maximum token bucket depth</td>
<td>250 gigabytes</td>
</tr>
<tr>
<td>TB_MAX_PEAK</td>
<td>Maximum peak rate</td>
<td>Positive infinity, defined as an IEEE single-precision floating-point number with an exponent of all ones (255) and a sign and mantissa of all zeros (see RFC 1832; see Appendix E, “Related protocol specifications,” on page 743 for information about accessing RFCs)</td>
</tr>
</tbody>
</table>

Type `TB_Tsp_parms_t` is defined by typedef as a structure to represent generic Tspec parameters, and has at least the following members:
<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>TB_Tspec_b</td>
<td>float32_t</td>
<td>Token bucket depth in bytes</td>
</tr>
<tr>
<td>TB_Tspec_m</td>
<td>u_int32_t</td>
<td>Minimum policed unit in bytes</td>
</tr>
<tr>
<td>TB_Tspec_M</td>
<td>u_int32_t</td>
<td>Maximum packet size in bytes</td>
</tr>
<tr>
<td>TB_Tspec_p</td>
<td>float32_t</td>
<td>Peak data rate in bytes per second</td>
</tr>
<tr>
<td>TB_Tspec_r</td>
<td>float32_t</td>
<td>Token bucket rate in bytes per second</td>
</tr>
</tbody>
</table>

- Type `gen_Tspec_t` is defined by typedef as a structure to represent a generic Tspec, and has at least the following members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>gen_Tspec_parms</td>
<td>TB_Tspec_parms_t</td>
<td></td>
</tr>
<tr>
<td>gen_Tspec_parm_hdr</td>
<td>IS_parm_hdr_t (IS_WKP_TB_TSPEC,)</td>
<td></td>
</tr>
<tr>
<td>gen_Tspec_serv_hdr</td>
<td>IS_serv_hdr_t (GENERAL_INFO, length)</td>
<td></td>
</tr>
</tbody>
</table>

- The following macros are defined with the values given below:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>gtspec_b</td>
<td>gen_Tspec_parms.TB_Tspec_b</td>
</tr>
<tr>
<td>gtspec_flags</td>
<td>gen_Tspec_parm_hdr.isph_flags</td>
</tr>
<tr>
<td>gtspec_len</td>
<td>(sizeof(gen_Tspec_t) - sizeof(IS_serv_hdr_t))</td>
</tr>
<tr>
<td>gtspec_len32b</td>
<td>gen_Tspec_parm_hdr.isph_len32b</td>
</tr>
<tr>
<td>gtspec_m</td>
<td>gen_Tspec_parms.TB_Tspec_m</td>
</tr>
<tr>
<td>gtspec_M</td>
<td>gen_Tspec_parms.TB_Tspec_M</td>
</tr>
<tr>
<td>gtspec_p</td>
<td>gen_Tspec_parms.TB_Tspec_p</td>
</tr>
<tr>
<td>gtspec_parmno</td>
<td>gen_Tspec_parm_hdr.isph_parm_num</td>
</tr>
<tr>
<td>gtspec_r</td>
<td>gen_Tspec_parms.TB_Tspec_r</td>
</tr>
</tbody>
</table>

**Integrated services data structures and macros formats for controlled-load service**

The following formats for controlled-load service apply to the integrated services data structures and macros:

- Type `CL_flowspec_t` is defined by typedef as a structure to represent a controlled-load flowspec, and has at least the following members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_spec_parms</td>
<td>TB_Tsp_parms_t</td>
<td></td>
</tr>
<tr>
<td>CL_spec_parm_hdr</td>
<td>IS_parm_hdr_t (IS_WKP_TB_TSPEC)</td>
<td></td>
</tr>
<tr>
<td>CL_spec_serv_hdr</td>
<td>IS_serv_hdr_t (CONTROLLED_LOAD_SERV, 0,len)</td>
<td></td>
</tr>
</tbody>
</table>
• The following macros are defined with the values given below:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLspec_b</td>
<td>CL_spec_parms.TB_Tspec_b</td>
</tr>
<tr>
<td>CLspec_flags</td>
<td>CL_spec_parm_hdr.isph_flags</td>
</tr>
<tr>
<td>CLspec_len</td>
<td>(sizeof(CL_flowspec_t) - sizeof(IS_serv_hdr_t))</td>
</tr>
<tr>
<td>CLspec_len32b</td>
<td>CL_spec_parm_hdr.isph_len32b</td>
</tr>
<tr>
<td>CLspec_m</td>
<td>CL_spec_parms.TB_Tspec_m</td>
</tr>
<tr>
<td>CLspec_M</td>
<td>CL_spec_parms.TB_Tspec_M</td>
</tr>
<tr>
<td>CLspec_p</td>
<td>CL_spec_parms.TB_Tspec_p</td>
</tr>
<tr>
<td>CLspec_parmno</td>
<td>CL_spec_parm_hdr.isph_parm_num</td>
</tr>
<tr>
<td>CLspec_r</td>
<td>CL_spec_parms.TB_Tspec_r</td>
</tr>
</tbody>
</table>

### Integrated services data structures and macros formats for guaranteed service

The following formats for guaranteed service apply to the integrated services data structures and macros:

• The following enumeration is defined for service-specific parameter identifiers and has at least the following members with the following values:

<table>
<thead>
<tr>
<th>Member</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS_GUAR_RSPEC</td>
<td>130</td>
</tr>
<tr>
<td>GUAR_ADSPARM_C</td>
<td>131</td>
</tr>
<tr>
<td>GUAR_ADSPARM_D</td>
<td>132</td>
</tr>
<tr>
<td>GUAR_ADSPARM_Ctot</td>
<td>133</td>
</tr>
<tr>
<td>GUAR_ADSPARM_Dtot</td>
<td>134</td>
</tr>
<tr>
<td>GUAR_ADSPARM_Csum</td>
<td>135</td>
</tr>
<tr>
<td>GUAR_ADSPARM_Dsum</td>
<td>136</td>
</tr>
</tbody>
</table>

• Type `guar_Rspec_t` is defined by typedef as a structure for guaranteed Rspec parameters, and has at least the following members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guar_R</td>
<td>float32_t</td>
<td>Guaranteed rate in bytes per second</td>
</tr>
<tr>
<td>Guar_S</td>
<td>u_int32_t</td>
<td>Slack term in microseconds</td>
</tr>
</tbody>
</table>

• Type `Guar_flowspec_t` is defined by typedef as a structure to represent a guaranteed flowspec, and has at least the following members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guar_Rspec</td>
<td>guar_Rspec_t</td>
<td>Guaranteed rate in Bytes per second</td>
</tr>
<tr>
<td>Guar_Rspec_hdr</td>
<td>IS_parm_hdr_t</td>
<td>(IS_GUAR_RSPEC)</td>
</tr>
<tr>
<td>Guar_serv_hdr</td>
<td>IS_serv_hdr_t</td>
<td>(GUARANTEED_SERV, 0, length)</td>
</tr>
<tr>
<td>Guar_Tspec_hdr</td>
<td>IS_parm_hdr_t</td>
<td>(IS_WKP_TB_TSPEC)</td>
</tr>
</tbody>
</table>
The following macros are defined with the values given below:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gspec_b</td>
<td>Guar_Tspec_parms.TB_Tspec_b</td>
</tr>
<tr>
<td>Gspec_len</td>
<td>(sizeof(Guar_flowspec_t) - sizeof(IS_serv_hdr_t))</td>
</tr>
<tr>
<td>Gspec_m</td>
<td>Guar_Tspec_parms.TB_Tspec_m</td>
</tr>
<tr>
<td>Gspec_M</td>
<td>Guar_Tspec_parms.TB_Tspec_M</td>
</tr>
<tr>
<td>Gspec_p</td>
<td>Guar_Tspec_parms.TB_Tspec_p</td>
</tr>
<tr>
<td>Gspec_r</td>
<td>Guar_Tspec_parms.TB_Tspec_r</td>
</tr>
<tr>
<td>Gspec_R</td>
<td>Guar_Rspec.Guar_R</td>
</tr>
<tr>
<td>Gspec_R_flags</td>
<td>Guar_Rspec.hdr.isph_flags</td>
</tr>
<tr>
<td>Gspec_R_len32b</td>
<td>Guar_Rspec.hdr.isph_len32b</td>
</tr>
<tr>
<td>Gspec_R_parmno</td>
<td>Guar_Rspec.hdr.isph_parm_num</td>
</tr>
<tr>
<td>Gspec_S</td>
<td>Guar_Rspec.Guar_S</td>
</tr>
<tr>
<td>Gspec_T_flags</td>
<td>Guar_Tspec.hdr.isph_flags</td>
</tr>
<tr>
<td>Gspec_T_len32b</td>
<td>Guar_Tspec.hdr.isph_len32b</td>
</tr>
<tr>
<td>Gspec_T_parmno</td>
<td>Guar_Tspec.hdr.isph_parm_num</td>
</tr>
</tbody>
</table>

Type `Gads_parms_t` is defined by typedef as a structure for guaranteed adspec parameters, and has the following members, which can be followed by override general parameter values:

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gads_Csum</td>
<td>u_int32_t</td>
<td></td>
</tr>
<tr>
<td>Gads_Csum_hdr</td>
<td>IS_parm_hdr_t</td>
<td>(GUAR_ADSPARM_Csum)</td>
</tr>
<tr>
<td>Gads_Ctot</td>
<td>u_int32_t</td>
<td></td>
</tr>
<tr>
<td>Gads_Ctot_hdr</td>
<td>IS_parm_hdr_t</td>
<td>(GUAR_ADSPARM_Ctot)</td>
</tr>
<tr>
<td>Gads_Dsum</td>
<td>u_int32_t</td>
<td></td>
</tr>
<tr>
<td>Gads_Dsum_hdr</td>
<td>IS_parm_hdr_t</td>
<td>(GUAR_ADSPARM_Dsum)</td>
</tr>
<tr>
<td>Gads_Dtot</td>
<td>u_int32_t</td>
<td></td>
</tr>
<tr>
<td>Gads_Dtot_hdr</td>
<td>IS_parm_hdr_t</td>
<td>(GUAR_ADSPARM_Dtot)</td>
</tr>
<tr>
<td>Gads_serv_hdr</td>
<td>IS_serv_hdr_t</td>
<td>(GUARANTEED_SERV, x, len)</td>
</tr>
</tbody>
</table>

**Integrated services data structures and macros basic adspec pieces**

The following basic adspec pieces apply to the integrated services data structures and macros:

- Type `genparm_parms_t` is defined by typedef as a structure for general path characterization parameters, and has at least the following members:
Integrated services flowspec
The following integrated services flowspecs apply to the integrated services data structures and macros:

- Type `IS_specbody_t` is defined by typedef as a structure to represent an integrated services flowspec, and has at least the following members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_spec</td>
<td>CL_flowspec_t</td>
<td>Controlled-load service</td>
</tr>
<tr>
<td>G_spec</td>
<td>Guar_flowspec_t</td>
<td>Guaranteed service</td>
</tr>
<tr>
<td>spec_mh</td>
<td>IS_main_hdr_t</td>
<td></td>
</tr>
<tr>
<td>spec_u</td>
<td>union</td>
<td></td>
</tr>
</tbody>
</table>

- The following macros are defined with the values given below:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISmh_len32b</td>
<td>spec_mh.ismh_len32b</td>
</tr>
<tr>
<td>ISmh_unused</td>
<td>spec_mh.ismh_unused</td>
</tr>
<tr>
<td>ISmh_version</td>
<td>spec_mh.ismh_version</td>
</tr>
</tbody>
</table>

Integrated services tspec
The following integrated services tspecs apply to the integrated services data structures and macros:

- Type `IS_tspbody_t` is defined by typedef as a structure to represent an Integrated Services Tspec, and has at least the following members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>st_mh</td>
<td>IS_main_hdr_t</td>
<td></td>
</tr>
<tr>
<td>tspec_u</td>
<td>union (1)</td>
<td></td>
</tr>
<tr>
<td>gen_stspec</td>
<td>gen_Tspec_t</td>
<td>Generic Tspec</td>
</tr>
</tbody>
</table>
Note:

(1) While service-dependent Tspecs are possible, there are none.

- The following macros are defined with the values given below:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IStmh_len32b</td>
<td>st_mh.ismh_len32b</td>
</tr>
<tr>
<td>IStmh_unused</td>
<td>st_mh.ismh_unused</td>
</tr>
<tr>
<td>IStmh_version</td>
<td>st_mh.ismh_version</td>
</tr>
</tbody>
</table>

**Integrated services adspec**

The following integrated services adspecs apply to the integrated services data structures and macros:

<table>
<thead>
<tr>
<th>Member</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>adspec_genparms</td>
<td>genparm_parms_t</td>
<td>General char parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fragment</td>
</tr>
<tr>
<td>adspec_mh</td>
<td>IS_main_hdr_t</td>
<td>Main header</td>
</tr>
</tbody>
</table>
Chapter 7. X Window System interface in the z/OS Communications Server environment

This topic describes the X Window System application programming interface (API). The X Window System API allows you to write applications in the MVS environment that can be displayed on X11 servers on a TCP/IP-based network, and provides the application with graphics capabilities as defined by the X Window System protocol.

X11 and Motif libraries are based on the X Window System Version 11 Release 6.6 and Motif Version 2.1.30. Applications are supported in 31-bit and 64-bit mode. For compatibility with applications written for prior releases, X11 R6.1 and Motif 1.2 libraries and corresponding header files are also provided.

X Window System and Motif

This topic describes the X Window System API. The X Window System API allows you to write applications in the z/OS UNIX System Services (z/OS UNIX) MVS environment.

The X Window System support includes the following APIs from the X Window System Version 11 Release 6.6:

- X11 Core distribution routines (X11)
- Inter-Client Exchange routines (ICE)
- Session Manager routines (SM)
- X Window System extended routines (Xext) including:
  - XC-MISC: Allows clients to get back ID ranges from the server
  - Big-Requests: Allows large length value in protocol requests
  - Shape: Allows nonrectangular windows
  - Sync: Lets clients synchronize through the X Server
- Authentication functions (Xau)
- X10 compatibility routines (oldX)
- X Toolkit (Xt)
- Utility functions used by Xaw (Xmu)
- Athena Widget set (Xaw)
- Header files needed for compiling X clients
- Selection of standard MIT X clients
- Sample X demonstrations
- Sample Motif demonstrations

The X Window System support provided also includes the APIs based on Motif Release 2.1.30:

- Motif-based widget set (Xm library)
- Motif Resource Manager (Mrm library)
- Motif User Interface language (uil library)
- Motif User Interface Language Compiler
- Header files needed for compiling clients using the Motif-based widget set
**DLL support for the X Window System**

The X Window System and Motif functions are provided as a set of archive files for applications that are statically linked and as a set of DLLs for applications that are dynamically linked. Dynamic linkage is recommended; it results in application binaries that are much smaller. All applications linked using these DLLs must be compiled with the DLL option. The examples shown in "Compiling and linking Motif and X Window System applications" on page 199 assume that c89 is using the z/OS C/C ++ Compiler.

Three sets of DLLs are provided. The first set ensures compatibility with applications compiled with previous releases of the X Window System and Motif. For this set of DLLs applications must be compiled in 31-bit mode with the DLL option; applications cannot be compiled with XPLINK. These DLLs are unchanged from the previous release and are compiled with IBM hexadecimal floating point support. New or changed applications should be migrated to the new X11R6.6 and Motif 2.1.30 versions of the libraries.

The following DLLs are provided to support applications that require X11R6.1 and Motif 1.2 function. These libraries are provided to ensure compatibility of applications written for previous releases of z/OS.

- X11.dll (contains the contents of libX11.a, libXau.a, liboldX.a, and libXext.a)
- SM.dll (contains the contents of libSM.a)
- ICE.dll (contains the contents of libICE.a)
- PEX5.dll (contains the contents of libPEX5.a)
- Xaw.dll (contains the contents of libXaw.a, libXmu.a, and libXt.a)
- Xm.dll (contains the contents of libXm.a and libXt.a)
- Mrm.dll (contains the contents of libMrm.a)
- Uil.dll (contains the contents of libUil.a)

The second set of DLLs provides X11R6.6 and Motif 2.1 function. To use this set of DLLs, the application must be compiled in 31-bit mode, with the DLL option and the XPLINK option. This set of DLLs is compiled with IEEE floating point support. These DLLs do not support applications compiled with enhanced ASCII support. The PEX5 library is no longer supported with these DLLs.

- X11_31.dll (contains the contents of libX11.a, libXau.a, liboldX.a, libXext.a, and libXp.a)
- SM_31.dll (contains the contents of libSM.a)
- ICE_31.dll (contains the contents of libICE.a)
- Xaw_31.dll (contains the contents of libXaw.a, libXmu.a, and libXt.a)
- Xm_31.dll (contains the contents of libXm.a and libXt.a)
- Mrm_31.dll (contains the contents of libMrm.a)
- Uil_31.dll (contains the contents of libUil.a)

The third set of DLLs provides X11R6.6 and Motif 2.1 function in 64-bit addressing mode. To use this set of DLLs, the application must be compiled in 64-bit mode, with the DLL option and the XPLINK option.

- X11_64.dll (contains the contents of libX11.a, libXau.a, liboldX.a, libXext.a, and libXp.a)
- SM_64.dll (contains the contents of libSM.a)
- ICE_64.dll (contains the contents of libICE.a)
- Xaw_64.dll (contains the contents of libXaw.a, libXmu.a, and libXt.a)
xMm_64.dll (contains the contents of libXm.a and libXt.a)
Mrm_64.dll (contains the contents of libMrm.a)
Uil_64.dll (contains the contents of libUil.a)

All DLLs, along with their sidedecks (.x), are symbolically linked from /usr/lib.

Rules:
• An application must use only one set of DLLs. You cannot mix 31-bit and 64-bit DLLs. An application should not attempt to mix old- and new-function DLLs.
• An application should use either the static libraries or the dynamic libraries, not both.

How the X Window System interface works in the MVS environment
The X Window System is a network-transparent protocol that supports windowing and graphics. The protocol is communicated between a client or application and an X server over a reliable bidirectional byte stream. This byte stream is provided by the TCP/IP communication protocol. In the MVS environment, X Window System support consists of a set of application calls that create the X protocol, as requested by the application. This application programming interface allows an application to be created, which uses the X Window System protocol to be displayed on an X server.

In an X Window System environment, the X server is generally located on the workstation, and distributes user input to and accepts requests from various client programs located either on the same system or elsewhere on a network. The X server provides access to the resources that are shared among many X applications, such as the screen, keyboard, mouse, fonts, and graphics contexts. A single X server can control more than one physical screen.

The application program that you create is the client part of a client-server relationship. The communication path from the MVS X Window System application to the server involves the client code and TCP/IP.

The X client code uses sockets to communicate with the X server. Each client can interact with multiple servers, and each server can interact with multiple clients.

If your application is written to the Xlib interface, it calls XOpenDisplay() to start communication with an X server on a workstation. The Xlib code opens a communication path called a socket to the X server, and sends the appropriate X protocol to initiate client-server communication.

The X protocol generated by the X Window System client code uses an ISO Latin-1 encoding for character strings, while the MVS encoding for character strings is EBCDIC. The X Window System client code in the MVS environment automatically transforms character strings from EBCDIC to ISO Latin-1 or from ISO Latin-1 to EBCDIC, as needed.

z/OS UNIX application resource file
The X Window System allows you to modify certain characteristics of an application at run time using application resources. Typically, application resources are set to tailor the appearance and possibly the behavior of an application. The application resources can specify information about an application’s window sizes, placement, coloring, font usage, and other functional details.
In the z/OS UNIX environment, this information can be found in the file
/u/user_id/.Xdefaults

where
/u/user_id

is found from the environment variable home.

**Identifying the target display in z/OS UNIX**
The DISPLAY environment variable is used by the X Window System to identify
the host name of the target display.

The following is the format of the DISPLAY environment variable:

```
host_name:target_server.target_screen
```

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host_name</td>
<td>Specifies the host name or IP address of the host machine on which the X Window System server is running.</td>
</tr>
<tr>
<td>target_server</td>
<td>Specifies the display number on the host machine. This is usually 0, unless the host machine is running multiple X servers.</td>
</tr>
<tr>
<td>target_screen</td>
<td>Specifies the screen to be used on the target server. This is optional and defaults to 0.</td>
</tr>
</tbody>
</table>

For more information about resolving a host name to an IP address, see the z/OS
XL C/C++ Programming Guide.

**X Window System programming considerations**
The X Window System toolkit includes files that define two macros for obtaining
the offset of fields in an X Window System Toolkit structure, XtOffset, and
XtOffsetOf. Programs written for, or ported to, z/OS UNIX MVS must use the
XtOffsetOf macro for this purpose.

**Porting Motif applications to z/OS UNIX MVS**
Some Motif widget and gadget resources have the type KeySym. In an
ASCII-based system the KeySym is the same as the ASCII character value. For
example, the character ’F’ has the ASCII hexadecimal value 46 and the KeySym
hexadecimal value 46.

However, on z/OS UNIX MVS, the character value of ’F’ is hexadecimal C6, while
the KeySym hexadecimal value is still 46. Remember to use true KeySym values
when specifying resources of type KeySym, whether in a defaults file or in a
function call.

In some cases, an X Window System server may have clients that are not running
on z/OS UNIX MVS. If a z/OS UNIX MVS X Window System application sends
nonstandard properties that contain text strings to the X Window System server,
and these properties might be accessed by clients that are not running on z/OS
UNIX MVS, the strings should be translated. The translation should be to the
server default character set before transmission to the server and to the appropriate
host character set when retrieved from the server. This translation is an application
responsibility.
Compiling and linking Motif and X Window System applications

The z/OS UNIX c89 or make commands should be used to compile and link X Window System and Motif programs. The following example shows how to use the c89 command to compile an X Window System program, xxx, which uses the Athena widget set, and create the executable file xxx. All code that uses the X Window System and Motif libraries must be compiled with the DLL option even if static linking is used.

c89 -o xxx -Wc,dll,xlink -Wl,xlink xxx.c /usr/lib/Xaw_31.x /usr/lib/SM_31.x
/usr/lib/ICE_31.x /usr/lib/X11_31.x

The following example shows how to compile the program xxx for use with the 64-bit DLLs. LP64 also requires the use of XPLINK.

c89 -o xxx -Wc,dll,xlink,LP64 -Wl,xlink xxx.c /usr/lib/Xaw_64.x /usr/lib/SM_64.x
/usr/lib/ICE_64.x /usr/lib/X11_64.x

The following example shows how to use the c89 command to compile an X Window System program, yyy, which uses the Motif widget set, and create an executable file yyy:

c89 -o yyy -Wc,dll,xlink -Wl,xlink yyy.c /usr/lib/Xm_31.x /usr/lib/SM_31.x
/usr/lib/ICE_31.x /usr/lib/X11_31.x

The following example shows how to use the c89 command to compile an X Window System program, yyy, which uses the Motif widget set, and create an executable file yyy. This example links with the previous function libraries (X 6.1 and Motif 1.2). You must explicitly tell the compiler where to pick up the header files for the previous function libraries with the -I option.

c89 -o yyy -Wc,dll -Wl,xlink yyy.c -I/usr/include/lpp/tcpip/X11R6/include
/usr/lib/Xm.x /usr/lib/SM.x /usr/lib/ICE.x /usr/lib/X11.x

For examples of the input to the make command, see the Makefile in each of these subdirectories:
/usrlpp/tcpip/X11R6/Xamples/demos
/usrlpp/tcpip/X11R6/Xamples/clients
/usrlpp/tcpip/X11R66/Xamples/demos
/usrlpp/tcpip/X11R66/Xamples/clients
/usrlpp/tcpip/X11R66/Xamples/motif

To build the samples for X11 and Motif, set the following environment variables:
• export _C89_CCMODE=1
• export _CC_CCMODE=1

Setting these environment variables causes the c89 and cc commands to relax requirements on the order of options and operands and makes the porting of makefiles from other platforms easier.

For more information about the z/OS UNIX c89 and make commands, see the z/OS UNIX System Services Command Reference.

Running an X Window System or Motif DLL-enabled application

When running an X Window System or Motif DLL-enabled application, ensure that the LIBPATH environment variable includes /usr/lib.
# X Window System environment variables

Table 4 provides a list of environment variables that can be set to affect the behavior of X Window System applications.

**Table 4. Environment variables for the X Window System interface**

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISPLAY</td>
<td>Contains the name of the display to be used. There is no default value. See note 1.</td>
</tr>
<tr>
<td>ICEAUTHORITY</td>
<td>This variable identifies where the authentication information is located.</td>
</tr>
<tr>
<td>LANG</td>
<td>Determines the locale category for native language, local customs, and coded character set in the absence of the LC_ALL and other LC_* (LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, LC_NUMERIC, LC_TIME) environment variables. See note 2.</td>
</tr>
<tr>
<td>LC_CTYPE</td>
<td>Determine the locale category for character handling functions, such as tolower(), toupper(), and isalpha(). This environment variable determines the interpretation of sequences of bytes of text data as characters (for example, single as opposed to multibyte characters), the classification of characters (for example, alpha, digit, graph), and the behavior of character classes.</td>
</tr>
<tr>
<td>SESSION_MANAGER</td>
<td>If defined, causes a Session Shell widget to connect to a session manager. There is no default value.</td>
</tr>
<tr>
<td>XAPPLRESDIR</td>
<td>Specifies the directory to search for files that contain application defaults.</td>
</tr>
<tr>
<td>XAUTHORITY</td>
<td>Specifies the name of the authority file on the local host.</td>
</tr>
<tr>
<td>XCMSDB</td>
<td>Specifies the name of a color name database file.</td>
</tr>
<tr>
<td>XENVIRONMENT</td>
<td>Contains the full path name of the file that contains resource defaults. There is no default value.</td>
</tr>
<tr>
<td>XFILESEARCHPATH</td>
<td>Used by XtResolvePathname as a default path. There is no default value.</td>
</tr>
<tr>
<td>XKEYSYMDB</td>
<td>Specifies the location of the XKEYSYMDB.</td>
</tr>
<tr>
<td>XLOCALEDIR</td>
<td>Specifies the directory to search for locale files. The default value is /usr/lib/X11/locale.</td>
</tr>
<tr>
<td>XMODIFIERS</td>
<td>Can be set to contain additional information important for the current locale setting. See note 3.</td>
</tr>
<tr>
<td>XUSERFILESEARCHPATH</td>
<td>Specifies where to find the personal X resources files used to configure an application.</td>
</tr>
<tr>
<td>XWTRACE</td>
<td>Controls the generation of socket-level communications traces between Xlib and the X Window System server. These traces are as follows:</td>
</tr>
<tr>
<td></td>
<td>- XWTRACE undefined or 0: No trace generated</td>
</tr>
<tr>
<td></td>
<td>- XWTRACE=1: Error messages</td>
</tr>
<tr>
<td></td>
<td>- XWTRACE&gt;=2: API function tracing for TRANS functions</td>
</tr>
<tr>
<td></td>
<td>There is no default value. The output is sent to stderr.</td>
</tr>
</tbody>
</table>
Table 4. Environment variables for the X Window System interface (continued)

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| XWTRACELENCE         | If defined, causes a trace of locale-sensitive routines. Possible values are:  
  • XWTRACELENCE undefined or 0: No trace generated  
  • XWTRACELENCE=1: Error messages  
  • XWTRACELENCE>=2: All available trace information  
  There is no default value. The output is sent to stderr.  
  See note 4. |

Windows

Notes:

1. In the following example, royal.csc.ibm.com is the name of the workstation running the X Window System server. The display is indicated by :0.0, and is specified this way in almost all cases.

   export DISPLAY=royal.csc.ibm.com:0.0

2. This can be used by applications to determine the language to use for error messages, instructions, collating sequences, date formats, and so on.

3. Typically set to @im=<input-method> to enable a particular input method.

4. If XWTRACELENCE is defined, a routine flow trace is generated. If XWTRACELENCE=2, more detailed information is provided.

**Motif environment variables**

Table 5 provides a list of environment variables that can be set to affect the behavior of Motif applications.

Table 5. Environment variables for Motif

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTICONBMSEARCHPATH</td>
<td>Contains the search path for icons on monochrome displays.</td>
</tr>
<tr>
<td>DTICONSEARCHPATH</td>
<td>Contains the search path for icons on color displays.</td>
</tr>
<tr>
<td>KBD_LANG</td>
<td>Specifies the value of LANG for applicable languages.</td>
</tr>
<tr>
<td>RESOURCE_NAME</td>
<td>Used by XtOpenDisplay as an alternative specification of an application name. There is no default value.</td>
</tr>
<tr>
<td>UILTRACE</td>
<td>Specifies whether UIL trace is on or off.</td>
</tr>
<tr>
<td>WMDDPATH</td>
<td>Specifies the WMD path.</td>
</tr>
<tr>
<td>XAPPLRESDIR</td>
<td>Specifies the directory to search for files that contain application defaults.</td>
</tr>
<tr>
<td>XMBINDDIR</td>
<td>Specifies the location of the xmbind.alias file.</td>
</tr>
<tr>
<td>XMICONBMSEARCHPATH</td>
<td>Used to locate desktop icons.</td>
</tr>
<tr>
<td>XMICONSEARCHPATH</td>
<td>Used to locate bitmap (2-color) desktop icons.</td>
</tr>
<tr>
<td>XPROPFORMATS</td>
<td>Specifies the name of the file from which additional formats are to be obtained.</td>
</tr>
</tbody>
</table>
EBCDIC/ASCII translation in the X Window System

Because the X Window System was designed primarily for an ASCII-based environment and z/OS UNIX MVS uses EBCDIC, it is necessary to provide translations between various servers and MVS clients. Translations must also be provided between locale-based coded character sets in z/OS UNIX MVS and the coded character sets used on the X Window System server. The following topics describe how this is accomplished.

EBCDIC/ASCII translation in the X Window System: Locale independent translation

All arguments for X Window System functions that are specified to be in the Host Portable Character Set are translated between EBCDIC and ASCII by a translation between code page IBM-1047 and code page ISO8859-1. All single-byte character set string arguments to X Window System function calls that are not locale-dependent (do not have names starting with Xmb or Xwc) are also translated between EBCDIC and ASCII using code page IBM-1047 and ISO8859-1. In addition, properties of type STRING passed to XChangeProperty are translated to ASCII before transmission to the server.

These translations are performed on data being transmitted to the server and on data received from the server that is being returned to the application.

The arguments to X Window System functions of the type XChar2b are not translated. This includes such functions as XDraw16, XDrawText16, and XTextExtents16.

EBCDIC/ASCII translation in the X Window System: Locale dependent translation

The string arguments to X Window System functions with names starting with Xmb or Xwc are translated between the current MVS z/OS UNIX locale code set (the value returned by nl_info(CODESET)) and the current XLocale. The MVS z/OS UNIX locale is mapped to the XLocale by an entry in /usr/lib/X11/locale/locale.alias. Properties passed to XChangeProperty with a type of the locale-encoding name atom are translated from the MVS z/OS UNIX locale-coded character set to the XLocale coded character set.

XTextProperty with COMPOUND_TEXT encoding

The XTextProperty structure returned by XmbTextListToProperty and XwcTextListToProperty has its property data translated from the MVS z/OS UNIX locale coded character set to the XLocale coded character set if the XTextProperty encoding is COMPOUND_TEXT. Similarly the reverse translation is performed for XmbTextPropertyToTextList and XwcTextPropertyToTextList if the XTextProperty has the encoding COMPOUND_TEXT.

Standard clients supplied with MVS z/OS UNIX X Window System support

The following standard clients are provided in /usr/lpp/tcpip/X11R6/Xamples/clients:

<table>
<thead>
<tr>
<th>Client</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>appres</td>
<td>Lists application resource database</td>
</tr>
<tr>
<td>atobm</td>
<td>Bit map conversion utility</td>
</tr>
<tr>
<td>bitmap</td>
<td>Bit map editor</td>
</tr>
<tr>
<td>Client</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>bmtoa</td>
<td>Bit map conversion utility</td>
</tr>
<tr>
<td>editres</td>
<td>Resource editor</td>
</tr>
<tr>
<td>iceauth</td>
<td>ICE authority file utility</td>
</tr>
<tr>
<td>oclock</td>
<td>Displays time of day</td>
</tr>
<tr>
<td>xauth</td>
<td>X authority file utility</td>
</tr>
<tr>
<td>xclipboard</td>
<td>Clipboard utility</td>
</tr>
<tr>
<td>xcutsel</td>
<td>Clipboard utility</td>
</tr>
<tr>
<td>clock</td>
<td>Analog and digital clock for X</td>
</tr>
<tr>
<td>xdpynfo</td>
<td>Display information utility for X</td>
</tr>
<tr>
<td>xfd</td>
<td>X font display utility</td>
</tr>
<tr>
<td>xlogo</td>
<td>Displays X logo</td>
</tr>
<tr>
<td>xlsatoms</td>
<td>Lists interned atoms defined on server</td>
</tr>
<tr>
<td>xlsclients</td>
<td>Lists client applications running on a display</td>
</tr>
<tr>
<td>xmag</td>
<td>Magnifies part of screen</td>
</tr>
<tr>
<td>xlsfonts</td>
<td>Lists server fonts</td>
</tr>
<tr>
<td>xprop</td>
<td>Property displayer for X</td>
</tr>
<tr>
<td>xwininfo</td>
<td>Window information utility for X</td>
</tr>
<tr>
<td>xwd</td>
<td>Dumps an image of an X window</td>
</tr>
<tr>
<td>xwud</td>
<td>Displays dumped image for X</td>
</tr>
<tr>
<td>xfindproxy</td>
<td>Find an LBX proxy</td>
</tr>
</tbody>
</table>

Use the `man` command to display information about these clients as shown below:
`man -M /usr/lpp/tcpip/X11R6/Xamples/man client`

**Demonstration programs supplied with MVS z/OS UNIX**

**X Window System support**

The following demonstration programs are supplied in `/usr/lpp/tcpip/X11R6/Xamples/demos`:

- **xsamp1**
  - Uses only Xlib

- **xsamp2**
  - Uses Athena widget set

- **xsamp3**
  - Uses Motif widget set

- **pexsamp**
  - Uses PEX5 library

**X Window System and Motif files locations**

The following topics provide X Window System and Motif locations.

**Previous function X11R6.1 and Motif 1.2**

- Previous function X11R6.1 and Motif 1.2 static libraries for 31-bit applications.
  Applications that want to link with these libraries must use the `-L` flag on the `cc` or `c89` command to specify the library directory.
• Previous function X11R6.1 and Motif 1.2 dynamic link libraries (DLLs); 31-bit, non-XPLINK:
  /usr/lib/ICE.dll -> symlink to /usr/lpp/tcpip/X11R6/lib/ICE.dll
  /usr/lib/SM.dll -> symlink to /usr/lpp/tcpip/X11R6/lib/SM.dll
  /usr/lib/Xaw.dll -> symlink to /usr/lpp/tcpip/X11R6/lib/Xaw.dll

• Header files for previous function X11R6.1 and Motif 1.2:
  /usr/lpp/tcpip/X11R6/include/X11
  /usr/lpp/tcpip/X11R6/include/X11/ICE
  /usr/lpp/tcpip/X11R6/include/X11/PEX5
  /usr/lpp/tcpip/X11R6/include/X11/SM
  /usr/lpp/tcpip/X11R6/include/X11/Xaw
  /usr/lpp/tcpip/X11R6/include/X11/Xmu
  /usr/lpp/tcpip/X11R6/include/X11/bitmaps
  /usr/lpp/tcpip/X11R6/include/X11/extensions
  /usr/lpp/tcpip/X11R6/include/Mrm (motif header files)
  /usr/lpp/tcpip/X11R6/include/Xm (motif header files)
  /usr/lpp/tcpip/X11R6/include/Uil (Uil header files)

• Other utilities and data files for the previous function X11R6.1 and Motif 1.2:
  /usr/lpp/tcpip/bin/X11/uil (uil compiler)
  /usr/lpp/tcpip/X11R6/lib/X11/locale (locale data files)
  /usr/lpp/tcpip/X11R6/lib/X11/XErrorDB (X Error message database)
  /usr/lpp/tcpip/X11R6/lib/X11/XKeysymDB (X keysym Database)
  /usr/lpp/tcpip/X11R6/lib/X11/app-defaults/ (application default files)

• Examples included for X11R6.1 and Motif 1.2:
  /usr/lpp/tcpip/X11R6/Xamples/man/cat1/ (man pages for Xamples programs)
  /usr/lpp/tcpip/X11R6/Xamples/demos/ (demonstration programs)
  /usr/lpp/tcpip/X11R6/Xamples/clients/ (selected standard clients)

New function X11R6.6 and Motif 2.1.30

• New function X11R6.6 and Motif 2.1 static libraries for 31-bit and 64-bit applications (these libraries are all XPLINK):

Notes:
1. PEX is no longer supported in these libraries.
2. Xp is a new library.
New function X11R6.6 and Motif 2.1 31-bit dynamic link libraries (DLLs):

- /usr/lib/libXm.a -> /usr/lpp/tcpip/X11R66/lib/libXm.a
- /usr/lib/libMrm.a -> /usr/lpp/tcpip/X11R66/lib/libMrm.a
- /usr/lib/libUil.a -> /usr/lpp/tcpip/X11R66/lib/libUil.a

- New function X11R6.6 and Motif 2.1 64-bit dynamic link libraries (DLLs):

- /usr/lib/Mrm_31.dll -> /usr/lpp/tcpip/X11R66/lib/Mrm_31.dll
- /usr/lib/Xm_31.dll -> /usr/lpp/tcpip/X11R66/lib/Xm_31.dll

- New function X11R6.6 and Motif 2.1 64-bit dynamic link libraries (DLLs):

- /usr/lib/X11_64.dll -> /usr/lpp/tcpip/X11R66/lib/X11_64.dll
- /usr/lib/ICE_64.dll -> /usr/lpp/tcpip/X11R66/lib/ICE_64.dll
- /usr/lib/SM_64.dll -> /usr/lpp/tcpip/X11R66/lib/SM_64.dll
- /usr/lib/Xaw_64.dll -> /usr/lpp/tcpip/X11R66/lib/Xaw_64.dll
- /usr/lib/Mrm_64.dll -> /usr/lpp/tcpip/X11R66/lib/Mrm_64.dll
- /usr/lib/Uil_64.dll -> /usr/lpp/tcpip/X11R66/lib/Uil_64.dll
- /usr/lib/Xm_64.dll -> /usr/lpp/tcpip/X11R66/lib/Xm_64.dll

- Header files for X11R6.6 and Motif 2.1:

- /usr/include/X11/ -> /usr/lpp/tcpip/X11R66/include/X11 (header files)
- /usr/include/X11/ICE -> /usr/lpp/tcpip/X11R66/include/X11/ICE (ICE specific header files)
- /usr/include/X11/SM -> /usr/lpp/tcpip/X11R66/include/X11/SM (SM specific header files)
- /usr/include/X11/Xaw -> /usr/lpp/tcpip/X11R66/include/X11/Xaw (Xaw specific header files)
- /usr/include/X11/Xmu -> /usr/lpp/tcpip/X11R66/include/X11/Xmu (Xmu specific header files)
- /usr/include/X11/extensions -> /usr/lpp/tcpip/X11R66/include/X11/extensions (extensions specific header files)
- /usr/include/X11/bitmaps -> /usr/lpp/tcpip/X11R66/include/X11/bitmaps (bitmaps for samples)
- /usr/include/Mrm -> /usr/lpp/tcpip/X11R66/include/Mrm (motif header files)
- /usr/include/Xm -> /usr/lpp/tcpip/X11R66/include/Xm (motif header files)
- /usr/include/X11/uil -> /usr/lpp/tcpip/X11R66/include/uil (Uil header files)

- Other utilities and Data files for the new function X11R6.6 and Motif 2.1:

- /bin/X11/uil -> /usr/lpp/tcpip/bin/X1166/uil (31-bit uil compiler)
- /bin/X11/uil64 -> /usr/lpp/tcpip/bin/X1166/uil64 (64-bit uil compiler)

- Examples included for X11R6.6 and Motif 2.1:

- /usr/lpp/tcpip/X11R66/Xamples/man/cat1 (man pages for Xamples programs)
- /usr/lpp/tcpip/X11R66/Xamples/demos/ (demonstration programs)
- /usr/lpp/tcpip/X11R66/Xamples/clients/ (selected standard clients)
- /usr/lpp/tcpip/X11R66/Xamples/motif (selected Motif examples)
Chapter 8. Remote procedure calls in the z/OS
Communications Server environment

This topic describes the high-level remote procedure calls (RPCs) implemented in TCP/IP including the RPC programming interface to the C language and communication between processes.

The RPC protocol permits remote execution of subroutines across a TCP/IP network. RPC, together with the eXternal Data Representation (XDR) protocol, defines a standard for representing data that is independent of internal protocols or formatting. RPCs can communicate between processes on the same or different hosts.

For more information about the RPC and XDR protocols, see the following:
• Sun Microsystems publication, Networking on the Sun Workstation: Remote Procedure Call Programming Guide
• RFC 1831
• RFC 1832
  XDR: External Data Representation Standard, R. Srinivasan, August 1995

See Appendix E, “Related protocol specifications,” on page 743 for information about accessing RFCs.

Tips:
• RPC is supported using the C/370 programming language and the TCP/IP C socket API. For more information about the C/370 socket API, see the z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference
• For more information about z/OS UNIX System Services sockets, see the z/OS XL C/C++ Run-Time Library Reference

The RPC interface

The RPC interface enables programmers to write distributed applications using high-level RPCs rather than lower-level calls based on sockets.

When you use RPCs, the client communicates with a server. The client invokes a procedure to send a call message to the server. When the message arrives, the server calls a dispatch routine, and performs the requested service. The server sends back a reply message, after which the original procedure call returns to the client program with a value derived from the reply message.

See Sample RPC programs for sample RPC client, server, and raw data stream programs. Figure 2 on page 208 and Figure 3 on page 209 provide an overview of the high-level RPC client and server processes from initialization through cleanup.
Figure 2. Remote procedure call (client)
Portmapper and rpcbind

Portmapper and rpcbind are the software that supply client programs with information about server programs. Portmapper returns port numbers of server programs and rpcbind returns universal addresses. A universal address is a text string representation of the transport dependent address. A universal address for rpcbind is defined in RFC 3530 as a text string of the IP address, a dot, then the
text string of the two octets of the port number. The following are examples of universal addresses for port 1024 (port 1024 = port 0x400):

- 9.1.1.4.0
- ::FFFF:9.1.1.1.4.0
- 2001:0DB8::10:1:1.4.0

You can communicate between different computer operating systems when messages are directed to port numbers or universal addresses rather than to targeted remote programs. Clients contact server programs by sending messages to the port numbers or universal addresses of remote processes. Because you make requests to the port number or universal address of a server rather than directly to a server program, client programs need a way to find this information about the server programs they are calling. Portmapper and rpcbind standardize the way clients locate information about the server programs that are supported on a network.

Portmapper and rpcbind use well-known port 111. See Appendix B, “Well-known port assignments,” on page 675, for other well-known TCP and UDP port assignments.

The port-to-program information maintained by portmapper is called the portmap. Clients ask portmapper or rpcbind about entries for servers on the network. Servers contact portmapper or rpcbind to add or update entries to the portmap.

**Contacting portmapper or rpcbind**

To find the port or universal addresses of a remote program, the client sends an RPC to well-known port 111 of the server’s host. If the server listening on port 111 (rpcbind or portmapper) has an entry for the remote program, it provides the port number or universal addresses in a return RPC. The client then contacts the remote program by sending an RPC to the port number or universal addresses provided.

Clients can save information about recently called remote programs to avoid having to contact portmapper or rpcbind for each request to a server. Some of the RPC function calls automatically contact portmapper or rpcbind on behalf of the client. This eliminates the need for the application code to perform this task.

To see all the servers currently registered with RPC binding protocol Version 2 with portmapper or rpcbind, use the RPCINFO command as follows:

```
RPCINFO -p host_name
```

For details about rpcbind, see the MODIFY command: RPCBIND information in the z/OS Communications Server: IP System Administrator's Commands. For more information about rpcinfo and portmapper, see the Rpcinfo information in the z/OS Communications Server: IP System Administrator's Commands.

**Portmapper and rpcbind target assistance**

Portmapper and rpcbind assist clients in contacting server programs. Either portmapper or rpcbind can be used on the same host, but not both. If the client sends an RPC with the target program number, version number, procedure number, and arguments to the server listening on port 111 (rpcbind or portmapper), that server locates the target server in its list of registered servers and passes the client’s message to the target server. When the target server returns information to portmapper or rpcbind, the information is passed to the client along
with the port number (or universal address, if rpbind is being used) of the remote program. The client can then contact the server directly.

**Requirements:** The following apply when the rpbind server runs on a multilevel secure host.

- The rpcinfo utility issues a target assistance request on behalf of the user who invoked it when rpcinfo is invoked with the -b parameter. When the SAF profile BPX.POE is defined on your host and rpbind is started, the rpbind user ID must be granted at least READ access to the profile to enable rpbind to respond to `rpcinfo -b` requests.
- When the SAF profile BPX.POE is defined in class FACILITY and the rpbind server is in use, the rpbind user ID must be granted at least READ access to the profile to enable the server to support target assistance requests.

**Rules:**

- The target assistance RPCs: PMAPPROC_CALLIT, RPCBPROC_CALLIT, RPCBPROC_BCAST, and RPCBPROC_INDIRECT, are defined in RFC 1833: *Binding Protocols for ONC RPC*.
- The following RPC library routines issue target assistance requests on behalf of the calling application: pmap_rmtcall() and clnt_broadcast().

### Registering with rpbind

RPC applications register with rpbind by sending an RPCBPROC_SET or PMAPPROC_SET RPC to rpbind, or by invoking an RPC library routine that sends one of these RPCs to rpbind on its behalf.

**Requirements:**

- Your registration request must originate from an IP address on the host where rpbind is running.
- When the SAF profile EZB.RPCBIND.sysname.rpcbindname.REGISTRY is defined in the SERVAUTH class, the user ID that is associated with the RPC server that registers with rpbind must be granted at least READ access to the profile. If your application sends a PMAPPROC_SET or RPCBPROC_SET request to rpbind, you must grant the user ID that is associated with your application at least READ access to the profile when the profile is defined.
- If your server registers an IPv4 IP address, you must register the address as an IPv4 address rather than an IPv4-mapped IPv6 address.

The following example assumes that your server is listening on IP address 1.2.3.4 and port 1024 and that the server uses stream sockets. In the rpcb specified with the RPCBPROC_SET procedure, specify the following rcb field values:

- `r_addr = 1.2.3.4.0` instead of `::FFFF:1.2.3.4.0`
- `r_netid = tcp` instead of `tcp6`

- If the following conditions apply to your server, you should register your application with both the IPv4 address, INADDR_ANY, and the IPv6 unspecified address (in6addr_any):
  - The server is listening on an AF_INET6 socket bound to the IPv6 unspecified address (in6addr_any)
  - The server host has both IPv4 and IPv6 interfaces
  - The server will serve IPv4 clients as well as IPv6 clients

This example assumes that your server uses datagram sockets and is listening to an AF_INET6 socket that is bound to the IPv6 unspecified address (in6addr_any)
on port 2048. The server host has both IPv4 and IPv6 interfaces and the server intends to accept requests from both IPv4 and IPv6 clients.

Register your application twice. In the rpcb that is specified by RPCBPROC_SET, specify the following rpcb field values on the first registration:

- \( r\_addr = 0.0.0.8.0 \)
- \( r\_netid = udp \)

Specify the following rpcb field values on the second registration:

- \( r\_addr = ::0.8.0 \)
- \( r\_netid = udp6 \)

- When processing an RPCBPROC_SET request, rpcbind ignores the \( r\_owner \) field of the input rpcb.

### Deregistering with rpcbind

RPC applications deregister with rpcbind by sending an RPCBPROC_UNSET or PMAPPROC_UNSET RPC request to rpcbind, or by invoking an RPC library routine that sends one of these RPCs to rpcbind on its behalf.

**Requirements:**

- Your deregistration request must originate from an IP address on the local host.
- When the SAF profile EZB.RPCBIND.sysname.rpcbindname.REGISTRY is defined in the SERVAUTH class, the user ID that is associated with the RPC server that deregisters with rpcbind must be granted at least READ access to the profile. If your application sends a PMAPPROC_UNSET or RPCBPROC_UNSET request to rpcbind, you must grant the user ID that is associated with your application at least READ access to the profile when the profile is defined.

### Obtaining address lists from the rpcbind server

RPC binding protocol V4 provides a procedure, RPCBPROC_GETADDRLIST, for obtaining a list of addresses supported by a service. When a client queries the z/OS rpcbind server using the UDP protocol over IPv4 or IPv6 transport, the rpcbind server confines the reply to fit within one UDP IPv4 datagram. To obtain all addresses supported by the service, the client should use TCP protocol when invoking the RPCBPROC_GETADDRLIST procedure. For more information on RPC binding protocol V4 and the RPCBPROC_GETADDRLIST procedure, see RFC 1833. See [Appendix E, “Related protocol specifications,” on page 743](#) for information on accessing RFCs.

**Result:** Your client might not be able to reach every address returned by the RPCBPROC_GETADDRLIST procedure and possibly might not be able to reach the service at all with the information provided by the RPCBPROC_GETADDRLIST procedure. Following are some examples:

- The service might register a specific address that is not reachable from the client.
- If you use UDP to query the rpcbind server, the addresses returned within the span of a single datagram might be unreachable by the client.

**Restriction:** If the service supports private network addresses, rpcbind returns those addresses in an RPCBPROC_GETADDRLIST reply. If your client resides in the private network with the service, your client can use these addresses to contact the service. However, if the service and the client reside in different private networks, unpredictable results will occur. See RFC 1918 for more information about private network addresses. See [Appendix E, “Related protocol specifications,” on page 743](#) for information on accessing RFCs.
**RPC servers in a CINET environment**

**Tip:** This topic applies only if your RPC server application registers with rpcbind. The portmapper does not recognize new stacks that join a CINET environment, so RPC servers that register with the portmapper are not affected.

The rpcbind server recognizes stacks that are started after rpcbind itself is started. If your RPC server does not recognize stacks that are started after your server establishes its listening socket, your server will not accept calls from the new stack. This is true of all servers in a CINET environment, not just RPC servers. If an RPC client reaches the rpcbind server from a newly started stack to obtain the universal address of your server, it is possible that the client will be unable to contact your server (because your server is not accepting connections from the new stack).

To avoid this problem, do the following:

- Avoid starting a new stack after your RPC server is started.
- Always stop and start your RPC server after starting a new stack.
- Code your server to do the following:
  - Detect a stack starting
  - Deregister your application with rpcbind
  - Close your listening socket
  - Establish a new listening socket and ephemeral port
  - Register the new port using rpcbind

For more information about detecting a stack that is starting, see the `setibmsockopt()` -- Set IBM Specific Options Associated with a Socket information in the z/OS XL C/C++ Run-Time Library Reference.

---

Chapter 8. Remote procedure calls in the z/OS Communications Server environment 213
RPCGEN command

**Purpose**
Use the RPCGEN command to generate the code to implement the RPC protocol.

**Format**
```
RPCGEN −c infile 
RPCGEN −h infile 
RPCGEN −l infile 
RPCGEN −m infile 
RPCGEN −o outfile infile 
RPCGEN −s transport infile 
```

**Parameters**
- `-c` Compiles into XDR routines.
- `-h` Compiles into C data definitions (a header file).
- `-l` Compiles into client-side stubs.
- `-m` Compiles into server-side stubs without generating a main routine. This option is useful for callback routines and for writing a main routine for initialization.
- `-o outfile` Specifies the name of the output data set. If none is specified, standard output is used for `-c`, `-h`, `-l`, `-m`, and `-s` modes.
- `infile` Specifies the name of the input data set written in the RPC language. The default is the data specified by the SYSIN DD statement.
- `-s transport` Compiles into server-side stubs, using the given transport. TCP and UDP are the supported transports. You can invoke this option more than once to compile a server that serves multiple transports. By default, RPCGEN creates server stubs that support both TCP and UDP.

RPCGEN is a tool that generates C code to implement an RPC protocol. The input to RPCGEN is a language similar to C, known as RPC language.

RPCGEN `infile` is normally used when you want to generate all four of the following output data sets. For example:
- If the `infile` is named proto.x, RPCGEN generates:
  - A header file called PROTO.H
  - XDR routines called PROTOX.C
  - Server-side stubs called PROTOS.C
  - Client-side stubs called PROTOC.C
- If the `infile` is named USERA.RPC.SOURCE(PROTO), RPCGEN generates:
  - A header file called USERA.RPC.H(PROTO)
- XDR routines called USERA.RPC.C(PROTOX)
- Server-side stubs called USERA.RPC.C(PROTOS)
- Client-side stubs called USERA.RPC.C(PROTOC)

RPCGEN obtains the file names for the C compiler for preprocessing input from the CCRPCGEN CLIST, which must be customized similar to the C installation procedure. For installation using the C/C++ compiler, the following would be an example of the values for the statements in CCRPCGEN that are used by RPCGEN:

```plaintext
SET CHD = &STR(CBC) /* PREFIX FOR SYSTEM FILES */
SET CVER = &STR( ) /* VERSION OF COMPILER */
SET COMPL = &STR(SCCNCMP) /* C COMPILER MODULES */
SET EDCMSGS = &STR(SCBCDMSG) /* C COMPILER MESSAGES */
SET LANG = &STR(CBCLMSGS) /* MESSAGE LANGUAGE */
SET SCEEHDRS = &STR(SCEEH) /* C SYSTEM HEADER FILES */
SET CMOD = &STR(CCNDRVR) /* C COMPILER EXECUTABLE MODULE */
SET WORKDA = &STR(SYSDA) /* UNIT TYPE FOR WORK FILES */
SET WRKSPC = &STR(1,1) /* CYLS ALLOCATED FOR WORK FILES */
```

The CCRPGEN clist must reside in the SYSPROC concatenation.

Notes:
1. A temporary file called PROTO.EXPANDED is created by the RPCGEN command. During normal operation, this file is also subsequently erased by the RPCGEN command.
2. The code generated by RPCGEN is not suitable for input to a C++ compiler.

For more information about the RPCGEN command, see the Sun Microsystems publication, *Network Programming*. 

Chapter 8. Remote procedure calls in the z/OS Communications Server environment 215
clnt_stat enumerated type

The clnt_stat enumerated type is defined in the CLNT.H file.

RPCs frequently return information in the form of a clnt_stat enumerated value. The following is the format and a description of the clnt_stat enumerated type:

```c
enum clnt_stat {
    RPC_SUCCESS=0, /* call succeeded */
    /*
     * local errors
     */
    RPC_CANTENCODERARGS=1, /* can't encode arguments */
    RPC_CANTDECODERES=2, /* can't decode results */
    RPC_CANTSEND=3, /* failure in sending call */
    RPC_CANTRECV=4, /* failure in receiving result */
    RPC_TIMEDOUT=5, /* call timed out */
    /*
     * remote errors
     */
    RPC_VERSMISMATCH=6, /* RPC versions not compatible */
    RPC_AUTHERROR=7, /* authentication error */
    RPC_PROGUNAVAIL=8, /* program not available */
    RPC_PROGVERSMISMATCH=9, /* program version mismatched */
    RPC_PROCUNAVAIL=10, /* procedure unavailable */
    RPC_CANTDECODERARGS=11, /* decode arguments error */
    RPC_SYSTEMERROR=12, /* generic "other problem" */
    /*
     * callrpc errors
     */
    RPC_UNKNOWNHOST=13, /* unknown host name */
    /*
     * create errors
     */
    RPC_PMAPFAILURE=14, /* the pmapper failed in its call */
    RPC_PROGNOTREGISTERED=15, /* remote program is not registered */
    /*
     * unspecified error
     */
    RPC_FAILED=16
};
```

Porting RPC applications

This topic contains information about porting RPC applications.

Remapping file names with MANIFEST.H

To conform to the MVS requirement that MVS data set names be eight characters or less in length, a file called MANIFEST.H remaps the RPC long names to eight-character derived names for internal processing.

The MANIFEST.H header file must be the first include file in the application, and it must be present at compile time. If it is not included, the application will fail to link-edit. If the preprocessor macro MVS is defined when the RPC.H file is included, RPC.H will implicitly include MANIFEST.H.

Note: #define Resolve_Via_Lookup must be specified before #include manifest.h to enable the following socket calls: endhostent(), gethostent(), gethostbyaddr(), gethostbyname(), and sethostent().
Accessing system return messages

To access system return values, you need only use the ERRNO.H include statement supplied with the compiler. To access network return values, you must add the following include statement:

```c
#include <tcperrno.h>
```

Printing system return messages

To print only system errors, use perror(), a procedure available in the C compiler run-time library. To print both system and network errors, use tcperror(), a procedure included with TCP/IP.

Enumerations

Both xdr_enum() and xdr_union() are macros to account for varying length enumerations. xdr_enum() and xdr_union cannot be referenced by callrpc(), svc_freargs(), svc_getargs(), or svc_sendreply(). An XDR routine for the specific enumeration or union must be created. For more information, see “xdr_enum()” on page 290.

Header files for remote procedure calls

The following header files are provided with TCP/IP. To compile your program, you must include certain header files; however, not all of them are necessary for every RPC application program.

```c
auth.h authuni.h bsdtime.h bsdtocms.h clnt.h h.h inet.h manifest.h netdb.h pmapcln.h
auth.h autouni.h bsdtime.h bsdtocms.h clnt.h h.h inet.h manifest.h netdb.h pmapcln.h
```

Note: When you compile your application program using RPC, you must include the RPC header files before the X Window System include files.

Compiling and linking RPC applications

You can use several methods to compile, link-edit, and execute your TCP/IP C source program in MVS. This topic contains information about the data sets that you must include to run your C source program under MVS batch, using IBM-supplied cataloged procedures.

The following data set name is used as an example in the sample JCL statements:

```c
USER.MYP REO.G.H
```

Contains user #include files.

Compatibility considerations when compiling and linking RPC applications

Unless noted in z/OS Communications Server: New Function Summary an application program compiled and link edited on a release of z/OS Communications Server IP can be used on higher level releases. That is, the API is upward compatible.
Application programs that are compiled and link edited on a release of z/OS Communications Server IP cannot be used on older releases. That is, the API is not downward compatible.

Sample compile cataloged procedure additions

Include the following in the compile step of your cataloged procedure. Cataloged procedures are included in the IBM-supplied samples for your MVS system.

- Add the following statement as the first //SYSLIB DD statement.
  
  //SYSLIB DD DSN=SEZACMAC,DISP=SHR

- Add the following //USERLIB DD statement.
  
  //USERLIB DD DSN=USER.MYPROG.H,DISP=SHR

Compiling and linking RPC applications: Nonreentrant modules

To compile and link nonreentrant RPC applications, the procedure is similar to the procedure for nonreentrant C applications as described in the topic on nonreentrant modules in the z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference.

One additional JCL statement is needed. Add the following SYSLIB statement after SEZACMTX statement in the link step:

// DD DSN=SEZARPCL,DISP=SHR

Compiling and linking RPC applications: Reentrant modules

To compile and link reentrant RPC applications, the procedure is similar to the procedure for reentrant C applications as described in the topic on reentrant modules in the z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference.

One additional JCL statement is needed. Add the following SYSLIB statement after the SEZARNT1 statement in the prelink-edit step:

// DD DSN=SEZARNT4,DISP=SHR

RPC global variables

These topics describe the three RPC global variables, rpc_createerr, svc_fds, and svc_fdset.
**rpc_createerr**

**Format**

```c
#include <rpc.h>

struct rpc_createerr rpc_createerr;
```

**Usage**

rpc_createerr is a global variable that is set when any RPC client creation routine fails. Use clnt_pcreateerror() to print the message.

**Context**

- clntraw_create()
- clnttcp_create()
- clntudp_create()

svc_fds

**Format**

```c
#include <rpc.h>
int svc_fds;
```

**Usage**

svc_fds is a global variable that specifies the read descriptor bit set on the service machine. This is of interest only if the service programmer decides to write an asynchronous event processing routine; otherwise svc_run() should be used. Writing asynchronous routines in the MVS environment is not simple, because there is no direct relationship between the descriptors used by the socket routines and the event control blocks commonly used by MVS programs for coordinating concurrent activities.

**Rule:** Do not modify this variable.

**Context**

- svc_getreq()
**svc_fdset**

**Format**

```c
#include <rpc.h>

fd_set svc_fdset;
```

**Usage**

`svc_fdset` is a global variable that specifies the read descriptor bit set on the service machine. This is of interest only if the service programmer decides to write an asynchronous event processing routine; otherwise `svc_run()` should be used. Writing asynchronous routines in the MVS environment is not simple, because there is no direct relationship between the descriptors used by the socket routines and the event control blocks commonly used by MVS programs for coordinating concurrent activities.

**Rule:** Do not modify this variable.

**Context**

- `svc_getreqset()`
Remote procedure and external data representation calls

These topics provide the syntax, parameters, and other appropriate information for each remote procedure and external data representation call supported by z/OS Communications Server.
auth_destroy()

**Format**

```c
#include <rpc.h>
void
auth_destroy(auth)
AUTH *auth;
```

**Parameters**

- `auth`
  
  Indicates a pointer to authentication information.

**Usage**

The `auth_destroy()` call deletes the authentication information for `auth`. Once this procedure is called, `auth` is undefined.

**Context**

- `authnone_create()`
- `authunix_create()`
- `authunix_create_default()`
authnone_create()

Format

```c
#include <rpc.h>
AUTH *
authnone_create()
```

Parameters

None.

Usage

The authnone_create() call creates and returns an RPC authentication handle. The handle passes the NULL authentication on each call.

Context

- auth_destroy()
- authunix_create()
- authunix_create_default()
authunix_create()

Format

```
#include <rpc.h>
AUTH *
authunix_create(host, uid, gid, len, aup_gids)
char *host;
int uid;
int gid;
int len;
int *aup_gids;
```

Parameters

- **host**
  - Specifies a pointer to the symbolic name of the host where the desired server is located.
- **uid**
  - Specifies the user’s user ID.
- **gid**
  - Specifies the user’s group ID.
- **len**
  - Indicates the length of the information pointed to by `aup_gids`.
- **aup_gids**
  - Specifies a pointer to an array of groups to which the user belongs.

Usage

The `authunix_create()` call creates and returns an authentication handle that contains UNIX-based authentication information.

Context

- `auth_destroy()`
- `authnone_create()`
- `authunix_create_default()`
authunix_create_default()

Format

```
#include <rpc.h>

AUTH *
authunix_create_default()
```

Parameters

None

Usage

The authunix_create_default() call invokes authunix_create() with default parameters.

Context

- auth_destroy()
- authnone_create()
- authunix_create()
callrpc()

Format

```c
#include <rpc.h>

enum clnt_stat
callrpc(host, prognum, versnum, procnum, inproc, in, outproc, out)
    char *host;
    u_long prognum;
    u_long versnum;
    u_long procnum;
    xdrproc_t inproc;
    char *in;
    xdrproc_t outproc;
    char *out;
```

Parameters

- **host**
  Specifies a pointer to the symbolic name of the host where the desired server is located.

- **prognum**
  Identifies the program number of the remote procedure.

- **versnum**
  Identifies the version number of the remote procedure.

- **procnum**
  Identifies the procedure number of the remote procedure.

- **inproc**
  Specifies the XDR procedure used to encode the arguments of the remote procedure.

- **in**
  Specifies a pointer to the arguments of the remote procedure.

- **outproc**
  Specifies the XDR procedure used to decode the results of the remote procedure.

- **out**
  Specifies a pointer to the results of the remote procedure.

Usage

The callrpc() calls the remote procedure described by `prognum`, `versnum`, and `procnum` running on the `host` system. callrpc() encodes and decodes the parameters for transfer.

Notes:

1. `clnt_perrno()` can be used to translate the return code into messages.
2. callrpc() cannot call the procedure xdr_enum. See "xdr_enum()" on page 290 for more information.
3. This procedure uses UDP as its transport layer. See "clntudp_create()" on page 247 for more information.
Return codes
A value of RPC_SUCCESS (0) indicates success; otherwise, an error has occurred as indicated by the value returned. The results of the remote procedure call are returned to \textit{out}.

Context
- clnt_broadcast()
- clnt_call()
- clnt_perrno()
- clntudp_create()
- clnt_sperrno()
- clnt_sperrno()
- xdr_enum()
clnt_broadcast()

Format

```c
#include <rpc.h>
enum clnt_stat
clnt_broadcast(prognum, versnum, procnum, inproc, in, outproc, out, eachresult)
    u_long prognum;
    u_long versnum;
    u_long procnum;
    xdrproc_t inproc;
    char *in;
    xdrproc_t outproc;
    char *out;
    resultproc_t eachresult;
```

Parameters

- **prognum**
  Identifies the program number of the remote procedure.
- **versnum**
  Identifies the version number of the remote procedure.
- **procnum**
  Identifies the procedure number of the remote procedure.
- **inproc**
  Identifies the XDR procedure used to encode the arguments of the remote procedure.
- **in**
  Specifies a pointer to the arguments of the remote procedure.
- **outproc**
  Specifies the XDR procedure used to decode the results of the remote procedure.
- **out**
  Specifies a pointer to the results of the remote procedure; however, the output of the remote procedure is decoded.
- **eachresult**
  Specifies the procedure called after each response.

**Note:** resultproc_t is a type definition.

```c
#include <rpc.h>
typedef bool_t (*resultproc_t)();
```

- **addr**
  Specifies the pointer to the address of the machine that sent the results.

Usage

The clnt_broadcast() call broadcasts the remote procedure described by `prognum`, `versnum`, and `procnum` to all locally connected broadcast networks. Each time clnt_broadcast() receives a response it calls eachresult(). The format of eachresult() is:
Format

```
#include <rpc.h>
bool_t eachresult(out, addr)
char *out;
struct sockaddr_in *addr;
```

Return codes

If eachresult() returns 0, clnt_broadcast() waits for more replies; otherwise, eachresult() returns the appropriate status.

**Note:** Broadcast sockets are limited in size to the maximum transfer unit of the data link.

Context

- callpc()
- clnt_call()
clnt_call()

Format

#include <rpc.h>

enum clnt_stat
clnt_call(clnt, procnum, inproc, in, outproc, out, tout)
CLIENT *clnt;
ulong procnum;
xdrproc_t inproc;
char *in;
xdrproc_t outproc;
char *out;
struct timeval tout;

Parameters

clnt
Specifies the pointer to a client handle that was previously obtained using
clntraw_create(), clnttcp_create(), or clntudp_create().

procnum
Identifies the remote procedure number.

inproc
Specifies the XDR procedure used to encode procnum arguments.

in
Specifies a pointer to the arguments of the remote procedure.

outproc
Indicates the XDR procedure used to decode the remote procedure results.

out
Specifies a pointer to the results of the remote procedure.

tout
Indicates the time allowed for the server to respond.

Usage

The clnt_call() calls the remote procedure (procnum) associated with the client
handle (clnt).

Return codes

A value of RPC_SUCCESS (0) indicates success; otherwise, an error has occurred as
indicated by the value returned. The results of the remote procedure call are
returned in out.

Context

• callrpc()
• clnt_broadcast()
• clnt_geterr()
• clnt_perror()
• clnt_sperror()
• clntraw_create()
- clnttcp_create()
- clntudp_create()
clnt_control()

Format

```
#include <rpc.h>

bool_t
clnt_control(clnt, request, info)
CLIENT *clnt;
int request;
void *info;
```

Parameters

- **clnt**
  
  Indicates the pointer to a client handle that was previously obtained using clntraw_create(), clnttcp_create(), or clntudp_create().

- **request**
  
  Determines the operation (either CLSET_TIMEOUT, CLGET_TIMEOUT, CLGET_SERVER_ADDR, CLSET_RETRY_TIMEOUT, or CLGET_RETRY_TIMEOUT).

- **info**
  
  Indicates the pointer to information used by the request.

Usage

The clnt_control() call performs one of the following control operations:

- Control operations that apply to both UDP and TCP transports:
  
  **CLSET_TIMEOUT**
  
  Sets timeout (info points to the timeval structure).

  **CLGET_TIMEOUT**
  
  Gets timeout (info points to the timeval structure).

  **CLGET_SERVER_ADDR**
  
  Gets server’s address (info points to the sockaddr_in structure).

- UDP only control operations:
  
  **CLSET_RETRY_TIMEOUT**
  
  Sets retry timeout (info points to the timeval structure).

  **CLGET_RETRY_TIMEOUT**
  
  Gets retry timeout (info points to the timeval structure). If you set the timeout using clnt_control(), the timeout parameter to clnt_call() is ignored in all future calls.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_create()
- clnt_destroy()
- clntraw_create()
- clnttcp_create()
- clntudp_create()
clnt_create()

Format

```c
#include <rpc.h>
CLIENT *
clnt_create(host, prognum, versnum, protocol)
char *host;
unsigned long prognum;
unsigned long versnum;
char *protocol;
```

Parameters

- **host**
  - Indicates the pointer to the name of the host at which the remote program resides.

- **prognum**
  - Specifies the remote program number.

- **versnum**
  - Specifies the version number of the remote program.

- **protocol**
  - Indicates the pointer to the protocol, which can be either tcp or udp.

Usage

The clnt_create() call creates an RPC client transport handle for the remote program specified by (`prognum`, `versnum`). The client uses the specified protocol as the transport layer. Default timeouts are set, but they can be modified using clnt_control().

Return codes

NULL indicates failure.

Context

- clnt_control()
- clnt_destroy()
- clnt_pcreateerror()
- clnt_spcreateerror()
- clnt_sperror()
- clntraw_create()
- clnttcp_create()
- clntudp_create()
clnt_destroy()

Format

```c
#include <rpc.h>
void
clnt_destroy(clnt)
    CLIENT *clnt;
```

Parameters

clnt

Specifies the pointer to a client handle that was previously created using
clntudp_create(), clnttcp_create(), or clntraw_create().

Usage

The clnt_destroy() call deletes a client RPC transport handle. This procedure
involves the deallocation of private data resources, including clnt. Once this
procedure is used, clnt is undefined. If the RPC library opened the associated
sockets, it also closes them. Otherwise, the sockets remain open.

Context

- clnt_control()
- clnt_create()
- clntraw_create()
- clnttcp_create()
- clntudp_create()
clnt_freeres()

Format

```c
#include <rpc.h>
bool_t
clnt_freeres(clnt, outproc, out)
CLIENT *clnt;
xdrproc_t outproc;
char *out;
```

Parameters

- **clnt**
  Indicates the pointer to a client handle that was previously obtained using clnt_create(), clntraw_create(), clnttcp_create(), or clntudp_create().

- **outproc**
  Specifies the XDR procedure used to decode the remote procedure’s results.

- **out**
  Specifies the pointer to the results of the remote procedure.

Usage

The clnt_freeres() call deallocates any resources that were assigned by the system to decode the results of an RPC.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_create()
- clntraw_create()
- clnttcp_create()
- clntudp_create()
clnt_geterr()

Format

```c
#include <rpc.h>

void
clnt_geterr(clnt, errp)
CLIENT *clnt;
struct rpc_err *errp;
```

Parameters

- **clnt**: Indicates the pointer to a client handle that was previously obtained using clnt_create(), clntraw_create(), clnttcp_create(), or clntudp_create().

- **errp**: Indicates the pointer to the address into which the error structure is copied.

Usage

The clnt_geterr() call copies the error structure from the client handle to the structure at address `errp`.

Context

- clnt_call()
- clnt_create()
- clnt_pcreateerror()
- clnt_perror()
- clnt_spcreateerror()
- clnt_sperror()
- clntraw_create()
- clnttcp_create()
- clntudp_create()
clnt_pcreateerror()

Format

```
#include <rpc.h>

void
clnt_pcreateerror(s)
char *s;
```

Parameters

- `s` Indicates a null or null-terminated character string. If `s` is nonnull, `clnt_pcreateerror()` prints the string `s` followed by a colon, followed by a space, followed by the error message, and terminated with a new line. If `s` is null or points to a null string, just the error message and the new line are output.

Usage

The `clnt_pcreateerror()` call writes a message to the standard error device, indicating why a client handle cannot be created. This procedure is used after `clntraw_create()`, `clnttcp_create()`, `clntudp_create()`, or `clnt_create()`, fails.

Context

- `clnt_create`
- `clnt_geterr`
- `clnt_perrno`
- `clnt_perror`
- `clnt_spcreateerror`
- `clnt_sperno`
- `clnt_sperror`
- `clntraw_create`
- `clnttcp_create`
- `clntudp_create`
clnt_perrno()

Format

```c
#include <rpc.h>
void
clnt_perrno(stat)
enum clnt_stat stat;
```

Parameters

`stat`
Indicates the client status.

Usage

The `clnt_perrno()` call writes a message to the standard error device corresponding to the condition indicated by `stat`. This procedure should be used after `callrpc()` if there is an error.

Context

- `callrpc()`
- `clnt_geterr()`
- `clnt_pcreateerror()`
- `clnt_perror()`
- `clnt_spcreateerror()`
- `clnt_sperrno()`
- `clnt_sperror()`
clnt_perror()

**Format**

```c
#include <rpc.h>
void
clnt_perror(clnt, s)
    CLIENT *clnt;
    char *s;
```

**Parameters**

- **clnt**
  Specifies the pointer to a client handle that was previously obtained using clnt_create(), clntudp_create(), clnttcp_create(), or clntraw_create().

- **s**
  Indicates a null or null-terminated character string. If s is nonnull, clnt_perror() prints the string s followed by a colon, followed by a space, followed by the error message, and terminated with a new line. If s is null or points to a null string, just the error message and the new line are output.

**Usage**

The clnt_perror() call writes a message to the standard error device, indicating why an RPC failed. This procedure should be used after clnt_call() if there is an error.

**Context**

- clnt_call()
- clnt_create()
- clnt_geterr()
- clnt_pcreateerror()
- clnt_perror()
- clnt_errno()
- clnt_spcreateerror()
- clnt_sperrno()
- clnt_sperror()
- clntraw_create()
- clnttcp_create()
- clntudp_create()
clnt_spcreateerror()

Format

```c
#include <rpc.h>

char *
clnt_spcreateerror(s)
char *s;
```

Parameters

- `s` Indicates a null or null-terminated character string. If `s` is nonnull, `clnt_spcreateerror()` prints the string `s` followed by a colon, followed by a space, followed by the error message, and terminated with a new line. If `s` is null or points to a null string, just the error message and the new line are output.

Usage

The `clnt_spcreateerror()` call returns the address of a message indicating why a client handle cannot be created. This procedure is used after `clnt_create()`, `clntraw_create()`, `clnttcp_create()`, or `clntudp_create()` fails.

Return codes

- Pointer to a character string ending with a new line.

Context

- `callrpc()`
- `clnt_geterr()`
- `clnt_errno()`
- `clnt_error()`
- `clnt_pccreateerror()`
- `clnt_serrno()`
- `clnt_serror()`
- `clntraw_create()`
- `clnttcp_create()`
- `clntudp_create()`
clnt_sperrno()

Format

```
#include <rpc.h>
char *
clnt_sperrno(stat)
enum clnt_stat stat;
```

Parameters

stat
Indicates the client status.

Usage

The clnt_sperrno() call returns the address of a message corresponding to the condition indicated by stat. This procedure should be used after callrpc(), if there is an error.

Return codes

Pointer to a character string ending with a new line.

Context

- clnt_call()
- clnt_geterr()
- clnt_pcreateerror()
- clnt_spcreateerror()
- clnt_sperror()
- clnt_perror()
clnt_sperror()

Format

```
#include <rpc.h>

char *
clnt_sperror(clnt, s)
CLIENT *clnt;
char *s;
```

Parameters

clnt
Indicates the pointer to a client handle that was previously obtained using clnt_create(), clntudp_create(), clnttcp_create(), or clntraw_create().

s Indicates a null or null-terminated character string. If s is nonnull, clnt_sperror() prints the string s followed by a colon, followed by a space, followed by the error message, and terminated with a new line. If s is null or points to a null string, just the error message and the new line are output.

Usage

The clnt_sperror() call returns the address of a message indicating why an RPC failed. This procedure should be used after clnt_call(), if there is an error.

Return codes

Pointer to a character string ending with a new line.

Context

- clnt_call()
- clnt_create()
- clnt_geterr()
- clnt_pcreateerror()
- clnt_perror()
- clnt_spcreateerror()
- clnt_sperror()
- clnt_sperrno()
- clntraw_create()
- clnttcp_create()
- clntudp_create()
clntraw_create()

Format

```
#include <rpc.h>
CLIENT *
clntraw_create(prognum, versnum)
u_long prognum;
u_long versnum;
```

Parameters

- **prognum**
  - Specifies the remote program number.
- **versnum**
  - Specifies the version number of the remote program.

Usage

The clntraw_create() call creates a dummy client for the remote double (**prognum**, **versnum**). Because messages are passed using a buffer within the address space of the local process, the server should also use the same address space, which simulates RPC programs within one address space. See “svcraw_create()” on page 277 for more information.

Return codes

NULL indicates failure.

Context

- clnt_call()
- clnt_create()
- clnt_destroy()
- clnt_freeres()
- clnt_geterr()
- clnt_pcreateerror()
- clnt_perror()
- clnt_spcreateerror()
- clnt_sperror()
- clntudp_create()
- clnttcp_create()
- svcraw_create()
clnttcp_create()

Format

```c
#include <rpc.h>
CLIENT *
clnttcp_create(addr, prognum, versnum, sockp, sendsz, recvsz)
struct sockaddr_in *addr;
ulong prognum;
ulong versnum;
int *sockp;
uint sendsz;
uint recvsz;
```

Parameters

**addr**
Indicates the pointer to the Internet address of the remote program. If addr points to a port number of 0, addr is set to the port on which the remote program is receiving.

**prognum**
Specifies the remote program number.

**versnum**
Specifies the version number of the remote program.

**sockp**
Indicates the pointer to the socket. If *sockp* is RPC_ANYSOCK, then this routine opens a new socket and sets *sockp*.

Requirements: If you use this handle to send the PMAPPROC_SET, PMAPPROC_UNSET, RPCBPROC_SET, or RPCBPROC_UNSET RPC to rpcbind, the following requirements apply:

- Your registration request must originate from an IP address on the local host.
- If the SAF profile EZB.RPCBIND.sysname.rpcbindname.REGISTRY is defined in the SERVAUTH class, your application user ID must be granted at least READ access to the profile.

**sendsz**
Specifies the size of the send buffer. Specify 0 to choose the default.

**recvsz**
Specifies the size of the receive buffer. Specify 0 to choose the default.

Usage

The clnttcp_create() call creates an RPC client transport handle for the remote program that is specified by (prognum, versnum). The client uses TCP as the transport layer.

Return codes

NULL indicates failure.

Context

- clnt_call()
- clnt_control()
- clnt_create()
- clnt_destroy()
- clnt_freeres()
- clnt_geterr()
- clnt_pcreateerror()
- clnt_perror()
- clnt_spcreateerror()
- clnt_sperror()
- clntraw_create()
- clntudp_create()
clntudp_create()

Format

```
#include <rpc.h>
CLIENT *
clntudp_create(addr, prognum, versnum, wait, sockp)
struct sockaddr_in *addr;
ulong prognum;
ulong versnum;
struct timeval wait;
int *sockp;
```

Parameters

- `addr` indicates the pointer to the Internet address of the remote program. If `addr` points to a port number of 0, `addr` is set to the port on which the remote program is receiving. The remote portmap service is used for this.

- `prognum` specifies the remote program number.

- `versnum` specifies the version number of the remote program.

- `wait` indicates that UDP resends the call request at intervals of `wait` time, until either a response is received or the call times out. The timeout length is set using the `clnt_call()` procedure.

- `sockp` specifies the pointer to the socket. If `*sockp` is `RPC_ANYSOCK`, this routine opens a new socket and sets `*sockp`.

Requirements: If you use this handle to send the `PMAPPROC_SET`, `PMAPPROC_UNSET`, `RPCBPROC_SET`, or `RPCBPROC_UNSET` RPC to `rpcbind`, the following requirements apply:

  - Your registration request must originate from an IP address on the local host.
  - If the SAF profile `EZB.RPCBIND.sysname.rpcbindname.REGISTRY` is defined in the `SERVAUTH` class, your application user ID must be granted at least `READ` access to the profile.

Usage

The `clntudp_create()` call creates a client transport handle for the remote program (`prognum`) with version (`versnum`). UDP is used as the transport layer.

Note: This procedure should not be used with procedures that use large arguments or return large results. While UDP packet size is configurable to a maximum of 64 - 1 kilobytes, the default UDP packet size is only 8 kilobytes.

Return codes

- `NULL` indicates failure.
Context

- call_rpc()
- clnt_call()
- clnt_control()
- clnt_create()
- clnt_destroy()
- clnt_freeres()
- clnt_geterr()
- clnt_pcreateerror()
- clnt_perror()
- clnt_spcreateerror()
- clnt_serror()
- clntraw_create()
- clnttcp_create()
get_myaddress()

Format

```c
#include <rpc.h>
void
get_myaddress(addr)
struct sockaddr_in *addr;
```

Parameters
addr
Indicates the pointer to the location where the local Internet address is placed.

Usage
The get_myaddress() call puts the local host Internet address into `addr`. The port number (`addr->sin_port`) is set to htons (PMAPPORT), which is 111.

Context
- clnttcp_create()
- getpcport()
- pmap_getmaps()
- pmap_getport()
- pmap_rmtcall()
- pmap_set()
- pmap_unset()
getrpcport()  

Format  

```c  
#include <rpc.h>  
u_short  
getrpcport(host, prognum, versnum, protocol)  
char *host;  
u_long prognum;  
u_long versnum;  
int protocol;  
```

Parameters  

- **host**  
  Specifies the pointer to the name of the foreign host.  

- **prognum**  
  Specifies the program number to be mapped.  

- **versnum**  
  Specifies the version number of the program to be mapped.  

- **protocol**  
  Specifies the transport protocol used by the program (IPPROTO_TCP or IPPROTO_UDP).  

Usage  

The getrpcport() call returns the port number associated with the remote program (prognum), the version (versnum), and the transport protocol (protocol).  

Return codes  

The value 1 indicates that the mapping does not exist or that the remote portmap could not be contacted. If portmapper cannot be contacted, rpc_createerr contains the RPC status.  

Context  

- get_myaddress()  
- pmap_getmaps()  
- pmap_getport()  
- pmap_rmtcall()  
- pmap_set()  
- pmap_unset()
pmap_getmaps()

Format

```c
#include <rpc.h>
#include <pmap_pro.h>
#include <pmap_cln.h>

struct pmaplist *
pmap_getmaps(
    struct sockaddr_in *addr;
);```

Parameters

`addr`
Indicates the pointer to the Internet address of the foreign host.

Usage

The `pmap_getmaps()` call returns a list of current program-to-port mappings on the foreign host specified by `addr`.

Return codes

Returns a pointer to a pmaplist structure, or NULL.

Context

- `getrpcport()`
- `pmap_getport()`
- `pmap_rmtcall()`
- `pmap_set()`
- `pmap_unset()`
pmap_getport()

Format

```c
#include <rpc.h>
#include <pmap_pro.h>
#include <pmap_cln.h>

u_short
pmap_getport(addr, prognum, versnum, protocol)
struct sockaddr_in *addr;
ulong prognum;
ulong versnum;
int protocol;
```

Parameters

- **addr**
  Indicates the pointer to the Internet address of the foreign host.

- **prognum**
  Specifies the program number to be mapped.

- **versnum**
  Specifies the version number of the program to be mapped.

- **protocol**
  Indicates the transport protocol used by the program (IPPROTO_TCP or IPPROTO_UDP).

Usage

The `pmap_getport()` call returns the port number associated with the remote program (`prognum`), the version (`versnum`), and the transport protocol (`protocol`).

Return codes

The value 1 indicates that the mapping does not exist or that the remote portmap could not be contacted. If portmapper cannot be contacted, `rpc_createerr` contains the RPC status.

Context

- `getrpcport()`
- `pmap_getmaps()`
- `pmap_remcall()`
- `pmap_set()`
- `pmap_unset()`
pmap_rmtcall()

Format

```c
#include <rpc.h>
#include <pmap_pro.h>
#include <pmap_cln.h>
enum clnt_stat
pmap_rmtcall(addr, progrnum,
versnum, procrnum, inproc, in, outproc, out, tout, portp)
struct sockaddr_in *addr;
u_long progrnum;
u_long versnum;
u_long procrnum;
xdrproc_t inproc;
char *in;
xdrproc_t outproc;
char *out;
struct timeval tout;
u_long *portp;
```

Parameters

- **addr**: Indicates the pointer to the Internet address of the foreign host.
- **progrnum**: Specifies the remote program number.
- **versnum**: Specifies the version number of the remote program.
- **procrnum**: Identifies the procedure to be called.
- **inproc**: Specifies the XDR procedure used to encode the arguments of the remote procedure.
- **in**: Specifies the pointer to the arguments of the remote procedure.
- **outproc**: Specifies the XDR procedure used to decode the results of the remote procedure.
- **out**: Indicates the pointer to the results of the remote procedure.
- **tout**: Specifies the timeout period for the remote request.
- **portp**: If the call from the remote portmap service is successful, `portp` contains the port number of the triple `(progrnum, versnum, procrnum)`.

Usage

The pmap_rmtcall() call instructs portmapper, on the host at `addr`, to make an RPC call to a procedure on that host. This procedure should be used only for ping-type functions.
Return codes
clnt_stat enumerated type.

Context
• getrpcport()
• pmap_getmaps()
• pmap_getport()
• pmap_set()
• pmap_unset()
pmap_set()

Format

```c
#include <rpc.h>
#include <pmap_pro.h>
#include <pmap_cln.h>

bool_t
pmap_set(prognum, versnum, protocol, port)
    u_long prognum;
    u_long versnum;
    int protocol;
    u_short port;
```

Parameters

- **prognum**: Specifies the local program number.
- **versnum**: Specifies the version number of the local program.
- **protocol**: Indicates the transport protocol used by the local program.
- **port**: Indicates the port to which the local program is mapped.

Usage

The `pmap_set()` call sets the mapping of the program (specified by `prognum`, `versnum`, and `protocol`) to `port` on the local machine. This procedure is automatically called by the `svc_register()` procedure.

Requirements: When your application registers with rpcbind rather than with portmapper, the following requirements apply:

- Your registration request must originate from an IP address on the local host.
- If you have defined the SAF profile `EZB.RPCBIND.sysname.rpcbindname.REGISTRY` in the SERVAUTH class, your application user ID must be granted at least READ access to permit this library call.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- `getrpcport`
- `pmap_getmaps`
- `pmap_getport`
- `pmap_rmtcall`
- `pmap_unset`
pmap_unset()

Format

```c
#include <rpc.h>
#include <pmap_pro.h>
#include <pmap_cln.h>

bool_t
pmap_unset(prognum, versnum)
u_long prognum;
u_long versnum;
```

Parameters

- **prognum**: Specifies the local program number.
- **versnum**: Specifies the version number of the local program.

Usage

The `pmap_unset()` call removes the mappings associated with `prognum` and `versnum` on the local machine. All ports for each transport protocol currently mapping the `prognum` and `versnum` are removed from the portmap service.

Requirements: When your application registers with `rpcbind` rather than with `portmapper`, the following requirements apply:

- Your registration request must originate from an IP address on the local host.
- If you have defined the SAF profile `EZB.RPCBIND.sysname.rpcbindname.REGISTRY` in the `SERVAUTH` class, your application user ID must be granted at least READ access to permit this library call.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- `getrpcport()`
- `pmap_getmaps()`
- `pmap_getport()`
- `pmap_rmtcall()`
- `pmap_set()`
registerrpc()

Format

```c
#include <rpc.h>
int
registerrpc(prognum, versnum, procnum, procname, inproc, outproc)
  u_long prognum;
  u_long versnum;
  u_long procnum;
  char *(*procname)();
  xdrproc_t inproc;
  xdrproc_t outproc;
```

Parameters

- **prognum**: Specifies the program number to register.
- **versnum**: Specifies the version number to register.
- **procnum**: Specifies the procedure number to register.
- **procname**: Indicates the procedure that is called when the registered program is requested. `procname` must accept a pointer to its arguments, and return a static pointer to its results.
- **inproc**: Specifies the XDR routine used to decode the arguments.
- **outproc**: Specifies the XDR routine that encodes the results.

Usage

The registerrpc() call registers a procedure `(prognum, versnum, procnum)` with the local portmapper, and creates a control structure to remember the server procedure and its XDR routine. The control structure is used by svc_run(). When a request arrives for the program `(prognum, versnum, procnum)`, the procedure `procname` is called. Procedures registered using registerrpc() are accessed using the UDP transport layer.

**Note**: `xdr_enum()` cannot be used as an argument to registerrpc(). See `xdr_enum()` on page 290 for more information.

Requirements: When your application registers with rpcbind rather than with portmapper, the following requirements apply:
- Your registration request must originate from an IP address on the local host.
- If you have defined the SAF profile `EZB.RPCBIND.sysname.rpcbindname.REGISTRY` in the SERVAUTH class, your application user ID must be granted at least READ access to permit this library call.

Return codes

The value 1 indicates success; the value -1 indicates an error.
Context

- svc_register()
- svc_run()
svc_destroy()

Format

```c
#include <rpc.h>
void
svc_destroy(xprt)
SVCXPRT *xprt;
```

Parameters

`xprt`

Specifies the pointer to the service transport handle.

Usage

The `svc_destroy()` call deletes the RPC service transport handle `xprt`, which becomes undefined after this routine is called.

Context

- `svccraw_create()`
- `svctcp_create()`
- `svcup_create()`
svc_freeargs()

Format

```c
#include <rpc.h>
bool_t
svc_freeargs(xprt, inproc, in)
SVCXPRT *xprt;
xdproc_t inproc;
char *in;
```

Parameters

- **xprt**
  - Specifies the pointer to the service transport handle.
- **inproc**
  - Specifies the XDR routine used to decode the arguments.
- **in**
  - Indicates the pointer to the input arguments.

Usage

The svc_freeargs() call frees storage allocated to decode the arguments to a service procedure using svc_getargs().

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- svc_getargs()
svc_getargs()

Format

```c
#include <rpc.h>
bool_t svc_getargs(xprt, inproc, in)
SVCXPRT *xprt;
xdrproc_t inproc;
char *in;
```

Parameters

- **xprt**
  - Specifies the pointer to the service transport handle.
- **inproc**
  - Specifies the XDR routine used to decode the arguments.
- **in**
  - Indicates the pointer to the decoded arguments.

Usage

The `svc_getargs()` call uses the XDR routine `inproc` to decode the arguments of an RPC request associated with the RPC service transport handle `xprt`. The results are placed at address `in`.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- `svc_freeargs()`
svc_getcaller()

**Format**

```c
#include <rpc.h>
struct sockaddr_in *
svc_getcaller(xprt)
SVCXPRT *xprt;
```

**Parameters**

`xprt`

Specifies the pointer to the service transport handle.

**Usage**

Macro obtains the network address of the client associated with the service transport handle `xprt`.

**Return codes**

This is a pointer to a sockaddr_in structure.

**Context**

- get_myaddress()
svc_getreq()

Format

```c
#include <rpc.h>
void
svc_getreq(rdfds)
int rdfds;
```

Parameters

- **rdfds**
  - Specifies the read descriptor bit set.

Usage

The svc_getreq() call is used, rather than svc_run(), to implement asynchronous event processing. The routine returns control to the program when all sockets have been serviced.

svc_getreq limits you to 32 socket descriptors, of which 3 are reserved. Use svc_getreqset if you have more than 29 socket descriptors.

Context

- svc_run()
svc_getreqset()

Format

```c
#include <rpc.h>
void
svc_getreqset(readfds)
  fd_set readfds;
```

Parameters

readfds
  Specifies the read descriptor bit set.

Usage

The svc_getreqset() call is used, rather than svc_run(), to implement asynchronous event processing. The routine returns control to the program when all sockets have been serviced.

A server would use a select() call to determine if there are any outstanding RPC requests at any of the sockets created when the programs were registered. The read bit descriptor set returned by select() is then used on the call to svc_getreqset().

Note that you should not pass the global bit descriptor set svc_fdset on the call to select(), because select() changes the values. Instead, you should make a copy of svc_fdset before you call select().

Context

- svc_run()
svc_register()

Format

```c
#include <rpc.h>
bool_t
svc_register(xprt, prognum, versnum, dispatch, protocol)
SVCXPRT *xprt;
u_long prognum;
u_long versnum;
void (*dispatch)();
int protocol;
```

Parameters

- **xprt**
  Specifies the pointer to the service transport handle.

- **prognum**
  Specifies the program number to be registered.

- **versnum**
  Specifies the version number of the program to be registered.

- **dispatch()**
  Indicates the dispatch routine associated with prognum and versnum.
  
  The structure of the dispatch routine is:
  ```c
  #include <rpc.h>
  
  dispatch(request, xprt)
  struct svc_req *request;
  SVCXPRT *xprt;
  
  protocol
  The protocol used. The value is generally one of the following:
  ```
  ```c
  • 0
  • IPPROTO_UDP
  • IPPROTO_TCP
  
  When the value 0 is used, the service is not registered with portmapper.
  ```
  
  Rule: When using a toy RPC service transport created with svcraw_create(), a call to xprt_register() must be made immediately after a call to svc_register().

Usage

The svc_register() call associates the program described by (prognum, versnum) with the service dispatch routine dispatch.

Requirements: When your application registers with rpcbind rather than with portmapper, the following requirements apply:

- Your registration request must originate from an IP address on the local host.
- If you have defined the SAF profile EZB.RPCBIND.sysname.rpcbindname.REGISTRY in the SERVAUTH class, your application user ID must be granted at least READ access to permit this library call.
Return codes
The value 1 indicates success; the value 0 indicates an error.

Context
- registerrpc()
- svc_unregister()
- xprt_register()
svc_run()

Format

```c
#include <rpc.h>
svc_run()
```

Parameters

None.

Usage

The svc_run() call does not return control. It accepts RPC requests and calls the appropriate service using svc_getreqset().

Context

svc_getreqset()
svc_sendreply()

Format

```c
#include <rpc.h>
bool_t
svc_sendreply(xprt, outproc, out)
SVCXPRT *xprt;
xdproc_t outproc;
char *out;
```

Parameters

- **xprt**
  - Indicates the pointer to the caller’s transport handle.
- **outproc**
  - Specifies the XDR procedure used to encode the results.
- **out**
  - Specifies the pointer to the results.

Usage

The svc_sendreply() call is called by the service dispatch routine to send the results of the call to the caller.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_call()
svc_unregister()

Format

```c
#include <rpc.h>
void
svc_unregister(prognum, versnum)
  u_long prognum;
  u_long versnum;
```

Parameters

- **prognum**
  - Specifies the program number that is removed.
- **versnum**
  - Specifies the version number of the program that is removed.

Usage

The `svc_unregister()` call removes all local mappings of prognum and versnum to dispatch routines and prognum, versnum, and * to port numbers.

**Requirements:** When your application registers with rpcbind rather than with portmapper, the following requirements apply:

- Your registration request must originate from an IP address on the local host.
- If you have defined the SAF profile `EZB.RPCBIND.sysname.rpcbindname.REGISTRY` in the SERVAUTH class, your application user ID must be granted at least READ access to permit this library call.
svcerr_auth()

Format

```c
#include <rpc.h>
void
svcerr_auth(xprt, why)
SVCXPRTP  xprt;
enum auth_stat  why;
```

Parameters

- `xprt`: Specifies the pointer to the service transport handle.
- `why`: Specifies the reason the call is refused.

Usage

The svcerr_auth() call is called by a service dispatch routine that refuses to execute an RPC request because of authentication errors.

Context

- svcerr_noproc()
- svcerr_noprog()
- svcerr_progvers()
- svcerr_systemerr()
- svcerr_weakauth()
svcerr_decode()

Format

```c
#include <rpc.h>
void
svcerr_decode(xprt)
SVCXPRTP *xprt;
```

Parameters

`xprt`

Indicates the pointer to the service transport handle.

Usage

The svcerr_decode() call is called by a service dispatch routine that cannot decode its parameters.

Context

- svcerr_noproc()
- svcerr_noprogr()
- svcerr_progvers()
- svcerr_systemerr()
- svcerr_weakauth()
svcerr_noproc()

Format

```c
#include <rpc.h>
void
svcerr_noproc(xprt)
SVCXPRTP *xprt;
```

Parameters

- `xprt` Indicates the pointer to the service transport handle.

Usage

The svcerr_noproc() call is called by a service dispatch routine that does not implement the requested procedure.

Context

- svcerr_decode()
- svcerr_noprog()
- svcerr_progvers()
- svcerr_systemerr()
- svcerr_weakauth()
svcerr_noprog()

Format

```c
#include <rpc.h>
void
svcerr_noprog(xprt)
SVCXPRT *xprt;
```

Parameters

- `xprt`  
  Indicates the pointer to the service transport handle.

Usage

The `svcerr_noprog()` call is used when the desired program is not registered.

Context

- `svcerr_decode()`
- `svcerr_noproc()`
- `svcerr_progvers()`
- `svcerr_systemerr()`
- `svcerr_weakauth()`
svcerr_progvers()

Format

```c
#include <rpc.h>
void
svcerr_progvers(xprt, low_vers, high_vers)
SVCXPRT *xprt;
ulong low_vers;
ulong high_vers;
```

Parameters

- **xprt**
  Indicates the pointer to the service transport handle.
- **low_vers**
  Specifies the low version number that does not match.
- **high_vers**
  Specifies the high version number that does not match.

Usage

The svcerr_progvers() call is called when the version numbers of two RPC programs do not match. The low version number corresponds to the lowest registered version, and the high version corresponds to the highest version registered on the portmapper.

Context

- svcerr_decode()
- svcerr_noproc()
- svcerr_noprog()
- svcerr_progvers()
- svcerr_systemerr()
- svcerr_weakauth()
svcerr_systemerr()

Format

```c
#include <rpc.h>
void
svcerr_systemerr(xprt)
SVCXPRTP *xprt;
```

Parameters

- **xprt**
  - Indicates the pointer to the service transport handle.

Usage

The `svcerr_systemerr()` call is called by a service dispatch routine when it detects a system error that is not handled by the protocol.

Context

- `svcerr_decode()`
- `svcerr_noproc()`
- `svcerr_noprog()`
- `svcerr_progvers()`
- `svcerr_weakauth()`
**svcerr_weakauth()**

**Format**

```c
#include <rpc.h>
void
svcerr_weakauth(xprt)
SVCXPRTP *xprt;
```

**Parameters**

**xprt**

Indicates the pointer to the service transport handle.

**Note:** This is the equivalent of svcerr_auth(xprt, AUTH_TOO_WEAK).

**Usage**

The svcerr_weakauth() call is called by a service dispatch routine that cannot execute an RPC because of correct but weak authentication parameters.

**Context**

- svcerr_decode()
- svcerr_noproc()
- svcerr_noprog()
- svcerr_progvers()
- svcerr_systemerr()
svcraw_create()  

Format

```c
#include <rpc.h>
SVCXPRTP * svcraw_create()
```

Parameters

None.

Usage

The svcraw_create() call creates a local RPC service transport used for timings, to which it returns a pointer. Messages are passed using a buffer within the address space of the local process; therefore, the client process must also use the same address space. This allows the simulation of RPC programs within one computer. See "clntraw_create()" on page 244 for more information.

Return codes

NULL indicates failure.

Context

- svc_destroy()
- svctcp_create()
- svcudp_create()
svctcp_create()

Format

```c
#include <rpc.h>
SVCXprt *
svctcp_create(sock, send_buf_size, recv_buf_size)
int sock;
_u_int send_buf_size;
_u_int recv_buf_size;
```

Parameters

- **sock**
  Specifies the socket descriptor. If sock is RPC_ANYSOCK, a new socket is
  created. If the socket is not bound to a local TCP port, it is bound to an
  arbitrary port.

- **send_buf_size**
  Specifies the size of the send buffer. Specify 0 to choose the default.

- **recv_buf_size**
  Specifies the size of the receive buffer. Specify 0 to choose the default.

Usage

The svctcp_create() call creates a TCP-based service transport to which it returns a
pointer. xprt—>xp_sock contains the transport socket descriptor. xprt—>xp_port
contains the transport port number.

Return codes

NULL indicates failure.

Context

- svcrw_create()
- svcudp_create()
svcudp_create()

Format

```
#include <rpc.h>
SVCXprt *
svcudp_create(sock, send_buf_size, recv_buf_size)
int sock;
uint send_buf_size;
uint recv_buf_size;
```

Parameters

- **sock**: Specifies the socket associated with the service transport handle. If sock is RPC_ANYSOCK, a new socket is created.
- **send_buf_size**: Specifies the size of the send buffer. Specify 0 to choose the default.
- **recv_buf_size**: Specifies the size of the receive buffer. Specify 0 to choose the default.

Usage

The svcudp_create() call creates a UDP-based service transport to which it returns a pointer. xprt->xp_sock contains the transport socket descriptor. xprt->xp_port contains the transport port number.

Return codes

NULL indicates failure.

Context

- svcraw_create()
- svctcp_create()
xdr_accepted_reply()

Format

```c
#include <rpc.h>
bool_t
xdr_accepted_reply(xdrs, ar)
XDR *xdrs;
struct accepted_reply *ar;
```

Parameters

- **xdrs**
  Specifies the pointer to an XDR stream.
- **ar**
  Specifies the pointer to the reply to be represented.

Usage

The `xdr_accepted_reply()` call translates RPC reply messages.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- `clnt_broadcast()`
- `clnt_call()`
- `clnt_freeres()`
- `pmap_rmtcall()`
- `registerrpc()`
- `svc_freeargs()`
- `svc_getargs()`
- `svc_sendreply()`
xdr_array()

Format

```c
#include <rpc.h>
bool_t
xdr_array(xdrs, arrp, sizep, maxsize, elsize, elproc)
XDR *xdrs;
char **arrp;
u_int *sizep;
u_int maxsize;
u_int elsize;
xdrproc_t elproc;
```

Parameters

- **xdrs**
  Specifies the pointer to an XDR stream.

- **arrp**
  Specifies the address of the pointer to the array.

- **sizep**
  Specifies the pointer to the element count of the array.

- **maxsize**
  Specifies the maximum number of elements accepted.

- **elsize**
  Specifies the size of each of the array’s elements, found using sizeof().

- **elproc**
  Specifies the XDR routine that translates an individual array element.

Usage

The xdr_array() call translates between an array and its external representation.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()
xdr_authunix_parms()

Format

```c
#include <rpc.h>
bool_t
xdr_authunix_parms(xdrs, aupp)
    XDR *xdrs;
    struct authunixParms *aupp;
```

Parameters

- **xdrs**
  Specifies the pointer to an XDR stream.
- **aupp**
  Indicates the pointer to the authentication information.

Usage

The xdr_authunix_parms() call translates UNIX-based authentication information.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()
xdr_bool()  

Format  

```c  
#include <rpc.h>  
bool_t  
xdr_bool(xdrs, bp)  
XDR *xdrs;  
bool_t *bp;  
```  

Parameters  

- `xdrs`  
  Specifies the pointer to an XDR stream.  
- `bp`  
  Indicates the pointer to the Boolean.  

Usage  

The `xdr_bool()` call translates between booleans and their external representation.  

Return codes  

The value 1 indicates success; the value 0 indicates an error.  

Context  

- clnt_broadcast()  
- clnt_call()  
- clnt_freeres()  
- pmap_rmtcall()  
- registerrpc()  
- svc_freeargs()  
- svc_getargs()  
- svc_sendreply()  

---  

Chapter 8. Remote procedure calls in the z/OS Communications Server environment  283
xdr_bytes()  

Format

```c
#include <rpc.h>
bool_t
xdrBytes(xdrs, sp, sizep, maxsize)
XDR *xdrs;
char **sp;
int *sizep;
int maxsize;
```

Parameters

- **xdrs**
  Specifies the pointer to an XDR stream.
- **sp**
  Specifies the pointer to the byte string.
- **sizep**
  Indicates the pointer to the byte string size.
- **maxsize**
  Specifies the maximum size of the byte string.

Usage

The xdr_bytes() call translates between byte strings and their external representations.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()
xdr_callhdr()

Format

```
#include <rpc.h>
void
xdr_callhdr(xdrs, chdr)
XDR *xdrs;
struct rpc_msg *chdr;
```

Parameters

- **xdrs**: Specifies the pointer to an XDR stream.
- **chdr**: Specifies the pointer to the call header.

Usage

The xdr_callhdr() call translates an RPC message header into XDR format.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()
xdr_callmsg()

**Format**

```c
#include <rpc.h>
bool_t
xdr_callmsg(xdrs, cmsg)
XDR *xdrs;
struct rpc_msg *cmsg;
```

**Parameters**

- **xdrs**
  Specifies the pointer to an XDR stream.
- **cmsg**
  Specifies the pointer to the call message.

**Usage**

The xdr_callmsg() call translates RPC messages (header and authentication, not argument data) to and from the XDR format.

**Return codes**

The value 1 indicates success; the value 0 indicates an error.

**Context**

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()
### xdr_char()

**Format**

```c
#include <rpc.h>

bool_t
xdr_char(xdrs, cp)
XDR *xdrs;
char *cp;
```

**Parameters**

- `xdrs` Specifies the pointer to an XDR stream.
- `cp` Specifies the pointer to the C character.

**Usage**

The `xdr_char()` call is a filter that translates between C characters and their external representations.

**Notes:**

1. Encoded characters are not packed, and they occupy 4 bytes each.
2. `xdr_string` and `xdr_text_char()` are the only supported routines that convert ASCII to EBCDIC. The `xdr_char` routine does not support conversion.

**Return codes**

The value 1 indicates success; the value 0 indicates an error.

**Context**

- `clnt_broadcast()`
- `clnt_call()`
- `clnt_freeres()`
- `pmap_rmtcall()`
- `registerrpc()`
- `svc_freeargs()`
- `svc_getargs()`
- `svc_sendreply()`
- `xdr_bytes()`
- `xdrOpaque()`
- `xdr_string()`
xdr_destroy()

Format

```
#include <rpc.h>
void
xdr_destroy(xdrs)
    XDR *xdrs;
```

Parameters

`xdrs`

Specifies the pointer to an XDR stream.

Usage

The xdr_destroy() call is a macro that invokes the destroy routine associated with the XDR stream, xdrs. Destruction usually involves freeing private data structures associated with the stream. Using xdrs after invoking xdr_destroy() is undefined.
xdr_double()

Format

```c
#include <rpc.h>
bool_t
xdr_double(xdrs, dp)
XDR *xdrs;
double *dp;
```

Parameters

- **xdrs**
  Indicates the pointer to an XDR stream.
- **dp**
  Indicates the pointer to a double-precision number.

Usage

The xdr_double() call translates between C double-precision numbers and their external representations.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()
xdr_enum()

Format

```
#include <rpc.h>
bool_t
xdr_enum(xdrs, ep)
XDR *xdrs;
enum_t *ep;
```

Parameters

- **xdrs**
  Indicates the pointer to an XDR stream.
- **ep**
  Indicates the pointer to the enumerated number. enum_t can be any
declarated as **enum colors** (black, brown, red).

Usage

The xdr_enum() call translates between C-enumerated groups and their external
representation. When calling the procedures callrpc() and registerrpc(), a stub
procedure must be created for both the server and the client before the procedure
of the application program using xdr_enum(). This procedure should look like the
following:

```
#include <rpc.h>
enum colors (black, brown, red)
void
static xdr_enum_t(xdrs, ep)
XDR *xdrs;
enum colors *ep;
{
    xdr_enum(xdrs, ep)
}
```

The xdr_enum_t procedure is used as the inproc and outproc in both the client and
server RPCs. For example:

- **An RPC client would contain the following lines:**

  ```
  ...
  error = callrpc(argv[1],ENUMRCVPROG,VERSION,ENUMRCVPROC,
    xdr_enum_t,&innumber,xdr_enum_t,&outnumber);
  ...
  
  - An RPC server would contain the following line:

    ```
    ...
    registerrpc(ENUMRCVPROG,VERSION,ENUMRCVPROC,xdr_enum_t,xdr_enum_t);
    ```

Return codes

The value 1 indicates success; the value 0 indicates an error.
Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()
xdr_float()

Format

#include <rpc.h>
bool_t
xdr_float(xdrs, fp)
XDR *xdrs;
float *fp;

Parameters
xdrs
  Specifies the pointer to an XDR stream.
fp
  Indicates the pointer to the floating-point number.

Usage
The xdr_float() call translates between C floating-point numbers and their external
representations.

Return codes
The value 1 indicates success; the value 0 indicates an error.

Context
  - clnt_broadcast()
  - clnt_call()
  - clnt_freeres()
  - pmap_rmtcall()
  - registerrpc()
  - svc_freeargs()
  - svc_getargs()
  - svc_sendreply()
xdr_free()

Format

```
#include <rpc.h>

void
xdr_free(proc, objp)
    xdrproc_t proc;
    char *objp;
```

Parameters

- **proc**: Specifies the XDR routine.
- **objp**: Indicates the pointer to the object being freed.

Usage

The xdr_free() call is a generic freeing routine.

**Note**: The pointer passed to this routine is not freed, but what it points to is freed (recursively).
xdr_getpos()

Format

```
#include <rpc.h>

u_int
xdr_getpos(xdrs)
    XDR *xdrs;
```

Parameters

- **xdrs**
  Specifies the pointer to an XDR stream.

Usage

The xdr_getpos() call is a macro that invokes the get-position routine associated with the XDR stream, xdrs. A desirable feature of XDR streams is that simple arithmetic works with this number, although the XDR stream instances do not guarantee this.

Return codes

An unsigned integer, which indicates the position of the XDR byte stream.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()
xdr_inline()

Format

```
#include <rpc.h>
long *
xdr_inline(xdrs, len)
XDR *xdrs;
int len;
```

Parameters

- **xdrs**
  - Indicates the pointer to an XDR stream.
- **len**
  - Specifies the byte length of the desired buffer.

Usage

The xdr_inline() call returns a pointer to a continuous piece of the XDR stream buffer. The value is `long *` rather than `char *`, because the external data representation of any object is always an integer multiple of 32 bits.

**Note:** xdr_inline() can return NULL if there is not sufficient space in the stream buffer to satisfy the request.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()
xdr_int()

Format

```c
#include <rpc.h>
bool_t
xdr_int(xdrs, ip)
XDR *xdrs;
int *ip;
```

Parameters

- `xdrs` Indicates the pointer to an XDR stream.
- `ip` Indicates the pointer to the integer.

Usage

The `xdr_int()` call translates between C integers and their external representations.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- `clnt_broadcast()`
- `clnt_call()`
- `clnt_freeres()`
- `pmap_rmtcall()`
- `registerrpc()`
- `svc_freeargs()`
- `svc_getargs()`
- `svc_sendreply()`
xdr_long()

Format

```c
#include <rpc.h>
bool_t
xdr_long(xdrs, lp)
  XDR *xdrs;
  long *lp;
```

Parameters

- `xdrs` Indicates the pointer to an XDR stream.
- `lp` Indicates the pointer to the long integer.

Usage

The `xdr_long()` call translates between C long integers and their external representations.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- `clnt_broadcast()`
- `clnt_call()`
- `clnt_freeres()`
- `pmap_rmtcall()`
- `registerrpc()`
- `svc_freeargs()`
- `svc_getargs()`
- `svc_sendreply()`
xdr_opaque()

Format

```c
#include <rpc.h>
bool_t
xdr_opaque(xdrs, cp, cnt)
    XDR *xdrs;
    char *cp;
    u_int cnt;
```

Parameters

- **xdrs**
  
  Indicates the pointer to an XDR stream.

- **cp**
  
  Indicates the pointer to the opaque object.

- **cnt**
  
  Specifies the size of the opaque object.

Usage

The `xdr_opaque()` call translates between fixed-size opaque data and its external representation.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- `clnt_broadcast()`
- `clnt_call()`
- `clnt_freeres()`
- `pmap_rmtcall()`
- `registerrpc()`
- `svc_freeargs()`
- `svc_getargs()`
- `svc_sendreply()`
xdr_opaque_auth()

Format

```c
#include <rpc.h>
bool_t
xdr_opaque_auth(xdrs, ap)
    XDR *xdrs;
    struct opaque_auth *ap;
```

Parameters

- **xdrs**: Indicates the pointer to an XDR stream.
- **ap**: Indicates the pointer to the opaque authentication information.

Usage

The xdr_opaque_auth() call translates RPC message authentications.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registrrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()
xdr_pmap()

Format

```c
#include <rpc.h>
#include <pmap_pro.h>
#include <pmap_cln.h>
bool_t
xdr_pmap(xdrs, regs)
XDR *xdrs;
struct pmap *regs;
```

Parameters

- **xdrs**: Indicates the pointer to an XDR stream.
- **regs**: Indicates the pointer to the portmap parameters.

Usage

The `xdr_pmap()` call translates an RPC procedure identification, such as is used in calls to portmapper.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- `clnt_broadcast()`
- `clnt_call()`
- `clnt_freeres()`
- `pmap_rmtcall()`
- `registerrpc()`
- `svc_freeargs()`
- `svc_getargs()`
- `svc_sendreply()`
xdr_pmaplist()

Format

```c
#include <rpc.h>
#include <pmap_pro.h>
#include <pmap_cln.h>
bool_t
xdr_pmaplist(xdrs, rp)
XDR *xdrs;
struct pmaplist **rp;
```

Parameters

- **xdrs**
  Indicates the pointer to an XDR stream.
- **rp**
  Indicates the pointer that points to a pointer to the portmap data array.

Usage

The `xdr_pmaplist()` call translates a variable number of RPC procedure identifications, such as portmapper creates.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- `clnt_broadcast()`
- `clnt_call()`
- `clnt_freeres()`
- `pmap_rmtcall()`
- `registerrpc()`
- `svc_freeargs()`
- `svc_getargs()`
- `svc_sendreply()`
xdr_pointer()

Format

```
#include <rpc.h>
bool_t
xdr_pointer(xdrs, pp, size, proc)
XDR *xdrs;
char **pp;
unsigned size;
xdrproc_t proc;
```

Parameters

- **xdrs**
  Indicates the pointer to an XDR stream.
- **pp**
  Indicates the pointer that points to a pointer.
- **size**
  Specifies the size of the target.
- **proc**
  Indicates the XDR procedure that translates an individual element of the type addressed by the pointer.

Usage

The `xdr_pointer()` call provides pointer-chasing within structures. This differs from the `xdr_reference()` call in that it can serialize or deserialize trees correctly.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- `clnt_broadcast()`
- `clnt_call()`
- `clnt_freeres()`
- `pmap_rmtcall()`
- `registerrpc()`
- `svc_freeargs()`
- `svc_getargs()`
- `svc_sendreply()`
xdr_reference()

Format

```c
#include <rpc.h>
bool_t xdr_reference(xdrs, pp, size, proc)
XDR *xdrs;
char **pp;
uint size;
xdrproc_t proc;
```

Parameters

- **xdrs**
  Indicates the pointer to an XDR stream.
- **pp**
  Indicates the pointer that points to a pointer.
- **size**
  Specifies the size of the target.
- **proc**
  Specifies the XDR procedure that translates an individual element of the type addressed by the pointer.

Usage

The `xdr_reference()` call provides pointer-chasing within structures.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- `clnt_broadcast()`
- `clnt_call()`
- `clnt_freeres()`
- `pmap_rmtcall()`
- `registerrpc()`
- `svc_freeargs()`
- `svc_getargs()`
- `svc_sendreply()`
xdr_rejected_reply()

Format

```c
#include <rpc.h>
bool_t
xdr_rejected_reply(xdrs, rr)
XDR *xdrs;
struct rejected_reply *rr;
```

Parameters

- `xdrs` : Indicates the pointer to an XDR stream.
- `rr` : Indicates the pointer to the rejected reply.

Usage

The `xdr_rejected_reply()` call translates rejected RPC reply messages.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- `clnt_broadcast()`
- `clnt_call()`
- `clnt_freeres()`
- `pmap_rmtcall()`
- `registerrpc()`
- `svc_freeargs()`
- `svc_getargs()`
- `svc_sendreply()`
xdr_replymsg()

Format

```c
#include <rpc.h>
bool_t
xdr_replymsg(xdrs, rmsg)
    XDR *xdrs;
    struct rpc_msg *rmsg;
```

Parameters

- **xdrs**: Indicates the pointer to an XDR stream.
- **rmsg**: Indicates the pointer to the reply message.

Usage

The xdr_replymsg() call translates RPC reply messages.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()
xdr_setpos()

Format

```c
#include <rpc.h>
int
xdr_setpos(xdrs, pos)
    XDR *xdrs;
    u_int pos;
```

Parameters

- **xdrs**: Indicates the pointer to an XDR stream.
- **pos**: Indicates the pointer to a set position routine.

Usage

The `xdr_setpos()` call is a macro that invokes the set position routine associated with the XDR stream `xdrs`. The parameter `pos` is a position value obtained from `xdr_getpos()`.

Return codes

The value 1 indicates that the XDR stream can be repositioned; the value 0 indicates otherwise.

**Note**: It is difficult to reposition some types of XDR streams; therefore, this routine might fail with one type of stream and succeed with another.

Context

- `clnt_broadcast()`
- `clnt_call()`
- `clnt_freeres()`
- `pmap_rmtcall()`
- `registerrpc()`
- `svc_freeargs()`
- `svc_getargs()`
- `svc_sendreply()`
xdr_short()

Format

```c
#include <rpc.h>
bool_t
xdr_short(xdrs, sp)
XDR *xdrs;
short *sp;
```

Parameters

- **xdrs**
  Indicates the pointer to an XDR stream.
- **sp**
  Indicates the pointer to the short integer.

Usage

The xdr_short() call translates between C short integers and their external representations.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()
**xdr_string()**

**Format**

```c
#include <rpc.h>
bool_t
xdr_string(xdrs, sp, maxsize)
    XDR *xdrs;
    char **sp;
    u_int maxsize;
```

**Parameters**

- **xdrs**
  - Indicates the pointer to an XDR stream.
- **sp**
  - Indicates the pointer that points to a pointer to the string.
- **maxsize**
  - Indicates the maximum size of the string.

**Usage**

The `xdr_string()` call translates between C strings and their external representations.

**Return codes**

The value 1 indicates success; the value 0 indicates an error.

**Context**

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()
**xdr_text_char()**

**Format**

```c
#include <rpc.h>
bool_t
xdr_text_char(xdrs, cp)
XDR *xdrs;
char *cp;
```

**Parameters**

- `xdrs` Specifies the pointer to an XDR stream.
- `cp` Specifies the pointer to the C character.

**Usage**

The `xdr_text_char()` call is a filter primitive that translates between C characters and their external representations.

**Notes:**
1. Encoded characters are not packed, and they occupy 4 bytes each.
2. `xdr_text_char()` converts ASCII to EBCDIC.

**Return codes**

The value 1 indicates success; the value 0 indicates an error.

**Context**

- `clnt_broadcast()`
- `clnt_call()`
- `clnt_freeres()`
- `pmap_rmtcall()`
- `registerrpc()`
- `svc_freeargs()`
- `svc_getargs()`
- `svc_sendreply()`
- `xdr_bytes()`
- `xdrOpaque()`
- `xdr_string()`
**xdr_u_char()**

**Format**

```c
#include <rpc.h>
bool_t
xdr_u_char(xdrs, ucp)
    XDR *xdrs;
    unsigned char *ucp;
```

**Parameters**

- **xdrs**: Indicates the pointer to an XDR stream.
- **ucp**: Indicates the pointer to an unsigned C character.

**Usage**

The `xdr_u_char()` call is a filter primitive that translates between unsigned C characters and their external representations.

**Return codes**

The value 1 indicates success; the value 0 indicates an error.

**Context**

- `clnt_broadcast()`
- `clnt_call()`
- `clnt_freeres()`
- `pmap_rmtcall()`
- `registerrpc()`
- `svc_freeargs()`
- `svc_getargs()`
- `svc_sendreply()`
xdr_u_int()

Format

```c
#include <rpc.h>
bool_t
xdr_u_int(xdrs, up)
XDR *xdrs;
up_int *up;
```

Parameters

- `xdrs` Indicates the pointer to an XDR stream.
- `up` Indicates the pointer to the unsigned integer.

Usage

The xdr_u_int() call translates between C unsigned integers and their external representations.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- `clnt_broadcast()`
- `clnt_call()`
- `clnt_freeres()`
- `pmap_rmtcall()`
- `registerrpc()`
- `svc_freeargs()`
- `svc_getargs()`
- `svc_sendreply()`
xdr_u_long()

Format

```c
#include <rpc.h>
bool_t
xdr_u_long(xdrs, ulp)
XDR *xdrs;
ulong *ulp;
```

Parameters

- **xdrs**
  
  Indicates the pointer to an XDR stream.

- **ulp**
  
  Indicates the pointer to the unsigned long integer.

Usage

The `xdr_u_long()` call translates between C unsigned long integers and their external representations.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()
xdr_u_short()

Format

#include <rpc.h>
bool_t
xdr_u_short(xdrs, usp)
XDR *xdrs;
u_short *usp;

Parameters

xdrs
   Indicates the pointer to an XDR stream.
usp
   Indicates the pointer to the unsigned short integer.

Usage

The xdr_u_short() call translates between C unsigned short integers and their external representations.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()
xdr_union()

Format

```
#include <rpc.h>
bool_t
xdr_union(xdrs, dscmp, unp, choices, dfault)
XDR *xdrs;
enum_t *dscmp;
char *unp;
struct xdr_discrim *choices;
xdrproc_t *default;
```

Parameters

**xdrs**
Indicates the pointer to an XDR stream.

**dscmp**
Indicates the pointer to the union discriminant. enum_t can be any enumeration type.

**unp**
Indicates the pointer to the union.

**choices**
Indicates the pointer to an array detailing the XDR procedure to use on each arm of the union.

**dfault**
Indicates the default XDR procedure to use.

Usage

The xdr_union() call translates between a discriminated C union and its external representation.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Examples

The following is an example of this call:

```
#include <rpc.h>
enum colors (black, brown, red);
bool_t
xdr_union(xdrs, dscmp, unp, choices, dfault)
XDR *xdrs;
enum colors *dscmp;
char *unp;
struct xdr_discrim *choices;
xdrproc_t *default;
```

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
• registerrpc()
• svc_freeargs()
• svc_getargs()
• svc_sendreply()
xdr_vector()

**Format**

```c
#include <rpc.h>
bool_t
xdr_vector(xdrs, basep, nelem, elemsize, xdr_elem)
XDR *xdrs;
char *basep;
_u_int nelem;
_u_int elemsize;
_xdrproc_t xdr_elem;
```

**Parameters**

- **xdrs**
  Indicates the pointer to an XDR stream.

- **basep**
  Indicates the base of the array.

- **nelem**
  Indicates the element count of the array.

- **elemsize**
  Specifies the size of each of array elements, found using sizeof().

- **xdr_elem**
  Specifies the XDR routine that translates an individual array element.

**Usage**

The xdr_vector() call translates between a fixed-length array and its external representation. Unlike variable-length arrays, the storage of fixed-length arrays is static and cannot be freed.

**Return codes**

The value 1 indicates success; the value 0 indicates an error.

**Context**

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()
xdr_void()

Format

```
#include <rpc.h>
bool_t
xdr_void()
```

Parameters

None.

Usage

The xdr_void call always returns 1. It may be passed to RPC routines that require a function parameter, where no action is required. This call can be placed in the inproc or outproc parameter of the clnt_call function when you do not need to move data.

Return codes

Always a value of 1.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registrrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()
xdr_wrapstring()

Format

```
#include <rpc.h>
bool_t
xdr_wrapstring(xdrs, sp)
XDR *xdrs;
char **sp;
```

Parameters

- `xdrs` Indicates the pointer to an XDR stream.
- `sp` Indicates the pointer that points to a pointer to the string.

Usage

The xdr_wrapstring() call is the same as calling xdr_string() with a maximum size of MAXUNSIGNED. It is useful, because many RPC procedures implicitly invoke two-parameter XDR routines, and xdr_string() is a three-parameter routine.

Return codes

The value 1 indicates success; the value 0 indicates an error.

Context

- clnt_broadcast()
- clnt_call()
- clnt_freeres()
- pmap_rmtcall()
- registerrpc()
- svc_freeargs()
- svc_getargs()
- svc_sendreply()
xdrmem_create()

Format

```c
#include <rpc.h>
void
xdrmem_create(xdrs, addr, size, op)
XDR *xdrs;
char *addr;
uint size;
enum xdr_op op;
```

Parameters

- **xdrs**
  Indicates the pointer to an XDR stream.
- **addr**
  Indicates the pointer to the memory location.
- **size**
  Specifies the maximum size of addr.
- **op**
  Determines the direction of the XDR stream (XDR_ENCODE, XDR_DECODE, or XDR_FREE).

Usage

The `xdrmem_create()` call creates an XDR stream in memory. It initializes the XDR stream pointed to by `xdrs`. Data is written to, or read from, `addr`. 

Chapter 8. Remote procedure calls in the z/OS Communications Server environment  319
xdrrec_create()

Format

```c
#include <rpc.h>

void
xdrrec_create(xdrs, sendsize, recvsize, handle, readit, writeit)
XDR *xdrs;
uint sendsize;
uint recvsize;
char *handle;
int (*readit) ();
int (*writeit) ();
```

Parameters

- **xdrs**: Indicates the pointer to an XDR stream.
- **sendsize**: Specifies the size of the send buffer. Specify 0 to choose the default.
- **recvsize**: Specifies the size of the receive buffer. Specify 0 to choose the default.
- **handle**: Specifies the first parameter passed to readit() and writeit().
- **readit()**: Called when a stream input buffer is empty.
- **writeit()**: Called when a stream output buffer is full.

Usage

The xdrrec_create() call creates a record-oriented stream and initializes the XDR stream pointed to by xdrs.

Notes:
1. The caller must set the x_op field.
2. This XDR procedure implements an intermediate record string.
3. Additional bytes in the XDR stream provide record boundary information.
xdrrec_endofrecord()

Format

```c
#include <rpc.h>
bool_t
xdrrec_endofrecord(xdrs, sendnow)
XDR *xdrs;
int sendnow;
```

Parameters

- `xdrs` Indicates the pointer to an XDR stream.
- `sendnow` Specify nonzero to write out data in the output buffer.

Usage

The `xdrrec_endofrecord()` call can be invoked only on streams created by `xdrrec_create()`. Data in the output buffer is marked as a complete record.

Return codes

The value 1 indicates success; the value 0 indicates an error.
**xdrrec_eof()**

**Format**

```c
#include <rpc.h>
bool_t
xdrrec_eof(xdrs)
xdr *xdrs;
```

**Parameters**

- `xdrs`
  - Indicates the pointer to an XDR stream.

**Usage**

The `xdrrec_eof()` call can be invoked only on streams created by `xdrrec_create()`.

**Return codes**

The value 1 indicates the current record has been consumed; the value 0 indicates continued input on the stream.
**xdrrec_skiprecord()**

**Format**

```c
#include <rpc.h>
bool_t
xdrrec_skiprecord(xdrs)
    XDR *xdrs;
```

**Parameters**

- **xdrs**
  Indicates the pointer to an XDR stream.

**Usage**

The `xdrrec_skiprecord()` call can be invoked only on streams created by `xdrrec_create()`. The XDR implementation is instructed to discard the remaining data in the input buffer.

**Return codes**

The value 1 indicates success; the value 0 indicates an error.
xdrstdio_create()

Format

```c
#include <rpc.h>
#include <stdio.h>
void
xdrstdio_create(xdrs, file, op)
XDR *xdrs;
FILE *file;
enum xdr_op op;
```

Parameters

- **xdrs**
  Indicates the pointer to an XDR stream.

- **file**
  Specifies the data set name for the input/output (I/O) stream.

- **op**
  Determines the direction of the XDR stream (either XDR_ENCODE, XDR_DECODE, or XDR_FREE).

Usage

The xdrstdio_create() call creates an XDR stream connected to a file through standard I/O mechanisms. It initializes the XDR stream pointed to by xdrs. Data is written to, or read from, file.
xprt_register()

Format

```c
#include <rpc.h>
void
xprt_register(xprt)
SVCXPRTP *xprt;
```

Parameters

- **xprt**
  - Indicates the pointer to the service transport handle.

Usage

The `xprt_register()` call registers service transport handles with the RPC service package. This routine also modifies the global variables `svc_fds` and `svc_fdset`.

Context

- `svc_fds`
xprt_unregister()

Format

```c
#include <rpc.h>
void
xprt_unregister(xprt)
SVCXPRT *xprt;
```

Parameters

- `xprt`
  Indicates the pointer to the service transport handle.

Usage

The `xprt_unregister()` call unregisters an RPC service transport handle. A transport handle should be unregistered with the RPC service package before it is destroyed. This routine also modifies the global variables svc_fds and svc_fdset.
Sample RPC programs

z/OS Communications Server provides sample RPC programs. The C source code can be found in the SEZAINST data set.

The following are sample C source modules:

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEZAINST(GENESEND)</td>
<td>RPC client</td>
</tr>
<tr>
<td>SEZAINST(GENESEVR)</td>
<td>RPC server</td>
</tr>
<tr>
<td>RAWEX</td>
<td>RAW client/server</td>
</tr>
</tbody>
</table>

Running RPC sample programs

This topic provides information needed to run the GENESERV, GENESEND, and RAWEX modules.

Starting the GENESERV server

To start the GENESERV server, run GENESERV on the other MVS address space (server).

Note: Portmapper must be running before you can run GENESERV.

Running GENESEND client

To start the GENESEND client, run GENESEND MVSX 4445 (MVSX is the name of the host machine where the GENESERV server is running, and 4445 is the integer to send and return).

The following output is displayed:

Value sent: 4445
Value received: 4445

Running the RAWEX module

To start RAWEX, run RAWEX 6667, (6667 is an integer chosen by you).

The following output is displayed:

Argument: 6667
Received: 6667
Sent: 6667
Result: 6667

RPC client

The following is an example of an RPC client program.

Note: The characters shown in this example might vary due to differences in character sets. This code is included as an example only.
Figure 4. RPC client program sample (Part 1 of 2)
RPC server

The following is an example of an RPC server program.

```c
fprintf(stderr,"usage: %s hostname integer\n", argv[0]);
exit (-1);
} /* endif */
innumber = atoi(argv[2]);
/*
 * Send the integer to the server. The server should
 * return the same integer.
 */
error = callrpc(argv[1],intrcvgorg,version,intrcvgvproc,xdr_int,
                 (char *)&innumber,xdr_int,(char *)&outnumber);
if (error != 0) {
    fprintf(stderr,"error: callrpc failed: %d \n",error);
    fprintf(stderr,"intrcvgorg: %d version: %d intrcvgvproc: %d",
            intrcvgorg, version,intrcvgvproc);
    exit(1);
} /* endif */
printf("value sent: %d value received: %d\n", innumber, outnumber);
exit(0);
}

Figure 4. RPC client program sample (Part 2 of 2)
```
Figure 5. RPC server program sample (Part 1 of 2)
The following is an example of an RPC raw data stream program.

```c
float *floatrcv();

/*REGISTER PROG, VERS AND PROC WITH THE PORTMAPPER*/

/*FIRST PROGRAM*/
registerrpc(intrcvprog,intrvers,intrcvcpl,intrcv,xdr_int,xdr_int);
printf("Intrcv Registration with Port Mapper completed\n");

/*OR MULTIPLE PROGRAMS*/
registerrpc(floatcho,fltvers,fltrcvproc,
floatrcv,xdr_float,xdr_float);
printf("Floatrcv Registration with Port Mapper completed\n");

/* svc_run will handle all requests for programs registered. */
svc_run();
printf("Error:svc_run returned!\n");
exit(1);
}

/* Procedure called by the server to receive and return an integer. */
int *
intrcv(in)
int *in;
{
    int *out;
    printf("integer received: %d\n",*in);
    out = in;
    printf("integer being returned: %d\n",*out);
    return(out);
}

/* Procedure called by the server to receive and return a float. */
float *
floatrcv(in)
float *in;
{
    float *out;
    printf("float received: %e\n",*in);
    out = in;
    printf("float being returned: %e\n",*out);
    return(out);
}
```

Figure 5. RPC server program sample (Part 2 of 2)
This program does not access an external interface. It provides
a test of the raw RPC interface allowing a client and server
program to be in the same process.

/*
 ifndef MVS
 define MVS
 endif
 #include <rpc.h>
 #include <stdio.h>

 #define rawprog ((u_long)150104)
 #define rawvers ((u_long)1)
 #define rawproc ((u_long)1)

 extern enum clnt_stat clntraw_call();
 extern void raw2();

 main(argc,argv)

 Figure 6. RPC raw data stream program sample (Part 1 of 3)
int argc;
char **argv[];
{
    SVCXPRT *transp;
    struct hostent *hp;
    struct timeval pertry_timeout, total_timeout;
    struct sockaddr_in server_addr;
    int bout, in;
    register CLIENT *clnt;
    enum clnt_stat cs;
    int addrlen;

    /* The only argument passed to the program is an integer to 
     * be transferred from the client to the server and back. 
     */
    if(argc!=2) {
        printf("usage: %s integer\n", argv[0]);
        exit(-1);
    }
    in = atoi(argv[1]);

    /* Create the raw transport handle for the server. 
     */
    transp = svcraw_create();
    if (transp == NULL) {
        fprintf(stderr, "can't create an RPC server transport\n");
        exit(-1);
    }

    /* In case the program is already registered, deregister it */
    pmap_unset(rawprog, rawvers);

    /* Register the server program with PORTMAPPER */
    if (!svc_register(transp, rawprog, rawvers, raw2, 0)) {
        fprintf(stderr, "can't register service\n");
        exit(-1);
    }

    /* The following registers the transport handle with internal 
     * data structures. 
     */
    xprt_register(transp);

    /* Create the client transport handle. 
     */
    if ((clnt = clntraw_create(rawprog, rawvers)) == NULL) {
        clnt_pcreateerror("clntudp_create");
        exit(-1);
    }

    total_timeout.tv_sec = 60;
    total_timeout.tv_usec = 0;
    printf("Argument: %d\n", in);
}

Figure 6. RPC raw data stream program sample (Part 2 of 3)
RPCGEN sample programs

This topic provides information about sample RPCGEN programs. The C source code can be found in the SEZAINST data set.

The following are sample C source files:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RG</td>
<td>RPCGEN user-generated input</td>
</tr>
<tr>
<td>RGUC</td>
<td>RPCGEN user-generated client</td>
</tr>
<tr>
<td>RGUS</td>
<td>RPCGEN user-generated server</td>
</tr>
</tbody>
</table>

Generating your own sequential data sets

The following steps describe how to generate your own sequential data sets:

1. Execute RPCGEN RG from the TSO command line.

The following sequential data sets are generated in your user space:

/* Make the call from the client to the server. */
cs=clnt_call(clnt,rawproc,xdr_int,
             (char *)&in,xdr_int,(char *)&bout,total_timeout);

printf("Result: %d",bout);
if(cs!=0) {
    clnt_perror(clnt,"Client call failed");
    exit(1);
}
exit(0);

/* Service procedure called by the server when it receives the client request. */
void raw2(rqstp,transp)
    struct svc_req *rqstp;
    SVCXPRT *transp;
{
    int in,out;
    if (rqstp->rq_proc=rawproc) {
        /* Unpack the integer passed by the client. */
        svc_getargs(transp,xdr_int,&in);
        printf("Received: %d\n",in);
        /* Send the integer back to the client. */
        out=in;
        printf("Sent: %d\n",out);
        if (!svc_sendreply(transp, xdr_int,&out)) {
            printf("Can't reply to RPC call.\n");
            exit(1);
        }
    }
}

Figure 6. RPC raw data stream program sample (Part 3 of 3)
Building client and server executable modules

Complete the following steps to build client and server executable modules:
1. Compile the RGUS C source program.
2. Compile the RGUC C source program.
3. Compile the RGS.C C source program generated by RPCGEN.
4. Compile the RGC.C C source program generated by RPCGEN.
5. Link-edit the sample source modules RGS and RGUS.
6. Link-edit the sample source modules RGUC and RGC.

Running RPCGEN sample programs

This topic provides information needed to run the sample programs in RPCGEN.
1. Execute RGS on the other MVS address space (server).
   No message is displayed.
2. Execute RGUC MVSX 6504 (MVSX is the host machine where the RGS server is running, and 6504 is the integer chosen by you).
   After executing the RGUC client, the following message is displayed:
   Output on the server session: 6504
Chapter 9. Remote procedure calls in the z/OS UNIX System Services environment

The z/OS UNIX files used by z/OS UNIX System Services RPC and their location in the z/OS UNIX file system are as follows:

- /usr/include/rpc: All header files are contained here.
- /usr/lib/librpclib.a: RPC archive files.
- orpcgen: ONC RPC protocol compiler.
- orpcinfo: Utility program for looking at portmaps of networked machines.
- oportmap: Network service program that maps ONC RPC program and version numbers to transport-specific port numbers.

Deviations from Sun RPC 4.0

z/OS UNIX System Services RPC deviates from Sun RPC 4.0 in the following ways:

- The source was modified to fit into 72 columns.
- xdr_enum()
  In z/OS UNIX System Services rpc xdr_enum() is a macro. This is a change identical to the changes in TCP/IP Version 2 for MVS and VM, and Version 3.1 for MVS. It is necessary because enumerations in C/370™ may have a length of 1, 2, or 4 bytes. The enum_t is not defined and xdr_enum() is replaced first by a call to _xdr_enum() that returns the entry to the appropriate XDR routine (xdr_char(), xdr_short(), or xdr_long()), which is then followed by a call to that routine. The xdr_union() is also modified into a macro, which separates the call for the discriminant from the remainder. The discriminant is processed as an enumeration, and then passed as a value to _xdr_union() to process the remaining union.
- xdr_string()
  As with previous 370 versions of TCP/IP, xdr_string() translates from EBCDIC to ASCII or reverse. With z/OS UNIX System Services the iconv() call is used, and data is translated directly into or out of the XDR buffers if sufficient buffers are available as indicated by an xdr_inline() call. With previous versions (or with z/OS UNIX System Services if the entire string will not fit into the buffer) it is necessary to allocate an additional buffer. While encoding, if the length of the data changes in the translation, xdr_setpos() is used to adjust the XDR buffer to reflect the actual amount of translated data. realloc() is used while decoding or for the temporary buffer, which may be necessary while encoding. The default translation is between ISO8859-1 and IBM-1047. This can be modified by iconv_open() calls during initialization, by specifying the external iconv_t variables xdr_hton_cd and xdr_ntoh_cd.
- xdr_float(), xdr_double()
  The format for S/370™ floating point data differs from the IEEE format specified for XDR. The xdr_float() and xdr_double() routines are modified to make the necessary conversions. For z/OS UNIX System Services, these routines utilize the C/370 library routines frexp() and ldexp() to extract and restore the exponent from the floating point number, rather than private subroutines.
Using z/OS UNIX System Services RPC

For RPC, a Sun ONC sample program is provided in /usr/lpp/tcpip/rpc/samples. To run the sample, you can run the Makefile facility in the rpc samples directory. Running make produces three executable files.

- **printmsg**
  The command **printmsg text** prints the message (text) on the local console. It can be displayed by viewing the system log.

- **msg_svc**
  msg_svc is an RPC server that enables the user at a remote station to put a message on the console of the server. The command **msg_svc &** starts this server.

- **rprintmsg**
  The command **rprintmsg rhost text** prints a message (text) on the console of host rhost.

**Note:** The _C89_LIBDIRS_ environmental variable must be set (for example, 
`export C89_LIBDIRS=/usr/lpp/tcpip/lib`) before the make is executed.

A sample makefile is provided: /usr/lpp/tcpip/rpc/samples/Makefile. To run make, use make -f /usr/lpp/tcpip/rpc/samples/Makefile from a writable directory.

New cache call function for RPC

```c
svcudp_enablecache(transp, size)
SVCXPR *transp;
ulong size;
```

where:

- **svcudp_enablecache** enables the caching of replies to remote calls using UDP. When a request due to a retry is received, and there is a reply to an earlier attempt in the cache, the cached reply is immediately returned to the client without calling the remote procedure.

- **transp** is the UDP service transport for which caching is to be enabled.

- **size** is the number of entries to be provided in the cache.

When issuing RPCGEN for a specification file that contains a %#, the following compiler error message may be displayed: ERROR EDC0401 abc.x:n The character is not valid, where abc.x is the name of the file and n is the line number containing a %#. This combination of characters is not accepted by the compiler.

Support for 64-bit integers

Four XDR functions support 64-bit integers in the z/OS UNIX System Services RPC API.

The function xdr_hyper() is equivalent to xdr_longlong_t(). The function xdr_u_hyper() is equivalent to xdr_u_longlong_t.

<table>
<thead>
<tr>
<th>XDR Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xdr_hyper()</td>
<td>Translates between C long longs and their external representatives.</td>
</tr>
<tr>
<td>xdr_u_hyper()</td>
<td>Translates between C unsigned long longs and their external representatives.</td>
</tr>
</tbody>
</table>
**XDR Function**

- **xdr_longlong_t()**
  - Translates between C long longs and their external representatives.

- **xdr_u_longlong_t()**
  - Translates between C unsigned long longs and their external representatives.

---

**UDP transport protocol CLIENT handles**

The function of `clntudp_bufcreate()` is similar to `clnttcp_create()` but creates UDP transport protocol CLIENT handles. The wait time for retries and timeouts is specified for the UDP transport. The total time allowed for RPC completion can be specified by `clnt_call()`. Buffer sizes may be specified or defaulted. The same potential for version number mismatch exists. Success returns the CLIENT handle, failure NULL.

```c
CLIENT *
clntudp_bufcreate(addr, prognum, versnum, wait, sockp,sendsz,recvsz)
    struct sockaddr_in *addr;
    u_long prognum, versnum;
    struct tmeval wait;
    int *sockp;
    u_int sendsz;
    u_int recvsz;
```

---

**RPC restrictions**

RPC does not support the Binary Floating Point Facility. If you install the BFP processor, you must compile your RPC applications to preclude use of the BFP hardware. You can do this by specifying compiler option `ARCH(0)`, (the default setting).
Chapter 10. Network Computing System

The Network Computing System (NCS) is a set of tools for heterogeneous distributed computing. These tools conform to the Network Computing Architecture. This topic introduces the Network Computing Architecture and NCS.

To use the NCS system calls, you must know C language programming. For more information about NCS, see the NCS for IBM AIX/ESA® Planning and Administration Guide and the NCS for IBM AIX/ESA Programming Reference.

NCS and the Network Computing Architecture

NCS is an implementation of the Network Computing Architecture, an architecture for distributing software applications across heterogeneous collections of computers, networks, and programming environments. Programs based on NCS can take advantage of computing resources throughout a network or internet, with different parts of each program executing on the computers best suited for the tasks.

The Network Computing Architecture supports distributed programs of many kinds. For example, one program might perform graphical input and output on a workstation while it does intense computation on a supercomputer. Another program might perform many independent calculations on a large set of data; it could distribute these calculations among any number of available processors on the network or internet.

NCS components

The components of NCS are written in portable C wherever possible. They are available in source code and in several binary formats. Currently, the NCS components are:

- Remote procedure call (RPC) runtime library
- Location Broker
- Network Interface Definition Language (NIDL) compiler

The RPC runtime library and the Location Broker provide runtime support for network computing. These two components, along with various utilities and files, make up the Network Computing Kernel (NCK), which contains all the software you need to run a distributed application.

The Network Interface Definition Language (NIDL) compiler is a tool for developing distributed applications.

Remote procedure call runtime library

The RPC runtime library is the backbone of the Network Computing System. It provides the calls that enable local programs to execute procedures on remote hosts. These calls transfer requests and responses between clients (the programs calling the procedures) and servers (the programs executing the procedures).

When you write NCS applications, you usually do not use many RPC runtime library calls directly. Instead, you write interface definitions in NIDL and use the NIDL compiler to generate most of the required calls to the runtime library.
**Location broker**

A broker is a server that provides information about resources. The location broker enables clients to locate specific objects (for example, databases) or specific interfaces (for example, data retrieval interfaces). Location broker software includes the global location broker (GLB), the local location broker (LLB), a client agent through which programs use GLB and LLB services, and administrative tools.

The GLB stores in a database the locations of objects and interfaces throughout a network or internet; clients can use the GLB to access an object or interface without knowing its location beforehand. The LLB stores in a local database similar information about resources on the local host; it also implements a forwarding facility that provides access by means of a single address to all of the objects and interfaces at the host.

**Network interface definition language compiler**

The NIDL compiler takes as input an interface definition written in NIDL. From this definition, the compiler generates source code in portable C for client and server stub modules. An interface definition specifies the interface between a user of a service and the provider of the service; it defines how a client sees a remote service and how a server sees requests for its service.

The stubs produced by the NIDL compiler contain nearly all of the remoteness in a distributed application. They perform data conversions, assemble and disassemble packets, and interact with the RPC run-time library. It’s much easier to write an interface definition in NIDL than it would be to write the stub code that the NIDL compiler generates from your definition.

**MVS implementation of NCS**

The following list indicates the NCS components that are available in MVS or z/OS UNIX.

- Network Interface Definition Language (NIDL) compiler 1.0
- Network Computing Kernel (NCK) 1.1

The IBM MVS implementation of NCS differs from the Apollo Computer, Inc. implementation of NCS. The following list summarizes the differences between the two implementations:

- The IBM MVS implementation of NCS contains support for the Non-Replicated Global Location Broker daemon (nrglbd). It does not contain support for the Global Location Broker daemon (gldb), which can be replicated on multiple hosts in the network.
- The IBM MVS implementation of NCS does not contain support for the Data Replication Manager Administrative Tool (drm_admin). This tool works only with the replicated version of the Global Location Broker, which is not supported in MVS NCS.
- The IBM MVS implementation of NCS does not support multitasking. Neither does it support forking or spawning a task. It does not support Apollo’s Concurrent Programming Support (CPS).
- The IBM MVS implementation of NCS supports AF_INET only.
- In NCS, the receiving machine (client or server) translates EBCDIC characters to ASCII and ASCII characters to EBCDIC. The IBM MVS implementation of NCS translates correctly, but the Apollo NCS Version 1.0 code has the following problems:
– The EBCDIC Null character 0x00 is incorrectly translated to the ASCII character 0x02. It should be translated to the ASCII character 0x00.
– The EBCDIC Delete character 0x07 is incorrectly translated to the ASCII character 0x10.
– The EBCDIC Line Feed character 0x25 is incorrectly translated to the ASCII character 0x3f.

These are the three significant errors in the EBCDIC to ASCII translation table that is part of NCS Version 1.0. EBCDIC to ASCII translation works correctly only if you do not use the previous characters or if the EBCDIC to ASCII translation table has already been fixed in the NCS program on the receiving side.

• NCS Version 1.0 does not correctly translate between IBM floating point and IEEE floating point. This includes both the translation from IEEE to IBM floating point and IBM to IEEE floating point. As with EBCDIC to ASCII translations, the receiver of the data performs the floating point conversion. Servers and clients can both act as receivers of data. Therefore, NCS programs on both sides need to contain correct support of IBM floating point if you pass floating point data to or from a system that uses IBM floating point.

• Apollo NCS Version 1.0 supports two enum data types: the short enum, which NCS assumes occupies 2 bytes in storage; and the regular enum, which occupies 4 bytes. The IBM C/370 compiler dynamically determines the size required for an enum variable as 1 byte, 2 bytes, or 4 bytes.

The NCS short enum data type works correctly on MVS, but the NCS regular enum data type does not. If for some reason you cannot use the short enum data type on MVS and must use the regular enum data type, then you must force the C/370 compiler to allocate 4 bytes for all enum variables.

If your Interface Definition Language (IDL) contains enum typedefs as input to the NIDL compiler, for example

```c
typedef enum {low, medium, high} word;
typedef enum {red, green, blue} colors;
```

then you must modify the header data set that gets generated by the NIDL compiler. If the header data set is to be used on MVS with the C/370 compiler, you must force the compiler to use fullword enumeration types:

```c
/* you should add the following define to the header data set */
#define INT_MAX (0x7fffffff)

/* you need to modify the declares for the enum data type to */
/* force the compiler to use 4 bytes (word) for regular enum. */
enum word {low, medium, high, word_expand_to_fullword = INT_MAX};
enum colors {red, green, blue, colors_expand_to_fullword = INT_MAX};
```

If you do not force the compiler to use fullword enumeration types, the compiler assigns either 1 byte or 2 bytes to your enum variables and the enum variables are not transmitted correctly using NCS.

**Note:** MVS NCS does not support C language pragma statements.

---

**NCS system IDL data sets**

The NCS System Interface Definition Language (IDL) data sets consist of several interface definition data sets that are distributed with NCS. These data sets define types and constants, or local or remote interfaces. Some of these data sets can be imported by your own IDL data set. The import declaration is an NIDL statement
similar to the C #include directive, which causes other IDL data sets to be included by the NIDL compiler. You do not need to run NIDL against the data sets to be imported.

- base.idl
- conv.idl
- glb.idl
- lb.idl
- llb.idl
- nbase.idl
- rpc.idl
- rrpc.idl
- socket.idl
- uuid.idl

For more information on IDL files, see the *NCS for IBM AIX/ESA Planning and Administration Guide*.

**NCS C header data sets and the Pascal include data set**

The following is a list of the C header data sets that you might need to include in your C source programs to use NCS. These data sets can also be included by the NIDL-generated stub code. These data sets are located in SEZACMAC and must be copied to your user ID.

The following is a list of the headers used by NCS:

```
base.h
conv.h
glb.h
bsdtocms.h
idl@base.h
lb.h
llb.h
nbase.h
ncsdefs.h
ncssock.h
pfm.h
rpc.h
rrpc.h
socket.h
uuid.h
```

IDL@BASE.COPY is the name of the Pascal include data set. This data set should be included in your client or server source code if it is written in Pascal.

**NCS RPC run-time library**

On MVS, all of the routines that make up the NCS RPC run-time library are stored in the SEZALIBN data set. This library must be specified on the SYSLIB DD statement of your link-edit job step.

**NCS portability issues**

There are several NCS-based portability issues of which you need to be aware.

**NCS defines NCSDEFS.H**

The linkage editor and loader on MVS restrict the number of characters in an external name to eight characters or less. This means that if you are porting an existing non-MVS program, and it contains external references that are longer than eight characters, you need to redefine these references into unique, eight-character
names. If you are writing new code on MVS and you create external references that are longer than eight characters, you also have to redefine these references into unique eight-character names.

A data set called NCSDEFS.H, contains the redefines of all the external references greater than eight characters in length that are part of the NCS RPC run-time library. This data set needs to be included in all of your code that uses NCS.

Figure 7 shows the lines of code that should be included in each NCS-based routine to maintain portability of your code.

```c
#ifdef IBM370
    # include "ncsdefs.h" /* NCS redefines for IBM 370.*/
#endif
Figure 7. Macro to maintain IBM System/370 portability
```

To compile the code on MVS, define IBM370 to the compiler by using the compile option DEFINE(IBM370). By isolating MVS-dependent sections of code, you can maintain code portability.

### Required user-defined USERDEFS.H

The NIDL compiler generates stub code. For this stub code to compile correctly on MVS, the external references greater than eight characters must be redefined to eight characters or less. The data set USERDEFS.H contains a template for the information that needs to be redefined.

The following are considerations when using the USERDEFS.H data set.

- The data set should be copied to your user ID and be renamed to something appropriate for your NCS-based code (for example, user_id.USERDEFS.H).
  
  This data set is a good place to put any code-specific external names longer than eight characters that need to be redefined.
- The data set must always contain the redefines for the server and client entry point vector (epv). See the example USERDEFS.H data set shown in this topic for more information about USERDEFS.H.
- The data set should be included in all your NCS-based source code
- The data set must be included by the NIDL-generated stubs and switches.
  
  To have NIDL automatically add this include, use the NIDL run-time option -inc.

Figure 8 shows the H data set in the stub and switch code. You should also follow this method for including the USERDEFS.H data set (or whatever you renamed it) in your NCS-based code.

```c
#ifdef IBM370
    # include "ncsdefs.h"
    # include "userdefs.h"
#endif
Figure 8. NCSDEFS.H and USERDEFS.H include statements
```

The following provides an example of the USERDEFS.H data set:

```c
/**********************************************************
* Template for User Redefines
* On IBM MVS or MVS operating systems external references longer
```

Chapter 10. Network Computing System 345
* than 8 characters must be redefined to 8 characters
* or less. This data set must be included in your Client or Server
* code, and you must provide the nidl compiler with the name of
* this data set when nidl is invoked so that the stub code can also
* include it.
***

```c
#define IDL_interface_name _server_epv xxxSEpv
#define IDL_interface_name: _client_epv xxxCEpv
```

The following is a description of the elements shown in the preceding example.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDL_interface_name</td>
<td>The interface name coded in your IDL data set. You must replace <code>IDL_interface_name</code> with this name.</td>
</tr>
<tr>
<td>xxx</td>
<td>A unique three-character sequence, starting with a letter, that makes this redefine name unique throughout your NCS-based programs. For example, the <code>xxx</code> could be replaced with the first 3 characters of the <code>IDL_interface_name</code>.</td>
</tr>
</tbody>
</table>

See the "NIDL compiler options" on page 348 for a description of NIDL run-time options.

### NCS: Preprocessing, compiling, and linking

The following topics provide information about how to compile and link-edit your program:

- NCS Preprocessor Programs
- Compiling and Linking NCS Programs

#### NCS preprocessor programs

The NIDL compiler translates an NIDL interface definition into the NCS client and server stub modules. Before the C/C++ for z/OS compiler can be run on NCS-based code, any $ (such as those in the NCS RPC run-time library routines) must be converted to an underscore (_). You can use CPP to do this conversion. For more information about CPP, see the "Converting C identifiers using the CPP program" on page 348.

#### NIDL compiler

The Network Interface Definition Language (NIDL) compiler is a member of SEZALOAD. MVS data sets written in NIDL must have the form `user_id.name.IDL`. The NIDL compiler generates a server stub data set, a client stub data set, a client switch data set, and a header data set.

For more information about NIDL, see the NCS for IBM AIX/ESA Planning and Administration Guide.

A command list (CLIST) called RUNNIDL is provided to assist you in invoking the NIDL compiler. RUNNIDL is a member of SEZAINST. The NIDL options specified in RUNNIDL CLIST are set to the most frequently used NIDL run-time options. If you do not want to run with these NIDL options, you can invoke the NIDL compiler directly.

The NIDL compiler does not support IDL include files that are members of a partitioned data set.
Any NCS system IDL files that are imported by your IDL data set must be copied from SEZAINST to your user ID. The following are the members of SEZAINST that you might need to copy.

<table>
<thead>
<tr>
<th>Member</th>
<th>Data set name</th>
</tr>
</thead>
<tbody>
<tr>
<td>basei</td>
<td>user_id.base.idl</td>
</tr>
<tr>
<td>convi</td>
<td>user_id.conv.idl</td>
</tr>
<tr>
<td>glbi</td>
<td>user_id.glb.idl</td>
</tr>
<tr>
<td>lbi</td>
<td>user_id.lb.idl</td>
</tr>
<tr>
<td>llbi</td>
<td>user_id.llb.idl</td>
</tr>
<tr>
<td>nbasei</td>
<td>user_id.nbase.idl</td>
</tr>
<tr>
<td>rpci</td>
<td>user_id.rpc.idl</td>
</tr>
<tr>
<td>rrpci</td>
<td>user_id.rrpc.idl</td>
</tr>
<tr>
<td>socketi</td>
<td>user_id.socket.idl</td>
</tr>
<tr>
<td>uuidi</td>
<td>user_id.uuid.idl</td>
</tr>
</tbody>
</table>

Use the RUNNIDL CLIST command in the following format:

```
RUNNIDL IDL_d_s_n IDL
```

Parameter Description

- **IDL_d_s_n**: Specifies the data set name of the NIDL data set.
- **IDL**: Specifies the data set type of the NIDL data set. The data set type must be IDL.
- **inc (d_s_n)**: Specifies the data set name of a header data set that contains redefines specific to your programs and stubs. The NIDL compiler generates code to include the user-specified-include data set name in the stub data set and switch code that it generates. The data set name defaults to the USERDEFS.H data set.
- **pascal**: Specifies that the NIDL compiler generates a Pascal language include data set as output. The server stub data set, client stub data set, client switch data set, and header data sets are generated in C language.

The following example invokes the NIDL compiler using the BANK.IDL data set as input. The header data set containing the redefines for BANK is in the data set BANKDEFS.H.

```
RUNNIDL BANK IDL inc (bankdefs)
```

**NIDL compiler limitations:** You should be aware of the following limitations concerning the NIDL compiler options on MVS.

- **−no_cpp**
  
  You cannot invoke the NCS CPP routine from within the NIDL compiler. If you invoke NIDL directly, you must specify the −no_cpp option.

- **−ext**
  
  The extension option is used to generate unique data set names for the NIDL output. The defaults for −ext on MVS are @C.C@CSTUB, @S.C@SSTUB, and @W.C@CSWTCH. The extension is appended to the data set name of the IDL data set to generate a unique data set name for the two stubs and the switch.
For example, the IDL data set name and default extension for a client switch are appended in the following format:

```
IDL_data_set_name@W.C@CSWTCH
```

**Note:** This default restricts the IDL data set name to 6 characters or less.

The following is a list of data set names and default low-level qualifiers for the NIDL generated output:

<table>
<thead>
<tr>
<th>Data set name</th>
<th>Low-level qualifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>IDL_data_set_name@C</code></td>
<td><code>C@CSTUB</code></td>
<td>Client stub</td>
</tr>
<tr>
<td><code>IDL_data_set_name@W</code></td>
<td><code>C@CSWTCH</code></td>
<td>Client switch</td>
</tr>
<tr>
<td><code>IDL_data_set_name@S</code></td>
<td><code>C@SSTUB</code></td>
<td>Server stub</td>
</tr>
<tr>
<td><code>IDL_data_set_name</code></td>
<td><code>H</code></td>
<td>C header data set</td>
</tr>
<tr>
<td><code>IDL_data_set_name</code></td>
<td><code>COPY</code></td>
<td>Pascal header data set (if the pascal option is used).</td>
</tr>
</tbody>
</table>

You can change this default by invoking NIDL directly and specifying your own `-ext` option. If you specify your own `-ext` option, the name of your data set is restricted to a maximum of 8 characters, and the extension is restricted to a maximum of 8 characters.

**NIDL compiler options:** The linkage editor and loader on MVS restricts the number of characters in an external name to 8 characters or less. For the code generated by the NIDL compiler to compile correctly on MVS, the external references greater than 8 characters need to be redefined to 8 characters or less. The data set USERDEFS.H contains a template for the information to be redefined.

The `-inc` option allows you to specify the data set name of a header data set that contains redefines specific to your programs and stubs. If the `-inc` option is specified, the NIDL compiler generates code to `#include` the user-specified `-inc` data set name in the stub and switch code that it generates.

For example, the BANK sample program has a BANKDEFS.H data set, where all of the BANK external names greater than 8 characters are redefined. When the NIDL compiler is run against the BANK.IDL data set, if you specified `-inc bankdefs`, the `#include` for this data set is automatically generated in the two stubs and switch programs. The following is an example of the code:

```
#ifdef IBM370
# include "ncsdefs.h"
# include "bankdefs.h"
#endif
```

**Converting C identifiers using the CPP program**

All of the NCS RPC run-time library routines and most of the NCS constants and data types contain a $ character. For example, the routine you call to register your server with RPC run-time is `rpc_register`. The routine you call to register your server with the location broker is `lb_register`.

IBM C/370, based on ANSI standards, does not allow a $ to be used as a correct character in a C identifier. The IBM C/370 preprocessor does not allow you to redefine a $ to another character. NCS provides a routine called CPP for systems that do not allow a $ in C identifiers. The NCS CPP program reads a C source data set, expands macros and include data sets, and writes an input for the C compiler. The most important function that the CPP program performs for MVS NCS users is that it converts every $ to an underscore (_) when it occurs in a C identifier.
Before any of your code or the stub code can be compiled, all occurrences of a $ in a C identifier must be converted to an underscore (_). NCS uses CPP to do this.

**Note:** Because CPP does not contain all the functions of the C/370 preprocessor, there can be times when you need to modify your code to make it acceptable to CPP, even though C/370 might have accepted it.

A CLIST called RUNCPP is provided to assist you in invoking the CPP program. You can use this CLIST, or invoke CPP directly. RUNCPP is a member of SEZAINST.

Use the RUNCPP CLIST command in the following format:

```
RUNCPP data_set_name data_set_type
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_set_name</td>
<td>Specifies the name of the data set used as input to NCS CPP.</td>
</tr>
<tr>
<td>data_set_type</td>
<td>Specifies the data set type.</td>
</tr>
</tbody>
</table>

To run CPP with the data set BANK.C@CSTUB as input, enter the following:

```
RUNCPP BANK C@CSTUB
```

The RUNCPP CLIST has the most frequently used CPP run-time options hard coded into it. IBM recommends using RUNCPP, but if you must use options that are not specified with RUNCPP, invoke CPP directly.

For portability reasons, you should leave the $ in all the RPC run-time routines, constants, and data types. CPP should be run against your code after you run NIDL. In this way, the client stub and switch or server stub can be moved to a system that supports the $. For portability to other systems, you should always maintain the version of your code that contains the $.

For programs that are not run on any system other than IBM MVS, you can permanently change $ to (_), so that you do not have to use CPP. Then, only the client stub and switch or the server stub has to be run through the CPP routine. In some cases, this is the preferred solution, especially if you need the full function of the C/C++ for z/OS preprocessor and compiler and CPP does not include this support. For example, many AD/Cycle C/370 header files contain preprocessor directives that CPP does not understand. If you are including AD/Cycle C/370 header files in your application, you should manually change $ to underscore (_) in your application and any included header files so that you do not have to run CPP.

CPP does not support C include files that are members of a partitioned data set. Any NCS C header files that are included by your data set must be copied to your user ID. The following are the members of SEZACMAC that you might need to copy:

<table>
<thead>
<tr>
<th>Member</th>
<th>Data set name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ncssock1</td>
<td>user_id.socket.h</td>
</tr>
<tr>
<td>ncsrpc</td>
<td>user_id.rpc.h</td>
</tr>
<tr>
<td>base</td>
<td>user_id.base.h</td>
</tr>
<tr>
<td>conv</td>
<td>user_id.conv.h</td>
</tr>
<tr>
<td>glb</td>
<td>user_id.glb.h</td>
</tr>
<tr>
<td>bsdto cms</td>
<td>user_id.bsdto cms.h</td>
</tr>
</tbody>
</table>
Any C/370 standard header files that are included by your data set must be copied from the C/370 product header partitioned data set (SEZACMAC).

### Compiling and linking NCS programs

Following are the steps needed to create, build, and execute an NCS application:

1. **Set up**
   - Copy RUNNIDL and RUNCPP from SEZAINST to one of your system-supported CLIST libraries.

2. **Write the IDL description of the client and server applications.**
   - Write your NIDL interface program and client or server code, and your userdefs-type header file that redefines your long names.

3. **Run NIDL**
   - Copy any imported NCS IDL files from SEZAINST to your user ID.
   - Run the NIDL compiler using your IDL data set as input.
     ```
     RUNNIDL middle_qualifier IDL INC(userdefs)
     ```
     If your data set is `user_id.SAMPLE.IDL` and your header file is `user_id.USERDEFS.H`, the command to run is:
     ```
     RUNNIDL SAMPLE IDL INC(userdefs)
     ```

4. **Convert $ to _**
   - You can convert any identifiers containing a $ either using CPP or manually.
   - **Run CPP**
     - Copy any included header files from the partitioned data set in which it resides to your user ID.
     - Run CPP against all of your code, the client stub and switch, and the server stub.
       ```
       RUNCPP middle_qualifier low_level_qualifier
       ```
       If your data set is `user_id.SAMPLE.C`, run the following command:
       ```
       RUNCPP SAMPLE C
       ```
   - **Manually convert $ to underscore (_)**:
     - Use an editor to convert all occurrences of $ to _ in all of your code, the client stub and switch.
     - Copy to a partitioned data set any C header files that contain a $ and that are included by your code, the client stub or switch, or the server stub. Edit the C header files in the partitioned data set to convert all occurrences of $ to _. During compilation, this partitioned data set must be specified on the SYSLIB statement ahead of SEZACMAC.

5. **Compile and Link**
   - You can use several methods to compile, link-edit, and execute your C/C++ for z/OS source program in MVS. This topic contains information about the
additional data sets that you must include to run the C data sets generated by
RUNCPP under MVS batch, using IBM-supplied cataloged procedures.
The following list contains data set names, which are used as examples in the
following JCL statements:

<table>
<thead>
<tr>
<th>Data set name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_id.SAMPLE.CPPOUT</td>
<td>Sequential data set that contains the C program generated by RUNCPP.</td>
</tr>
<tr>
<td>user_id.OBJ</td>
<td>A partitioned data set that contains the compiled versions of C programs as its members.</td>
</tr>
<tr>
<td>user_id.LOADLIST</td>
<td>A partitioned data set that contains the loadlist as its members.</td>
</tr>
<tr>
<td>user_id.LOAD</td>
<td>A partitioned data set that contains the link-edited versions of C programs as its members.</td>
</tr>
<tr>
<td>user_id.HDRS</td>
<td>A partitioned data set that contains C header files as its members.</td>
</tr>
</tbody>
</table>

**NCS: Sample compile cataloged procedure additions**

Include the following in the compile step of your cataloged procedure. Cataloged procedures are included in the IBM-supplied samples for your z/OS system.

Add the following to the CPARM parameter:

```
CPARM='DEF(IBM390)'  
```

Add the following statement as the first //SYSLIB DD statement.

```
//SYSLIB DD DSN=SEZACMAC,DISP=SHR  
```

**Note:** If you do not run CPP and your C source file includes either socket.h or rpc.h, you must copy the NCS versions of these files (ncsock1 and ncsrpc) from SEZACMAC to user_id.HDRS and rename them to socket and rpc. user_id.HDRS must then be specified on the SYSLIB statement ahead of SEZACMAC.

```
//SYSLIB DD DSN=user_id.HDRS,DISP=SHR  
DD DSN=SEZACMAC,DISP=SHR  
```

**NCS: Sample link-edit cataloged procedure additions**

Include the following in the link-edit step of your cataloged procedure.

- Add the following statements as the first //SYSLIB DD statement:
  
  ```
  //       DD DSN=SEZALIBN,DISP=SHR  
  //       DD DSN=SEZACMTX,DISP=SHR  
  ```

- Add the following // USERLIB DD statement:

  ```
  //USERLIB   DD DSN=user_id.OBJ,DISP=SHR  
  ```

All entry points are not defined as external references in SEZALIBN. You must include the following when you link-edit your application code.

```
INCLUDE SYSLIB(RPCOS)  
INCLUDE SYSLIB(RPCSEQ)  
INCLUDE SYSLIB(RPCOUTIL)  
INCLUDE SYSLIB(SOCKET)  
```

- Create a member SAMPLE of partitioned data set user_id.LOADLIST and add the necessary objects to link to SAMPLE.
For example, to create SAMPLE load module with three objects (SAMPLE, SAMPLE@C, SAMPLE@W), the corresponding contents of SAMPLE in user_id.LOADLIST would be:

```
INCLUDE SYSLIB(RPC@S)
INCLUDE SYSLIB(RPC@SEQ)
INCLUDE SYSLIB(RPC@UTIL)
INCLUDE SYSLIB(SOCKET)
INCLUDE USERLIB(SAMPLE)
INCLUDE USERLIB(SAMPLE@C)
INCLUDE USERLIB(SAMPLE@W)
MODE AMODE(31)
ENTRY CEESTART
```

**Note:** For more information about compiling and linking, see the [z/OS XL C/C++ User’s Guide](#).

---

### Running UUID@GEN

The NCS program UUID@GEN generates universal unique identifiers. The UUID@GEN data set is a member of SEZALOAD.

For more information about using UUID@GEN, see the [NCS for IBM AIX/ESA Planning and Administration Guide](#).

Use the following format to invoke the UUID@GEN.

```
$UIDGEN
```

---

### NCS sample programs

The source code for the following NCS sample programs is included in SEZAINST:

- BANK
- NCSSMP
- BINOP

See [“Compiling and linking NCS programs” on page 350](#) for step-by-step instructions on compiling, link-editing, and running the sample programs. For specific instructions on building and running each sample, see [“Compiling, linking, and running the sample BINOP program” on page 353](#), [“Compiling, linking, and running the NCSSMP program” on page 357](#), and [“Compiling, linking, and running the sample BANK program” on page 362](#).

Implement the BINOP sample program on your system, then run either the NCSSMP program or BANK. BINOP uses a well-known port rather than the NCS location broker. The BINOP sample program can help verify NCS on your system.

When running the NIDL compiler against any of the sample program IDL data sets, ensure that you specify the include data set. For example, to run NIDL against the BANK.IDL data set, enter the following:

```
RUNNIDL BANK IDL inc (bankdefs)
```
The NCSSMP sample program

The following is an example of an NCS sample program. It includes the following program segments:

- NCS redefines for this sample program
- Instructions to compile and run the sample program on MVS

The source code for the following program segments are included in SEZAINST:

- NCSSERV1 (NCS server)
- NCSCLNT1 (NCS client)
- NCSSMPI (NCS NIDL interface)

NCS sample redefines

The following is an example of a redefine data set that is needed if this NCS sample program is to run on MVS:

```c
/***************************************************************
* Redefines for NCS Sample Program                             *
* On IBM VM or MVS operating systems external references longer *
* than 8 characters must be redefined to 8 characters or less. *
* This file must be included in the Sample Programs and stubs. *
***************************************************************

#define binop_server_epv  binSEpv
#define binop_client_epv  binCEpv
#define binop_add         binAdd
#define getNCShandle      binGtHnd
```

Compiling, linking, and running the sample BINOP program

The NCS sample program BINOP consists of the following data sets, which are members of SEZAINST:

<table>
<thead>
<tr>
<th>Sample data set</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINOPR</td>
<td>Describes how to run the BINOP sample program.</td>
</tr>
<tr>
<td>BINOPSC</td>
<td>Contains C source code for the BINOP server program.</td>
</tr>
<tr>
<td>BINOPCC</td>
<td>Contains C source code for the BINOP client program.</td>
</tr>
<tr>
<td>BINOP</td>
<td>Contains C source code for the BINOP remote subroutine.</td>
</tr>
<tr>
<td>BINODIS</td>
<td>Contains the interface definition language data set for BINOP sample programs used as input to the NIDL compiler.</td>
</tr>
<tr>
<td>BINDEFS</td>
<td>Indicates the header data set containing the redefines of external references, greater than 8 characters in length, used in the BINOP sample programs.</td>
</tr>
</tbody>
</table>

The following topics describe steps required to run the sample BINOP program successfully:

- “Setting up the sample BINOP program” on page 354
- “Compiling the sample BINOP program” on page 355
- “Linking the sample BINOP program” on page 356
- “Running the sample BINOP program” on page 357
Note: If you have a problem with any of these steps, you must resolve them before you can go on to the next step. If you encounter a problem, first ensure that TCP/IP for MVS or z/OS CS has been installed and is operational on your system.

Setting up the sample BINOP program

Before you begin: You need to know how to access data sets and copy files.

Perform the following steps as prerequisites to compiling, linking, and running the sample BINOP program.

1. Copy the sample data sets from SEZAINST to your user ID.

<table>
<thead>
<tr>
<th>From location</th>
<th>To location</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEZAINST(BINOP)</td>
<td>user_id.binop.c</td>
</tr>
<tr>
<td>SEZAINST(BINOPCC)</td>
<td>user_id.binopcc</td>
</tr>
<tr>
<td>SEZAINST(BINOPSC)</td>
<td>user_id.binops.c</td>
</tr>
<tr>
<td>SEZAINST(BINDEFS)</td>
<td>user_id.bindefs.h</td>
</tr>
<tr>
<td>SEZAINST(BINOPI)</td>
<td>user_id.binop.idl</td>
</tr>
</tbody>
</table>

2. Copy the imported data sets from SEZAINST to your user ID.

<table>
<thead>
<tr>
<th>From location</th>
<th>To location</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEZAINST(BASEI)</td>
<td>user_id.base.idl</td>
</tr>
<tr>
<td>SEZAINST(NBASEI)</td>
<td>user_id.nbase.idl</td>
</tr>
<tr>
<td>SEZAINST(RPCI)</td>
<td>user_id.rpc.idl</td>
</tr>
</tbody>
</table>

3. To generate stubs, run NIDL using the following command:

   RENNIDL BINOP IDL INC(BINDEFS)

4. Copy the included C header files to your user ID.

<table>
<thead>
<tr>
<th>From location</th>
<th>To location</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEZACMAC(BASE)</td>
<td>user_id.base.h</td>
</tr>
<tr>
<td>SEZACMAC(NBASE)</td>
<td>user_id.nbase.h</td>
</tr>
<tr>
<td>SEZACMAC(NCSDEFS)</td>
<td>user_id.ncsdefs.h</td>
</tr>
<tr>
<td>SEZACMAC(TYPES)</td>
<td>user_id.types.h</td>
</tr>
<tr>
<td>SEZACMAC(BSDTIME)</td>
<td>user_id.bsdtime.h</td>
</tr>
<tr>
<td>SEZACMAC(BSDTOCMS)</td>
<td>user_id.bsdtocms.h</td>
</tr>
<tr>
<td>SEZACMAC(BSDTYPES)</td>
<td>user_id.bsdtypes.h</td>
</tr>
<tr>
<td>SEZACMAC(IDL@BASE)</td>
<td><a href="mailto:user_id.idl@base.h">user_id.idl@base.h</a></td>
</tr>
<tr>
<td>SEZACMAC(PFM)</td>
<td>user_id.pfm.h</td>
</tr>
<tr>
<td>‘C’ library</td>
<td>user_id.setjmp.h</td>
</tr>
<tr>
<td>‘C’ library</td>
<td>user_id_stdio.h</td>
</tr>
<tr>
<td>‘C’ library</td>
<td>user_id_time.h</td>
</tr>
</tbody>
</table>
**Note:** C library header files depend on the compiler you are using. For example:

- C370 2.2
- AD/Cycle C/370

5. You must run CPP to change $ to _ before you can compile this code. To run CPP, enter the following commands:

   - RUNCPP BINOPS C
   - RUNCPP BINOPC C
   - RUNCPP BINOPS COSSTUB
   - RUNCPP BINOP0C COSSTUB
   - RUNCPP BINOP0W COSWTCH
   - RUNCPP BINOP C

You know you are done when RUNCPP completes with no errors.

### Compiling the sample BINOP program

**Before you begin:** You need to have completed the steps in “Setting up the sample BINOP program” on page 354.

You can use several methods to compile, link-edit, and execute your program in MVS. The following explains how to compile your C data sets generated by RUNCPP under MVS batch, using IBM-supplied cataloged procedures.

The following list contains data set names, which are used as examples in the following JCL statements:

<table>
<thead>
<tr>
<th>Data set name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_id.OBJ</td>
<td>A partitioned data set that contains the compiled versions of C programs as its members.</td>
</tr>
<tr>
<td>user_id.LOADLIST</td>
<td>A partitioned data set that contains the loadlist as its members.</td>
</tr>
<tr>
<td>user_id.LOAD</td>
<td>A partitioned data set that contains the link-edited versions of C programs as its members.</td>
</tr>
</tbody>
</table>

In order for the program to compile correctly, you must make changes to the EDCC cataloged procedure, which is supplied with IBM C for zSeries Compiler Licensed Program (5688-187).

Perform the following steps to compile your program.

1. Remove the OUTFILE and OUTDCB parameters.

2. Add the following to the CPARM parameter:

   ```
   CPARM='DEF(IBMCPP,IBM370)',
   ```

3. Replace the //SYSIN DD statement and the //SYSLIN statement with the following:

   ```
   //SYSIN DD DSN=user_id..&INFILE..CPPOUT,DISP=SHR
   //SYSLIN DD DSN=user_id.OBJ(&MEM),DISP=SHR
   ```

4. Add the following //SYSLIB DD statement:
5. Submit the compile job at the Spool Display and Search Facility (SDSF) command panel, by entering the following:

/s EDCC,INFILE=BINOPS
/s EDCC,INFILE=BINOPC
/s EDCC,INFILE=BINOPPS
/s EDCC,INFILE=BINOPQC
/s EDCC,INFILE=BINOPAW
/s EDCC,INFILE=BINOP

You know you are done when no errors are received.

Linking the sample BINOP program

Before you begin: You need to have completed the steps in “Setting up the sample BINOP program” on page 354 and “Compiling the sample BINOP program” on page 355.

In order for the program to link correctly, you must make changes to the EDCL cataloged procedure, which is supplied with IBM C for zSeries Compiler Licensed Program (5688-187).

Perform the following steps to link-edit your program.

1. Remove the OUTFILE parameter.

2. Add the following statements after the //SYSLIB DD statement:

   // DD DSN=SEZALIBN,DISP=SHR
   // DD DSN=SEZACMTX,DISP=SHR

3. Add the following //USERLIB DD statement:

   //USERLIB DD DSN=user_id.OBJ,DISP=SHR

4. Replace the //SYSLIN DD statement with the following:

   //SYSLIN DD DSN=user_id.OBJ(&MEM),DISP=SHR
   // DD DSN=user_id.LOADLIST(&MEM),DISP=SHR

5. Include the following lines when you link-edit your application code, because not all entry points are defined as external references in SEZALIBN.

   INCLUDE SYSLIB(RPC@S)
   INCLUDE SYSLIB(RPC@SEQ)
   INCLUDE SYSLIB(RPC@UTIL)
   INCLUDE SYSLIB(SOCKET)

6. Replace the //SYSLMOD DD statement with the following:

   //SYSLMOD DD DSN=user_id.LOAD(&MEM),DISP=SHR

7. Create one member of the partitioned data set user_id.LOADLIST, by adding the following lines to the data set BINOPC.

   INCLUDE SYSLIB(RPC@S)
   INCLUDE SYSLIB(RPC@SEQ)
   INCLUDE SYSLIB(RPC@UTIL)
   INCLUDE SYSLIB(SOCKET)
8. Create a second member of the partitioned data set user_id.LOADLIST, by adding the following lines to the data set BINOPS.

   INCLUDE SYSLIB(RPC@S)
   INCLUDE SYSLIB(RPC@SEQ)
   INCLUDE SYSLIB(RPC@UTIL)
   INCLUDE SYSLIB(SOCKET)
   INCLUDE USERLIB(BINOP@S)
   INCLUDE USERLIB(BINOP)
   MODE AMODE(31)
   ENTRY CEESTART

9. Submit the link-edit job at the SDSF command panel, by entering the following:

   /s EDCL,MEM=BINOPC
   /s EDCL,MEM=BINOPS

You know you are done when no errors are received.

Running the sample BINOP program

Before you begin: You need to have completed the steps in “Setting up the sample BINOP program” on page 354, “Compiling the sample BINOP program” on page 355, and “Linking the sample BINOP program” on page 356.

Perform the following steps to run your program.

1. Start the NCS server sample program on one MVS user ID by entering the following command:

   CALL 'user_id.LOAD(BINOPS)' '2'

2. Start the NCS client on a different MVS user ID by entering the following command:

   CALL 'user_id.LOAD(BINOPC)' 'hostname 2 3'

   where hostname is the name of the system that the server is running on.

You know you are done when the program runs successfully.

Compiling, linking, and running the NCSSMP program

The NCSSMP sample program consists of the following data sets, which are members of SEZAINST:

- **NCSSMPR**: Describes the NCS sample program.
- **NCSSERV1**: Contains C source code for the server for the NCS sample program.
- **NCSCLNT1**: Contains C source code for the client for the NCS sample program.
- **NCSSMPI**: Contains the interface definition language data set for the NCS sample program used as input to the NIDL compiler.
NSMPDEFS  Indicates the header data set containing the redefines of external references, greater than 8 characters in length, used in the NCS sample program.

For an example of the source code, see "The NCSSMP sample program" on page 353.

The following topics describe steps required to run the NCSSMP program successfully.

- "Setting up the NCSSMP program"
- "Compiling the NCSSMP program" on page 359
- "Linking the NCSSMP program" on page 360
- "Running the NCSSMP program" on page 361

Note: If you have a problem with any of these steps, you must resolve them before you can go on to the next step. If you encounter a problem, first ensure that TCP/IP for MVS or z/OS CS has been installed and is operational on your system. Also, ensure that the NCS Global Location Broker is running somewhere on your network.

Setting up the NCSSMP program

Before you begin: You need to know how to access data sets and copy files.

Perform the following steps as prerequisites to compiling, linking, and running the NCSSMP program.

1. Copy the sample data sets from SEZAINST to your user ID.

<table>
<thead>
<tr>
<th>From location</th>
<th>To location</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEZAINST(NCSSERV1)</td>
<td>user_id.ncsserv1.c</td>
</tr>
<tr>
<td>SEZAINST(NCSCLNT1)</td>
<td>user_id.ncscnt1.c</td>
</tr>
<tr>
<td>SEZAINST(NCSSMPI)</td>
<td>user_id.ncssmp.idl</td>
</tr>
<tr>
<td>SEZAINST(NSMPDEFS)</td>
<td>user_id.nsmpdefs.h</td>
</tr>
</tbody>
</table>

2. Copy the imported data sets from SEZAINST to your user ID.

<table>
<thead>
<tr>
<th>From location</th>
<th>To location</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEZAINST(RPCI)</td>
<td>user_id.rpc.idl</td>
</tr>
<tr>
<td>SEZAINST(BASEI)</td>
<td>user_id.base.idl</td>
</tr>
<tr>
<td>SEZAINST(NBASEI)</td>
<td>user_id.nbase.idl</td>
</tr>
</tbody>
</table>

3. To generate stubs, run NIDL using the following command:

   RUNNIDL NCSSMP IDL INC(nsmpdefs)

4. Copy the data sets included by CPP to your user ID.

<table>
<thead>
<tr>
<th>From location</th>
<th>To location</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEZACMAC(NCSDEFS)</td>
<td>user_id.ncsdefs.h</td>
</tr>
<tr>
<td>SEZACMAC(BSDTOCMS)</td>
<td>user_id.bsdtocms.h</td>
</tr>
</tbody>
</table>
### Compiling the NCSSMP program

**Before you begin:** You need to have completed the steps in “Setting up the NCSSMP program” on page 358.

You can use several methods to compile, link-edit, and execute your program in MVS. This topic explains how to compile your C data sets generated by RUNCPP under MVS batch, using IBM-supplied cataloged procedures.

The following list contains data set names, which are used as examples in the following JCL statements:

- **user_id.OBJ** A partitioned data set that contains the compiled versions of C programs as its members.
- **user_id.LOADLIST** A partitioned data set that contains the loadlist as its members.
- **user_id.LOAD** A partitioned data set that contains the link-edited versions of C programs as its members.

In order for the program to compile correctly, you must make changes to the EDCC cataloged procedure, which is supplied with IBM C for zSeries, Compiler Licensed Program (5688-187).
Perform the following steps to compile your program.

1. Remove the OUTFILE and OUTDCB parameters.

2. Add the following to the CPARM parameter:
   
   ```
   CPARM='DEF(IBMCPP,IBM370)',
   ```

3. Replace the //SYSIN DD statement and the //SYSLIN statement with the following:
   
   ```
   //SYSIN DD DSN=user_id.&MEM..CPPOUT,DISP=SHR
   //SYSLIN DD DSN=user_id.OBJ(&MEM),DISP=SHR
   ```

4. Add the following //SYSLIB DD statement:
   
   ```
   //SYSLIB DD DSN=SEZACMAC,DISP=SHR
   ```

5. Submit the compile job at the Spool Display and Search Facility (SDSF) command panel, by entering the following:
   
   ```
   /s EDCC,MEM=NCSSERV1
   /s EDCC,MEM=NCSCLNT1
   /s EDCC,MEM=NCSSMP0S
   /s EDCC,MEM=NCSSMP0C
   /s EDCC,MEM=NCSSMP0W
   ```

You know you are done when no errors are received.

**Linking the NCSSMP program**

**Before you begin:** You need to have completed the steps in “Setting up the NCSSMP program” on page 358 and “Compiling the NCSSMP program” on page 359.

In order for the program to link correctly, you must make changes to the EDCL cataloged procedure, which is supplied with IBM C for zSeries, Compiler Licensed Program (5688-187).

Perform the following steps to link-edit your program.

1. Remove the OUTFILE parameter.

2. Add the following statements after the //SYSLIB DD statement:

   ```
   // DD DSN=SEZALIBN,DISP=SHR
   // DD DSN=SEZACMTX,DISP=SHR
   ```

3. Add the following //USERLIB DD statement:
   
   ```
   //USERLIB DD DSN=user_id.OBJ,DISP=SHR
   ```

4. Replace the //SYSLIN DD statement with the following:

   ```
   //SYSLIN DD DSN=user_id.OBJ(&MEM),DISP=SHR
   // DD DSN=user_id.LOADLIST(&MEM),DISP=SHR
   ```

5. Include the following when you link-edit your application code, because not all entry points are defined as external references in SEZALIBN.
6. Replace the //SYSLMOD DD statement with the following:
   //SYSLMOD DD DSN=user_id.LOAD(MEM),DISP=SHR

7. Create one member of the partitioned data set userid.LOADLIST by adding the following lines to the data set NCSCLNT1.
   INCLUDE SYSLIB(RPC@S)
   INCLUDE SYSLIB(RPC@SEQ)
   INCLUDE SYSLIB(RPC@UTIL)
   INCLUDE SYSLIB(SOCKET)
   INCLUDE USERLIB(NCSSMP@C)
   INCLUDE USERLIB(NCSSMP@W)
   MODE AMODE(31)
   ENTRY CEESTART

8. Create a second member of the partitioned data set userid.LOADLIST by adding the following lines to the data set NCSSERV1.
   INCLUDE SYSLIB(RPC@S)
   INCLUDE SYSLIB(RPC@SEQ)
   INCLUDE SYSLIB(RPC@UTIL)
   INCLUDE SYSLIB(SOCKET)
   INCLUDE USERLIB(NCSSMP@S)
   MODE AMODE(31)
   ENTRY CEESTART

9. Submit the link-edit job at the SDSF command panel, by entering the following:
   /s EDCL,MEM=NCSCLNT1
   /s EDCL,MEM=NCSSERV1

You know you are done when no errors are received.

**Running the NCSSMP program**

**Before you begin:** You need to have completed the steps in “Setting up the NCSSMP program” on page 358, “Compiling the NCSSMP program” on page 359, and “Linking the NCSSMP program” on page 360.

Perform the following steps to run your program.

1. Make sure that the Local and Global Location Brokers are running.

2. Start the NCS server sample program on one MVS user ID by entering the following command:
   CALL 'user_id.LOAD(NCCESSRV1)'

3. Start the NCS client on a different MVS user ID by entering the following command:
   CALL 'user_id.LOAD(NCSCLNT1) '5 32'

You know you are done when the program runs successfully.
Compiling, linking, and running the sample BANK program

The NCS sample program BANK consists of the following data sets, which are members of SEZAINST:

<table>
<thead>
<tr>
<th>Sample data set</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BANKR</td>
<td>Describes how to run the BANK sample program.</td>
</tr>
<tr>
<td>BANKDC</td>
<td>Contains C language source code for the BANK server program.</td>
</tr>
<tr>
<td>BANKC</td>
<td>Contains C language source code for the BANK client program.</td>
</tr>
<tr>
<td>UTILC</td>
<td>Contains utility routines used by the BANK server and client programs.</td>
</tr>
<tr>
<td>UTILH</td>
<td>Indicates a header data set used in the BANK sample program.</td>
</tr>
<tr>
<td>UUIDBIND</td>
<td>Contains autobind and unbind source code routines used by the BANK server and client programs.</td>
</tr>
<tr>
<td>BANKIDL</td>
<td>Contains the interface definition language data set for the BANK sample programs used as input to the NIDL compiler.</td>
</tr>
<tr>
<td>SHAWMUT</td>
<td>Contains input data for BANK server program.</td>
</tr>
<tr>
<td>BAYBANKS</td>
<td>Contains input data for BANK server program.</td>
</tr>
<tr>
<td>BANKDEFS</td>
<td>Indicates a header data set containing the redefines of external references, greater than 8 characters in length, used in the BANK sample programs.</td>
</tr>
</tbody>
</table>

The following topics describe steps required to run the sample BANK program successfully.

- "Setting up the sample BANK program”
- "Compiling the sample BANK program” on page 364
- "Linking the sample BANK program” on page 365
- "Running the sample BANK program” on page 366

**Note:** If you have a problem with any of these steps, you must resolve them before you can go on to the next step. If you encounter a problem, first ensure that TCP/IP for MVS or z/OS Communications Server has been installed and is operational on your system. Also, ensure that the NCS Global Location Broker is running somewhere on your network and the Local Location Broker is running on the client system.

**Setting up the sample BANK program**

**Before you begin:** You need to know how to access data sets and copy files.

Perform the following steps as prerequisites to compiling, linking, and running the BANK program.
1. Copy the sample data sets from SEZAINST to your user ID.

<table>
<thead>
<tr>
<th>From location</th>
<th>To location</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEZAINST(BANKDC)</td>
<td>user_id.bankd.c</td>
</tr>
<tr>
<td>SEZAINST(BANKC)</td>
<td>user_id.bank.c</td>
</tr>
<tr>
<td>SEZAINST(UTILC)</td>
<td>user_id.util.c</td>
</tr>
<tr>
<td>SEZAINST(UUIDBIND)</td>
<td>user_id.uuidbind.c</td>
</tr>
<tr>
<td>SEZAINST(UTILH)</td>
<td>user_id.util.h</td>
</tr>
<tr>
<td>SEZAINST(BANKIDL)</td>
<td>user_id.bank.idl</td>
</tr>
<tr>
<td>SEZAINST(SHAWMUT)</td>
<td>user_id.shawmut.bank</td>
</tr>
<tr>
<td>SEZAINST(BAYBANK)</td>
<td>user_id.baybank.bank</td>
</tr>
<tr>
<td>SEZAINST(BANKDEFS)</td>
<td>user_id.bankdefs.h</td>
</tr>
</tbody>
</table>

2. Copy the data sets imported by IDL from SEZAINST to your user ID.

<table>
<thead>
<tr>
<th>From location</th>
<th>To location</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEZAINST(BASEI)</td>
<td>user_id.base.idl</td>
</tr>
<tr>
<td>SEZAINST(NBASEI)</td>
<td>user_id.nbase.idl</td>
</tr>
<tr>
<td>SEZAINST(RPCI)</td>
<td>user_id.rpc.idl</td>
</tr>
</tbody>
</table>

3. To generate stubs, run NIDL using the following command:
   ```
   RUNNIDL BANK IDL INC(bankdefs)
   ```

4. Copy the data sets included by CPP to your user ID.

<table>
<thead>
<tr>
<th>From location</th>
<th>To location</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEZACMAC(NCSDEFS)</td>
<td>user_id.ncsdefs.h</td>
</tr>
<tr>
<td>SEZACMAC(BSDTOCMS)</td>
<td>user_id.bsdtocms.h</td>
</tr>
<tr>
<td>SEZACMAC(BASE)</td>
<td>user_id.base.h</td>
</tr>
<tr>
<td>SEZACMAC(IDL@BASE)</td>
<td><a href="mailto:user_id.idl@base.h">user_id.idl@base.h</a></td>
</tr>
<tr>
<td>SEZACMAC(NBASE)</td>
<td>user_id.nbase.h</td>
</tr>
<tr>
<td>SEZACMAC(LB)</td>
<td>user_id.lb.h</td>
</tr>
<tr>
<td>SEZACMAC(GLB)</td>
<td>user_id.glb.h</td>
</tr>
<tr>
<td>SEZACMAC(TYPES)</td>
<td>user_id.types.h</td>
</tr>
<tr>
<td>SEZACMAC(BSDTYPES)</td>
<td>user_id.bsdtypes.h</td>
</tr>
<tr>
<td>SEZACMAC(BSDTIME)</td>
<td>user_id.bsdtime.h</td>
</tr>
<tr>
<td>SEZACMAC(PFM)</td>
<td>user_id.pfm.h</td>
</tr>
<tr>
<td>SEZACMAC(UUID)</td>
<td>user_id.uuid.h</td>
</tr>
<tr>
<td>'C' library</td>
<td>user_id.stdio.h</td>
</tr>
<tr>
<td>'C' library</td>
<td>user_id.setjmp.h</td>
</tr>
<tr>
<td>'C' library(ERRNO)</td>
<td>user_id.errno.h</td>
</tr>
<tr>
<td>'C' library(TIME)</td>
<td>user_id.time.h</td>
</tr>
</tbody>
</table>
From location To location

<table>
<thead>
<tr>
<th>From location</th>
<th>To location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Note: ‘C’ library header files depend on the compiler you are using. For example:</td>
<td></td>
</tr>
<tr>
<td>• C370 2.2</td>
<td></td>
</tr>
<tr>
<td>• AD/Cycle C/370</td>
<td></td>
</tr>
</tbody>
</table>

5. You must run CPP to change $ to _ before you can compile this code. To run CPP, enter the following commands:

```bash
RUNCPP UTIL C
RUNCPP UUIDBIND C
RUNCPP BANKD C
RUNCPP BANK C
RUNCPP BANK@S C@STUB
RUNCPP BANK@C C@CSTUB
RUNCPP BANK@T C@CSWTCH
```

You know you are done when RUNCPP completes with no errors.

**Compiling the sample BANK program**

**Before you begin:** You need to have completed the steps in “Setting up the sample BANK program” on page 362.

You can use several methods to compile, link-edit, and execute your program in MVS. This topic explains how to compile your C data sets generated by RUNCPP under MVS batch, using IBM-supplied cataloged procedures.

The following list contains data set names, which are used as examples in the following JCL statements:

<table>
<thead>
<tr>
<th>Data set name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>user_id.OBJ</code></td>
<td>A partitioned data set that contains the compiled versions of C programs as its members.</td>
</tr>
<tr>
<td><code>user_id.LOADLIST</code></td>
<td>A partitioned data set that contains the loadlist as its members.</td>
</tr>
<tr>
<td><code>user_id.LOAD</code></td>
<td>A partitioned data set that contains the link-edited versions of C programs as its members.</td>
</tr>
</tbody>
</table>

In order for the program to compile correctly, you must make changes to the EDCC cataloged procedure, which is supplied with IBM C for zSeries, Compiler Licensed Program (5688-187).

Perform the following steps to compile your program.

1. Remove the OUTFILE and OUTDCB parameters.

2. Add the following to the CPARM parameter:

   ```bash
   CPARM='DEF(IBMCPP,IBM370)',
   ```

3. Replace the ```//SYSIN DD``` statement and the ```//SYSLIN``` statement with the following:

   ```bash
   //SYSIN DD DSN=user_id.&MEM..CPPOUT,DISP=SHR
   //SYSLIN DD DSN=user_id.OBJ(&MEM),DISP=SHR
   ```
4. Add the following //SYSLIB DD statement:
   
   ```
   //SYSLIB DD DSN=SEZACMAC,DISP=SHR
   ```

5. Submit the compile job at the Spool Display and Search Facility (SDSF) command panel, by entering the following:
   
   ```
   /s EDCC,MEM=BANKD
   /s EDCC,MEM=BANK
   /s EDCC,MEM=BANK0S
   /s EDCC,MEM=BANK0C
   /s EDCC,MEM=BANKW
   /s EDCC,MEM=UTIL
   /s EDCC,MEM=UUIDBIND
   ```

You know you are done when no errors are received.

**Linking the sample BANK program**

**Before you begin:** You need to have completed the steps in "Setting up the sample BANK program" on page 362 and "Compiling the sample BANK program" on page 364.

In order for the program to link correctly, you must make changes to the EDCL cataloged procedure, which is supplied with IBM C for zSeries, Compiler Licensed Program (5688-187).

Perform the following steps to link-edit your program.

1. Remove the OUTFILE parameter.

2. Add the following statements after the //SYSLIB DD statement:
   
   ```
   //DD DSN=SEZALIBN,DISP=SHR
   //DD DSN=SEZACMTX,DISP=SHR
   ```

3. Add the following //USERLIB DD statement:
   
   ```
   //USERLIB DD DSN=user_id.OBJ,DISP=SHR
   ```

4. Replace the //SYSLIN DD statement with the following:
   
   ```
   //SYSLIN DD DSN=user_id.OBJ(&MEM),DISP=SHR
   //DD DSN=user_id.LOADLIST(&MEM),DISP=SHR
   ```

5. Include the following when you link-edit your application code, because not all entry points are defined as external references in SEZALIBN.
   
   ```
   INCLUDE SYSLIB(RPC@S)
   INCLUDE SYSLIB(RPC@SEQ)
   INCLUDE SYSLIB(RPC@UTIL)
   INCLUDE SYSLIB(SOCKET)
   ```

6. Replace the //SYSLMOD DD statement with the following:
   
   ```
   //SYSLMOD DD DSN=user_id.LOAD(&MEM),DISP=SHR
   ```

7. Create one member of the partitioned data set `user_id.LOADLIST` by adding the following lines to the data set BANK:

Chapter 10. Network Computing System 365
8. Create a second member of the partitioned data set *user_id*.LOADLIST by adding the following lines to the data set BANKD:

```
INCLUDE SYSLIB(RPC@S)
INCLUDE SYSLIB(RPC@SEQ)
INCLUDE SYSLIB(RPC@UTIL)
INCLUDE SYSLIB(SOCKET)
INCLUDE USERLIB(BANK@C)
INCLUDE USERLIB(BANK@W)
INCLUDE USERLIB(UTIL)
INCLUDE USERLIB(UUIDBIND)
MODE AMODE(31)
ENTRY CEESTART
```

9. Submit the link-edit job at the SDSF command panel, by entering the following:

```
/s EDCL,MEM=BANK
/s EDCL,MEM=BANKD
```

You know you are done when no errors are received.

**Running the sample BANK program**

**Before you begin:** You need to have completed the steps in "Setting up the sample BANK program" on page 362, "Compiling the sample BANK program" on page 364, and "Linking the sample BANK program" on page 365.

Perform the following steps to run your program.

1. Make sure that the Local and Global Location Brokers are running.

2. Start the NCS server sample program on one MVS user ID. To do so, enter the following command:

```
CALL 'user_id.LOAD(BANKD)' 'ip shawmut shawmut.bank' asis
```

3. Start the NCS client on a different MVS user ID. To do so, enter the following command:

```
CALL 'user_id.LOAD(BANK)' 'inquire shawmut Leach' asis
```

You know you are done when the program runs successfully.
Chapter 11. Running the sample mail filter program

This topic explains how to run the sample mail filter program, lf_smpl.c. A mail filter is designed to provide more functionality for a sendmail daemon. These functions might include adding a recipient, scanning for viruses, rejecting a disallowed recipient address, and so on. This sample mail filter creates a file in /tmp named msg.XXXXXXXX (where X represents any combination of letters and numbers) to log the message body and headers.

Compiling and linking the lf_smpl.c source code

The following source is needed to compile and link the sample filter:

- /usr/lpp/tcpip/samples/sendmail/milter/lf_smpl.c - sample filter program
- /usr/include/libmilter/mfapi.h - header file needed for lf_smpl.c
- /usr/include/libmilter/mfdef.h - header file needed for lf_smpl.c
- /usr/lib/libmilter.a - milter API library

The sample filter program is documented in the end of /usr/lpp/tcpip/samples/sendmail/libmilter/README.

Note: The milter API library libmilter.a is built in the IBM-1047 environment. The sample filter must be compiled and linked in IBM-1047 to assure correct data exchange between the sample filter and the milter API. The sample filter also must be executed in the environment with codepage IBM-1047.

The following example shows how to use the cc command to compile and link the sample filter.

cc -I. -o filter lf_smpl.c libmilter.a

Specifying filters in the sendmail configuration file

To use filters in sendmail, filters must be declared in the sendmail configuration file (sendmail.cf). For more information about this sendmail configuration file, see the z/OS Communications Server: IP Configuration Guide.

Running the sample mail filter program

Error messages for the sample filter are written to a log file. The log file is defined in lf_smpl.c as follows:

openlog(NULL, LOG_PID, LOG_LOCAL7)

To get error messages, first create a log file. The log reference in the source code can be modified to reference the log file you created. For more information about error messages, see z/OS Communications Server: IP Diagnosis Guide.

The lf_smpl.c sample program accepts the -p argument as follows:

-p socket_reference

Specifying the socket information of the filter, the parameter should be formatted according to the socket specification in the sendmail configuration file.
For example, use the command `filter -p inet:3333@localhost` with the following configuration:

```
Xfilter, S=inet:3333@localhost
O InputMailFilters=filter
```

### Library control functions

The following are sample mail filter program functions.

#### smfi_register

```c
#include <libmilter/mfapi.h>
int smfi_register(
    smfiDesc_str descr
);
```

#### smfi_register description

Register a set of filter callbacks. When called, smfi_register creates a filter using the information given in the smfiDesc_str argument.

#### Notes:

1. smfi_register must be called before smfi_main.
2. Multiple calls to smfi_register within a single process are not allowed.

#### smfi_register parameters

- **descr** A filter descriptor of type smfiDesc_str describing the filter’s functions. The structure has the following members:
  ```c
  struct smfiDesc
  {
    char *xxffi_name; /* filter name */
    int  xxffi_version; /* version code -- do not change */
    unsigned long xxffi_flags; /* flags */
    /* connection info filter */
    sfsistat (*xxffi_connect)(SMFICTX *, char *, _SOCK_ADDR *);
    /* SMTP HELO command filter */
    sfsistat (*xxffi_helo)(SMFICTX *, char *);
    /* envelope sender filter */
    sfsistat (*xxffi_envfrom)(SMFICTX *, char **);
    /* envelope recipient filter */
    sfsistat (*xxffi_envrcpt)(SMFICTX *, char **);
    /* header filter */
    sfsistat (*xxffi_header)(SMFICTX *, char *, char *);
    /* end of header */
    sfsistat (*xxffi_eoh)(SMFICTX *);
    /* body block */
    sfsistat (*xxffi_body)(SMFICTX *, unsigned char *, size_t);
    /* end of message */
    sfsistat (*xxffi_eom)(SMFICTX *);
    /* message aborted */
    sfsistat (*xxffi_abort)(SMFICTX *);
    /* connection cleanup */
    sfsistat (*xxffi_close)(SMFICTX *);
  };
  ```

A NULL value for any callback function indicates that the filter does not process the given type of information, simply returning SMFIS_CONTINUE.
**smfi_register result**

smfi_register can return MI_FAILURE for any of the following reasons:

- Memory allocation failed.
- Incompatible version or illegal flags value.

The xxfi_flags field should contain the bitwise OR of zero or more of the following values, describing the actions the filter might take:

**SMFIF_ADDHdra**
This filter can add headers.

**SMFIF_CHGHdR**
This filter can change and delete headers.

**SMFIF_CHGBODY**
This filter can replace the body during filtering. This can have significant performance impact if other filters do body filtering after this filter.

**SMFIF_ADDRCPT**
This filter can add recipients to the message.

**SMFIF_DELRCPT**
This filter can remove recipients from the message.

**smfi_setconn**

```c
#include <libmilter/mfapi.h>
int smfi_setconn(
    char *oconn;
);
```

**smfi_setconn description**

The smfi_setconn API sets the socket through which the filter communicates with sendmail. The smfi_setconn API must be called once before smfi_main.

**smfi_setconn parameters**

- **oconn**  
The address of the desired communication socket. The address should be a NULL-terminated string in proto:address format as follows:
  - [unix|local]/path/to/file - A named pipe
  - inet:port@[hostname|ip-address] - An IPv4 socket
  - inet6:port@[hostname|ip-address] - An IPv6 socket

**smfi_setconn result**

smfi_setconn does not fail on an address that is not valid. A failure is detected only in smfi_main.

**Notes:**

1. If possible, filters should not run as root when communicating over UNIX/local domain sockets.
2. UNIX/local sockets should have their permissions set to 0600 (read/write permission only for the socket's owner).

**smfi_settimeout**

```c
#include <libmilter/mfapi.h>
int smfi_settimeout(
    int otimeout
);
```
**smfi_settimeout description**
Sets the number of seconds libmilter will wait for an MTA connection before timing out a socket. If smfi_settimeout is not called, a default timeout of 1800 seconds is used.

Note: The smfi_settimeout API should be called only before smfi_main.

**smfi_settimeout parameters**
- `otimeout`
  - The number of seconds to wait before timing out (value must be greater than zero). Zero means no wait, rather than wait forever.

**smfi_settimeout result**
smfi_settimeout always returns MI_SUCCESS.

**smfi_main**

```c
#include <libmilter/mfapi.h>
int smfi_main(
);
```

**smfi_main description**
The smfi_main API is called after a filter's initialization is complete. smfi_main passes control to the milter event loop.

**smfi_main parameters**
smfi_main has no parameters.

**smfi_main result**
smfi_main returns MI_FAILURE if it fails to establish a connection. This can occur for a number of reasons (for instance, if an address that is not valid is passed to smfi_setconn). The reason for the failure is logged. Otherwise, smfi_main returns MI_SUCCESS.

---

**Data access functions**
The following are mail filter data access functions.

**smfi_getsymval**

```c
#include <libmilter/mfapi.h>
char* smfi_getsymval(
    SMFICTX *ctx,  
    char *symname
);
```

**smfi_getsymval description**
Get the value of a sendmail macro. smfi_getsymval can be called from within any of the xxfi_* callbacks. Which macros are defined depends on when smfi_getsymval is called.

**smfi_getsymval parameters**
- `ctx` The opaque context structure.
- `symname` The name of a sendmail macro, optionally enclosed in braces ({} and {}). See below for default macros.
**smfi_getsymval result**

smfi_getsymval returns the value of the given macro as a null-terminated string, or returns NULL if the macro is not defined.

**Notes:**

1. By default, the following macros are valid in the given contexts:

<table>
<thead>
<tr>
<th>Sent With</th>
<th>Macros</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxfi_connect</td>
<td>daemon_name, if_name, if_addr, j, _</td>
</tr>
<tr>
<td>xxfi_helo</td>
<td>tls_version, cipher, cipher_bits, cert_subject, cert_issuer</td>
</tr>
<tr>
<td>xxfi_envfrom</td>
<td>i, auth_type, auth_authen, auth_ssf, auth_author, mail_mailer, mail_host, mail_addr</td>
</tr>
<tr>
<td>xxfi_envrcpt</td>
<td>rcpt_mailer, rcpt_host, rcpt_addr</td>
</tr>
</tbody>
</table>

2. All macros remain in effect from the point they are received until the end of the connection for the first two sets, the end of the message for the third set (xxfi_envfrom), and for each recipient for the final set (xxfi_envrcpt).

3. The macro list can be changed using the confMILTER_MACROS_* options in sendmail.mc. The scopes of such macros are determined by when they are set by sendmail. For descriptions of macro values, see the *Sendmail Installation and Operation Guide* provided with the sendmail distribution.

**smfi_getpriv**

```c
#include <libmilter/mfapi.h>
void* smfi_getpriv(
        SMFICTX *ctx
    );
```

**smfi_getpriv description**

Get the connection-specific data pointer for this connection. smfi_getpriv can be called in any of the xxfi_* callbacks.

**smfi_getpriv parameters**

- **ctx** Opaque context structure.

**smfi_getpriv result**

smfi_getpriv returns the private data pointer stored by a prior call to smfi_setpriv, or returns NULL if none has been set.

```c
#include <libmilter/mfapi.h>
int smfi_setpriv(
        SMFICTX *ctx,     
        void *privatedata
    );
```

**smfi_setpriv description**

Set the private data pointer for this connection.

**smfi_setpriv parameters**

- **ctx** Opaque context structure.
- **privatedata** Pointer to private data. This value is returned by subsequent calls to smfi_getpriv using ctx.
**smfi_setpriv result**

smfi_setpriv returns MI_FAILURE if the context of ctx is not valid. Otherwise, it returns MI_SUCCESS.

**Notes:**
1. There is only one private data pointer per connection; multiple calls to smfi_setpriv with different values cause previous values to be lost.
2. Before a filter terminates it should release the private data and set the pointer to NULL.

**smfi_setreply**

```c
#include <libmilter/mfapi.h>
int smfi_setreply(
    SMFICTX *ctx,
    char *rcode,
    char *xcode,
    char *message
);
```

**smfi_setreply description**

Directly set the SMTP error reply code for this connection. This code is used on subsequent error replies resulting from actions taken by this filter. smfi_setreply can be called from any of the xxfi_ callbacks.

**smfi_setreply parameters**

- **ctx**: Opaque context structure.
- **rcode**: The three-digit SMTP reply code, as a null-terminated string. rcode cannot be NULL and must be a valid reply code.
- **xcode**: The extended reply code. If xcode is NULL, no extended code is used.
- **message**: The text part of the SMTP reply. If message is NULL, an empty message is used.

**smfi_setreply result**

smfi_setreply fails and returns MI_FAILURE if:
- The rcode or xcode argument is not valid.
- A memory-allocation failure occurs.

Otherwise, it returns MI_SUCCESS.

**Notes:**
1. Values passed to smfi_setreply are not checked for standards compliance.
2. For details about reply codes and their meanings, see the documentation provided with the sendmail distribution.

---

**Message modification functions**

The following are mail filter message modification functions.
smfi_addheader

```c
#include <libmilter/mfapi.h>
int smfi_addheader(
    SMFICTX *ctx,
    char *headerf,
    char *headerv
);
```

**smfi_addheader description**
The smfi_addheader API adds a header to the current message. It is called only from xxfi_eom.

**smfi_addheader parameters**
- `ctx` Opaque context structure.
- `headerf` The header name, a non-NULL, null-terminated string.
- `headerv` The header value to be added, a non-NULL, null-terminated string. This can be the empty string.

**smfi_addheader result**
smfi_addheader returns MI_FAILURE in the following cases:
- `headerf` or `headerv` is NULL.
- Adding headers in the current connection state is not valid.
- Memory allocation fails.
- A network error occurs.
- SMFIF_ADDHDRS was not set when smfi_register was called.
Otherwise, it returns MI_SUCCESS.

**Notes:**
1. smfi_addheader does not change a message’s existing headers. To change a header’s current value, use smfi_chgheader.
2. A filter that calls smfi_addheader must have set the SMFIF_ADDHDRS flag in the smfiDesc_str passed to smfi_register.
3. Filter order is important for smfi_chgheader; later filters see the header changes made by previous filters.
4. Neither the name nor the value of the header is checked for standards compliance. However, each line of the header must be less than 2048 characters in length and should be less than 998 characters in length. If longer headers are required, make them multiline. It is the filter writer’s responsibility to ensure that no standards are violated.

smfi_chgheader

```c
#include <libmilter/mfapi.h>
int smfi_chgheader(
    SMFICTX *ctx,
    char *headerf,
    mi_int32 hdridx,
    char *headerv
);
```

**smfi_chgheader description**
The smfi_chgheader API changes a header’s value for the current message. It is called only from xxfi_eom.
**smfi_chgheader parameters**

ctx  Opaque context structure.

headerf  The header name, a non-NULL, null-terminated string.

hdridx  Header index value (1-based). A hdridx value of 1 modifies the first occurrence of a header named headerf. If hdridx is greater than the number of times headerf appears, a new copy of headerf is added.

headerv  The new value of the given header. A headerv value of NULL implies that the header should be deleted.

**smfi_chgheader result**

smfi_chgheader returns MI_FAILURE in the following cases:

- headerf is NULL.
- Modifying headers in the current connection state is not valid.
- Memory allocation fails.
- A network error occurs.
- SMFIF_CHGHDRS was not set when smfi_register was called.

Otherwise, it returns MI_SUCCESS.

**Notes:**

1. While smfi_chgheader can be used to add new headers, it is more efficient and safer to use smfi_addheader to add new headers.
2. A filter that calls smfi_chgheader must have set the SMFIF_CHGHDRS flag in the smfiDesc_str passed to smfi_register.
3. Filter order is important for smfi_chgheader; later filters see the header changes made by previous filters.
4. Neither the name nor the value of the header is checked for standards compliance. However, each line of the header must be less than 2048 characters in length and should be less than 998 characters in length. If longer headers are needed, make them multiline. It is the filter writer’s responsibility to ensure that no standards are violated.

**smfi_addrcpt**

```
#include <libmilter/mfapi.h>

int smfi_addrcpt(SMFICTX *ctx, char *rcpt);
```

**smfi_addrcpt description**

The smfi_addrcpt API adds a recipient to the message envelope. It is called only from xxfi_eom.

**smfi_addrcpt parameters**

ctx  Opaque context structure.

rcpt  The new recipient’s address.
**smfi_addrcpt result**

smfi_addrcpt fails and returns MI_FAILURE if the following conditions occur:

- Adding headers in the current connection state is not valid.
- A network error occurs.
- SMFIF_ADDRCPT was not set when smfi_register was called.
- rcpt is NULL.

Otherwise, smfi_addrcpt returns MI_SUCCESS.

*Note:* A filter that calls smfi_addrcpt must have set the SMFIF_ADDRCPT flag in the smfiDesc_str passed to smfi_register.

**smfi_delrcpt**

```
#include <libmilter/mfapi.h>
int smfi_delrcpt(
    SMFICTX *ctx;
    char *rcpt;
);
```

**smfi_delrcpt description**

The smfi_delrcpt API removes the named recipient from the current message’s envelope. It is called only from xxfi_eom.

**smfi_delrcpt parameters**

- **ctx** Opaque context structure.
- **rcpt** The recipient address to be removed, a non-NULL, null-terminated string.

**smfi_delrcpt result**

smfi_delrcpt fails and returns MI_FAILURE if any of the following conditions occur:

- rcpt is NULL.
- Adding headers in the current connection state is not valid.
- A network error occurs.
- SMFIF_DELRCPT was not set when smfi_register was called.

Otherwise, it returns MI_SUCCESS.

*Note:* The addresses to be removed must match exactly (for example, an address and its expanded form must match).

**smfi_replacebody**

```
#include <libmilter/mfapi.h>
int smfi_replacebody(
    SMFICTX *ctx,
    unsigned char *bodyp,
    int bodylen
);
```

**smfi_replacebody description**

The smfi_replacebody API replaces the body of the current message. It is called only from xxfi_eom and can be called more than once. If it is called multiple times, subsequent calls result in data being appended to the new body.

**smfi_replacebody parameters**

- **ctx** Opaque context structure.
**bodyp** A pointer to the start of the new body data, which does not have to be null-terminated. If bodyp is NULL, it is treated as having length equal to 0. Body data should be in CR/LF form.

**bodylen** The number of data bytes bodyp points to.

**smfi_replacebody result**

smfi_replacebody fails and returns MI_FAILURE if any of the following conditions occur:

- bodyp is equal to NULL and bodylen is greater than 0.
- Changing the body in the current connection state is not valid.
- A network error occurs.
- SMFIF_CHGBODY was not set when smfi_register was called.

Otherwise, it will return MI_SUCCESS.

**Notes:**

1. Since the message body can be very large, setting SMFIF_CHGBODY might significantly affect filter performance.
2. If a filter sets SMFIF_CHGBODY but does not call smfi_replacebody, the original body remains unchanged.
3. Filter order is important for smfi_replacebody; later filters see the new body contents created by previous filters.

---

### Mail filter callbacks

Each of these callbacks should return one of the values that is defined in Table 6. Any value other than those listed constitutes an error and causes sendmail to terminate its connection to the offending filter.

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMFIS_CONTINUE</td>
<td>Continue processing the current connection, message, or recipient.</td>
</tr>
<tr>
<td>SMFIS_REJECT</td>
<td>For a connection-oriented routine, reject this connection; call xxfi_close. For a message-oriented routine (except xxfi_eom or xxfi_abort), reject this message. For a recipient-oriented routine, reject the current recipient (but continue processing the current message).</td>
</tr>
<tr>
<td>SMFIS_DISCARD</td>
<td>For a message or recipient-oriented routine, accept this message, but silently discard it. SMFIS_DISCARD should not be returned by a connection-oriented routine.</td>
</tr>
<tr>
<td>SMFIS_ACCEPT</td>
<td>For a connection-oriented routine, accept this connection without further filter processing; call xxfi_close. For a message or recipient-oriented routine, accept this message without further filtering.</td>
</tr>
<tr>
<td>SMFIS_TEMPFAIL</td>
<td>Return a temporary failure; the corresponding SMTP command will return an appropriate 4xx status code. For a message-oriented routine (except xxfi_envfrom), fail for this message. For a connection-oriented routine, fail for this connection; call xxfi_close. For a recipient-oriented routine, fail only for the current recipient; continue message processing.</td>
</tr>
</tbody>
</table>
xxfi_connect - Connection information

```c
#include <libmilter/mfapi.h>
sfsistat (*xxfi_connect)(
    SMFICTX *ctx,
    char *hostname,
    _SOCK_ADDR *hostaddr);
```

**xxfi_connect description**
Called once, at the start of each SMTP connection. Default behavior is to do nothing and return SMFIS_CONTINUE.

**xxfi_connect parameters**

- **ctx**  
  The opaque context structure.

- **hostname**  
  The host name of the message sender, as determined by a reverse lookup on the host address. If the reverse lookup fails, hostname will contain the message sender’s IP address enclosed in square brackets (for example, [a.b.c.d]).

- **hostaddr**  
  The host address, as determined by a getpeername() call on the SMTP socket. NULL if the type is not supported in the current version.

**xxfi_connect result**
If a previous filter rejects the connection in its xxfi_connect() routine, this filter’s xxfi_connect() is not called.

xxfi_helo - SMTP HELO/EHLO command

```c
#include <libmilter/mfapi.h>
sfsistat (*xxfi_helo)(
    SMFICTX *ctx,
    char * helohost
);
```

**xxfi_helo description**
Handle the HELO/EHLO command. xxfi_helo is called whenever the client sends a HELO/EHLO command. It can therefore be called between zero and three times. Default is to do nothing and return SMFIS_CONTINUE.

**xxfi_helo parameters**

- **ctx**  
  Opaque context structure.

- **helohost**  
  Value passed to HELO/EHLO command, which should be the domain name of the sending host (but is, in practice, anything the sending host wants to send).

xxfi_envfrom - Envelope sender

```c
#include <libmilter/mfapi.h>
sfsistat (*xxfi_envfrom)(
    SMFICTX *ctx,
    char ** argv
);
```
**xxfi_envfrom description**
Handle the envelope FROM command. xxfi_envfrom is called once at the beginning of each message, before xxfi_envrcpt. The default behavior is to do nothing and return SMFIS_CONTINUE.

**xxfi_envfrom parameters**
- **ctx**  
  Opaque context structure.
- **argv**  
  Null-terminated SMTP command arguments; argv[0] is guaranteed to be the sender address. Later arguments are the ESMTP arguments.

**xxfi_envfrom result**
Can return the following values:

- **SMFIS_TEMPFAIL**  
  Temporarily fail for this recipient; further recipients can still be sent. xxfi_abort is not called.
- **SMFIS_REJECT**  
  Reject this particular recipient; further recipients can still be sent. xxfi_abort is not called.
- **SMFIS_DISCARD**  
  Accept and discard the message. xxfi_abort is called.
- **SMFIS_ACCEPT**  
  Accept the message. xxfi_abort is not called.

**xxfi_envrcpt - Envelope recipient**

```c
#include <libmilter/mfapi.h>
sfsistat (*xxfi_envrcpt)(
   SMFICTX * ctx,
   char ** argv
);
```

**xxfi_envrcpt description**
Handle the envelope RCPT command. xxfi_envrcpt is called once per recipient (one or more times per message), immediately after xxfi_envfrom. The default behavior is to do nothing and return SMFIS_CONTINUE.

**xxfi_envrcpt parameters**
- **ctx**  
  Opaque context structure.
- **argv**  
  Null-terminated SMTP command arguments; argv[0] is guaranteed to be the recipient address. Later arguments are the ESMTP arguments.

**xxfi_envrcpt result**
Can return the following values:

- **SMFIS_TEMPFAIL**  
  Temporarily fail for this particular recipient; further recipients can still be sent. xxfi_abort is not called.
- **SMFIS_REJECT**  
  Reject this particular recipient; further recipients can still be sent. xxfi_abort is not called.
- **SMFIS_DISCARD**  
  Accept and discard the message. xxfi_abort is called.
- **SMFIS_ACCEPT**  
  Accept the message. xxfi_abort is not called.
SMFIS_ACCEPT
Accept recipient. xxfi_abort is not called.

**xxfi_header - Header**

```c
#include <libmilter/mfapi.h>
sfsistat (*xxfi_header)(
    SMFICTX * ctx,
    char * headerf,
    char * headerv
);
```

**xxfi_header description**
Handle a message header. xxfi_header is called zero or more times between xxfi_envrcpt and xxfi_eoh, once per message header. The default behavior is to do nothing and then return SMFIS_CONTINUE.

**xxfi_header parameters**
- **ctx**
  Opaque context structure.
- **headerf**
  Header field name.
- **headerv**
  Header field value. The content of the header can include folded white space (multiple lines with following white space). The trailing line terminator (CR/LF) is removed.

**Notes:**
1. Later filters see any header changes or additions made by previous filters.
2. For more detail about header format, see sendmail documentation.

**xxfi_eoh - End of header**

```c
#include <libmilter/mfapi.h>
sfsistat (*xxfi_eoh)(
    SMFICTX * ctx
);
```

**xxfi_eoh description**
Handle the end of a message header. xxfi_eoh is called once after all headers have been sent and processed. The default behavior is to do nothing and then return SMFIS_CONTINUE.

**xxfi_eoh parameters**
- **ctx**
  Opaque context structure.

**xxfi_body - body block**

```c
#include <libmilter/mfapi.h>
sfsistat (*xxfi_body)(
    SMFICTX * ctx,
    unsigned char * bodyp,
    size_t len
);
```

**xxfi_body description**
Handle a piece of a message’s body. xxfi_body is called zero or more times between xxfi_eoh and xxfi_eom. The default behavior is to do nothing and then return SMFIS_CONTINUE.
**xxfi_body parameters**

- **ctx**  
  Opaque context structure.

- **bodyp**  
  Pointer to the start of this block of body data. bodyp is not valid outside this call to xxfi_body.

- **len**  
  The amount of data pointed to by bodyp.

**Notes:**

1. Since message bodies can be very large, defining xxfi_body can significantly impact filter performance.
2. End-of-lines are represented as received from SMTP (normally CR/LF).
3. Later filters see body changes made by previous filters.
4. Message bodies might be sent in multiple chunks, with one call to xxfi_body per chunk.

**xxfi_eom - End of message**

```c
#include <libmilter/mfapi.h>
sfsistat (*xxfi_eom)(
    SMFICTX * ctx
);
```

**xxfi_eom description**

End of a message. xxfi_eom is called once for a given message, after all calls to xxfi_body. The default behavior is to do nothing and then return SMFIS_CONTINUE.

**xxfi_eom parameters**

- **ctx**  
  Opaque context structure.

**Note:** A filter is required to make all its modifications to the message headers, body, and envelope in xxfi_eom. Modifications are made using the smfi_* routines.

**xxfi_abort - Message aborted**

```c
#include <libmilter/mfapi.h>
sfsistat (*xxfi_abort)(
    SMFICTX * ctx
);
```

**xxfi_abort description**

Handle the current message being aborted. xxfi_abort can be called at any time during message processing (between some message-oriented routine and xxfi_eom).

**xxfi_abort parameters**

- **ctx**  
  Opaque context structure.

**Notes:**

1. xxfi_abort must reclaim any resources allocated on a per-message basis, and must be tolerant of being called between any two message-oriented callbacks.
2. Calls to xxfi_abort and xxfi_eom are mutually exclusive.
3. xxfi_abort is not responsible for reclaiming connection-specific data, since xxfi_close is always called when a connection is closed.
4. Since the current message is already being aborted, the return value is currently ignored.
5. xxfi_abort is called only if the message is aborted outside the filter's control and the filter has not completed its message-oriented processing. For example, if a filter has already returned SMFIS_ACCEPT, SMFIS_REJECT or SMFIS_DISCARD from a message-oriented routine, xxfi_abort is not called, even if the message is later aborted outside its control.

**xxfi_close - Connection cleanup**

```c
#include <libmilter/mfapi.h>
sfsistat (xxfi_close)(
    SMFICTX *ctx
);
```

**xxfi_close description**
Provides notification that the current connection is being closed. xxfi_close is always called once at the end of each connection. The default behavior is to do nothing and then return SMFIS_CONTINUE.

**xxfi_close parameters**

- **ctx** Opaque context structure.

**Notes:**
1. xxfi_close is called on close even if the previous mail transaction was aborted.
2. xxfi_close is responsible for freeing any resources allocated on a per-connection basis.
3. Since the connection is already closing, the return value is currently ignored.
Chapter 12. Policy API (PAPI)

The Policy Agent includes an application programming interface (API) known as the Policy API or PAPI.

The PAPI interface allows user applications to connect to the Policy Agent through a UNIX socket connection and access policy related data. Data returned from the Policy Agent are queued in the user’s address space. A set of PAPI functions is defined to access specific portions of the returned data. The interface also provides for terminating the connection and cleaning up resources obtained while the API was in use.

Currently, the only function provided by PAPI is to retrieve policy performance data.

API outline for retrieving data from Policy Agent

Using the PAPI interface, an application uses the papi_connect() call to define an API connection and to register with the Policy Agent. The papi_get_perf_data() call is used to retrieve the policy performance data from the Policy Agent. An application can then use a set of helper functions to access performance information returned on the papi_get_perf_data() call. The set of helper functions is:

- papi_get_policy_instance() - Returns the policy instance number for the set of policies in the performance data returned.
- papi_get_rules_count() - Returns the number of policy rules in the performance data returned.
- papi_get_actions_count() - Returns the number of policy actions in the performance data returned.
- papi_get_rule_perf_info() - Returns a policy rule entry based on the rule number that contains the rule performance information.
- papi_get_rule_perf_by_id() - Returns a policy rule entry based on the rule ID that contains the rule performance information.
- papi_get_action_perf_info() - Returns a policy action entry based on the action number that contains the action performance information.
- papi_get_action_perf_by_id() - Returns a policy action entry based on the action ID that contains the action performance information.
- papi_strerror() - Returns a string describing a PAPI return code value, similar to the C strerror() function.

When the application is done using the data returned on the papi_get_perf_data() call, it can call papi_free_perf_data() to free the data. When the application no longer wants to retrieve policy performance data from the Policy Agent, it can call papi_disconnect() to end the connection.

Compiling and linking PAPI applications

To use the PAPI interface, an application must perform the following steps:

1. Include the <papiuser.h> header file, which is available in the /usr/include directory.
2. Compile the application with the DLL compiler option. See the z/OS XL C/C++ User’s Guide for more information about how to specify compiler options.

3. Include the PAPI definition side deck (papiuser.x), which is available in the /usr/lib directory, when prelinking or binding the application.

4. If the Binder is used instead of the C prelinker, specify the Binder DYNAM=DLL option. See the z/OS MVS Program Management: User's Guide and Reference for information about specifying Binder options.

Running PAPI applications

At execution time, the PAPI application must have access to the PAPI DLL (papi.dll), which is available in the /usr/lib directory. Ensure that the LIBPATH environment variable includes this directory when running the application. The PAPI application must either run with superuser authority to use PAPI, or must have security product authority in the SERVAUTH class. These security product profiles can be defined by TCP/IP stack (TcpImage) and policy type (only ptype = QOS is applicable). Wildcarding of profile names is allowed. The security product profiles take the following form:

```
EZB.PAGENT.<sysname>.<TcpImage>.<ptype>
```

where:
- `sysname` - System name defined in sysplex
- `TcpImage` - TCP name for the requested policy information
- `ptype` - Policy Type that is being requested (QOS)

Note: Wildcarding is allowed on segments of the profile name.

See the EZARACF sample in SEZAINST for sample commands needed to create the profile name and permit users access to it.

PAPI return codes

The following return codes may be returned from PAPI functions.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAPI_OK</td>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>PAPI_HELPER_RETURN_NULL</td>
<td>NULL</td>
<td>NULL return from PAPI helper function.</td>
</tr>
<tr>
<td>PAPI_HELPER_RETURN_ZERO</td>
<td>0</td>
<td>Zero return from PAPI helper function.</td>
</tr>
<tr>
<td>PAPI_NOK</td>
<td>1</td>
<td>Generic error.</td>
</tr>
<tr>
<td>PAPI_INVALID_PARAMETER_VALUE</td>
<td>2</td>
<td>Parameter has an invalid value.</td>
</tr>
<tr>
<td>PAPI_CLIENT_ALREADY_INITIALIZED</td>
<td>4</td>
<td>User already issued papi_connect().</td>
</tr>
<tr>
<td>PAPI_CLIENT_ALREADY_REGISTERED</td>
<td>5</td>
<td>User already issued papi_connect().</td>
</tr>
<tr>
<td>PAPI_FUNC_NOT_READY</td>
<td>8</td>
<td>PAPI function not ready - try again later.</td>
</tr>
<tr>
<td>PAPI_INVALID_ACCEPTABLE_CACHED_TIME</td>
<td>10</td>
<td>The acceptableCachedTime input parameter is ignored because it is less than the MinimumSamplingInterval configured to the Policy Agent.</td>
</tr>
<tr>
<td>PAPI_PERF_COLL_TYPE_MISMATCH</td>
<td>11</td>
<td>Some or all of the requested type of performance data (rules, actions, or both) is not being collected by the Policy Agent.</td>
</tr>
</tbody>
</table>
Table 7. PAPI function return codes (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAPI_MALLOC_FAILED</td>
<td>16</td>
<td>PAPI could not allocate memory in user’s address space.</td>
</tr>
<tr>
<td>PAPI_MALLOC_IN_PAGENT_FAILED</td>
<td>17</td>
<td>Policy Agent could not allocate memory.</td>
</tr>
<tr>
<td>PAPI_PAGENT_INTERNAL_ERROR</td>
<td>18</td>
<td>Internal error encountered in Policy Agent.</td>
</tr>
<tr>
<td>PAPI_INTERNAL_ERROR</td>
<td>19</td>
<td>Internal error encountered in PAPI.</td>
</tr>
<tr>
<td>PAPI_CLIENT_NOT_REGISTERED</td>
<td>20</td>
<td>User did not issue papi_connect().</td>
</tr>
<tr>
<td>PAPI_NOT_VALID_AUTHORIZATION</td>
<td>21</td>
<td>User not authorized to issue PAPI function.</td>
</tr>
<tr>
<td>PAPI_VERSION_INCORRECT</td>
<td>22</td>
<td>Incompatibility between the version of the PAPI DLL (papi.dll) and the version of Policy Agent.</td>
</tr>
<tr>
<td>PAPI_CONNECT_FAILED</td>
<td>30</td>
<td>Connect to Policy Agent failed.</td>
</tr>
<tr>
<td>PAPI_READ_FAILED</td>
<td>31</td>
<td>Read from Policy Agent failed.</td>
</tr>
<tr>
<td>PAPI_SOCKET_NOT_READABLE</td>
<td>32</td>
<td>Socket for Policy Agent connection is not readable.</td>
</tr>
<tr>
<td>PAPI_WRITE_FAILED</td>
<td>33</td>
<td>Write to Policy Agent failed.</td>
</tr>
<tr>
<td>PAPI_READ_TIMEOUT</td>
<td>34</td>
<td>Read from Policy Agent timed out.</td>
</tr>
<tr>
<td>PAPI_SOCKET_FAILED</td>
<td>35</td>
<td>Could not open socket for Policy Agent connection.</td>
</tr>
<tr>
<td>PAPI_FCNTL_FAILED</td>
<td>36</td>
<td>fcntl() on connection socket failed.</td>
</tr>
<tr>
<td>PAPI_NULL_INPUT</td>
<td>50</td>
<td>A required parameter is not specified.</td>
</tr>
<tr>
<td>PAPI_TCPIMAGE_NOT_VALID</td>
<td>51</td>
<td>The specified kernel name is not known to the Policy Agent.</td>
</tr>
<tr>
<td>PAPI_TCPIMAGE_INVALID_LENGTH</td>
<td>52</td>
<td>The specified kernel name is too long.</td>
</tr>
<tr>
<td>PAPI_FILTERNAME_INVALID_LENGTH</td>
<td>53</td>
<td>The specified filter name is too long.</td>
</tr>
<tr>
<td>PAPI_KERNEL_NOT_AVAILABLE</td>
<td>54</td>
<td>The TCP/IP stack is not available to process a request, or an error occurred while obtaining data from the stack.</td>
</tr>
</tbody>
</table>

PAPI client library services

The Policy Agent API provides the following client library calls to connect, disconnect, get, and free storage for policy performance data.

- papi_connect()
- papi_debug()
- papi_disconnect()
- papi_free_perf_data()
- papi_get_perf_data()

The Policy Agent API provides the following helper functions to access the policy performance data.

- papi_get_action_perf_by_id()
- papi_get_action_perf_info()
- papi_get_actions_count()
- papi_get_policy_instance()
To use these calls, the application must include the file papiuser.h.

---

**PAPI: Connecting and retrieving data**

Use the following PAPI functions for connecting and retrieving data.

**papi_connect - Connect to Policy Agent**

```c
#include <papiuser.h>

extern int papi_connect(void **papiHandle, void *regReq);
```

**papi_connect description**

This function is used to open a connection and register with the Policy Agent. The parameters it takes are a pointer to a void pointer, which is used to return the handle, and a void pointer to pass in the registration information. The registration information is currently not used. All information about this connection is stored internally using the handle pointer as a reference. Most other PAPI functions require that this handle be passed in as input. A call should subsequently be made to papi_disconnect() to release the resources used by papi_connect().

**papi_connect parameters**

- **papiHandle**
  - This is an output parameter that points to the handle to identify this papi_connect().

- **regReq**
  - This is an input parameter that points to the registration information. This pointer should be NULL.

**papi_connect result**

If the connection is successful, the call returns a return code of PAPI_OK, and papiHandle is set.

If the connect fails, the call returns a non-PAPI_OK return code value.

**papi_connect example**

```c
void *mainHandle;
int nRc;

nRc = papi_connect(&mainHandle, NULL);
if (nRc != PAPI_OK)
{
    printf("Error in papi_connect : %d\n", nRc);
}
else
{
    /* everything is ok so far ... */
}
papi_debug - Set debug capability

```c
#include <papiuser.h>

extern int papi_debug(papiDebug_t *debugValue);
```

**papi_debug description**
This function allows debug information to be displayed for PAPI functions. This function can be called by the application to turn debug on or off anytime during the PAPI processing.

**papi_debug parameters**
*debugValue*
This is a pointer to an input parameter that is used to turn debug on or off.

If `papi_debug()` is not issued, then no debug information is displayed.

If debug is being used, then the application can pass in a user exit in the `papiDebug_t papiLogFunc` field.

If `papiLogFunc` is NULL, then all messages are logged using `printf()`.

The following defines are located in `papiuser.h`.

```c
typedef struct {
    unsigned int papiDebugOpt; /* Debug On/Off */
    void *papiUserValue; /* User Define value */
    papiLogUserExit_t papiLogFunc; /* Logging Function */
} papiDebug_t; /* Input papi_debug()*/
```

Set `papiDebugOpt`:
- `#define PAPI_DEBUG_OFF 0 /* Debug Off */`
- `#define PAPI_DEBUG_ON 1 /* Debug On */`

**papi_debug result**
If the debug is successful, the call returns a return code of PAPI_OK.

If the debug fails, the call returns a non-PAPI_OK return code value.

papi_disconnect - Disconnect from the Policy Agent

```c
#include <papiuser.h>

extern int papi_disconnect(void *papiHandle);
```

**papi_disconnect description**
This function is used to terminate a connection with the Policy Agent. The only parameter it takes is a pointer to the handle that was set in the `papi_connect()` call.

**papi_disconnect parameters**
*papiHandle*
This is an input parameter of type void. It is the responsibility of the caller to disconnect from the Policy Agent based on the `papiHandle` returned on the `papi_connect()` API.

**papi_disconnect result**
If the disconnect is successful, the call returns a return code of PAPI_OK.

If the disconnect fails, the call returns a non-PAPI_OK return code value.
### papi_disconnect example

```c
void *mainHandle;
int nRc;

nRc = papi_disconnect(mainHandle);
if (nRc != PAPI_OK)
{
    printf("Error in papi_disconnect : %d\n", nRc );
}
else
{
    /* everything is ok so far ... */
}
```

### papi_free_perf_data - Free retrieved QoS performance data

```c
#include <papiuser.h>
extern int papi_free_perf_data(void *perfDataHandle);
```

#### papi_free_perf_data description

This function is used to free the memory associated with the policy performance data returned by the papi_get_perf_data() API. This API should be invoked with the handle to free the memory allocated to hold the performance information.

#### papi_free_perf_data parameters

- `*perfDataHandle`
  - This is an input parameter of type `void` that points to the memory obtained from the papi_get_perf_data() API.

#### papi_free_perf_data result

If the free is successful, the call returns a return code of PAPI_OK.

If the free fails, the call returns a non-PAPI_OK return code value.

### papi_free_perf_data example

```c
int nRc;
void *perfDataHandle;

/* Initialization and obtaining data to be done here */

nRc = papi_free_perf_data(perfDataHandle);
if (nRc != PAPI_OK)
{
    printf("Error in papi_free_perf_data : %d\n", nRc );
}
else
{
    /* everything is ok so far ... */
}
```

### papi_get_perf_data - Retrieve QoS performance data

```c
#include <papiuser.h>

extern int papi_get_perf_data(void *papiHandle,
   int typeFlag,
   void *filter,
   int *acceptableCachedTime,
   char *kernelName,
   void **perfDataHandle);
```

---

388 z/OS V1R11.0 Comm Svrs: IP Programmer’s Guide and Reference
papi_get_perf_data description
This function is used to retrieve the policy performance data from the Policy Agent.

Policy performance metrics collected by the kernel are affected by the FLUSH/NOFLUSH parameter on the Policy Agent TcpImage configuration statement. FLUSH causes the metrics values to be reset to 0 at the following times:
- When a new TcpImage statement is processed for the first time, including Policy Agent starting. This should not be a concern in most cases.
- When a MODIFY REFRESH command is entered.

Metrics are never reset when NOFLUSH is specified. See the z/OS Communications Server: IP Configuration Reference and the policy-based networking information in the z/OS Communications Server: IP Configuration Guide for more information.

Note: Changes to policy definitions might not cause immediate changes in performance metrics, due to averaging and smoothing over several sampling intervals. A period of time must elapse after a policy change in order to reach a new steady state.

papi_get_perf_data parameters

*papiHandle
This is an input parameter of type void that points to the handle to identify the associated papi_connect().

typeFlag
This is an input parameter of type int that specifies the type of performance data that is requested. This field is treated as a bit stream and multiple data types can be specified by turning on the required bits. Turning on the bits will return all the performance data of that type (for example, turning on the rules bit returns all rule performance data). The supported bit definitions are:

PAPI_RULES  
Indicates to retrieve performance data from policy rules.

PAPI_ACTIONS  
Indicates to retrieve performance data from policy actions.

PAPI_ALL  
Indicates to retrieve performance data from policy rules and policy actions.

If this API is issued to get data for a type (rule and action) that has not been configured by the DataCollection parameter on the PolicyPerformanceCollection statement, this API is able to return only data that is configured to be collected by the Policy Agent. A PAPI_PERF_COLL_TYPE_MISMATCH return code indicates that the request type was not collected.

*filter
This field of type void is reserved for future use. The only type of filtering that is supported is through the use of the typeFlag field. This parameter must be specified as NULL.

*acceptableCachedTime
This is an input and output parameter of type int. This parameter is specified in seconds and is used to determine whether Policy Agent returns to the caller any performance data that has been cached. If the time that
had elapsed after the data was retrieved and cached is greater than the
acceptableCachedTime, Policy Agent retrieves new data from the stack and
returns this new data to the caller. The acceptableCachedTime is set to
MinimumSamplingInterval. This new performance data will now be
cached.

If this API specifies an acceptableCachedTime that is less than the
MinimumSamplingInterval parameter on the PolicyPerformanceCollection
statement, the MinimumSamplingInterval is used to determine whether
data needs to be retrieved from the stack. An error
(PAPI_INVALID_ACCEPTABLE_CACHED_TIME) is returned stating that
the acceptableCachedTime has been ignored. Data will be returned and the
MinimumSamplingInterval will be returned as the acceptableCachedTime.
See the z/OS Communications Server: IP Configuration Reference for setting the
MinimumSamplingInterval parameter on the PolicyPerformanceCollection
statement in the Policy Agent configuration file.

Note: This is a required parameter.

*kernelName
This is an input parameter. It is a pointer to a character string of the kernel
name whose policy performance data will be returned. The kernel name
must be eight characters or less in length. If kernelName is NULL, the
default kernel name, as determined using the standard resolver search
order, will be used. If kernelName is invalid, the return code of
PAPI_TCPIMAGE_NOT_VALID will be returned.

**perfDataHandle
This is an output parameter that points to the handle to identify this
papi_get_perf_data(). It is the responsibility of the caller to free this
memory by calling the papi_free_perf_data() API. The values within this
memory should be obtained using the helper functions provided.

papi_get_perf_data result
If the retrieve is successful, the call returns a return code of PAPI_OK, and
perfDataHandle is set.

If the retrieve fails, the call returns a non-PAPI_OK return code value.

papi_get_perf_data example
void *mainHandle;
int  nRc, type;
int  cacheTime = 10;
void *perfDataHandle;
static char kernelName[9] = {'\0'};

/* initialization to be done here */

strcpy(kernelName, "TCPNAME");
type = PAPI_RULES_DATA | PAPI_ACTION_DATA;
nRc = papi_get_perf_data(mainHandle,
              type,
              NULL,
              &cachetime,
              kernelName,
              &perfDataHandle);

if (nRc != PAPI_OK)
{
   printf("Error in papi_get_perf_data : %d\n", nRc );
}
else
{
    /* everything is ok so far ... */
}

---

**PAPI helper functions**

The following are PAPI helper functions.

**papi_get_action_perf_by_id** - Obtain performance information on the action specified by the action ID

```c
#include <papiuser.h>

extern ActionPerfInfo *papi_get_action_perf_by_id( void *perfDataHandle,
                                              int actionId);
```

**papi_get_action_perf_by_id description**

This function is used to obtain the performance information on a particular action specified by the action ID.

The performance information is returned as an ActionPerfInfo structure, as described in "papi_get_action_perf_info - Obtain performance information on a particular action" (the recordId will match the actionId field).

**papi_get_action_perf_by_id parameters**

*perfDataHandle*

This is an input parameter of type void. This parameter points to the performance data returned by the papi_get_perf_data() API.

*actionId*

This is an input parameter of type int. This parameter identifies a particular action.

**papi_get_action_perf_by_id result**

If the function is successful, the ActionPerfInfo is returned. This pointer should not be freed.

If the function fails, it returns NULL.

**papi_get_action_perf_info** - Obtain performance information on a particular action

```c
#include <papiuser.h>

extern ActionPerfInfo *papi_get_action_perf_info( void *perfDataHandle,
                                              int actionNum);
```

**papi_get_action_perf_info description**

This function is used to obtain the performance information on a particular action. The action number is specified by actionNum. When multiple policy rules refer to a given policy action, the performance information in the action is an aggregate of all the rules that refer to it.

The performance information is returned as an ActionPerfInfo structure, defined as:

```c
typedef struct {
    char name[MAX_POLICY_NAME]; /* Rule / Action name */
    Bit32 recordType;       /* Rule / Action */
    Bit32 recordId;         /* Rule / Action Id */
} ActionPerfInfo;
```
papi_get_action_perf_info parameters

*perfDataHandle
This is an input parameter of type void. This parameter points to the performance data returned by the papi_get_perf_data() API.

actionNum
This is an input parameter of type int. This parameter points to a particular action. The action number starts from 0. For example, if the number of actions returned by the papi_get_actions_count() function is 6, specify 0 through 5 for the actionNum parameter.

papi_get_action_perf_info result
If the function is successful, the ActionPerfInfo is returned. This pointer should not be freed.

If the function fails, it returns NULL.

papi_get_actions_count - Obtain number of actions in the policy performance data
#include <papiuser.h>

extern int papi_get_actions_count( void *perfDataHandle );
**papi_get_actions_count description**
This function is used to obtain the number of actions in the policy performance data returned by the papi_get_perf_data() function.

**papi_get_actions_count parameters**

*perfDataHandle
This is an input parameter of type void. This parameter points to the performance data returned by the papi_get_perf_data() API.

**papi_get_actions_count result**
If the function is successful, the number of actions in the policy performance data is returned.

If the function fails, 0 is returned.

---

**papi_get_policy_instance - Obtain policy instance number for policies in the policy performance data**

```c
#include <papiuser.h>
extern int papi_get_policy_instance( void *perfDataHandle );
```

**papi_get_policy_instance description**
This function is used to obtain the policy instance number for the set of policies in the policy performance data returned by the papi_get_perf_data() function. The instance number is a value that applies to an entire set of policies, and changes only when a change has been made to the set of policies (for example, when policies are added or deleted).

**papi_get_policy_instance parameters**

*perfDataHandle
This is an input parameter of type void. This parameter points to the performance data returned by the papi_get_perf_data() API.

**papi_get_policy_instance result**
If the function is successful, the policy instance number for the set of policies in the policy performance data is returned. The instance number is a positive integer.

If the function fails, 0 is returned.

---

**papi_get_rule_perf_by_id - Obtain performance information on the rule specified by the rule ID**

```c
#include <papiuser.h>
extern RulePerfInfo *papi_get_rule_perf_by_id( void *perfDataHandle, int ruleId);
```

**papi_get_rule_perf_by_id description**
This function is used to obtain the performance information on the rule that is specified by the rule ID.

The performance information is returned as a RulePerfInfo structure, as described in "papi_get_rule_perf_info - Obtain performance information on a particular rule" on page 394 (the recordId will match the ruleId field).
**papi_get_rule_perf_by_id parameters**

*perfDataHandle*

This is an input parameter of type void. This parameter points to the performance data returned by the papi_get_perf_data() API.

**ruleId** This is an input parameter of type int. This parameter identifies a particular rule.

**papi_get_rule_perf_by_id result**

If the function is successful, the RulePerfInfo is returned. This pointer should not be freed.

If the function fails, it returns NULL.

**papi_get_rule_perf_info - Obtain performance information on a particular rule**

```c
#include <papiuser.h>

extern RulePerfInfo *papi_get_rule_perf_info( void *perfDataHandle,
                                          int ruleNum);
```

**papi_get_rule_perf_info description**

This function is used to obtain the performance information on a particular rule. The rule number is specified by `ruleNum`.

The performance information is returned as a RulePerfInfo structure, defined as:

```c
typedef struct {
    char name[MAX_POLICY_NAME]; /* Rule / Action name */
    Bit32 recordType; /* Rule / Action */
    Bit32 recordId; /* Rule / Action Id */
    time_t firstActivated; /* Time first activated */
    time_t lastMapped; /* Time last mapped */
    Bit64 bytesXmitted; /* Total bytes transmitted */
    Bit64 packetsXmitted; /* Total packets transmitted */
    Bit32 activeConnections; /* Active connections count */
    Bit32 reserved4; /* Reserved */
    Bit64 acceptedConnections; /* Total accepted connections */
    Bit32 smoothedRttAvg; /* Average smoothed RTT */
    Bit32 smoothedRttMdev; /* MDEV of smoothed RTT */
    Bit64 bytesRexmitted; /* Total bytes retransmitted */
    Bit64 packetsRexmitted; /* Total packets retransmitted */
    Bit32 smoothedConnDelayAvg; /* Average smoothed conn delay */
    Bit32 smoothedConnDelayMdev; /* MDEV of smoothed conn delay */
    Bit32 acceptQDelayAvg; /* Average accept queue delay */
    Bit32 acceptQDelayMdev; /* MDEV of accept queue delay */
    Bit64 packetsXmittedInProfile; /* Outbound in profile packets count */
    Bit64 bytesXmittedInProfile; /* Outbound in profile bytes count */
    Bit64 reserved2; /* Reserved */
    Bit64 reserved3; /* Reserved */
    Bit64 packetsReceived; /* Total packets */
} RulePerfInfo;
```
Bit64 bytesReceived;  /* Total bytes received @Q1A*/
Bit64 packetsXmittedTimedOut;  /* Total transmitted packets timed out @Q1A*/
Bit64 deniedConnections;  /* Total denied connections @Q1A*/

} RulePerfInfo, ActionPerfInfo;

**papi_get_rule_perf_info parameters**

*perfDataHandle*
   This is an input parameter of type void. This parameter points to the performance data returned by the papi_get_perf_data() API.

*ruleNum*
   This is an input parameter of type int. This parameter points to a particular rule. The rule number starts from 0. For example, if the number of rules returned by the papi_get_rules_count() function is 5, specify 0 through 4 for the ruleNum parameter.

**papi_get_rule_perf_info result**

If the function is successful, the RulePerfInfo is returned. This pointer should not be freed.

If the function fails, it returns NULL.

**papi_get_rules_count - Obtain number of rules in the policy performance data**

```c
#include <papiuser.h>
extern int papi_get_rules_count( void *perfDataHandle );
```

**papi_get_rules_count description**

This function is used to obtain the number of rules in the policy performance data returned by the papi_get_perf_data() function.

**papi_get_rules_count parameters**

*perfDataHandle*
   This is an input parameter of type void. This parameter points to the performance data returned by the papi_get_perf_data() API.

**papi_get_rules_count result**

If the function is successful, the number of rules in the policy performance data is returned.

If the function fails, 0 is returned.

**papi_strerror - Return string describing PAPI return code value**

```c
#include <papiuser.h>
extern char *papi_strerror( int papiReturnCode );
```

**papi_strerror description**

This function is used to obtain a string describing a PAPI return code value. It is similar to the C strerror() function.
**papi_strerror parameters**

**papiReturnCode**

This is an input parameter of type int. This parameter contains a PAPI return code value.

**papi_strerror result**

If the return code is known, a string describing the return code value is returned.

If the return code is not known, a generic unknown error string is returned.
Chapter 13. FTP Client Application Programming Interface (API)

This topic describes the FTP Client Application Programming Interface (API) to the z/OS FTP client. This topic explains how to initialize the interface, how to use the interface to submit a subcommand to the client, how to retrieve results of a request, and how to terminate the interface.

The following terms apply:

- **Subcommand** refers to z/OS FTP client subcommands.
- **Request** refers to a request sent to the interface (see “Sending requests to the FTP client API” on page 418).

A subcommand might result from a request, because a request can do the following:
- Invoke a specific subcommand (on an SCMD request)
- Result in an implicit subcommand (OPEN resulting from INIT)
- Automatically generate a subcommand (QUIT sent by TERM)
- Result in no subcommand (INIT with no host name or IP address included in start parameters; TERM issued after the user has explicitly issued SCMD QUIT, GETL, or POLL)

**Guideline:** Subcommands are processed by the z/OS FTP client. Some subcommands result in one or more FTP commands being sent to the FTP server. Examples of subcommands and commands are:

- **LOCSTAT** is a subcommand. No command is sent to the server for this subcommand.
- **SYSTEM** is a subcommand; a SYSTEM subcommand causes the client to send a SYST command to the server.
- **GET** is a subcommand. A data connection establishment command (PORT, PASV, or EPSV) might be sent to the server; then a RETR command is sent to the server.

**Tip:** FTP subcommands in [z/OS Communications Server: IP User's Guide and Commands](https://www.ibm.com/support/knowledgecenter/SSSHTM_1.12.0/com.ibm.zos筣08/zosip1cmd.3.0/index.html) describes the subcommands that are supported by the z/OS FTP client.

The interface to the z/OS FTP client enables a user program to send subcommands for the client to process. The user program can multitask to different instances of the interface by requesting no-wait mode when processing a subcommand. The interface also enables the user program to retrieve output that includes the messages from the client, replies from the FTP server, and other data generated as the result of the request.

The interface requires the use of an FTP Client Application Interface (FCAI) control block that is created by the user program [see “FTP Client Application Interface (FCAI) control block” on page 404](https://www.ibm.com/support/knowledgecenter/SSSHTM_1.12.0/com.ibm.zos筣08/zosip1cmd.3.0/index.html). The FCAI is a parameter on all calls to the interface and it is used to pass information between the interface and the user program.

The following topics are included in this topic:

- “FTP client API compatibility considerations” on page 398
FTP client API compatibility considerations

Unless noted in the z/OS Communications Server: New Function Summary, an application program that is compiled and link edited on a particular release of z/OS Communications Server IP can be used on higher level releases. Application programs that are compiled and link edited on a particular release of z/OS Communications Server IP cannot be used on older releases.

FTP client API guidelines and requirements

This topic lists the usage guidelines, requirements, and restrictions for the FTP Client Application Programming Interface (API) for user application programs.

Table 8 describes the programming requirements that apply to the FTP client API.

Table 8. Programming requirements for the FTP client API

<table>
<thead>
<tr>
<th>Function</th>
<th>Restriction</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>SRB mode</td>
<td>The API can be invoked only in TCB mode (task mode).</td>
</tr>
<tr>
<td>Cross-memory mode</td>
<td>The API can be invoked only in a non-cross-memory environment (PASN=SASN=HASN).</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>No locks should be held when issuing these calls.</td>
</tr>
<tr>
<td>Control parameters</td>
<td>Parameter lists and the FCAI control block must reside in primary storage that is accessible by the API to prevent ABENDs in the EZAFTPKS interface program.</td>
</tr>
<tr>
<td>Functional Recovery Routine (FRR)</td>
<td>Do not invoke the API with an FRR set. This can cause system recovery routines to be bypassed and severely damage the system.</td>
</tr>
</tbody>
</table>
### Table 8. Programming requirements for the FTP client API (continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage</td>
<td>Storage acquired for the purpose of containing data returned from an FTP client API call must be obtained in the same key as the application program status word (PSW) at the time of the call.</td>
</tr>
<tr>
<td>Nested FTP client API calls</td>
<td>You cannot issue nested FTP client API calls within the same task. If a request block (RB) issues an FTP client API call and is interrupted by an interrupt request block (IRB) in an STIMER exit, no additional FTP client API calls can be issued by the IRB.</td>
</tr>
<tr>
<td>Addressability mode (AMODE)</td>
<td>The API must be invoked while the caller is in 31-bit addressability mode.</td>
</tr>
</tbody>
</table>

**Guidelines:**

- The FTP client API is re-entrant.
- The user program can have more than one FTP Client Application Interface control block initialized and active in a single address space (see “FTP Client Application Interface (FCAI) control block” on page 404).
- The FTP client API spawns a child process for the z/OS FTP client. If you have a signal handler, you might see the SIGCHLD signal raised when the FTP client terminates; no action is required.
- The z/OS FTP client contains handlers for various asynchronous signals. The FTP client API does not contain any signal handlers, nor does it block or explicitly raise any signals. See “Programming notes for the FTP client API” on page 466 for more information about errors in the z/OS FTP client process.

**Requirements:**

- The application must supply an accessible parameter list and FCAI in primary storage to the FTP client API. ABENDs can occur in the interface if the application fails to comply with this requirement.
- Other ABENDs that occur due to inaccessible storage are trapped by the interface and returned to the application program as an interface error [see FCAI_IE and its associated values in “FTP Client Application Interface (FCAI) control block” on page 404]. To enable the interface to trap these ABENDs, specify TRAP(ON,NOSPIE) to disable invocation of the ESPIE macro when the application program executes within a Language Environment® enclave. For example, specify the following execution parameter for a COBOL application program:
  
  PARM=’/TRAP(ON,NOSPIE)’
  
  For instructions on specifying runtime options and parameters for Language Environment languages, see the information about using runtime options in z/OS Language Environment Programming Guide.
- All of the requests using the same FCAI control block must be made from the same thread.
- The user program must use a standard call interface. Samples are provided for COBOL, C, PL/I, and assembler (see “FTP client API sample programs” on page 474).
- The user program must execute in 31-bit addressing mode (AMODE 31). Other addressing modes are not supported by the interface. The program can reside at any location (RMODE can be 24 or ANY).
The application must have an OMVS segment defined (or defaulted).
The interface module EZAFTPKI must be accessible to the application in the
linklist or in a STEPLIB or JOBLIB DD statement.
You can either statically link the FTP client API stub program (EZAFTPKS) into
the user application program or load it dynamically for execution. The stub
program resides in SYSn.CSSLIB and is designed to maintain upward
compatibility.
For a PL/I program, include this statement before your first call instruction:
DCL EZAFTPKS ENTRY OPTIONS(RETCODE,ASM,INTER) EXT;

Java call formats

The FTP client API for Java provides an interface to the z/OS FTP client that
enables a user program written in Java to send subcommands for the client to
process. The user program can also use this interface to retrieve output that
includes the messages from the client, replies from the FTP server, and other data
that is generated as the result of the request.

Each instance of the interface is represented by an FTPClient object. A user
program can create multiple instances of the FTPClient object. A single user
program can use these objects to establish multiple simultaneous connections to
the same FTP server or to different FTP servers. The user program can multitask to
different instances of the interface by requesting that the API not wait for the
completion of an FTP subcommand before it returns control.

The z/OS FTP client that is used by the FTP client API is described in File Transfer
Protocol (FTP) information in z/OS Communications Server: IP User's Guide and
Commands and in the File Transfer Protocol information in z/OS Communications
Server: IP Configuration Reference. The z/OS FTP client, when started with the FTP
client API for Java, operates essentially the same as it does when invoked in an
interactive environment under the z/OS UNIX shell. See "z/OS FTP client
behavior when invoked from the FTP client API" on page 402 for a description of
the differences.

FTP client API for Java package uses the Java Native Interface (JNI) to interface
with the z/OS FTP client using the C Java FTP client API. See "FTP client API for
C functions" on page 435 for more information about the FTP client API for C.

The FTP client API for Java uses the Java logging API (java.util.logging.Logger) to
generate debug information. See documentation about the java.util.logging package
for details about using the Java logging API.

Guidelines:
• The user program can have more than one FTPClient object initialized and
active in a single address space.
• The FTP client API spawns a child process for the z/OS FTP client. If you have
a signal handler, you might see the SIGCHILD signal raised when the FTP client
terminates; no action is required.
• The z/OS FTP client contains handlers for various asynchronous signals. The
FTP client API does not contain any signal handlers and does not block or
explicitly raise any signals.

Requirements:
• All requests that use the same FTPClient object must be made from the same thread.
• The Java JVM in which the application runs must operate in 31-bit addressing mode. No other addressing modes are supported by the interface.
• The application must have an OMVS segment defined (or set by default).
• The interface module EZAFTPCLI must be accessible to the application in the link list or in a STEPLIB or JOBLIB DD statement.
• To use this package, you must include the EZAFTP.jar file in your classpath. In addition, the libEZAFTP.so file must be located in $LIBPATH so that the JNI methods can be found. The EZAFTP.jar file is installed into the directory /usr/include/java_classes and the libEZAFTP.so file is installed into the directory /usr/lib.

For more information about the FTP client API for Java, see the JavaDoc that is included in the EZAFTPDoc.jar file, which is installed into the directory /usr/include/java_classes. Download the jar file to a workstation, unpack it, and read it in a Web browser.

**COBOL, C, REXX, assembler, and PL/I call formats**

The FTP client API is invoked by calling the EZAFTPKS program. The following list shows formats for the COBOL, C, assembler, and PL/I languages.

• COBOL language call format
  
  The EZAFTPKS call format for COBOL programs is the following:
  ```
  "---CALL EZAFTPKS USING FCAI-Map, request_type, parm1, parm2, ... ---->
  ```

• C language call format
  
  See "FTP client API for C functions" on page 435 for in-line functions that can be used with C/C++ programs. These in-line functions provide the calls to EZAFTPKS.

• REXX™ language call format
  
  See "FTP client API for REXX function" on page 439 for an external REXX function that can be used with REXX programs.

• Assembler language call format
  
  The EZAFTPKS call format for assembler language programs is the following:
  ```
  "---CALL EZAFTPKS,(FCAI_Map, request_type, parm1, parm2, ... ),VL ---->
  ```

• PL/I language call format
  
  The EZAFTPKS call format for PL/I programs is the following:
  ```
  "---CALL EZAFTPKS (FCAI_Map, request_type, parm1, parm2, ... ); ---->
  ```

The following parameter definitions apply for each of the call formats:

**FCAI-Map or FCAI_Map**

The name of the FTP Client Application Interface block storage that describes an instance of use of the interface, or a pointer to the storage. The storage for this space is acquired by the calling program. COBOL and assembler callers can append storage within the calling program to the area defined in the COBOL copy member or assembler macro. PL/I or C callers must alter the INCLUDE member to add user storage to the area.

**request_type**

The type of processing requested by the invocation of the interface.

**parm_n**

A variable number of parameters, depending on the request type.
Some parameters are optional depending on the request. When an optional parameter is omitted but more parameters follow, use a placeholder appropriate for the language:

- COBOL uses the special name OMITTED in place of the missing parameter.
- C and PL/I use the special name NULL in place of the missing parameter.
- Assembler language uses a comma to indicate the position of the missing parameter.

### Converting parameter descriptions

The coding examples in this topic use IBM Enterprise COBOL for z/OS language syntax and conventions. The application program should use the syntax and conventions that are appropriate for the language in which it is written.

Example storage definition statements for COBOL, C, PL/I, and assembler language programs are:

- **IBM Enterprise COBOL for z/OS**
  
  PIC S9(4) COMP-5 HALFWORD BINARY VALUE  
  PIC S9(8) COMP-5 FULLWORD BINARY VALUE  
  PIC X(n) CHARACTER FIELD OF n BYTES

- **C**
  
  short int /* HALFWORD BINARY VALUE */  
  long int /* FULLWORD BINARY VALUE */  
  char x[n] /* CHARACTER FIELD OF n BYTES */

- **PL/I declare statement**
  
  DCL HALF FIXED BIN(15), HALFWORD BINARY VALUE  
  DCL FULL FIXED BIN(31), FULLWORD BINARY VALUE  
  DCL CHARACTER CHAR(n) CHARACTER FIELD OF n BYTES

- **Assembler declaration**
  
  DS H HALFWORD BINARY VALUE  
  DS F FULLWORD BINARY VALUE  
  DS CLn CHARACTER FIELD OF n BYTES

### z/OS FTP client behavior when invoked from the FTP client API

The z/OS FTP client that is used by the FTP client API is described in the File Transfer Protocol (FTP) information in the z/OS Communications Server: IP User’s Guide and Commands and in the File Transfer Protocol information in the z/OS Communications Server: IP Configuration Reference. The z/OS FTP client, when started with the FTP client API, operates essentially as it does when invoked in an interactive environment under the z/OS UNIX shell.

The following are the differences in the behavior of the z/OS FTP client when it is invoked by the FTP client API:

- When the z/OS FTP client starts, options (parameters) are processed that affect the operation of the client. The user program uses the START-PARM parameter on the INIT request to pass its options to the FTP client API, which passes the options on to the client (see “INIT” on page 419). All of the options that are defined for the z/OS FTP client are accepted when the client is started with the FTP client API. However, the following conditions apply:
  - The -e and the EXIT options are ignored by the FTP client API.

  These options are intended to affect the operation of the FTP client by causing it to stop when an eligible subcommand encounters an error. In the FTP client API, those errors are passed back to the user program as a client error code.
The user program can process the error and decide whether to continue or to end the client process.

- The -i option to disable prompting for the subcommands MGET, MPUT, MDELETE has no effect on the API.

See "Prompts from the client" on page 462 for a discussion on how prompts are handled when the z/OS FTP client is invoked from the FTP client API.

- When the z/OS FTP client is invoked within the z/OS UNIX shell, a backslash (\) is required before the open parenthesis ([ ]) that signals the start of the MVS-type parameters. Do not use the backslash when invoking the client with the FTP client API.

- When the z/OS FTP client is invoked from a batch job or from TSO, data sets and files can be allocated to DD names for use by the client. When the z/OS FTP client is spawned from the FTP client API, DD names that are associated with the application are not available to the client process. Specifically, the use of the following DD names is not supported by the FTP client API:
  
  - SYSFTPD and SYSTCPD
  - NETRC
  - INPUT (SYSIN) and OUTPUT

Transfer of data sets by DD name is not possible in the spawned client process. If the application sends a transfer subcommand (PUT, GET, and so on) that includes //DD:ddname, the client returns an error such as FCAI_CEC_FILE_ACCESS.

Changing local site defaults using FTP.DATA in

- Changing local site defaults using FTP.DATA in z/OS Communications Server: IP User’s Guide and Commands describes how to change local site defaults by using FTP.DATA. The search order for locating the FTP.DATA configuration file for the client under the FTP client API is as follows:
  1. -f parameter
  2. $HOME/ftp.data
  3. userid.FTP.DATA
  4. /etc/ftp.data
  5. SYS1.TCPPARMS(FTPDATA)
  6. tcpip.hlq.FTP.DATA

Restriction: The -f parameter cannot be a DD name when the FTP client is invoked from the FTP client API.

Tip: The $HOME variable is taken from the user’s RACF® user profile OMVS segment. The $HOME variable can be modified with the environment variable list passed during FTP client API initialization. Initialization is performed with an ftpapi('init') request in the REXX environment; with a call to the EZAFTPKS stub program with the INIT keyword for assembler, COBOL, and PL/1; or with a call to the FAPI_INIT function for C/C++.

- The FTP.DATA statements that can be used to change local site defaults for the z/OS FTP client are defined in the FTP.DATA statements information in the z/OS Communications Server: IP Configuration Reference. One of the statements is CLIENTERRCODES, which controls return code settings in the client. When the client is started by the FTP client API, the value on the CLIENTERRCODES statement does not affect the reporting of results. See “Interpreting results from an interface request” on page 464 for a complete description of how results from the interface and the client are reported.

- When the z/OS FTP client or server prompts for a password or accounting information, the prompt must be satisfied before any other subcommand or command is accepted. Under the FTP client API, the user program has the...
option to issue GETL or TERM even when a password or accounting information is expected. If the request is TERM, the interface generates a QUIT subcommand, which is accepted and stops the client process. See “GETL” on page 427, “TERM” on page 433, and “Prompts from the client” on page 462 for more information about how the FTP client API handles prompts.

- The FTP client API requires a secondary subcommand parameter with an SCMD PROXY request. See “SCMD” on page 422 and “Prompts from the client” on page 462 for more information.

**FTP Client Application Interface (FCAI) control block**

The user program written in Cobol, C, assembler, and PL/I and the FTP Client Application Programming Interface use the FCAI control block to describe an instance of use of the interface. The space for this control block is acquired by the user program.

**Tip:** REXX programs do not use an FCAI control block. For REXX programs, see “FTP Client Application Interface (FCAI) stem variables” on page 412.

**Requirement:** The FCAI control block must be aligned on at least a fullword boundary and reside in primary storage.

**Guideline:** FCAI_Map can be altered to embed in a structure that generates multiple copies of the FCAI. If this is done, ensure that additional storage in FCAI_UserArea is acquired in fullword increments to preserve the alignment of each copy of the FCAI control block.

Table 9 is a layout of the control block. The **Type** column indicates the type of value that the field contains: text (all text fields must be in EBCDIC), binary, or undefined. This column also contains the following information:

- (I) to indicate input from the user program. If the field is defined by values that appear in a table following Table 9, there is a reference to that table.
- (O) to indicate output from the interface program. If the field is defined by values that appear in a table following Table 9, there is a reference to that table.
- (R) to indicate a reserved field.
- (U) to indicate user area.

The field names in this table are the names used for assembler and PL/I. All sections of this topic use this name syntax with the following exceptions:

- “Sending requests to the FTP client API” on page 418 uses the COBOL syntax; the field names contain a dash (-) instead of an underscore (_).
- “FTP client API for C functions” on page 435 uses the C syntax; the field names are identical to assembler and PL/I, but the constant definitions are all in upper case.

Other than these differences for C and COBOL, the field names in the supplied macros and samples for each language are similar.

**Table 9. FCAI control block**

<table>
<thead>
<tr>
<th>Field name</th>
<th>Length</th>
<th>Offset</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCAI_Map</td>
<td>256 and higher</td>
<td>0</td>
<td>FCAI control block.</td>
<td>various</td>
</tr>
<tr>
<td>FCAI_DefinedFields</td>
<td>76</td>
<td>0</td>
<td>Fields defined to the interface.</td>
<td>various</td>
</tr>
</tbody>
</table>
Table 9. FCAI control block (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Length</th>
<th>Offset</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCAI_Eyecatcher</td>
<td>4</td>
<td>0</td>
<td>Eyecatcher= FCAI; this field is required.</td>
<td>(I) text</td>
</tr>
<tr>
<td>FCAI_Size</td>
<td>2</td>
<td>4</td>
<td>Size of FCAI; this field is required and has a minimum value of 256.</td>
<td>(I) binary</td>
</tr>
<tr>
<td>FCAI_Version</td>
<td>1</td>
<td>6</td>
<td>Version of FCAI; this field is required.</td>
<td>(I - see Table 10 on page 406) binary</td>
</tr>
<tr>
<td>FCAI_PollWait</td>
<td>1</td>
<td>7</td>
<td>POLL wait timer in seconds (see “FCAI_PollWait: Specifying a wait time before POLL” on page 468).</td>
<td>(I) binary</td>
</tr>
<tr>
<td>FCAI_ReqTimer</td>
<td>1</td>
<td>8</td>
<td>Request timer in seconds or 0 for none (see “FCAI_ReqTimer: Controlling requests that retrieve results from the spawned z/OS FTP client process” on page 468).</td>
<td>(I) binary</td>
</tr>
<tr>
<td>FCAI_TraceIt</td>
<td>1</td>
<td>9</td>
<td>Trace indicator for this request (see “Using the FTP client API trace” on page 470).</td>
<td>(I - see Table 11 on page 407) binary</td>
</tr>
<tr>
<td>FCAI_TraceID</td>
<td>3</td>
<td>10</td>
<td>ID used in a trace record. This value is used only when a request initiates the interface trace function and does not change thereafter.</td>
<td>(I) text</td>
</tr>
<tr>
<td>FCAI_TraceCAPI</td>
<td>1</td>
<td>13</td>
<td>TRACECAPI value on FTP.DATA statement.</td>
<td>(O - see Table 12 on page 407) binary</td>
</tr>
<tr>
<td>FCAI_TraceStatus</td>
<td>1</td>
<td>14</td>
<td>Status of the trace (see “FCAI_Status_TraceFailed and FCAI_TraceStatus: Reporting failures in the interface trace function” on page 467).</td>
<td>(O - see Table 13 on page 407) binary</td>
</tr>
<tr>
<td>FCAI_TraceSClass</td>
<td>1</td>
<td>15</td>
<td>SYSPUT class for trace file. This value is used only when a request initiates the interface trace function and does not change thereafter.</td>
<td>(I) text</td>
</tr>
<tr>
<td>FCAI_TraceName</td>
<td>8</td>
<td>16</td>
<td>ddname of the trace file.</td>
<td>(O) text</td>
</tr>
<tr>
<td>FCAI_Token</td>
<td>4</td>
<td>24</td>
<td>Interface token (do not alter after INIT).</td>
<td>(O) binary</td>
</tr>
<tr>
<td>FCAI_RequestID</td>
<td>4</td>
<td>28</td>
<td>Last request (for example, SCMD).</td>
<td>(O) text</td>
</tr>
<tr>
<td>FCAI_RCV</td>
<td>16</td>
<td>32</td>
<td>Request completion values (see “Interpreting results from an interface request” on page 464).</td>
<td>(O) binary</td>
</tr>
<tr>
<td>FCAI_Result</td>
<td>1</td>
<td>32</td>
<td>Request result (the return code register also contains this value).</td>
<td>(O - see Table 14 on page 407) binary</td>
</tr>
<tr>
<td>FCAI_Status</td>
<td>1</td>
<td>33</td>
<td>Status code.</td>
<td>(O - see Table 15 on page 408) binary</td>
</tr>
</tbody>
</table>
Table 9. FCAI control block (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Length</th>
<th>Offset</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCAI_IE</td>
<td>1</td>
<td>34</td>
<td>Interface error.</td>
<td>(O - see Table 16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>on page 408)</td>
<td>binary</td>
</tr>
<tr>
<td>FCAI_CEC</td>
<td>1</td>
<td>35</td>
<td>Client error code (see FTP return codes in the z/OS Communications Server: IP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>User's Guide and Commands).</td>
<td>(O) binary</td>
</tr>
<tr>
<td>FCAI_ReplyCode</td>
<td>2</td>
<td>36</td>
<td>Server reply code or 0 if no reply (see FTPD reply codes in z/OS Communications Server: IP and SNA Codes).</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(O) binary</td>
<td></td>
</tr>
<tr>
<td>FCAI_SCMD</td>
<td>1</td>
<td>38</td>
<td>Subcommand code (see FTP subcommand codes in z/OS Communications Server: IP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>User's Guide and Commands).</td>
<td>(O) binary</td>
</tr>
<tr>
<td>Reserved</td>
<td>1</td>
<td>39</td>
<td>Reserved.</td>
<td>(R) undefined</td>
</tr>
<tr>
<td>FCAI_ReturnCode</td>
<td>4</td>
<td>40</td>
<td>Return code (see Table 13 on page 407 and Table 16 on page 408 for errors that have associated return code data).</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(O) binary</td>
<td></td>
</tr>
<tr>
<td>FCAI_ReasonCode</td>
<td>4</td>
<td>44</td>
<td>Reason code (see Table 13 on page 407 and Table 16 on page 408 for errors that have associated reason code data).</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(O) binary</td>
<td></td>
</tr>
</tbody>
</table>

Summary fields for output lines that are held in the interface buffer

<table>
<thead>
<tr>
<th>Field name</th>
<th>Length</th>
<th>Offset</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCAI_NumberLines</td>
<td>4</td>
<td>48</td>
<td>Number of output lines returned by the request.</td>
<td>(O) binary</td>
</tr>
<tr>
<td>FCAI_LongestLine</td>
<td>4</td>
<td>52</td>
<td>Size of the longest line.</td>
<td>(O) binary</td>
</tr>
<tr>
<td>FCAI_SizeAll</td>
<td>4</td>
<td>56</td>
<td>Size of all output lines.</td>
<td>(O) binary</td>
</tr>
<tr>
<td>FCAI_SizeMessages</td>
<td>4</td>
<td>60</td>
<td>Size of all message lines.</td>
<td>(O) binary</td>
</tr>
<tr>
<td>FCAI_SizeReplies</td>
<td>4</td>
<td>64</td>
<td>Size of all reply lines.</td>
<td>(O) binary</td>
</tr>
<tr>
<td>FCAI_SizeList</td>
<td>4</td>
<td>68</td>
<td>Size of all list lines.</td>
<td>(O) binary</td>
</tr>
<tr>
<td>FCAI_SizeTrace</td>
<td>4</td>
<td>72</td>
<td>Size of all trace lines.</td>
<td>(O) binary</td>
</tr>
<tr>
<td>FCAI_PID</td>
<td>4</td>
<td>76</td>
<td>Process ID of FTP client.</td>
<td>(O) binary</td>
</tr>
</tbody>
</table>

Reserved and user areas

<table>
<thead>
<tr>
<th>Field name</th>
<th>Length</th>
<th>Offset</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCAI_ReservedForInterface</td>
<td>176</td>
<td>80</td>
<td>Reserved.</td>
<td>(R) undefined</td>
</tr>
<tr>
<td>FCAI_UserArea</td>
<td>0 to unlimited</td>
<td>256</td>
<td>Start of user area. It is not necessary to add the size of the user area to the value in FCAI_Size.</td>
<td>(U) undefined</td>
</tr>
</tbody>
</table>

Table 10. FCAI_Version field value

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCAI_Version_Number</td>
<td>1</td>
<td>Version number</td>
</tr>
</tbody>
</table>

For more information about the values found in Table 11 on page 407 see “Using the FTP client API trace” on page 470.
Table 11. FCAI_TraceIt field value

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCAI_TraceIt_No</td>
<td>0</td>
<td>Do not trace this request.</td>
</tr>
<tr>
<td>FCAI_TraceIt_Yes</td>
<td>1</td>
<td>Trace this request.</td>
</tr>
</tbody>
</table>

Table 12. FCAI_TraceCAPI field value

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCAI_TraceCAPI_C</td>
<td>0</td>
<td>Trace according to FCAI_TraceIt.</td>
</tr>
<tr>
<td>FCAI_TraceCAPI_A</td>
<td>1</td>
<td>Trace all events.</td>
</tr>
<tr>
<td>FCAI_TraceCAPI_N</td>
<td>2</td>
<td>Trace no events.</td>
</tr>
</tbody>
</table>

For more information about the values found in Table 13, see “FCAI_Status_TraceFailed and FCAI_TraceStatus: Reporting failures in the interface trace function” on page 467.

Table 13. FCAI_TraceStatus field value

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
<th>Additional information returned with FCAI_Status_TraceFailed</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCAI_TraceStatus_OK</td>
<td>0</td>
<td>Tracing OK or not started</td>
<td>None</td>
</tr>
<tr>
<td>FCAI_TraceStatus_StorageErr</td>
<td>1</td>
<td>Failed to acquire or access storage</td>
<td>FCAI_ReturnCode = GETMAIN return code</td>
</tr>
<tr>
<td>FCAI_TraceStatus_AllocErr</td>
<td>2</td>
<td>Allocation error</td>
<td>FCAI_ReturnCode = S99ERROR value or 8; FCAI_ReasonCode = S99ERSN for SMS error, S99INFO otherwise</td>
</tr>
<tr>
<td>FCAI_TraceStatus_OpenErr</td>
<td>3</td>
<td>Open error</td>
<td>FCAI_ReturnCode contains the OPEN return code or FCAI_ReasonCode contains the ABEND code</td>
</tr>
<tr>
<td>FCAI_TraceStatus_WriteErr</td>
<td>4</td>
<td>Write error</td>
<td>FCAI_ReasonCode=ABEND code</td>
</tr>
<tr>
<td>FCAI_TraceStatus_CloseErr</td>
<td>5</td>
<td>Close error</td>
<td>FCAI_ReturnCode contains the CLOSE return code or FCAI_ReasonCode contains the ABEND code</td>
</tr>
<tr>
<td>FCAI_TraceStatus_SysoutClassErr</td>
<td>6</td>
<td>FCAI_TraceSClass contains a Sysout output class that is not valid</td>
<td>None</td>
</tr>
</tbody>
</table>

For more information about the values found in Table 14, see “Interpreting results from an interface request” on page 464.

Table 14. FCAI_Result field value

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCAI_Result_OK</td>
<td>0</td>
<td>OK with no additional status.</td>
</tr>
<tr>
<td>FCAI_Result_Status</td>
<td>1</td>
<td>Status code returned in FCAI_Status.</td>
</tr>
<tr>
<td>FCAI_Result_IE</td>
<td>2</td>
<td>Interface error returned in FCAI_IE.</td>
</tr>
<tr>
<td>FCAI_Result_CEC</td>
<td>3</td>
<td>Client Error Code returned in FCAI_CEC.</td>
</tr>
<tr>
<td>FCAI_Result_NoMatch</td>
<td>4</td>
<td>GETL request has no matches.</td>
</tr>
</tbody>
</table>

Chapter 13. FTP Client Application Programming Interface (API)  407
Table 14. FCAI_Result field value (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCAI_Result_UnusableFCAI</td>
<td>17</td>
<td>FCAI is not usable.</td>
</tr>
<tr>
<td>FCAI_Result_TaskMismatch</td>
<td>18</td>
<td>Task is not the same as INIT task.</td>
</tr>
<tr>
<td>FCAI_Result_CliProcessKill</td>
<td>32</td>
<td>TERM issued BPX1KIL to end the client process. This is informational.</td>
</tr>
</tbody>
</table>

For more information about the values found in Table 15, see “Prompts from the client” on page 462 and “Interpreting results from an interface request” on page 464.

Table 15. FCAI_Status field values

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCAI_Status_InProgress</td>
<td>1</td>
<td>Subcommand is in-progress.</td>
<td>This status is returned for an SCMD issued in no-wait mode or if FCAI_ReqTimer expires on an SCMD issued in wait mode, and on any subsequent POLL requests until the SCMD completes.</td>
</tr>
<tr>
<td>FCAI_Status_PromptPass</td>
<td>2</td>
<td>Request prompted for a PASS subcommand.</td>
<td>The interface accepts only SCMD PASS or a GETL request until the prompt is satisfied or this instance of the interface is terminated.</td>
</tr>
<tr>
<td>FCAI_Status_Acct</td>
<td>3</td>
<td>Request prompted for an ACCT subcommand.</td>
<td>The interface accepts only SCMD ACCT or a GETL request until the prompt is satisfied or this instance of the interface is terminated.</td>
</tr>
<tr>
<td>FCAI_Status_TraceFailed</td>
<td>200</td>
<td>The interface trace failed on this request and has been disabled.</td>
<td>This status is added to any other status returned. See “FCAI_Status_TraceFailed and FCAI_TraceStatus: Reporting failures in the interface trace function” on page 467 for more information.</td>
</tr>
</tbody>
</table>

Table 16. FCAI_IE field values

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>General interface errors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FCAI_IE_RequestMissing</td>
<td>1</td>
<td>Request ID is missing.</td>
<td>Request ID parameter not passed to EZAFTPKS.</td>
</tr>
<tr>
<td>FCAI_IE_RequestUnknown</td>
<td>2</td>
<td>Unknown request.</td>
<td>Request ID not INIT, TERM, POLL, GETL, or SCMD.</td>
</tr>
<tr>
<td>FCAI_IE_ParmMissing</td>
<td>3</td>
<td>Parameter missing.</td>
<td>Required parameter not passed to EZAFTPKS.</td>
</tr>
<tr>
<td>FCAI_IE_ParmStorageErr</td>
<td>4</td>
<td>Storage error for a parameter.</td>
<td>Parameter list points to inaccessible storage.</td>
</tr>
</tbody>
</table>
Table 16. FCAI_IE field values (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCAI_IE_TooManyParameters</td>
<td>5</td>
<td>More parameters were passed than are defined for this request type.</td>
<td>Failure to include VL on an assembler language call to EZAFTPKS is one cause.</td>
</tr>
<tr>
<td>FCAI_IE_InternalErr</td>
<td>7</td>
<td>Internal error in the interface.</td>
<td>For example, allocated buffer not found in chain; see FCAI_IE_InternalErr: Unanticipated exceptional conditions in the interface on page 469.</td>
</tr>
<tr>
<td>FCAI_IE_LengthInvalid</td>
<td>8</td>
<td>Negative or zero length.</td>
<td>For example, zero buffer length with GETL; see FCAI_IE_LengthInvalid: Improper lengths passed to the interface on page 467.</td>
</tr>
<tr>
<td>INIT errors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FCAI_IE_APIAlreadyInit</td>
<td>16</td>
<td>Interface already initialized.</td>
<td>This FCAI was used on a prior INIT request.</td>
</tr>
<tr>
<td>FCAI_IE_InitParmTooBig</td>
<td>17</td>
<td>INIT parameter is too big.</td>
<td>FTP start parms string exceeds 2393 bytes.</td>
</tr>
<tr>
<td>FCAI_IE_APILoadFailed</td>
<td>18</td>
<td>The load of the interface failed.</td>
<td>FCAI_ReturnCode and FCAI_ReasonCode contain values set by the BLDL service.</td>
</tr>
<tr>
<td>FCAI_IE_NoTokenAddr</td>
<td>19</td>
<td>Token address is 0.</td>
<td>This FCAI has not been initialized and the current request is not INIT.</td>
</tr>
<tr>
<td>FCAI_IE_BadTokenAddr</td>
<td>20</td>
<td>Bad token field.</td>
<td>FCAI_Token is not valid.</td>
</tr>
<tr>
<td>FCAI_IE_GetWorkareaFailed</td>
<td>21</td>
<td>Error acquiring workarea.</td>
<td>FCAI_ReturnCode contains the value returned by the GETMAIN service.</td>
</tr>
<tr>
<td>FCAI_IE_ReqTimerExpired</td>
<td>22</td>
<td>INIT timed out waiting for output from the client.</td>
<td>See FCAI_ReqTimer: Controlling requests that retrieve results from the spawned z/OS FTP client process on page 468 for more information.</td>
</tr>
<tr>
<td>FCAI_IE_TooManyInitParms</td>
<td>23</td>
<td>More than 30 separate tokens were passed in the start parameters.</td>
<td>Tokens are defined as characters or punctuation surrounded by whitespace.</td>
</tr>
<tr>
<td>FCAI_IE_TooManyEnvVars</td>
<td>24</td>
<td>More than nine environment variables were passed on the INIT.</td>
<td>Use _CEE_ENVFILE= hfs_filename to pass more than nine environment variables.</td>
</tr>
<tr>
<td>FCAI_IE_CreatePipeErr</td>
<td>26</td>
<td>Error creating pipe to the client.</td>
<td>FCAI_ReturnCode and FCAI_ReasonCode contain values set by BPX1PIP in the z/OS UNIX System Services Programming: Assembler Callable Services Reference.</td>
</tr>
</tbody>
</table>
### Table 16. FCAI_IE field values (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCAI_IE_SpawnErr</td>
<td>27</td>
<td>Error spawning the client.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>FCAI_ReturnCode and FCAI_ReasonCode contain values set by BPXISP in the</td>
<td><a href="https://www.ibm.com/docs/en/zos/2.5.1">z/OS UNIX System Services Programming: Assembler Callable Services Reference</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>z/OS UNIX System Services Programming: Assembler Callable Services Reference</em></td>
<td></td>
</tr>
<tr>
<td>SCMD errors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FCAI_IE_ScmdParmTooBig</td>
<td>32</td>
<td>SCMD subcommand string too long.</td>
<td>SCMD subcommand parameter string must not exceed 2064 bytes.</td>
</tr>
<tr>
<td>FCAI_IE_UNKMode</td>
<td>33</td>
<td>Mode parameter value incorrect.</td>
<td>Mode parameter value must be W or N.</td>
</tr>
<tr>
<td>FCAI_IE_PassPromptErr</td>
<td>34</td>
<td>The current SCMD request is in error because PASS is required.</td>
<td>A prior request set FCAI_Status_PromptPass and the current SCMD is not PASS.</td>
</tr>
<tr>
<td>FCAI_IE_AcctPromptErr</td>
<td>35</td>
<td>The current SCMD request is in error because ACCT is required.</td>
<td>A prior request set FCAI_Status_PromptAcct and the current SCMD is not ACCT.</td>
</tr>
<tr>
<td>FCAI_IE_AlgReadyInProgress</td>
<td>37</td>
<td>The current SCMD request is in error because an SCMD is in-progress.</td>
<td>A prior SCMD returned FCAI_Status_InProgress. Issue a POLL request to complete the prior SCMD.</td>
</tr>
<tr>
<td>FCAI_IE_CliProcessStopped</td>
<td>38</td>
<td>The current request is in error because the client process was stopped normally with a QUIT subcommand.</td>
<td>Only GETL can be issued prior to TERM when the client has processed a QUIT subcommand; the current request is not GETL or TERM.</td>
</tr>
<tr>
<td>FCAI_IE_WriteErr</td>
<td>41</td>
<td>Error writing to the client.</td>
<td>FCAI_ReturnCode and FCAI_ReasonCode contain values set by BPX1WRT in the <a href="https://www.ibm.com/docs/en/zos/2.5.1">z/OS UNIX System Services Programming: Assembler Callable Services Reference</a></td>
</tr>
<tr>
<td>INIT, SCMD, and POLL errors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FCAI_IE_UnknownOperation</td>
<td>64</td>
<td>GETL OPERATION parameter is not recognized.</td>
<td>OPERATION must be FIND or COPY.</td>
</tr>
</tbody>
</table>

### INIT, SCMD, and POLL errors

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCAI_IE_ReadErr</td>
<td>42</td>
<td>Error reading from the client.</td>
<td>FCAI_ReturnCode and FCAI_ReasonCode contain values set by BPX1RED in the <a href="https://www.ibm.com/docs/en/zos/2.5.1">z/OS UNIX System Services Programming: Assembler Callable Services Reference</a></td>
</tr>
<tr>
<td>FCAI_IE_CliProcessBroken</td>
<td>47</td>
<td>Client process broken; send a TERM request.</td>
<td>A previous error was encountered when communicating with the client or the client has terminated unexpectedly. Only GETL or TERM are accepted when this occurs.</td>
</tr>
<tr>
<td>POLL errors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FCAI_IE_NotInProgress</td>
<td>48</td>
<td>A POLL request was issued when no subcommand was in-progress.</td>
<td>Processing can continue normally with the next request.</td>
</tr>
<tr>
<td>GETL errors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FCAI_IE_UnknownOperation</td>
<td>64</td>
<td>GETL OPERATION parameter is not recognized.</td>
<td>OPERATION must be FIND or COPY.</td>
</tr>
</tbody>
</table>
Table 16. FCAI_IE field values  (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCAI_IE_UnknownType</td>
<td>65</td>
<td>GETL TYPE parameter is not recognized.</td>
<td>TYPE must be one of the following: M - client message R - server reply T - client trace L - LIST/NLST output A - any</td>
</tr>
<tr>
<td>FCAI_IE_UnknownSequence</td>
<td>66</td>
<td>GETL FIND SEQUENCE parameter is not recognized.</td>
<td>Sequence must be one of the following: F - first L - last N - next</td>
</tr>
<tr>
<td>FCAI_IE_VectorStorageErr</td>
<td>67</td>
<td>The buffer described by the vector cannot be accessed.</td>
<td>See the description of the VECTOR parameter in “Parameter values that are set by the application” on page 439</td>
</tr>
<tr>
<td>FCAI_IE_BufferTooSmall</td>
<td>68</td>
<td>The buffer described by the vector is too small to hold the first line of returned output.</td>
<td>See GETL on page 423 for more information.</td>
</tr>
<tr>
<td>FCAI_IE_TraceIDTooBig</td>
<td>69</td>
<td>Length of traceID must be 0 - 3 characters.</td>
<td>Set only by the FTP client API for REXX.</td>
</tr>
<tr>
<td>FCAI_IE_TraceSClassTooBig</td>
<td>70</td>
<td>Length of traceSClass value must be 0 - 3 characters.</td>
<td>Set only by the FTP client API for REXX.</td>
</tr>
<tr>
<td>FCAI_IE_UnknownTraceIt</td>
<td>71</td>
<td>The traceIt value is not recognized.</td>
<td>Set only by the FTP client API for REXX.</td>
</tr>
<tr>
<td>FCAI_IE_ReqTimerInvalid</td>
<td>72</td>
<td>The request timer value is not in the range 0 - 255.</td>
<td>Set only by the FTP client API for REXX.</td>
</tr>
<tr>
<td>FCAI_IE_LinesParmTooBig</td>
<td>73</td>
<td>The GETL lines stem name is more than 200 characters in length.</td>
<td>Set only by the FTP client API for REXX.</td>
</tr>
<tr>
<td>FCAI_IE_PollWaitInvalid</td>
<td>74</td>
<td>The pollWait value is not in the range 0 - 255.</td>
<td>Set only by the FTP client API for REXX.</td>
</tr>
<tr>
<td>FCAI_IE_NumTraceInvalid</td>
<td>75</td>
<td>The numTrace value is not in the range 1 - 100000.</td>
<td>Set only by the FTP client API for REXX.</td>
</tr>
<tr>
<td>FCAI_IE_FcaiMapParmTooBig</td>
<td>76</td>
<td>The FCAI stem name is more than 250 characters in length.</td>
<td>Set only by the FTP client API for REXX.</td>
</tr>
<tr>
<td>FCAI_IE_EnvVarStorageErr</td>
<td>77</td>
<td>Unable to allocate storage for an environment variable.</td>
<td>Set only by the FTP client API for REXX.</td>
</tr>
<tr>
<td>FCAI_IE_SysoutClassErr</td>
<td>78</td>
<td>FCAI_TraceSClass contains a Sysout output class that is not valid.</td>
<td>Set only by the FTP client API for REXX.</td>
</tr>
</tbody>
</table>

Define the space for the FCAI by including the appropriate macro or source copy book in your program as follows:
FTP Client Application Interface (FCAI) stem variables

The user-written REXX program uses an FCAI stem to represent an instance of use of the interface.

Guideline: When passing the REXX stem to the FTP client API for REXX, include the terminating period (.).

Table 17 describes the stem variables that are created from the REXX stem. The Type column indicates the type of value that the field contains, which can be decimal or binary.

All stem variables, with the exception of stem.FCAI_Map, are output fields. The stem.FCAI_Map variable is used internally by the FTP client API for REXX function package.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>stem.FCAI_Result</td>
<td>Decimal</td>
<td>This stem variable corresponds to the FCAI_Result field in the FCAI_Map control block.</td>
<td>See Table 15 on page 408.</td>
</tr>
<tr>
<td>stem.FCAI_IE</td>
<td>Decimal</td>
<td>This stem variable corresponds to the FCAI_IE field in the FCAI_Map control block.</td>
<td>See Table 16 on page 408.</td>
</tr>
<tr>
<td>stem.FCAI_CEC</td>
<td>Decimal</td>
<td>This stem variable corresponds to the FCAI_CEC field in the FCAI_Map control block.</td>
<td>See FTP return codes in the z/OS Communications Server: IP User's Guide and Commands.</td>
</tr>
<tr>
<td>stem.FCAI_ReplyCode</td>
<td>Decimal</td>
<td>This stem variable corresponds to the FCAI_ReplyCode in the FCAI_Map control block.</td>
<td>See FTPD reply codes in z/OS Communications Server: IP and SNA Codes.</td>
</tr>
<tr>
<td>stem.FCAI_TraceStatus</td>
<td>Decimal</td>
<td>This stem variable corresponds to the FCAI_TraceStatus field in the FCAI_Map control block. It can be used to determine whether the last FTP client API trace succeeded or failed.</td>
<td>See Table 13 on page 407.</td>
</tr>
<tr>
<td>stem.FCAI_ReturnCode</td>
<td>Decimal</td>
<td>This stem variable corresponds to the FCAI_ReturnCode field in the FCAI_Map control block.</td>
<td>See Table 13 on page 407 and Table 16 on page 408.</td>
</tr>
</tbody>
</table>
Table 17. FCAI stem variables (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>stem.FCAI_ReasonCode</td>
<td>Decimal</td>
<td>This stem variable corresponds to the FCAI_ReasonCode field in the FCAI_Map control block.</td>
<td>See Table 13 on page 407 and Table 16 on page 408.</td>
</tr>
<tr>
<td>stem.FCAI_SCMD</td>
<td>Decimal</td>
<td>This stem variable corresponds to the FCAI_SCMD field in the FCAI_Map control block.</td>
<td>See FTP subcommand codes in the z/OS Communications Server: IP User’s Guide and Commands</td>
</tr>
<tr>
<td>stem.FCAI_Map</td>
<td>Binary</td>
<td>This stem variable contains a binary representation of the FCAI_Map control block, plus additional fields used by the FTP client API for REXX.</td>
<td>This stem variable must not be modified by the REXX program.</td>
</tr>
</tbody>
</table>

Predefined REXX variables

Predefined REXX variables make symbolic references easier and more consistent. Instead of using a numeric or non-numeric value, you can use the predefined variable, which defines that value for you. Table 18 shows the data type and value for each predefined variable. The predefined variables are created on the first CREATE request issued by a REXX program.

Requirement: The REXX program must treat the predefined variables as read only, and must not assign any values to them.

The predefined variables, listed alphabetically, are shown in Table 18.

Table 18. Predefined REXX variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCAL_CEC_ALREADY_CONNECTED</td>
<td>Decimal</td>
<td>6</td>
<td>Can be stored in stem.FCAI_CEC</td>
</tr>
<tr>
<td>FCAL_CEC_AUTHENTICATION</td>
<td>Decimal</td>
<td>17</td>
<td>Can be stored in stem.FCAI_CEC</td>
</tr>
<tr>
<td>FCAL_CEC_CLIENT_ERR</td>
<td>Decimal</td>
<td>24</td>
<td>Can be stored in stem.FCAI_CEC</td>
</tr>
<tr>
<td>FCAL_CEC_CONNECT_FAILED</td>
<td>Decimal</td>
<td>8</td>
<td>Can be stored in stem.FCAI_CEC</td>
</tr>
<tr>
<td>FCAL_CEC_CONVERSION</td>
<td>Decimal</td>
<td>21</td>
<td>Can be stored in stem.FCAI_CEC</td>
</tr>
<tr>
<td>FCAL_CEC_EOD BEFORE_EOF</td>
<td>Decimal</td>
<td>25</td>
<td>Can be stored in stem.FCAI_CEC</td>
</tr>
<tr>
<td>FCAL_CEC_FILE_ACCESS</td>
<td>Decimal</td>
<td>18</td>
<td>Can be stored in stem.FCAI_CEC</td>
</tr>
<tr>
<td>FCAL_CEC_FILE_READ</td>
<td>Decimal</td>
<td>19</td>
<td>Can be stored in stem.FCAI_CEC</td>
</tr>
<tr>
<td>FCAL_CEC_FILE_WRITE</td>
<td>Decimal</td>
<td>20</td>
<td>Can be stored in stem.FCAI_CEC</td>
</tr>
<tr>
<td>FCAL_CEC_INPUT_ERR</td>
<td>Decimal</td>
<td>12</td>
<td>Can be stored in stem.FCAI_CEC</td>
</tr>
<tr>
<td>FCAL_CEC_INTERNAL_ERROR</td>
<td>Decimal</td>
<td>1</td>
<td>Can be stored in stem.FCAI_CEC</td>
</tr>
<tr>
<td>FCAL_CEC_INVALID_ENVIRONMENT</td>
<td>Decimal</td>
<td>15</td>
<td>Can be stored in stem.FCAI_CEC</td>
</tr>
<tr>
<td>FCAL_CEC_INVALID_PARAM</td>
<td>Decimal</td>
<td>4</td>
<td>Can be stored in stem.FCAI_CEC</td>
</tr>
<tr>
<td>FCAL_CEC_LOGIN_FAILED</td>
<td>Decimal</td>
<td>11</td>
<td>Can be stored in stem.FCAI_CEC</td>
</tr>
<tr>
<td>FCAL_CEC_NEEDS_CONNECTION</td>
<td>Decimal</td>
<td>26</td>
<td>Can be stored in stem.FCAI_CEC</td>
</tr>
<tr>
<td>FCAL_CEC_NOT_ENABLED</td>
<td>Decimal</td>
<td>16</td>
<td>Can be stored in stem.FCAI_CEC</td>
</tr>
<tr>
<td>FCAL_CEC_NOTFOUND</td>
<td>Decimal</td>
<td>14</td>
<td>Can be stored in stem.FCAI_CEC</td>
</tr>
<tr>
<td>FCAL_CEC_OPEN_IOSTREAM_FAILED</td>
<td>Decimal</td>
<td>5</td>
<td>Can be stored in stem.FCAI_CEC</td>
</tr>
<tr>
<td>FCAL_CEC_PROXY_ERR</td>
<td>Decimal</td>
<td>22</td>
<td>Can be stored in stem.FCAI_CEC</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Description</td>
<td>Additional Information</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------</td>
<td>-------------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>FCAI_CEC_SERVER_ERROR</td>
<td>Decimal</td>
<td>2</td>
<td>Can be stored in stem.FCAI_CEC</td>
</tr>
<tr>
<td>FCAI_CEC_SESSION_ERROR</td>
<td>Decimal</td>
<td>10</td>
<td>Can be stored in stem.FCAI_CEC</td>
</tr>
<tr>
<td>FCAI_CEC_SQL_ERR</td>
<td>Decimal</td>
<td>23</td>
<td>Can be stored in stem.FCAI_CEC</td>
</tr>
<tr>
<td>FCAI_CEC_TIMEOUT</td>
<td>Decimal</td>
<td>9</td>
<td>Can be stored in stem.FCAI_CEC</td>
</tr>
<tr>
<td>FCAI_CEC_USAGE</td>
<td>Decimal</td>
<td>7</td>
<td>Can be stored in stem.FCAI_CEC</td>
</tr>
<tr>
<td>FCAI_ERROR_CEC</td>
<td>Decimal</td>
<td>-3</td>
<td>Can be returned by FTPAPI call</td>
</tr>
<tr>
<td>FCAI_ERROR_IE</td>
<td>Decimal</td>
<td>-2</td>
<td>Can be returned by FTPAPI call</td>
</tr>
<tr>
<td>FCAI_GETL_ANY_LINE</td>
<td>Char</td>
<td>A</td>
<td>Value for type parameter on GETL_FIND or GETL_COPY requests</td>
</tr>
<tr>
<td>FCAI_GETL_FIND_FIRST</td>
<td>Char</td>
<td>F</td>
<td>Value for sequence parameter on GETL_FIND requests</td>
</tr>
<tr>
<td>FCAI_GETL_FIND_LAST</td>
<td>Char</td>
<td>L</td>
<td>Value for sequence parameter on GETL_FIND requests</td>
</tr>
<tr>
<td>FCAI_GETL_FIND_NEXT</td>
<td>Char</td>
<td>N</td>
<td>Value for sequence parameter on GETL_FIND requests</td>
</tr>
<tr>
<td>FCAI_GETL_LIST_LINE</td>
<td>Char</td>
<td>L</td>
<td>Value for sequence parameter on GETL_FIND requests</td>
</tr>
<tr>
<td>FCAI_GETL_MESSAGE_LINE</td>
<td>Char</td>
<td>M</td>
<td>Value for type parameter on GETL_FIND or GETL_COPY requests</td>
</tr>
<tr>
<td>FCAI_GETL_REPLY_LINE</td>
<td>Char</td>
<td>R</td>
<td>Value for type parameter on GETL_FIND or GETL_COPY requests</td>
</tr>
<tr>
<td>FCAI_GETL_TRACE_LINE</td>
<td>Char</td>
<td>T</td>
<td>Value for type parameter on GETL_FIND or GETL_COPY requests</td>
</tr>
<tr>
<td>FCAI_IE_ACCTPROMPTERR</td>
<td>Decimal</td>
<td>35</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_ALREADYINPROGRESS</td>
<td>Decimal</td>
<td>37</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_APIALREADYINIT</td>
<td>Decimal</td>
<td>16</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_APILOADFAILED</td>
<td>Decimal</td>
<td>18</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_BADTOKENADDR</td>
<td>Decimal</td>
<td>20</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_BUFFERTOOMALL</td>
<td>Decimal</td>
<td>68</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_CLIPROCESSBROKEN</td>
<td>Decimal</td>
<td>47</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_CLIPROCESSSTOPPED</td>
<td>Decimal</td>
<td>38</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_CONTROLERR</td>
<td>Decimal</td>
<td>6</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_CREATETPIPEERR</td>
<td>Decimal</td>
<td>26</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_ENVVARSSTORAGEERR</td>
<td>Decimal</td>
<td>77</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_GETWORKAREAFALIED</td>
<td>Decimal</td>
<td>21</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_INITPARMTOOBIG</td>
<td>Decimal</td>
<td>17</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_INTERNALERR</td>
<td>Decimal</td>
<td>7</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_LENGTHININVALID</td>
<td>Decimal</td>
<td>8</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_LINESPARMTOOBIG</td>
<td>Decimal</td>
<td>73</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_NOTINPROGRESS</td>
<td>Decimal</td>
<td>48</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_NOTOKENADDR</td>
<td>Decimal</td>
<td>19</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_PARMMISSING</td>
<td>Decimal</td>
<td>3</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_PARMSTOREAGEERR</td>
<td>Decimal</td>
<td>4</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_PASSPROMPTERR</td>
<td>Decimal</td>
<td>34</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_POLLWAITINVALID</td>
<td>Decimal</td>
<td>74</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_READERR</td>
<td>Decimal</td>
<td>42</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_REQTIEREXPIRED</td>
<td>Decimal</td>
<td>22</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_REQTIMERRIVALD</td>
<td>Decimal</td>
<td>72</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_REQUESTMISSING</td>
<td>Decimal</td>
<td>1</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_REQUESTUNKNOWN</td>
<td>Decimal</td>
<td>2</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_SCMDPARMTOOBIG</td>
<td>Decimal</td>
<td>32</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_SPAWNERR</td>
<td>Decimal</td>
<td>27</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_SYSOUTCLASSERR</td>
<td>Decimal</td>
<td>78</td>
<td>Can be stored in fcaiMap.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_TOOMANYENVVARS</td>
<td>Decimal</td>
<td>24</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_TOOMANYINITPARMS</td>
<td>Decimal</td>
<td>23</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>FCAI_IE_TOOMANYPARAMETERS</td>
<td>Decimal</td>
<td>5</td>
<td>Can be stored in stem.FCAI_IE</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Description</td>
<td>Additional Information</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------</td>
<td>------------------------------------------</td>
<td>-------------------------------------------------------------</td>
</tr>
<tr>
<td>FCAL_IE_TRACEIDTOOBIG</td>
<td>Decimal 69</td>
<td>Can be stored in stem.FCAL_IE</td>
<td></td>
</tr>
<tr>
<td>FCAL_IE_TRACESCLASSTOOBIG</td>
<td>Decimal 70</td>
<td>Can be stored in stem.FCAL_IE</td>
<td></td>
</tr>
<tr>
<td>FCAL_IE_UNKMODE</td>
<td>Decimal 33</td>
<td>Can be stored in stem.FCAL_IE</td>
<td></td>
</tr>
<tr>
<td>FCAL_IE_UNKNOWNNOPERATION</td>
<td>Decimal 64</td>
<td>Can be stored in stem.FCAL_IE</td>
<td></td>
</tr>
<tr>
<td>FCAL_IE_UNKNOWNSEQUENCE</td>
<td>Decimal 66</td>
<td>Can be stored in stem.FCAL_IE</td>
<td></td>
</tr>
<tr>
<td>FCAL_IE_UNKNOWNTRACEIT</td>
<td>Decimal 71</td>
<td>Can be stored in stem.FCAL_IE</td>
<td></td>
</tr>
<tr>
<td>FCAL_IE_UNKNOWNTYPE</td>
<td>Decimal 65</td>
<td>Can be stored in stem.FCAL_IE</td>
<td></td>
</tr>
<tr>
<td>FCAL_IE_VECTORSTORAGEERR</td>
<td>Decimal 67</td>
<td>Can be stored in stem.FCAL_IE</td>
<td></td>
</tr>
<tr>
<td>FCAL_IE_WRITEERR</td>
<td>Decimal 41</td>
<td>Can be stored in stem.FCAL_IE</td>
<td></td>
</tr>
<tr>
<td>FCAL_MODE_NOWAIT</td>
<td>Char N</td>
<td>Can be stored in stem.FCAL_IE</td>
<td></td>
</tr>
<tr>
<td>FCAL_MODE_WAIT</td>
<td>Char W</td>
<td>Can be stored in stem.FCAL_IE</td>
<td></td>
</tr>
<tr>
<td>FCAL_RESULT_CEC</td>
<td>Decimal -3</td>
<td>Can be returned by an FTPAPI call</td>
<td></td>
</tr>
<tr>
<td>FCAL_RESULT_IE</td>
<td>Decimal -2</td>
<td>Can be returned by an FTPAPI call</td>
<td></td>
</tr>
<tr>
<td>FCAL_RESULT_INPROGRESS</td>
<td>Decimal 1</td>
<td>Can be stored in stem.FCAL_IE</td>
<td></td>
</tr>
<tr>
<td>FCAL_RESULT_NOMATCH</td>
<td>Decimal 4</td>
<td>Can be stored in stem.FCAL_Result or can be returned by an FTPAPI call</td>
<td></td>
</tr>
<tr>
<td>FCAL_RESULT_OK</td>
<td>Decimal 0</td>
<td>Can be stored in stem.FCAL_Result or returned by an FTPAPI call</td>
<td></td>
</tr>
<tr>
<td>FCAL_RESULT_PROMPTACCT</td>
<td>Decimal 3</td>
<td>Can be stored in stem.FCAL_Result or returned by an FTPAPI call</td>
<td></td>
</tr>
<tr>
<td>FCAL_RESULT_PROMPTPASS</td>
<td>Decimal 2</td>
<td>Can be stored in stem.FCAL_Result or returned by an FTPAPI call</td>
<td></td>
</tr>
<tr>
<td>FCAL_RESULT_REXXERROR</td>
<td>Decimal -19</td>
<td>Can be returned by an FTPAPI call</td>
<td></td>
</tr>
<tr>
<td>FCAL_RESULT_USBABLEFCAI</td>
<td>Decimal -17</td>
<td>Can be returned by an FTPAPI call</td>
<td></td>
</tr>
<tr>
<td>FCAL_SCMD_ACCT</td>
<td>Decimal 3</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAI_MAP request</td>
<td></td>
</tr>
<tr>
<td>FCAL_SCMD_AMBIGUOUS</td>
<td>Decimal 1</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAI_MAP request</td>
<td></td>
</tr>
<tr>
<td>FCAL_SCMD_APP</td>
<td>Decimal 4</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAI_MAP request</td>
<td></td>
</tr>
<tr>
<td>FCAL_SCMD_ASCII</td>
<td>Decimal 5</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAI_MAP request</td>
<td></td>
</tr>
<tr>
<td>FCAL_SCMD_BIG5</td>
<td>Decimal 57</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAI_MAP request</td>
<td></td>
</tr>
<tr>
<td>FCAL_SCMD_BINARY</td>
<td>Decimal 6</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAI_MAP request</td>
<td></td>
</tr>
<tr>
<td>FCAL_SCMD_BLOCK</td>
<td>Decimal 58</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAI_MAP request</td>
<td></td>
</tr>
<tr>
<td>FCAL_SCMD_CCC</td>
<td>Decimal 77</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAI_MAP request</td>
<td></td>
</tr>
<tr>
<td>FCAL_SCMD_CD</td>
<td>Decimal 7</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAI_MAP request</td>
<td></td>
</tr>
<tr>
<td>FCAL_SCMD_CDUP</td>
<td>Decimal 51</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAI_MAP request</td>
<td></td>
</tr>
<tr>
<td>FCAL_SCMD_CLEAR</td>
<td>Decimal 72</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAI_MAP request</td>
<td></td>
</tr>
<tr>
<td>FCAL_SCMD_CLOSE</td>
<td>Decimal 8</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAI_MAP request</td>
<td></td>
</tr>
<tr>
<td>FCAL_SCMD_COMPRESS</td>
<td>Decimal 59</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAI_MAP request</td>
<td></td>
</tr>
<tr>
<td>FCAL_SCMD_CPPROTECT</td>
<td>Decimal 73</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAI_MAP request</td>
<td></td>
</tr>
<tr>
<td>FCAL_SCMD_DEBUG</td>
<td>Decimal 11</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAI_MAP request</td>
<td></td>
</tr>
<tr>
<td>FCAL_SCMD_DELE</td>
<td>Decimal 13</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAI_MAP request</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Description</td>
<td>Additional Information</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------</td>
<td>-------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>FCAL_SCMD_DELIMIT</td>
<td>Decimal</td>
<td>12</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_DIR</td>
<td>Decimal</td>
<td>14</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_DUMP</td>
<td>Decimal</td>
<td>70</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_EBCDIC</td>
<td>Decimal</td>
<td>15</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_EUCKANJI</td>
<td>Decimal</td>
<td>46</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_FEAT</td>
<td>Decimal</td>
<td>78</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_FILE</td>
<td>Decimal</td>
<td>60</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_GENHELP</td>
<td>Decimal</td>
<td>2</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_GET</td>
<td>Decimal</td>
<td>16</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_GLOB</td>
<td>Decimal</td>
<td>65</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_HANGEUL</td>
<td>Decimal</td>
<td>53</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_HELP</td>
<td>Decimal</td>
<td>17</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_IBMKANJI</td>
<td>Decimal</td>
<td>47</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_JIS78kJ</td>
<td>Decimal</td>
<td>48</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_JIS83kJ</td>
<td>Decimal</td>
<td>49</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_KSC5601</td>
<td>Decimal</td>
<td>54</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_LANG</td>
<td>Decimal</td>
<td>79</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_LCD</td>
<td>Decimal</td>
<td>41</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_LMKDIR</td>
<td>Decimal</td>
<td>45</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_LOCSITE</td>
<td>Decimal</td>
<td>42</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_LOSTAT</td>
<td>Decimal</td>
<td>18</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_LPWD</td>
<td>Decimal</td>
<td>43</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_LS</td>
<td>Decimal</td>
<td>20</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_MDELETE</td>
<td>Decimal</td>
<td>21</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_MGET</td>
<td>Decimal</td>
<td>22</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_MKD</td>
<td>Decimal</td>
<td>44</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_MKFIFO</td>
<td>Decimal</td>
<td>82</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_MODE</td>
<td>Decimal</td>
<td>23</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_MPUT</td>
<td>Decimal</td>
<td>24</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>FCAL_SCMD_NOOP</td>
<td>Decimal</td>
<td>25</td>
<td>Can be stored in fcaiMap.FCAL_SCMD by a GET_FCAL_MAP request</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Description</td>
<td>Additional Information</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------</td>
<td>----------------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>FCAI_SCMD_OEEXCL</td>
<td>Decimal</td>
<td>68</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_OPEN</td>
<td>Decimal</td>
<td>10</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_PASS</td>
<td>Decimal</td>
<td>26</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_PRIVATE</td>
<td>Decimal</td>
<td>74</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_PROMPT</td>
<td>Decimal</td>
<td>66</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_PROTECT</td>
<td>Decimal</td>
<td>75</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_PROXY</td>
<td>Decimal</td>
<td>61</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_PUT</td>
<td>Decimal</td>
<td>27</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_PWD</td>
<td>Decimal</td>
<td>28</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_QUIT</td>
<td>Decimal</td>
<td>29</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_QUOTE</td>
<td>Decimal</td>
<td>30</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_RECORD</td>
<td>Decimal</td>
<td>62</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_RENAME</td>
<td>Decimal</td>
<td>31</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_RLIST</td>
<td>Decimal</td>
<td>56</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_RMD</td>
<td>Decimal</td>
<td>52</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_SAFE</td>
<td>Decimal</td>
<td>76</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_SCHINESE</td>
<td>Decimal</td>
<td>63</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_SENDPORT</td>
<td>Decimal</td>
<td>32</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_SENDSITE</td>
<td>Decimal</td>
<td>33</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_SITE</td>
<td>Decimal</td>
<td>34</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_SJSJK</td>
<td>Decimal</td>
<td>50</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_SREST</td>
<td>Decimal</td>
<td>80</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_STAT</td>
<td>Decimal</td>
<td>35</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_STREAM</td>
<td>Decimal</td>
<td>64</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_STRU</td>
<td>Decimal</td>
<td>36</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_SUNIQUE</td>
<td>Decimal</td>
<td>37</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_SYST</td>
<td>Decimal</td>
<td>38</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_TCHINESE</td>
<td>Decimal</td>
<td>55</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_TSO</td>
<td>Decimal</td>
<td>9</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_TYPE</td>
<td>Decimal</td>
<td>40</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
</tbody>
</table>
### Table 18. Predefined REXX variables (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCAI_SCMD_UCS2</td>
<td>Decimal</td>
<td>67</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_UNKNOWN</td>
<td>Decimal</td>
<td>99</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_USER</td>
<td>Decimal</td>
<td>19</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_SCMD_VERBOSE</td>
<td>Decimal</td>
<td>71</td>
<td>Can be stored in fcaiMap.FCAI_SCMD by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_STATUS_INPROGRESS</td>
<td>Decimal</td>
<td>1</td>
<td>Can be returned by an FTPAPI call</td>
</tr>
<tr>
<td>FCAI_STATUS_PROMPTACCT</td>
<td>Decimal</td>
<td>3</td>
<td>Can be returned by an FTPAPI call</td>
</tr>
<tr>
<td>FCAI_STATUS_PROMPTPASS</td>
<td>Decimal</td>
<td>2</td>
<td>Can be returned by an FTPAPI call</td>
</tr>
<tr>
<td>FCAI_STATUS_TRACEFAILED</td>
<td>Decimal</td>
<td>200</td>
<td>Can be returned by an FTPAPI call</td>
</tr>
<tr>
<td>FCAI_TASK_CLIPROCESSKILL</td>
<td>Decimal</td>
<td>-32</td>
<td>Can be stored in stem.FCAI_Result or returned by an FTPAPI function call</td>
</tr>
<tr>
<td>FCAI_TASK_TASKMISMATCH</td>
<td>Decimal</td>
<td>-18</td>
<td>Can be stored in stem.FCAI_Result or returned by an FTPAPI function call</td>
</tr>
<tr>
<td>FCAI_TRACE_DATASET_NAME</td>
<td>String</td>
<td></td>
<td>Data set name to which the REXX FTP Client trace is written if the FTP client API for REXX is active, or an empty string (&quot;&quot;) if the trace is not active.</td>
</tr>
<tr>
<td>FCAI_TRACE_DATASET_RETCODE</td>
<td>Integer</td>
<td></td>
<td>Return code from FTP client API for REXX tracing from the last invocation of the FTP client API for REXX</td>
</tr>
<tr>
<td>FCAI_TRACE_WORKAREA</td>
<td>Binary</td>
<td></td>
<td>Binary data used by the FTP client API for REXX when writing trace records.</td>
</tr>
<tr>
<td>Rule:</td>
<td></td>
<td></td>
<td>This variable must not be modified by the REXX program.</td>
</tr>
<tr>
<td>FCAI_TRACECAPI_A</td>
<td>Decimal</td>
<td>1</td>
<td>Can be stored in fcaiMap.FCAI_TraceCAPI by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_TRACECAPI_C</td>
<td>Decimal</td>
<td>0</td>
<td>Can be stored in fcaiMap.FCAI_TraceCAPI by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_TRACECAPI_N</td>
<td>Decimal</td>
<td>2</td>
<td>Can be stored in fcaiMap.FCAI_TraceCAPI by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_TRACESTATUS_ALLOCERR</td>
<td>Decimal</td>
<td>2</td>
<td>Can be stored in fcaiMap.FCAI_TraceStatus by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_TRACESTATUS_CLOSEERR</td>
<td>Decimal</td>
<td>5</td>
<td>Can be stored in fcaiMap.FCAI_TraceStatus by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_TRACESTATUS_OK</td>
<td>Decimal</td>
<td>0</td>
<td>Can be stored in fcaiMap.FCAI_TraceStatus by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_TRACESTATUS_OPENERR</td>
<td>Decimal</td>
<td>3</td>
<td>Can be stored in fcaiMap.FCAI_TraceStatus by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_TRACESTATUS_STORAGEERR</td>
<td>Decimal</td>
<td>1</td>
<td>Can be stored in fcaiMap.FCAI_TraceStatus by a GET_FCAI_MAP request</td>
</tr>
<tr>
<td>FCAI_TRACESTATUS_WRITEERR</td>
<td>Decimal</td>
<td>4</td>
<td>Can be stored in fcaiMap.FCAI_TraceStatus by a GET_FCAI_MAP request</td>
</tr>
</tbody>
</table>

### Sending requests to the FTP client API

This topic contains the description, syntax, parameters, and other related information for each of the following requests submitted to this API:

- INIT
- SCMD
- POLL
- GETL
- TERM

**Note:** The text in this topic sometimes uses shorthand when referring to results. When a request is said to return FCAI_IE_LengthInvalid, for example, it
means that FCAI_Result contains FCAI_Result_IE and FCAI_IE contains FCAI_IE_LengthInvalid. See “Interpreting results from an interface request” on page 464.

Tip: This topic provides information on the Cobol, C, assembler and PL/I programming languages. For REXX programs, see “FTP client API for REXX function” on page 439.

INIT

The user program issues the INIT request to initialize the FTP client API. This call is the first call to the interface and is made only one time for each FCAI control block that defines an instance of use of the interface.

Rules:

- Align the FCAI on at least a fullword boundary.
- The FCAI must reside in primary space (not a dataspace).
- Initialize the FCAI by performing the steps listed in step 3 in “Application tasks for the INIT request” on page 421.
- The caller can specify the number of seconds to wait by setting an FCAI_ReqTimer value (or specify that no timer is to be used). See “FCAI_ReqTimer: Controlling requests that retrieve results from the spawned z/OS FTP client process” on page 468.
- Before exiting, your application should issue a TERM request for each INIT request.

Example of the INIT call instruction

WORKING-STORAGE SECTION.

COPY EZAFTPKC.
01 REQUEST-INIT PIC X(4) VALUE IS 'INIT'.
01 START-Parm.
   05 PARM-LEN PIC 9(2) COMP-5 VALUE IS 6.
   05 PARM-VAL PIC X(6) VALUE IS '-n tls'.

01 ENV-VAR-LIST.
   05 ENV-VAR-COUNT PIC 9(8) COMP-5 VALUE IS 3.
   05 ENV-VAR1-LEN PIC 9(8) COMP-5.
   05 ENV-VAR2-LEN PIC 9(8) COMP-5.
   05 ENV-VAR3-LEN PIC 9(8) COMP-5.
   05 ENV-VAR1-P USAGE IS POINTER.
   05 ENV-VAR2-P USAGE IS POINTER.
   05 ENV-VAR3-P USAGE IS POINTER.

01 ENV-VAR-VALUES.
   05 ENV-VAR1.
      10 ENV-VAR PIC X(17) VALUE IS '_CEE_DMPTARG=/etc'.
      10 FILLER PIC X(1) VALUE LOW-VALUES.
   05 ENV-VAR2.
      10 ENV-VAR PIC X(18) VALUE IS '_BPX_JOBNAME=MYJOB'.
      10 FILLER PIC X(1) VALUE LOW-VALUES.
   05 ENV-VAR3.
      10 ENV-VAR PIC X(20) VALUE IS 'NLSPATH=/u/user79/%N'.
      10 FILLER PIC X(1) VALUE LOW-VALUES.

PROCEDURE DIVISION.

MOVE LOW-VALUES TO FCAI-Map.
MOVE FCAI-C-Eycatcher TO FCAI-Eycatcher.
MOVE FCAI-C-Version-1 TO FCAI-Version.
MOVE LENGTH OF FCAI-Map TO FCAI-Size.

MOVE LENGTH OF ENV-VAR1 TO ENV-VAR1-LEN.
SET ENV-VAR1-P TO ADDRESS OF ENV-VAR1.
MOVE LENGTH OF ENV-VAR2 TO ENV-VAR2-LEN.
SET ENV-VAR2-P TO ADDRESS OF ENV-VAR2.
MOVE LENGTH OF ENV-VAR3 TO ENV-VAR3-LEN.
SET ENV-VAR3-P TO ADDRESS OF ENV-VAR3.

CALL 'EZAFTPKS' USING FCAI-Map REQUEST-INIT START-PARM ENV-VAR-LIST.

Since both START-PARM and ENV-VAR-LIST are optional,
use OMITTED for START-PARM if it is not to be passed on
a CALL that passes ENV-VAR-LIST:

CALL 'EZAFTPKS' USING FCAI-Map REQUEST-INIT OMITTED ENV-VAR-LIST.

For equivalent PL/I and assembler language declarations, see “Converting
parameter descriptions” on page 402.

Parameter values that are set by the application

FCAI-Map
Storage area (defined in EZAFTPKC for COBOL) used to save information
about the requests using the FTP client API.

REQUEST-TYPE
A 4-byte field that contains INIT.

START-PARM
An optional parameter that comprises a 2-byte length followed by a string
that contains parameters that are valid to enter on a z/OS FTP command.

PARM-LEN
A 2-byte binary field that contains the length of PARM-VAL.

PARM-VAL
Storage containing the parameters for the z/OS FTP command. See
the FTP subcommands information in the z/OS Communications
Server: IP User's Guide and Commands for the parameters that are
valid when using the FTP command.

Requirement: Align START-PARM on at least a halfword boundary.

Tip: Start parameters that include a host name or IP address cause the client to
perform an implicit OPEN to connect to that host, which suspends the application
until the connection is complete. If this delay is undesirable during INIT, use a
subsequent SCMD OPEN instead of specifying a host in the start parameters.

ENV-VAR-LIST
An optional parameter that comprises a series of contiguous fullwords (4
bytes each) that are used to describe environment variables that are passed
on the spawn of the FTP Client.

See the following for information about FTP environment variables:
• Defining environment variables for the FTP server (optional) in the z/OS
Communications Server: IP Configuration Guide
• FTP server environment variables in z/OS Communications Server: IP
Configuration Reference
• Environment variables in z/OS Communications Server: IP User's Guide and
Commands
Also see the z/OS UNIX System Services and z/OS Language Environment libraries of publications for information concerning environment variables.

**ENV-VAR-COUNT**
Count of environment variables (n) to be passed. There can be 1–9 environment variables.

**ENV-VAR1_LEN through ENV-VARn-LEN (maximum of nine)**
The fullword length of each environment variable found in ENV-VAR1 through ENV-VARn.

**ENV-VAR1_P through ENV-VARn-P (maximum of nine)**
The address of each environment variable found in ENV-VAR1 through ENV-VARn.

Rules:
- Each environment variable passed in the ENV-VAR-LIST must be a NULL terminated string; that is, X'00' follows the last text character. Set the corresponding ENV-VARn-LEN field to the length of the environment variable text, plus 1 for the NULL terminator.
- Ensure that no duplicate environment variables are specified.
- Do not pass environment variable _CEE_RUNOPTS on INIT. The environment variables are established too late in the spawn() process for run-time options to be honored. See the z/OS Language Environment Programming Guide for information about using CEEDOPT or CEEBXITA to specify run-time options for the FTP client process.
- Do not pass environment variables in the _CEE_ENVFILE file that are required to be used in the spawn() process. The _CEE_ENVFILE variable is processed after the spawn() processing is complete. Variables like _BPX_JOBNAME should be specified as one of the nine environment variables in the ENV-VAR-LIST.
- Run-time options and environment variables that are specified on the EXEC statement or in CEEUOPT for the application program are not available to the spawned FTP client process.
- Align ENV-VAR-LIST on at least a fullword boundary.

**Parameter values that are returned to the application**
The results of the request are returned in the FCAI-Result field. See “Interpreting results from an interface request” on page 464. See “FTP client API: Other output that is returned to the application” on page 461 for a discussion of the output and statistics returned by the request.

**Guidelines for INIT results:**
- If the INIT request returns FCAI-IE-CliProcessBroken, check FCAI-EC for a client error code that might have been returned to explain the failure.
- FCAI-Token is zeros when the interface fails to initialize.

**Application tasks for the INIT request**
**Before you begin:** Create an FCAI control block for use by the interface.

Perform the following steps to issue the INIT request:
1. Specify start parameters for the FTP client (optional).
2. Specify a list of environment variables to pass to the FTP client (optional).
3. Initialize the FCAI.
   a. Clear the entire block to 0.
   b. Set the eyecatcher (or FCAI-Eyecatcher) to FCAI.
   c. Set the size field (or FCAI-Size) to 256 or greater.
   d. Set the version number (or FCAI-Version) to 1.

4. Set FCAI-ReqTimer to the desired value.

5. Set FCAI-Tracelt as desired for tracing. A request that initiates the interface trace also uses FCAI-TraceID and FCAI-TraceSClass.

6. Issue the INIT request.

7. Check the result of the request.

8. Check the results from an implicit OPEN, if one was performed.

9. Retrieve the FTP client output with GETL (optional).

After the INIT request completes, you can issue a subsequent SCMD or TERM request.

**SCMD**

The user program issues the SCMD request to send a subcommand to the FTP client.

**Example of the SCMD call instruction**

```plaintext
WORKING-STORAGE SECTION.

COPY EZAFTPKC.
01 REQUEST-SCMD PIC X(4) VALUE IS 'SCMD'.
01 SCMD-MODE-W PIC X(1) VALUE IS 'W'.
01 SCMD-MODE-N PIC X(1) VALUE IS 'N'.
01 SUBCOMMAND.
   05 SUBCOMMAND-LEN PIC 9(2) COMP-5 VALUE IS 13.
   05 SUBCOMMAND-VAL PIC X(13) VALUE IS 'open hostname'.

PROCEDURE DIVISION.

CALL 'EZAFTPKS' USING FCAI-Map REQUEST-SCMD SUBCOMMAND
SCMD-MODE-W.
```

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 402.
Parameter values that are set by the application

FCAI-Map
Storage area (defined in EZAFTPKC for COBOL) used to save information about the requests using the FTP client API.

REQUEST-TYPE
A 4-byte field that contains SCMD.

SUBCOMMAND
A required parameter that comprises a 2-byte field, followed by a string with a z/OS FTP subcommand. See the introductory material at the beginning of this topic for a discussion of FTP subcommands and commands.

SUBCOMMAND-LEN
A 2-byte binary field that contains the length of SUBCOMMAND-VAL.

SUBCOMMAND-VAL
Storage that contains a z/OS FTP subcommand. Leading spaces are not allowed; begin the subcommand in the first column of this storage area. A null terminator is unnecessary but is accepted if included. See [FTP subcommands] in the z/OS Communications Server: IP User's Guide and Commands for the supported subcommands and their parameters.

Rule: Align SUBCOMMAND on at least a halfword boundary.

MODE
An optional parameter that indicates whether the interface should wait for the subcommand to complete before returning to the caller. If the parameter is not present, the default is to wait. The parameter is a 1-byte field that contains W for wait mode or N for no-wait mode.

An SCMD issued in no-wait mode returns to the caller as soon as the interface sends the subcommand. Issue a POLL request to retrieve the results from the subcommand. If you choose no-wait mode, you cannot start a new SCMD request with this FCAI until you execute a POLL against this FCAI that indicates that the outstanding subcommand is complete.

An SCMD issued in wait mode essentially polls the client for you until the subcommand completes. The automatic polling operates similarly to the [progressive wait timer described in "FCAI_PollWait: Specifying a wait time before POLL" on page 468] It polls initially after 1 second and then doubles the wait time before each read to a maximum interval of 16 seconds until data is returned or the subcommand completes.

If you choose wait mode, you can limit the length of time the interface waits for completion by using FCAI-ReqTimer. See [FCAI_ReqTimer: Controlling requests that retrieve results from the spawned z/OS FTP client process] on page 468. If FCAI-ReqTimer causes the subcommand to be interrupted, a POLL is required to complete and retrieve the results of the subcommand before another subcommand can be issued.

Processing rules for SCMD QUIT:
• The MODE parameter is ignored for SCMD QUIT. The QUIT subcommand is always issued in wait mode.
The interface accepts a TERM and generates a successful SCMD QUIT even when the interface does not accept an explicit SCMD QUIT request. See "TERM" on page 433 and "Prompts from the client" on page 462 for more information.

After SCMD QUIT has successfully completed, only GETL and TERM are accepted for this FCAI.

Failure to issue SCMD QUIT can strand the client process. See "TERM" on page 433 for tips on preventing stranded clients.

SCMD and POLL requests between SCMD QUIT and TERM return the interface error FCAI_IE_CliProcessStopped.

Parameter values that are returned to the application

The results of the request are returned in the FCAI-Result field; see "Interpreting results from an interface request" on page 464. See "FTP client API: Other output that is returned to the application" on page 461 for a description of the output and statistics that are returned by the request.

Guidelines for SCMD results:

- If FCAI-Result contains FCAI-Result-Status, additional processing is required by the user program:
  - If the FCAI-Status field is FCAI-Status-InProgress, the request was issued in no-wait mode or the FCAI-ReqTimer value was exceeded. Issue a POLL request to obtain the final results.
  - If the FCAI-Status field is FCAI-Status-PromptPass, the request was accepted but the next SCMD request must be a PASS subcommand. This status is applicable for the USER subcommand.
  - If the FCAI-Status field is FCAI-Status-PromptAcct, the request was accepted, but the next SCMD request must be an ACCT subcommand. This status is applicable for the USER, PASS, and CD subcommands.
  - If FCAI-Status contains FCAI-Status-TraceFailed or higher, the interface trace function failed on this request. Subtract FCAI-Status-TraceFailed from FCAI-Status to obtain the remaining value for FCAI-Status. See "FCAI_Status_TraceFailed and FCAI_TraceStatus: Reporting failures in the interface trace function" on page 467.

- After an INIT that results in an implicit OPEN, FCAI-SCMD contains FCAI-SCMD-OPEN, and FCAI-ReplyCode contains the last reply from the server.

See "Prompts from the client" on page 462 for a discussion of how prompts are handled when the z/OS FTP client is invoked from the FTP client API.

Application tasks for the SCMD request

Before you begin: Have an initialized interface defined by an FCAI and an active z/OS FTP client process (not stopped or broken).

Perform the following steps to issue the SCMD request:

1. Specify a MODE parameter (optional).

2. Specify a subcommand parameter string.

3. Set FCAI-ReqTimer to the desired value.
4. Set FCAI-TraceIt as desired for tracing. A request that initiates the interface trace also uses FCAI-TraceID and FCAI-TraceSClass.

5. Issue the SCMD request.

6. Check the result of the request.

7. Issue one or more POLL requests to complete the subcommand, if necessary.

8. Check the result of the completed subcommand.

9. Retrieve the FTP client output with GETL (optional).

After the SCMD request completes, you can issue a subsequent SCMD or TERM request.

**POLL**

The user program issues the POLL request to complete and retrieve the results from a prior SCMD request.

The POLL request is rejected if a prior subcommand is not in-progress. That is, a prior SCMD or POLL request must have returned the status code FCAI-Status-InProgress; otherwise, the current POLL request returns FCAI-IE-NotInProgress.

Issuing an SCMD request in no-wait mode enables you to perform other program functions while the subcommand is running. While you are performing these functions, the FTP client might be writing a large amount of data to the interface.

**Guideline:** Avoid large intervals of time between POLL requests. Large time intervals can cause a pipe overflow and lead to an error or a wait in the client until the results from the subcommand are retrieved.

Each POLL request copies up to 32 KB of data from the client to the interface buffer even if the subcommand has not yet completed. The actual amount that is returned depends upon the size of the output, the timing of the request, and how much space is available in the current part of the interface buffer.

See "GETL" on page 427 for a discussion of the interface buffer. See "FCAI_PollWait: Specifying a wait time before POLL" on page 468 for more information about managing POLL requests.

**Example of the POLL call instruction**

```
WORKING-STORAGE SECTION.

COPY EZAFTPKC.
01 REQUEST-POLL PIC X(4) VALUE IS 'POLL'.
```
PROCEDURE DIVISION.

CALL 'EZAFTPKS' USING FCAI-Map REQUEST-POLL.

For equivalent PL/I and assembler language declarations, see “Converting parameter descriptions” on page 402.

Parameter values that are set by the application

FCAI-Map
   Storage area (defined in EZAFTPKC for COBOL) used to save information about the requests using the FTP client API.

REQUEST-TYPE
   A 4-byte field that contains POLL.

Parameter values that are returned to the application

The results of the request are returned in the FCAI-Result field. See “Interpreting results from an interface request” on page 464. See “FTP client API: Other output that is returned to the application” on page 461 for a discussion of the output and statistics returned by the request.

Guidelines for POLL results:
• If FCAI-Result contains FCAI-Result-Status, additional processing is required by the user program:
  – If the FCAI-Status field is FCAI-Status-InProgress, a prior SCMD request issued in no-wait mode or one that exceeded the FCAI-ReqTimer value is still in-progress. Obtain the final results with another POLL request.
  – If the FCAI-Status field is FCAI-Status-PromptPass, the prior request has now completed but the next SCMD request must be a PASS subcommand. This status is applicable for the USER subcommand.
  – If the FCAI-Status field is FCAI-Status-PromptAcct, the prior request has now completed but the next SCMD request must be an ACCT subcommand. This status is applicable for the USER, PASS, and CD subcommands.
  – If FCAI-Status contains FCAI-Status-TraceFailed or higher, the interface trace function failed on this request. Subtract FCAI-Status-TraceFailed from FCAI-Status to obtain the remaining value for FCAI-Status. See “FCAI_Status_TraceFailed and FCAI_TraceStatus: Reporting failures in the interface trace function” on page 467 for information about FCAI-Status-TraceFailed.
• Aside from interface errors, POLL itself returns only FCAI-Status-InProgress. All other results are those of the prior SCMD request.
• If the application issues POLL until FCAI-IE-NotInProgress is returned, the results from the prior subcommand are lost. Issue POLL only until FCAI-Status-InProgress is no longer returned.

Application tasks for the POLL request

Before you begin: Have an initialized interface defined by an FCAI and an active z/OS FTP client process (not stopped or broken). Ensure that a prior SCMD or POLL request returned FCAI-Status-InProgress.

Perform the following steps to issue the POLL request:
1. Set FCAI-PollWait to the desired value.
2. Set FCAI-TraceIt as desired for tracing. A request that initiates the interface trace also uses FCAI-TraceID and FCAI-TraceSClass.

3. Issue the POLL request.

4. Check the result of the request.

5. Repeat the POLL request to complete the subcommand, if necessary.

6. Check the result of the completed subcommand.

7. Retrieve the FTP client output with GETL (optional).

After the POLL request completes, you can issue a subsequent SCMD or TERM request.

**GETL**

The user program issues the GETL request to find and copy lines of output that are returned from the FTP client by an INIT, SCMD, or POLL request. The request copies output from the client into a space that is acquired by the user and described to the interface by a vector. The vector includes the address of the space, the ALET for the space (or 0), and the length of the space. In the description that follows, this space is referred to as the *user’s buffer*.

The internal buffers maintained by the FTP client API are referred to collectively as the *interface buffer*. The interface buffer is populated upon return from an INIT, an SCMD issued in wait mode, or a POLL issued for an SCMD that did not complete. The interface acquires the number of 32 KB interface buffer parts that are needed to contain the output from the request.

- INIT and SCMD in wait mode acquire interface buffer parts automatically until the request completes.
- POLL either fills the remainder of the current part of the interface buffer or acquires a new part before reading from the client. The actual amount returned for each POLL request varies depending on available output and current buffer utilization.
- The interface retains and reuses all interface buffer parts until TERM is issued to minimize acquiring and freeing storage. If the user program does not wish to retain acquired buffer parts, it should issue TERM to free them and then optionally reinitialize the interface.
- An SCMD QUIT that is generated by TERM uses the interface buffer, but only to allow the interface trace function to print the subcommand results. All interface buffer parts are freed by TERM and output is not available to the user program after that time.

See “FTP client API: Other output that is returned to the application” on page 461 for a discussion of the statistics that can be used in preparing to retrieve the output (such as setting the size of your buffer).
GETL has an OPERATION parameter that specifies whether to FIND and retrieve one line or to COPY many lines of output.

GETL has a TYPE parameter that specifies the type of lines to copy to the user’s buffer. Lines are classified into the following types: client messages, server replies, DIR or LS subcommand output, and trace output.

Each line of output that is written to the user’s buffer has a 3-byte prefix that consists of a 1-byte ID and a 2-byte length field. The IDs are defined as follows:

<table>
<thead>
<tr>
<th>ID</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>Message from the client</td>
</tr>
<tr>
<td>R</td>
<td>Reply from the server</td>
</tr>
<tr>
<td>T</td>
<td>Trace output that is generated as a result of activating FTP trace and extended trace with DEBUG and DUMP subcommands</td>
</tr>
<tr>
<td>L</td>
<td>Data from a DIR or LS subcommand</td>
</tr>
</tbody>
</table>

A last line marker is appended to the user’s buffer upon completion of a successful COPY request. It has a length of zero and one of the following IDs:

<table>
<thead>
<tr>
<th>ID</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>Last line of output for a request so far (The subcommand is not complete or you ran out of room in the user’s buffer. More output will be or is already available to be retrieved.)</td>
</tr>
<tr>
<td>Z</td>
<td>Last line of output for the request</td>
</tr>
</tbody>
</table>

There are two concepts about the output lines in the user’s buffer and the interface buffer that are important to understand. The first concept is the last line marker of output in the user’s buffer after a successful COPY operation. (A successful FIND copies exactly one line without writing a last line marker.) The last line marker has an ID of either Y or Z and a length field of x’0000.

When the last line marker ID is Y, partial output has been copied to the user’s buffer and the user program must now handle one or both of the following conditions:

- A prior subcommand has not yet completed, and more results must be retrieved with POLL.
- The COPY operation found more lines of output than can fit in the user’s buffer.

**Tip:** A last line marker with an ID of Y does not guarantee that any more lines of the type specified on the COPY were generated by the request.

When the last line marker ID is Z, the last line of output for the request has been retrieved.

See “FTP client API: Other output that is returned to the application” on page 461 and “POLL” on page 425.

**Rule:** A GETL request that specifies the same user’s buffer overwrites the contents of that buffer; it does not append lines to the user’s buffer.

**Guideline:** If the returned output must be preserved, move it or update the buffer pointer before a subsequent GETL request. This does not have to be done when a COPY immediately follows a FIND using the same criteria, as discussed in the remainder of this topic.
The second concept is that of the current line in the interface buffer. Initially, the current line is the first line of the interface buffer. The current line changes as you successfully FIND and COPY lines of output, as follows:

- After you successfully FIND a line, that line is copied into the buffer addressed by your vector and becomes the current line in the interface buffer.
  - An immediate COPY of one or more lines includes the line that was found on the preceding FIND when the TYPE parameter is the same on both operations.
  - A subsequent FIND (NEXT, FIRST, or LAST) advances the current line pointer to the appropriate matching line and copies that line.
- When you successfully COPY lines, the current line pointer first advances to the next matching line of the type specified on the COPY. At the end of the COPY operation, the current line pointer advances to the location that immediately follows the last copied line, which can be a line of any TYPE or the end of the interface buffer.
- When the current line pointer advances past a line in the interface buffer, that line can be located only by resetting the current line to a point at or before the desired line with a FIND FIRST with TYPE A (any) or the TYPE that matches the desired line.
- If FIND or COPY does not locate any matching lines, GETL returns FCAI-Result-NoMatch in FCAI-Result. This result or any result other than FCAI-Result-OK means that the current line pointer was unchanged by the request.

See "Using the FIND and COPY operations" on page 432 for more information about FIND and COPY.

**Example of the GETL call instruction**

```
WORKING-STORAGE SECTION.

COPY EZAFTPKC.
01 REQUEST-GETL PIC X(4) VALUE IS 'GETL'.
01 OPERATION PIC X(4).
01 TYPE PIC X(1).
01 SEQUENCE PIC X(1).
01 VECTOR.
   05 BUFF-ADDR USAGE IS POINTER.
   05 BUFF-ALET USAGE IS POINTER.
   05 BUFF-LEN PIC 9(8) COMP-5.
01 BUFFER PIC X(100).

PROCEDURE DIVISION.

MOVE 'FIND' TO OPERATION.
MOVE 'M' TO TYPE.
MOVE 'F' TO SEQUENCE.

SET BUFF-ADDR TO THE ADDRESS OF BUFFER.
MOVE ZEROS TO BUFF-ALET.
MOVE SIZE OF BUFFER TO BUFF-LEN.

CALL 'EZAFTPKS' USING FCAI-Map REQUEST-GETL OPERATION TYPE SEQUENCE VECTOR.

MOVE 'COPY' TO OPERATION.
MOVE 'M' TO TYPE.

SET BUFF-ADDR TO THE ADDRESS OF BUFFER.
MOVE ZEROS TO BUFF-ALET.
```
MOVE SIZE OF BUFFER TO BUFF-LEN.

CALL 'EZAFTPKS' USING FCAI-Map REQUEST-GETL OPERATION
TYPE VECTOR.

For equivalent PL/I and assembler language declarations, see "Converting parameter descriptions" on page 402.

Parameter values that are set by the application

FCAI-Map
Storage area (defined in EZAFTPKC for COBOL) used to save information about the requests using the FTP client API.

REQUEST-TYPE
A 4-byte field that contains GETL.

OPERATION
A 4-byte field that contains the operation to be performed. The OPERATION values are:

FIND  Find an output line matching TYPE and copy it into the user’s buffer.

COPY   Copy all remaining output lines matching TYPE into the user’s buffer.

TYPE A 1-byte field that indicates what type of output is requested by OPERATION. The TYPE values are:

M  Message from the client.

R  Reply from the server.

L  List data from a DIR or LS subcommand.

T  Trace output from debug or dump.

A  Any type of output line.

SEQUENCE (FIND operation only)
A 1-byte parameter that indicates which of the output lines of TYPE to FIND. SEQUENCE values are:

F  Find the first line of the requested TYPE.

N  Find the next line of the requested TYPE.

L  Find the last line of the requested TYPE.

Tip: The SEQUENCE parameter is not included on the call for a COPY operation, which always begins with the first line that matches TYPE at or after the current line.

VECTOR
A 3-word vector that describes the user’s buffer that receives a copy of an output line. VECTOR values are:

BUFF-ADDR
The address of the buffer used in the operation.

BUFF-ALET
The ALET of the buffer pointed to by BUFF-ADDR. This can be 0. If not 0, the ALET must reside in the PASN-AL for the application or be a public entry in the DU-AL. All programs that the application invokes must have the authority to access the space.
See the z/OS MVS Programming: Extended Addressability Guide for information about using ALETs to access data spaces.

**BUFF-LEN**
The length of the buffer used in the operation. For a COPY operation the interface reserves 4 bytes at the end of the buffer to ensure that there is room for the Y or Z line.

**Rule:** Align VECTOR on at least a fullword boundary.

**Parameter values that are returned to the application**
The results of the request are returned in the FCAI-Result field. See “Interpreting results from an interface request” on page 464.

**Guidelines for GETL results:**
- If FCAI-Result contains FCAI-Result-NoMatch:
  - If OPERATION was FIND and SEQUENCE was FIRST or LAST, no lines of the specified type currently exist in the interface buffer.
  - If OPERATION was FIND and SEQUENCE was NEXT, or if OPERATION was COPY, no lines of the specified type exist between the current line pointer and the end of the interface buffer.
- FCAI-IE-BufferTooSmall indicates that the user’s buffer will not hold the first (or only) matching line that was found in the interface buffer.

**Tip:** FCAI-IE-TooManyParameters might indicate that a SEQUENCE parameter was included for a COPY operation.

**Application tasks for the GETL request**
**Before you begin:** Have an initialized interface defined by an FCAI.

Perform the following steps to issue the GETL request:
1. Specify the desired operation (FIND or COPY).
2. Specify the desired line type.
3. Specify the desired sequence (for FIND).
4. Specify the buffer vector that describes the user’s buffer.
5. Set FCAI-TraceIt as desired for tracing. A request that initiates the interface trace also uses FCAI-TraceID and FCAI-TraceSClass.
6. Issue the GETL request.
7. Check the result of the request.
8. Issue one or more POLL requests to retrieve output for a subcommand that is in-progress.
9. Issue another GETL request, optionally changing the operation, sequence, or type of line.

After the GETL request completes, you can issue a subsequent SCMD or TERM request.

**Using the FIND and COPY operations**

**The FIND operation syntax for COBOL is as follows:**

```cobol
CALL 'EZAFTPKS'
USING FCAI-Map REQUEST-GETL OPERATION TYPE SEQUENCE VECTOR.
```

A FIND operation copies one line from the interface buffer into the user's buffer described by the vector. The FIND locates and copies the line indicated by the sequence parameter [F (first), L (last), or N (next)] that matches the requested TYPE.

**The COPY operation syntax for COBOL is as follows:**

```cobol
CALL 'EZAFTPKS'
USING FCAI-Map REQUEST-GETL OPERATION TYPE VECTOR.
```

A COPY operation copies one or more lines from the interface buffer into the user's buffer described by the vector. The COPY searches for a line at or after the current line that matches the TYPE parameter. It copies from that line through the last line of output in the interface buffer that matches TYPE.

The COPY stops when the user's buffer is full or when there are no more lines of the requested type in the interface buffer. The last line marker is written into the user's buffer after the last output line of the requested type (the interface reserves 4 bytes in the user's buffer to ensure that there is room for the marker).

If the FIND or COPY operation does not locate any matching lines, GETL returns FCAI-Result-NoMatch in FCAI-Result and does not change the current line pointer. The user's buffer contents are not predictable when this result is returned.

**Tips:**

- If you want to set the current line pointer to the top of the interface buffer, use a GETL request with TYPE set to A (any type of output line) and SEQUENCE set to F (first).
- GETL FIND and COPY cannot find output that has not yet been retrieved from the client and copied to the interface buffer. When handling the results from an incomplete subcommand, any POLL might append data to the interface buffer, but the results are only complete after a POLL request does not return FCAI-Status-InProgress.

**Assembler language GETL example:** Assume that an SCMD was issued with the subcommand `LS a*`. Following are the lines of output that are in the buffer at the end of the SCMD request. (The lines have been numbered for this example.)

```plaintext
1 T 0026 CA0149 dirlist: entered for ls command
2 T 0017 CA0798 getList: entered
3 M 0018 >>> PORT 9,42,105,93,4,8
4 R 0014 200 Port request OK.
5 M 000B >>> NLST a*
6 R 0013 125 List started OK
7 T CA1275 rcvListData: entered
8 L 0001 a
```
The following fields are also defined in the user program:

GETL DC C'GETL'
FIND DC C'FIND'
COPY DC C'COPY'
FIRST DC C'F'
NEXT DC C'N'
LAST DC C'L'
MESSAGE DC C'M'
REPLY DC C'R'
LIST DC C'L'
ANY DC C'A'
*  
BUFFVEC DS 0XL12
BUFFADDR DC A(BUFFER) THIS BUFFER IS LARGER THAN TOTAL OUTPUT RETURNED
   DC F'0'
BUFFLENG DC A(L'BUFFER)
BUFFER DC CL'4096'

These calls retrieve the following output lines:

CALL EZAFTPKS,(FCAI_Map,GETL,FIND,MESSAGE,FIRST,BUFFVEC)
copies line 3: M 0018 >>> PORT 9,42,105,93,4,8
CALL EZAFTPKS,(FCAI_Map,GETL,FIND,MESSAGE,NEXT,BUFFVEC)
copies line 5: M 000B >>> NLST a*
CALL EZAFTPKS,(FCAI_Map,GETL,FIND,MESSAGE,NEXT,BUFFVEC)
copies line 12: M 0018 Command(00-20-NLST-250):
CALL EZAFTPKS,(FCAI_Map,GETL,FIND,MESSAGE,NEXT,BUFFVEC)
copies line 12: M 0018 Command(00-20-NLST-250):
returns FCAI_Result_NoMatch
CALL EZAFTPKS,(FCAI_Map,GETL,FIND,REPLY,LAST,BUFFVEC)
copies line 10: R 0020 250 List completed successfully.
CALL EZAFTPKS,(FCAI_Map,GETL,FIND,LIST,BUFFVEC)
copies line 8: L 0001 a
CALL EZAFTPKS,(FCAI_Map,GETL,COPY,LIST,BUFFVEC)
copies line 8: L 0001 a
copies line 9: L 0002 ab
copies line 13: Z 0000

TERM

The user program issues the TERM request to terminate this instance of interface use. A TERM request is accepted by the interface at any time.

For the most orderly termination of the interface, the client program should ensure that no SCMDs are in-progress and then issue SCMD with the QUIT subcommand before using the TERM request. To assist the caller with exceptional conditions, TERM performs the following steps unless the client process is broken:

1. If the TERM request detects that a prior SCMD is in-progress, it generates up to three POLL requests at 16-second intervals in an attempt to retrieve the results from the client. Any generated POLL requests appear in the interface trace (if active) with (Generated by TERM) appended to the request record along with any results that are retrieved.

2. If the TERM request detects that a QUIT has not yet been issued to the client, it generates an SCMD QUIT request. The generated SCMD QUIT request appears
in the interface trace (if active) with (generated by TERM) appended to the request record along with any results that are retrieved.

**Rule:** The caller can specify the number of seconds to wait by setting a FCAI_ReqTimer value (or specify that no timer is to be used). See "FCAI_ReqTimer: Controlling requests that retrieve results from the spawned z/OS FTP client process" on page 468.

If the client process is broken, or the interface fails to complete a subcommand that was in-progress, or a generated QUIT fails to complete successfully, then the interface issues BPX1KIL with no signal to kill the client process. The reason for the failure is reported on a prior request or in the interface trace.

When TERM completes, FCAI-Token is zeros and only an INIT request is accepted by the interface using this FCAI.

**Rule:** Never update FCAI-Token or attempt to reinstate it after TERM. If FCAI-Token is corrupted, you must terminate your application to free acquired storage and stop the client process.

**Guidelines:**
- To ensure that complete results are returned, always check for FCAI-Status-InProgress upon return from an SCMD request and POLL to complete the request if it did not complete. Although TERM generates POLLS to the client, this does not guarantee that the prior subcommand will complete or that all results will be returned. The three POLL requests can retrieve a maximum of 96 KB. The actual amount that is retrieved depends upon the timing of the request and current buffer utilization.
- Set FCAI-TraceIt to FCAI-TraceIt-Yes on TERM to see the results from any generated POLLS or QUIT. The results are available only in the trace because TERM frees the interface buffer.
- If an active FCAI is cleared and reused on an INIT request without first issuing an SCMD QUIT or TERM, the FTP client process that was associated with it is stranded. The client child process can persist until the parent process terminates. Failure to issue TERM after SCMD QUIT retains acquired storage until the parent task terminates. It is especially important in long-running application processes to terminate instances of the interface that are no longer in use.

**Example of the TERM call instruction**

```
WORKING-STORAGE SECTION.

COPY EZAFTPKC.
    01 REQUEST-TERM PIC X(4) VALUE IS 'TERM'.

PROCEDURE DIVISION.

CALL 'EZAFTPKS' USING FCAI-Map REQUEST-TERM.
```

**Parameter values that are set by the application**

**FCAI-Map**

Storage area (defined in EZAFTPKC for COBOL) used to save information about the requests using the FTP client API.
REQUEST-TYPE
A 4-byte field that contains TERM.

Parameter values that are returned to the application
The results of the request are returned in the FCAI-Result field. See "Interpreting results from an interface request" on page 464.

Guidelines for TERM results:
- See the results from any generated POLL or SCMD QUIT requests in the interface trace output (if active).
- FCAI-Result-OK means the interface terminated normally (whether or not additional requests were automatically generated).
- FCAI-Result-CliProcessKill means that BPXKIL was issued to end the client process.
- FCAI-Result-Status means that FCAI-Status-TraceFailed was returned.
- FCAI-Status-TraceFailed can also be returned when FCAI-Result contains FCAI-Result-CliProcessKill.

Application tasks for the TERM request
Perform the following steps to issue the TERM request:

1. Set FCAI-ReqTimer to the desired value (for a generated SCMD QUIT).
2. Set FCAI-TraceIt as desired for tracing. A request that initiates the interface trace also uses FCAI-TraceID and FCAI-TraceSClass.
3. Issue the TERM request.
4. Check the result of the request.
5. See the interface trace output for the results of any generated POLL or SCMD QUIT requests.

After the TERM request completes, you can reinitialize the FCAI and issue a subsequent INIT request.

FTP client API for C functions
For the C programming language, in-line static functions are provided in the ftpcapi header file. These functions provide typecasting compiler checking and tools to facilitate calling the interface from a C language program.

The following are in-line static functions:
- FAPI_INIT initializes the interface.
- FAPI_SCMD sends an FTP subcommand.
- FAPI_POLL checks the status of an outstanding subcommand.
- FAPI_GETL_COPY retrieves output related to a subcommand and copies to a user buffer.
FAPI_GETL_FIND retrieves output related to a subcommand and searches for a line of a specific type of output.

FAPI_TERM ends the interface.

FAPI_INIT

Use the FAPI_INIT function to initialize the FTP client API. See "Sending requests to the FTP client API" on page 418 for more information about an INIT request.

FAPI_INIT accepts the following parameters:

**FCAI_MAP**
A pointer to an fcai_map_t structure used to save information about the request using the FTP client API.

**startparm**
A pointer to a NULL terminated string that contains the parameters for the z/OS FTP command. See the FTP command — Entering the FTP environment information in the z/OS Communications Server: IP User's Guide and Commands for descriptions of valid FTP command parameters.

**envVars**
A pointer to an fcai_envvarlist_t structure that contains a count followed by an array of up to nine pointers to NULL terminated strings. Each string represents an environment variable definition.

**FAPI_INIT example:**

```c
#define OPENSTRING "-w 300 127.0.0.1 21 (trace"

fcai_map_t fcai;
my_envvars.envVarCount = 3;
my_envvars.envVarEnt[0] = "_CEE_DMPTARG=/etc"
my_envvars.envVarEnt[1] = "_BPX_JOBNAME=MYJOB"
my_envvars.envVarEnt[2] = "NLSPATH=/u/myuser/%N"

memset(&fcai, 0, sizeof(fcai));
fcai.FCAI_Eyecatcher = FCAI_EYECATCHER;
fcai.FCAI_Size = FCAI_NUMINTERFACEBYTES;
fcai.FCAI_Version = FCAI_VERSION;
rc = FAPI_INIT(&fcai, OPENSTRING, &my_envars);
```

FAPI_SCMD

Use the FAPI_SCMD function to issue the SCMD request to send a subcommand to the FTP client API. See "Sending requests to the FTP client API" on page 418 for more information about an SCMD request.

FAPI_SCMD accepts the following parameters:

**FCAI_MAP**
A pointer to an fcai_map_t structure used to save information about the request using the FTP client API.

**subcommand**
A pointer to a NULL-terminated string that contains a z/OS FTP client subcommand with its parameters.

**mode**
A required character that indicates whether the interface should wait for the subcommand to complete before returning to the caller. Valid values are FAPI_MODE_WAIT or FAPI_MODE_NOWAIT.
FAPI_SCMD example:
/* Assume FAPI_INIT was successfully called with &fcai */
/* prior to calling FAPI_SCMD */
rc = FAPI_SCMD(&fcai, "USER user1", FAPI_MODE_WAIT);

FAPI_POLL

Use the FAPI_POLL function to issue the POLL request to complete and retrieve
the results from a prior FAPI_SCMD request. See "Sending requests to the FTP
client API" on page 418 for more information on a POLL request.

FAPI_POLL accepts the following parameter:
FCAI_MAP
A pointer to an fcai_map_t structure used to save information about the
request using the FTP client API.

FAPI_POLL example:
/* Assume FAPI_SCMD was successfully called with &fcai */
/* prior to calling FAPI_POLL */
rc = FAPI_POLL(&fcai);

FAPI_GETL_COPY

Use the FAPI_GETL_COPY function to issue the GETL COPY request to copy lines
of output that are returned from the FTP client by an FAPI_INIT, FAPI_SCMD, or
FAPI_POLL functions. See "Sending requests to the FTP client API" on page 418
for more information on a GETL request.

FAPI_GETL_COPY accepts the following parameters:
FCAI_MAP
A pointer to an fcai_map_t structure used to save information about the
request using the FTP client API.
type
A character that indicates what type of output is requested by the copy.
Valid type values include:

<table>
<thead>
<tr>
<th>Field constant defined in ftpcapi.h</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAPI_GETL_MESSAGE_LINE</td>
<td>M</td>
<td>Message from the client.</td>
</tr>
<tr>
<td>FAPI_GETL_REPLY_LINE</td>
<td>R</td>
<td>Reply from the server.</td>
</tr>
<tr>
<td>FAPI_GETL_TRACE_LINE</td>
<td>T</td>
<td>Trace output from debug or dump.</td>
</tr>
<tr>
<td>FAPI_GETL_LIST_LINE</td>
<td>L</td>
<td>List data from a DIR or LS subcommand.</td>
</tr>
<tr>
<td>FAPI_GETL_ANY_LINE</td>
<td>A</td>
<td>Any type of output line.</td>
</tr>
</tbody>
</table>

buffer_len
The length of the buffer used for the copy.

buffer
The address of the buffer used in the operation.

FAPI_GETL_COPY example:
/* Assume FAPI_INIT was successfully called with &fcai */
/* prior to calling FAPI_GETL_COPY */
char buffer[4096];
rc = FAPI_GETL_COPY(&fcai, FAPI_GETL_LIST_LINE,
sizeof(buffer), buffer);
**FAPI_GETL_FIND**

Use the FAPI_GETL_FIND function to issue the GETL FIND request to find one line of output returned from the FTP client by an FAPI_INIT, FAPI_SCMD, or FAPI_POLL. See “Sending requests to the FTP client API” on page 418 for more information on a GETL request.

FAPI_GETL_FIND accepts the following parameters:

**FCAI_MAP**
A pointer to an fcai_map_t structure used to save information about the request using the FTP client API.

**type**
A character that indicates what type of output is requested by the copy. Possible type values include:

<table>
<thead>
<tr>
<th>Field constant defined in ftpcapi.h</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAPI_GETL_MESSAGE_LINE</td>
<td>M</td>
<td>Message from the client.</td>
</tr>
<tr>
<td>FAPI_GETL_REPLY_LINE</td>
<td>R</td>
<td>Reply from the server.</td>
</tr>
<tr>
<td>FAPI_GETL_TRACE_LINE</td>
<td>T</td>
<td>Trace output from debug or dump.</td>
</tr>
<tr>
<td>FAPI_GETL_LIST_LINE</td>
<td>L</td>
<td>List data from a DIR or LS subcommand.</td>
</tr>
<tr>
<td>FAPI_GETL_ANY_LINE</td>
<td>A</td>
<td>Any type of output line.</td>
</tr>
</tbody>
</table>

**buffer_len**
The length of the buffer used for the copy.

**buffer**
The address of the buffer used in the operation.

**sequence**
A character that indicates which of the output lines of type should be found. Valid sequence values include:

<table>
<thead>
<tr>
<th>Field constant defined in ftpcapi.h</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAPI_GETL_FIND_FIRST</td>
<td>F</td>
<td>Find the first line of the requested type.</td>
</tr>
<tr>
<td>FAPI_GETL_FIND_NEXT</td>
<td>N</td>
<td>Find the next line of the requested type.</td>
</tr>
<tr>
<td>FAPI_GETL_FIND_LAST</td>
<td>L</td>
<td>Find the last line of the requested type.</td>
</tr>
</tbody>
</table>

**FAPI_GETL_FIND example:**

```c
/* Assume FAPI_INIT was successfully called with &fcai */
/* prior to calling FAPI_GETL_FIND */
char buffer[4096];
rc = FAPI_GETL_FIND(&fcai, FAPI_GETL_LIST_LINE,
sizeof(buffer), buffer, FAPI_GETL_FIND_LAST);
```

**FAPI_TERM**

Use the FAPI_TERM function to terminate the FTP client API instance associated with this FCAI_MAP. See “Sending requests to the FTP client API” on page 418 for more information on a TERM request.

FAPI_TERM allows the following parameter:

**FCAI_MAP**
A pointer to an fcai_map_t structure used to save information about the request using the FTP client API.
**FAPI_TERM example:**

```c
/* Assume FAPI_INIT was successfully called with &fcai */
/* prior to calling FAPI_TERM */
rc = FAPI_TERM(&fcai);
```

**FTP client API for REXX function**

A REXX language program requires an intermediate routine to translate from the string format used within REXX programs to the binary format used by the EZAFTPKS program. An external function package is provided that serves as this intermediary, facilitating calling the FTP client API from a REXX language program.

The FTP client API for REXX is included as a default system function package, and is included in the default parameter modules IRXTSPRM, IRXISPRM, and IRXPARMS.

The FTP client API for REXX has been verified to run in the following environments:
- TSO exec
- Batch environment
- ISPF
- UNIX shell

The FTP client API for REXX might run in additional environments, but it has not been tested and verified to work in other environments.

**Handling of SIGCHLD signals**

The FTP client API spawns a child process for the z/OS FTP client for each successful INIT request. When running in a Posix environment, the SIGCHILD signal is raised when the FTP client terminates. To avoid creating a zombie process when the child process terminates, the SIGCHILD signal must be caught or ignored.

If, when invoking the first CREATE request, the REXX program does not have a SIGCHLD signal handler and is running in a Posix environment, the FTP client API for REXX requests that the SIGCHLD signal be ignored. If the REXX program has a SIGCHLD handler, no change is made.

**FTP client API for REXX trace**

The FTP client API for REXX is used to debug problems in the FTP client API for REXX function package. The interface cannot be used to debug errors in the trace itself or any error that prevents the interface from accessing the trace data set. The trace writes records for requests to the interface and the results of interface requests.

**Tip:** The FTP client API for REXX trace is separate from the FTP client API trace. It uses a different output file and has a different enabling mechanism. The FTP client API for REXX trace is used to debug problems in the FTP client API for REXX function package, while the FTP client API trace is used to debug problems in the underlying EZAFTPKS interface and record activity and data that are returned to the interface that might not otherwise be available to the application.

The FTP client API for REXX trace is activated by specifying the z/OS UNIX FTP_REXX_TRACE_FILE environment variable or allocating the FTPRXTRC DD.
name. The FTP client API for REXX first looks for the FTP_REXX_TRACE_FILE environment variable (z/OS UNIX environment only) and then for the FTPRXTRC DD allocation.

The FTP client API for REXX trace can be written to any of the following:
- JES SYSOUT
- An MVS sequential data set (a member of a PDS is not supported); the data set must already exist or be allocated as new with DCB characteristics of an LRECL value in the range 80 – 256 and a RECFM value of Fixed Block
- A z/OS UNIX file. The file can be either an existing file or a file dynamically allocated by the FTP client API for REXX when needed

**Restriction:** In order for the FTP client API for REXX to be able to write trace records, the output data set or file must meet the following conditions:
- The data set or file must be a fixed block data set.
- For an MVS data set, the record format must be fixed.
- The data set or file record length must be in the range 8 – 256 bytes.
- The data set or file block size must be a multiple of the record length.
- The data set or file must not be block mode.

Data sets and files created by the FTP client API for REXX meet these conditions by default. If a file or data set is created by some other means, then you must ensure these conditions are met or no trace records will be written.

**Specifying the FTP client API for REXX trace output location**

Your environment determines the method you use to specify the FTP client API for REXX trace output location.

**Specifying the FTP client API for REXX trace output location: TSO environment:** In the TSO environment, the location specified by the FTPRXTRC DD statement is used as the FTP client API for REXX trace output location. Use the TSO ALLOCATE command to associate FTPRXTRC with these outputs. Following is an example:

```
ALLOC FILE(FTPRXTRC) DA(FTP.TRACE.OUT) NEW LRECL(80) RECFM(F B) TRACK SPACE(10 10)
```

See the [z/OS TSO/E Command Reference](https://www.ibm.com/docs/en/zos) for more details about using the ALLOCATE command.

**Specifying the FTP client API for REXX trace output location: z/OS UNIX shell environment:** In the z/OS UNIX shell environment, use one of the following to specify the FTP client API for REXX trace output location.

- For a new z/OS UNIX file or an existing MVS data set, enter the following:
  ```
  export FTP_REXX_TRACE_FILE=/tmp/myjob.ftapi.trace
  export FTP_REXX_TRACE_FILE="/appl.ftprxtrc"
  ```

  **Tip:** When the specified MVS data set name is not fully qualified, the user ID is added as the first qualifier for the data set. For example, if USER3 enters this command, it is equivalent to the following:

  ```
  export FTP_REXX_TRACE_FILE="/USER3.APPL.FTPRXTRC"
  ```

- For a non-qualified z/OS UNIX file (myjob.ftapi.trace) with a current working directory /u/user1/, the file used for tracing is /u/user1/myjob.ftapi.trace. Enter the following:
  ```
  export FTP_REXX_TRACE_FILE=ftpapi.trace
  ```
For a z/OS UNIX file or an MVS data set that is already allocated to a ddname, enter the following:

```bash
export FTP_REXX_TRACE_FILE="/dd:ddname"
```

**Restriction:** When using the FTP_REXX_TRACE_FILE environment variable, the maximum length for an MVS data set name or a z/OS UNIX file name is 64 characters. If the data set name or file name length exceeds 64 characters, then the name is truncated. If the MVS data set is not qualified, the 64 character limit is applied after the high-level qualifier is added. If the z/OS UNIX file path is a relative path, then the 64 character limit is calculated after the current working directory name is added.

**Specifying the FTP client API for REXX trace output location: MVS batch job environment:** In the MVS batch environment, an FTPRXTRC DD must be specified in the JCL for trace output to be written. You can write trace output as follows using the JES SYSOUT facility:

```bash
//FTPRXTRC DD SYSOUT=* 
```

**Specifying the FTP client API for REXX trace output location: z/OS UNIX environment batch job:** When using the z/OS UNIX environment from a batch job, use one of the following methods to specify the FTP client API for REXX trace output location:

- If the application exists in a file system, is invoked using the BPXBATSL utility, and does not perform any fork calls, use the FTPRXTRC DD statement to specify the output location as you would with an MVS batch job.
- In all other cases, the FTP_REXX_TRACE_FILE environment variable must be set. When using BPXBATSL or BXPBATCH utilities, set this and any other required environment variables using the STDENV DD statement as follows:

```bash
//STDENV DD JCL statement
```

Following is an example:

```bash
//STDENV DD DISP=SHR,DSN=USER3.APPL.ENVIRON
```

The STDENV data set can a fixed or variable (nospanned) record format type. It can contain multiple environment variables, as shown in the following sample:

```bash
FTP_REXX_TRACE_FILE="/USER3.APPL.RESTRACE"
_BPXK_SETIBMOPT_TRANSPORT=TCPCS
```

**Guidelines:**
- Environment variables must start in column 1, and the data set must not contain any sequence numbers (sequence numbers would be treated as part of the environment variable).
- For the FTP_REXX_TRACE_FILE environment variable, any blanks from a fixed format STDENV data set are removed. Because this might not be true for all variables, you should use a variable record format data set.
- For applications that fork, you should use an MVS data set. If you use a file system file, a C03 ABEND might occur when the forked process ends.

**Restriction:** When using the FTP_REXX_TRACE_FILE environment variable, the maximum length for the MVS data set name or the z/OS UNIX file name is 64 characters. If the data set name or file name length exceeds 64 characters, the name is truncated. If the MVS data set is not qualified, the 64 character limit is applied after the high-level qualifier is added. If the z/OS UNIX file path is a relative path, then the 64 character limit is calculated after the current working directory name is added.
See the z/OS UNIX System Services Command Reference for additional considerations when using the BPXBATCH or BPXBATSL utilities.

**FTP client API requests**

All FTP client API for REXX requests use the same format:

```plaintext
result = ftpapi(stem, request_type, parm1, parm2, ...);
```

The first parameter is the name of the REXX stem that refers to a specific instance of the FTP client environment. The second parameter identifies the API request that is being made (for example INIT or SCMD). The remaining parameters are used to pass data which is sometimes optional for the specific API request that is being made.

**Tip:** When passing string literals, enclose them in single or double quotes.

All FTP client API for REXX return codes use a consistent format, as shown in Table 19.

<table>
<thead>
<tr>
<th>Code value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0</td>
<td>A negative return code indicates that the FTP client API for REXX function failed to complete because of an error.</td>
</tr>
<tr>
<td>0</td>
<td>The call completed successfully and there is no additional status available.</td>
</tr>
<tr>
<td>&gt;0</td>
<td>The call completed successfully and there is additional information available.</td>
</tr>
</tbody>
</table>

The following requests are supported by the FTP client API for REXX:

- CREATE: Creates a new instance of the interface.
- INIT: Initializes the interface.
- SCMD: Sends an FTP subcommand.
- POLL: Checks the status of an outstanding subcommand.
- GETL_FIND: Retrieves output related to a subcommand and searches for a line of a specific type of output.
- GETL_COPY: Retrieves output related to a subcommand and copies it to a user buffer.
- SET_TRACE: Enables or disables the tracing of subsequent FTP client API calls within the EZAFTPKS program.
- SET_REQUEST_TIMER: Sets the length of time that the FTP client API waits for a INIT, SCMD, or TERM request to complete.
- GET_FCAI_MAP: Returns the complete contents of the FCAI_Map structure.
- TERM: Ends the interface.
CREATE request

Format:

```
ftpapi (stem, 'create', trace_id, traceSClass, traceNum)
```

Purpose: Creates a new instance of the FTP client API.

Parameters:

*stem*

The name of a stem used to return the FTP client environment. This stem is included as the first parameter on all subsequent FTP client API for REXX calls.

*create*

Requests the creation of a new FTP client API stem. The string literal is not case sensitive.

*trace_id*

The identifier to be used in trace records. The ID is a 3-character ID string that is written as the first three characters of each trace record. If not specified, then three blank characters (') are used. The string literal is case sensitive.

*traceSClass*

The SYSOUT class for the trace. Valid values are in the range A – Z and 0 – 9. The default value is A. The string literal is case sensitive.

*traceNum*

The number of trace records written to the REXX trace file before the file is closed and then reopened. This value can be any decimal value in the range 1 – 1 000 000. The default is 1 000 000 records.

Results:

<table>
<thead>
<tr>
<th>Value</th>
<th>Return codes</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0</td>
<td></td>
<td>The call did not complete successfully. The FTP client API stem variable was not created. The possible failure return codes are listed in the Return codes column.</td>
</tr>
<tr>
<td>-2 (FCAI_RESULT_IE)</td>
<td></td>
<td>An interface error occurred. The interface error code is stored in the <code>stem.FCAI_IE</code> stem variable.</td>
</tr>
<tr>
<td>-3 (FCAI_RESULT_CEC)</td>
<td></td>
<td>A client error occurred. The client error code is stored in the <code>stem.FCAI_CEC</code> field stem variable.</td>
</tr>
<tr>
<td>-17 (FCAI_RESULT_UNUSABLESTEM)</td>
<td></td>
<td>An error occurred in creating the stem variable.</td>
</tr>
<tr>
<td>-19 (FCAI_RESULT_REXXERROR)</td>
<td></td>
<td>An error occurred when storing, fetching, or dropping a REXX variable.</td>
</tr>
<tr>
<td>≥0</td>
<td></td>
<td>The call completed successfully. The FTP client API stem variable was successfully created.</td>
</tr>
<tr>
<td>0 (FCAI_RESULT_OK)</td>
<td></td>
<td>No additional information is available.</td>
</tr>
</tbody>
</table>
Examples:
rc = ftpapi('fcai.', 'create', 'PAZ')
INIT request

Format:

```
ftpapi(------stem------,------'init'------,------initString------,------envVar------)
```

Purpose: Initializes a new instance of the FTP client API.

Parameters:

stem
The name of a stem used to return the FTP client environment. The stem must have been passed on a prior successful CREATE function call.

‘init’
Requests the initialization of an FTP client API environment. The string literal is not case sensitive.

initString
Start parameters that are valid to enter on a z/OS FTP client command. The string literal is case sensitive.

eenvVar
Zero to nine environment variables that can be passed to the FTP client API. The string literals are case sensitive. See the following for information about FTP environment variables:

- [FTP server environment variables](https://www.ibm.com) in the [z/OS Communications Server: IP Configuration Reference](https://www.ibm.com)

Also see the z/OS UNIX System Services library and z/OS Language Environment library of publications for information concerning environment variables.

Rules:

- Do not specify duplicate environment variables.
- Do not pass environment variable _CEE_RUNOPTS on an INIT request. The environment variables are established too late in the spawn() process for run-time options to be honored. See the [z/OS Language Environment Programming Guide](https://www.ibm.com) for information about using the CEEDOPT program or the CEEBXITA exit to specify run-time options for the FTP client process.
- Do not pass environment variables like _BPX_JOBNAME in the _CEE_ENVFILE file because the _CEE_ENVFILE file processing is too late in the spawn() process to set the job name. Specify _BPX_JOBNAME as one of the nine environment variables in the envVar list.
### Results:

**Table 21. FTP client INIT request return codes**

<table>
<thead>
<tr>
<th>Value</th>
<th>Return codes</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0</td>
<td></td>
<td>The call did not complete successfully. The subcommand was not successfully executed. The possible failure return codes are listed in the Return codes column.</td>
</tr>
<tr>
<td>-2 (FCAI_RESULT_IE)</td>
<td></td>
<td>An interface error occurred. The interface error code is stored in the \textit{stem}.FCAI_IE stem variable.</td>
</tr>
<tr>
<td>-3 (FCAI_RESULT_CEC)</td>
<td></td>
<td>A client error occurred. The client error code is stored in the \textit{stem}.FCAI_CEC field stem variable.</td>
</tr>
<tr>
<td>-17 (FCAI_RESULT_UNUSABLESTEM)</td>
<td></td>
<td>The stem variable is not usable.</td>
</tr>
<tr>
<td>-19 (FCAI_RESULT_REXXERROR)</td>
<td></td>
<td>An error occurred when storing, fetching, or dropping a REXX variable.</td>
</tr>
<tr>
<td>≥0</td>
<td></td>
<td>The call completed successfully. The FTP client environment was initialized.</td>
</tr>
<tr>
<td>0 (FCAI_RESULT_OK)</td>
<td></td>
<td>No additional information is available.</td>
</tr>
</tbody>
</table>

**Examples:**

```plaintext
rc = ftpapi('fcai.', 'init', '-w 300 127.0.0.1 21', '_CEE_DMPTARG=/tmp', '_BPX_JOBNAME=MYJOB')
```

---

446  z/OS V1R11.0 Comm Svrv: IP Programmer’s Guide and Reference
SCMD request

Format:

```
ftpapi(stem, 'scmd', cmd, mode)
```

**Purpose:** Send a subcommand to the FTP client.

**Parameters:**

*stem*

The name of a stem that represents the FTP client environment. The stem must have been passed on a prior successful INIT function call.

*'scmd'*

Requests that a subcommand be sent to the FTP client. The string literal is not case sensitive.

*cmd*

The z/OS FTP subcommand. The string literal is case sensitive.

*mode*

Indicates whether the interface should wait ('W') for the subcommand to complete before returning to the caller, or return immediately ('N') regardless of whether the subcommand has completed. The string literal is not case sensitive.

**Results:**

*Table 22. FTP client SCMD request return codes*

<table>
<thead>
<tr>
<th>Value</th>
<th>Return codes</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0</td>
<td></td>
<td>The call did not complete successfully. The subcommand was not successfully executed. The possible failure return codes are listed in the Return codes column.</td>
</tr>
<tr>
<td>-2 (FCAI_RESULT_IE)</td>
<td>An interface error occurred. The interface error code is stored in the stem.FCAI_IE stem variable.</td>
<td></td>
</tr>
<tr>
<td>-3 (FCAI_RESULT_CEC)</td>
<td>A client error occurred. The client error code is stored in the stem.FCAI_CEC field stem variable.</td>
<td></td>
</tr>
<tr>
<td>-17 (FCAI_RESULT_UNUSABLESTEM)</td>
<td>The stem variable is not usable.</td>
<td></td>
</tr>
<tr>
<td>-18 (FCAI_TASK_TASKMISMATCH)</td>
<td>The task is not the same as the INIT task.</td>
<td></td>
</tr>
<tr>
<td>-19 (FCAI_RESULT_REXXERROR)</td>
<td>An error occurred when storing, fetching, or dropping a REXX variable.</td>
<td></td>
</tr>
<tr>
<td>-32 (FCAI_TASK_CLIPROCESSKILL)</td>
<td>A TERM request issued BPX1KIL to end the client process.</td>
<td></td>
</tr>
<tr>
<td>≥0</td>
<td></td>
<td>The call completed successfully.</td>
</tr>
<tr>
<td>0 (FCAI_RESULT_OK)</td>
<td>The prior request has completed successfully; there are no restrictions on the next subcommand.</td>
<td></td>
</tr>
<tr>
<td>1 (FCAI_RESULT_INPROGRESS)</td>
<td>The prior request is still in-progress.</td>
<td></td>
</tr>
</tbody>
</table>
Table 22. FTP client SCMD request return codes (continued)

<table>
<thead>
<tr>
<th>Value</th>
<th>Return codes</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>(FCAI_RESULT_PROMPTPASS)</td>
<td>The prior request has completed successfully but the next request must be a PASS subcommand.</td>
</tr>
<tr>
<td>3</td>
<td>(FCAI_RESULT_PROMPTACCT)</td>
<td>The prior request has completed successfully but the next request must be an ACCT subcommand.</td>
</tr>
</tbody>
</table>

Examples:

```c
rc = ftpapi('fcai.', 'scmd', 'DIR /tmp/*', 'N')
```
**POLL request**

**Format:**

```
ftpapi(stem, 'poll', pollWait)
```

**Purpose:** Completes and retrieves the results from a prior SCMD request.

**Parameters:**

- **stem**
  The name of a stem that represents the FTP client environment. The stem must have been passed on a prior successful INIT function call and the previous request must have been an SCMD request with the nowait option.

- **'poll’**
  Requests that the program wait for a prior SCMD request to complete and retrieve the results. The string literal is not case sensitive.

- **pollWait**
  Controls the length of time that the REXX program waits before it checks for output. This parameter is equivalent to the FCAI_PollWait field in the FCAI_MAP structure used by the C and callable FTP client APIs. Valid values are any decimal value in the range 0 – 255.

**Results:**

*Table 23. FTP client POLL request return codes*

<table>
<thead>
<tr>
<th>Value</th>
<th>Return codes</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0</td>
<td></td>
<td>The call did not complete successfully. The possible failure return codes are listed in the Return codes column.</td>
</tr>
<tr>
<td>-2 (FCAI_RESULT_IE)</td>
<td>An interface error occurred. The interface error code is stored in the stem.FCAI_IE stem variable.</td>
<td></td>
</tr>
<tr>
<td>-3 (FCAI_RESULT_CEC)</td>
<td>A client error occurred. The client error code is stored in the stem.FCAI_CEC field stem variable.</td>
<td></td>
</tr>
<tr>
<td>-17 (FCAI_RESULT_UNUSABLESTEM)</td>
<td>The stem variable is not usable.</td>
<td></td>
</tr>
<tr>
<td>-18 (FCAI_TASK_TASKMISMATCH)</td>
<td>The task is not the same as the INIT task.</td>
<td></td>
</tr>
<tr>
<td>-19 (FCAI_RESULT_REXXERROR)</td>
<td>An error occurred when storing, fetching, or dropping a REXX variable.</td>
<td></td>
</tr>
<tr>
<td>-32 (FCAI_TASK_CLIPROCESSKILL)</td>
<td>A TERM request issued BPX1KIL to end the client process.</td>
<td></td>
</tr>
<tr>
<td>≥0</td>
<td></td>
<td>The call completed successfully; the next request can be any subcommand.</td>
</tr>
<tr>
<td>0 (FCAI_RESULT_OK)</td>
<td>The prior request has completed successfully; there are no restrictions on the next subcommand.</td>
<td></td>
</tr>
<tr>
<td>1 (FCAI_RESULT_INPROGRESS)</td>
<td>The prior request is still in-progress.</td>
<td></td>
</tr>
</tbody>
</table>
### Table 23. FTP client POLL request return codes (continued)

<table>
<thead>
<tr>
<th>Value</th>
<th>Return codes</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 (FCAI_RESULT_PROMPTPASS)</td>
<td></td>
<td>The prior request has completed successfully but the next request must be a PASS subcommand.</td>
</tr>
<tr>
<td>3 (FCAI_RESULT_PROMPTACCT)</td>
<td></td>
<td>The prior request has completed successfully but the next request must be an ACCT subcommand.</td>
</tr>
</tbody>
</table>

**Examples:**

r = ftpapi('fcai.', 'poll')
GETL_FIND request

Format:

```plaintext
>>-ftpapi-(--stem--,--getl_find--,--lines--
,--'A'--,
,--'N'--)
```

**Purpose:** Finds and returns a line of output that was returned from the FTP client by an INIT, SCMD, or POLL request.

**Parameters:**

*stem*

The name of a stem that represents the FTP client environment. The stem must have been passed on a prior successful INIT function call.

*getl_find*

Requests the return of a single line of output that was returned from the FTP client by an INIT, SCMD, or POLL request. The string literal is not case sensitive.

*lines*

The stem where the output lines are to be copied. The results are:

*lines.0*

The number of output lines returned. This is 1 if the function result is 0, and 0 if the function result is 4.

*lines.id.1*

If the lines.0 field is 1, this contains a 1-character line identifier. Possible values are M, R, T, and L. These are described under the *type* parameter.

*lines.1*

If the lines.0 field is 1, this contains the contents of the output line.

*type*

A 1-character parameter that indicates what type of output is requested. The string literal is not case sensitive. Possible values are:

- **M**  Message from the client
- **R**  Reply from the server
- **L**  List data from a DIR or LS subcommand
- **T**  Trace output from debug or dump routine
- **A**  Any type of output

*sequence*

A 1-character parameter that indicates which of the output lines specified by the *type* parameter to find. The string literal is not case sensitive. Possible values are:

- **F**  Find the first line of the requested type.
- **N**  Find the next line of the requested type.
- **L**  Find the last line of the requested type.

The default value is **N**.
### Table 24. FTP client GETL_FIND request return codes

<table>
<thead>
<tr>
<th>Value</th>
<th>Return codes</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0</td>
<td></td>
<td>The call did not complete successfully. The possible failure return codes are listed in the Return codes column.</td>
</tr>
<tr>
<td>-2</td>
<td>FCAI_RESULT_IE</td>
<td>An interface error occurred. The interface error code is stored in the stem.FCAI_IE stem variable.</td>
</tr>
<tr>
<td>-3</td>
<td>FCAI_RESULT_CEC</td>
<td>A client error occurred. The client error code is stored in the stem.FCAI_CEC field stem variable.</td>
</tr>
<tr>
<td>-17</td>
<td>FCAI_RESULT_UNUSABLESTEM</td>
<td>The stem variable is not usable.</td>
</tr>
<tr>
<td>-18</td>
<td>FCAI_TASK_TASKMISMATCH</td>
<td>The task is not the same as the INIT task.</td>
</tr>
<tr>
<td>-19</td>
<td>FCAI_RESULT_REXXERROR</td>
<td>An error occurred when storing, fetching, or dropping a REXX variable.</td>
</tr>
<tr>
<td>-32</td>
<td>FCAI_TASK_CLIPROCESSKILL</td>
<td>A TERM request issued BPX1KIL to end the client process.</td>
</tr>
<tr>
<td>≥0</td>
<td></td>
<td>The call completed successfully.</td>
</tr>
<tr>
<td>0</td>
<td>FCAI_RESULT_OK</td>
<td>A matching output line was found and returned in the lines stem variable.</td>
</tr>
<tr>
<td>4</td>
<td>FCAI_RESULT_NOMATCH</td>
<td>The prior request has completed successfully but no matching output line was found.</td>
</tr>
</tbody>
</table>

### Examples:

```
rc = ftpapi('fcai.', 'getl_find', 'lines.', 'L')
```
GETL_COPY request

Format:

```
>> ftpapi (--stem--) ('getl_copy' --,'lines' --,'type')
```

Purpose: Finds and returns all lines of the output that was returned from the FTP client by an INIT, SCMD, or POLL request.

Parameters:

*stem*

The name of a stem that represents the FTP client environment. The stem must have been passed on a prior successful INIT function call.

*getl_copy"

Requests the return of all remaining lines of output of type *type* that were returned from the FTP client by an INIT, SCMD, or POLL request. The string literal is not case sensitive.

*lines*

The stem where the output lines are to be copied. The results are:

*lines.0*

The number of output lines returned. This is ≥ 1 if the function result is 0, and 0 if the function result is 4.

*lines.id.*

The 1-character identifier for line *i* of output. Possible values are M, R, T, and L. These are described under the *type* parameter.

*lines.*

The contents of line *i* of output.

*type*

A 1-character parameter that indicates what type of output is requested. The string literal is not case sensitive. Possible values are:

M  Message from the client
R  Reply from the server
L  List data from a DIR or LS subcommand
T  Trace output from debug or dump routine
A  Any type of output

The default value is A.

Results:

Table 25. FTP client GETL_COPY request return codes

<table>
<thead>
<tr>
<th>Value</th>
<th>Return codes</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0</td>
<td></td>
<td>The call did not complete successfully. The possible failure return codes are listed in the Return codes column.</td>
</tr>
</tbody>
</table>
Table 25. FTP client GETL_COPY request return codes (continued)

<table>
<thead>
<tr>
<th>Value</th>
<th>Return codes</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2 (FCAI_RESULT_IE)</td>
<td>An interface error occurred. The interface error code is stored in the stem.FCAI_IE stem variable.</td>
<td></td>
</tr>
<tr>
<td>-3 (FCAI_RESULT_CEC)</td>
<td>A client error occurred. The client error code is stored in the stem.FCAI_CEC field stem variable.</td>
<td></td>
</tr>
<tr>
<td>-17 (FCAI_RESULT_UNUSABLESTEM)</td>
<td>The stem variable is not usable.</td>
<td></td>
</tr>
<tr>
<td>-18 (FCAI_TASK_TASKMISMATCH)</td>
<td>The task is not the same as the INIT task.</td>
<td></td>
</tr>
<tr>
<td>-19 (FCAI_RESULT_REXXERROR)</td>
<td>An error occurred in storing, fetching, or dropping a REXX variable.</td>
<td></td>
</tr>
<tr>
<td>-32 (FCAI_TASK_CLIPROCESSKILL)</td>
<td>A TERM request issued BPX1KIL to end the client process.</td>
<td></td>
</tr>
<tr>
<td>≥0</td>
<td>The call completed successfully.</td>
<td></td>
</tr>
<tr>
<td>0 (FCAI_RESULT_OK)</td>
<td>The matching output lines were found and returned in the lines stem variable.</td>
<td></td>
</tr>
<tr>
<td>4 (FCAI_RESULT_NOMATCH)</td>
<td>The prior request has completed successfully but no matching output lines were found.</td>
<td></td>
</tr>
</tbody>
</table>

Examples:

rc = ftpapi('fcai.', 'getl_copy', 'lines.', 'A')
SET_TRACE request

Format:

```plaintext
ftpapi( stem , 'set_trace' , traceit )
```

**Purpose:** Enables or disables the tracing of subsequent FTP client API calls. After each subsequent call, you can examine the FCAI_TraceStatus variable to determine whether the tracing of that specific call was successful.

**Parameters:**

- **stem**
  The name of a stem that represents the FTP client environment. The stem must have been passed on a prior successful INIT function call.

- **set_trace**
  Indicates whether the trace should be enabled or disabled on subsequent FTP client API calls. The string literal is not case sensitive.

- **traceit**
  Sets the status of the FTP client API trace. The string literal is not case sensitive. Possible values are:

  - **ON**
    Begin tracing FTP client API calls.
  - **OFF**
    Stop tracing FTP client API calls.

**Results:**

*Table 26. FTP client SET_TRACE request return codes*

<table>
<thead>
<tr>
<th>Value</th>
<th>Return codes</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0</td>
<td></td>
<td>The call did not complete successfully. The possible failure return codes are listed in the Return codes column.</td>
</tr>
<tr>
<td>-2 (FCAI_RESULT_IE)</td>
<td>An interface error occurred. The interface error code is stored in the stem.FCAI_IE stem variable.</td>
<td></td>
</tr>
<tr>
<td>-17 (FCAI_RESULT_UNUSABLESTEM)</td>
<td>The stem variable is not usable.</td>
<td></td>
</tr>
<tr>
<td>-19 (FCAI_RESULT_REXXERROR)</td>
<td>An error occurred when storing, fetching, or dropping a REXX variable.</td>
<td></td>
</tr>
<tr>
<td>≥0</td>
<td></td>
<td>The call completed successfully.</td>
</tr>
<tr>
<td>0 (FCAI_RESULT_OK)</td>
<td>No additional information is provided.</td>
<td></td>
</tr>
</tbody>
</table>

**Examples:**

```plaintext
rc = ftpapi('fcai.', 'set_trace', 'ON')
```
**SET_REQUEST_TIMER request**

**Format:**

```text
>>> ftpapi(‘stem’, ‘set_request_timer’, ‘reqtimer’)
```

**Purpose:** Sets the length of time that the FTP client API waits for an INIT, SCMD, or TERM request to complete.

**Parameters:**

- `stem`  
  The name of a stem that represents the FTP client environment. The stem must have been passed on a prior successful INIT function call.

- `'set_request_timer'`  
  Requests that the request timer be set for subsequent INIT, SCMD, or TERM requests. The string literal is not case sensitive.

- `reqtimer`  
  The number of seconds to wait for the request to complete. Valid values are in the range 0 – 256. The value corresponds to the FCAI_RqTimer field used by the C and callable FTP client APIs.

**Results:**

*Table 27. FTP client SET_REQUEST_TIMER request return codes*

<table>
<thead>
<tr>
<th>Value</th>
<th>Return codes</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0</td>
<td></td>
<td>The call did not complete successfully. The possible failure return codes are listed in the Return codes column.</td>
</tr>
<tr>
<td>-2</td>
<td>(FCAI_RESULT_IE)</td>
<td>An interface error occurred. The interface error code is stored in the <code>stem.FCAI_IE</code> stem variable.</td>
</tr>
<tr>
<td>-17</td>
<td>(FCAI_RESULT_UNUSABLESTEM)</td>
<td>The stem variable is not usable.</td>
</tr>
<tr>
<td>-19</td>
<td>(FCAI_RESULT_REXXERROR)</td>
<td>An error occurred when storing, fetching, or dropping a REXX variable.</td>
</tr>
<tr>
<td>≥0</td>
<td></td>
<td>The call completed successfully.</td>
</tr>
<tr>
<td>0</td>
<td>(FCAI_RESULT_OK)</td>
<td>No additional information is provided.</td>
</tr>
</tbody>
</table>

**Examples:**

```text
rc = ftpapi(‘fcai.’, ‘set_request_timer’, 1)
```
GET_FCAI_MAP request

Format:

```
>>> ftpapi(--stem--,--'get_fcai_map'--,--fcaimap--) <<<
```

Purpose: Returns the contents of the full FCAI_Map structure.

Parameters:

*stem*  
The name of a stem that represents the FTP client environment. The stem must have been passed on a prior successful INIT function call.

'*get_fcai_map'*  
Requests the return of the full FCAI_Map structure. The string literal is not case sensitive.

*fcaimap*  
The name of the stem where the FCAI_Map structure should be placed. The map elements shown in Table 28 can be returned.

**Table 28. FCAI_Map structure elements**

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fcaiMap.FCAI_EyeCatcher</td>
<td>Eyecatcher='FCAI'</td>
</tr>
<tr>
<td>fcaiMap.FCAI_Size</td>
<td>Size of the FCAI=256</td>
</tr>
<tr>
<td>fcaiMap.FCAI_Version</td>
<td>Version of the FCAI=1</td>
</tr>
<tr>
<td>fcaiMap.FCAI_PollWait</td>
<td>POLL wait timer in seconds</td>
</tr>
<tr>
<td>fcaiMap.FCAI_ReqTimer</td>
<td>Request timer in seconds or 0 for none</td>
</tr>
<tr>
<td>fcaiMap.FCAI_TraceIt</td>
<td>Trace indicator for this request</td>
</tr>
<tr>
<td>fcaiMap.FCAI_TraceID</td>
<td>ID used in a trace record</td>
</tr>
<tr>
<td>fcaiMap.FCAI_TraceCAPI</td>
<td>TRACECAPI value on FTP.DATA statement</td>
</tr>
<tr>
<td>fcaiMap.FCAI_TraceStatus</td>
<td>Status of the trace</td>
</tr>
<tr>
<td>fcaiMap.FCAI_TraceSClass</td>
<td>SYSOUT class for trace file</td>
</tr>
<tr>
<td>fcaiMap.FCAI_TraceName</td>
<td>ddname of the trace file</td>
</tr>
<tr>
<td>fcaiMap.FCAI-Token</td>
<td>Interface token</td>
</tr>
<tr>
<td>fcaiMap.FCAI_RequestID</td>
<td>Last request processed by the EZAFTPKS program (for example, 'SCMD')</td>
</tr>
<tr>
<td>fcaiMap.FCAI_Result</td>
<td>Request result</td>
</tr>
<tr>
<td>fcaiMap.FCAI_IE</td>
<td>Interface error</td>
</tr>
<tr>
<td>fcaiMap.FCAI_CEC</td>
<td>Client error code (see FTP return codes in z/OS Communications Server: IP User’s Guide and Commands)</td>
</tr>
<tr>
<td>fcaiMap.FCAI_ReplyCode</td>
<td>Server reply code or 0 if no reply (see FTPD reply codes in z/OS Communications Server: IP and SNA Codes)</td>
</tr>
<tr>
<td>fcaiMap.FCAI_SCMD</td>
<td>Subcommand code (see FTP subcommand codes in z/OS Communications Server: IP User’s Guide and Commands)</td>
</tr>
<tr>
<td>fcaiMap.FCAI_ReturnCode</td>
<td>Return code</td>
</tr>
</tbody>
</table>
Table 28. FCAI_Map structure elements (continued)

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fcaiMap.FCAI_ReasonCode</td>
<td>Reason code</td>
</tr>
<tr>
<td>fcaiMap.FCAI_NumberLines</td>
<td>Number of output lines returned by the request</td>
</tr>
<tr>
<td>fcaiMap.FCAI_LongestLine</td>
<td>Size of the longest line</td>
</tr>
<tr>
<td>fcaiMap.FCAI_SizeAll</td>
<td>Size of all output lines</td>
</tr>
<tr>
<td>fcaiMap.FCAI_SizeMessages</td>
<td>Size of all message lines</td>
</tr>
<tr>
<td>fcaiMap.FCAI_SizeReplies</td>
<td>Size of all reply lines</td>
</tr>
<tr>
<td>fcaiMap.FCAI_SizeList</td>
<td>Size of all list lines</td>
</tr>
<tr>
<td>fcaiMap.FCAI_SizeTrace</td>
<td>Size of all trace lines</td>
</tr>
<tr>
<td>fcaiMap.FCAI_PID</td>
<td>Process ID of FTP client</td>
</tr>
</tbody>
</table>

Results:

Table 29. FTP client GET_FCAI_MAP request return codes

<table>
<thead>
<tr>
<th>Value</th>
<th>Return codes</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0</td>
<td></td>
<td>The call did not complete successfully. The possible failure return codes are listed in the Return codes column.</td>
</tr>
<tr>
<td>-2</td>
<td>(FCAI_RESULT_IE)</td>
<td>An interface error occurred. The interface error code is stored in the stem.FCAI_IE stem variable.</td>
</tr>
<tr>
<td>-17</td>
<td>(FCAI_RESULT_UNUSABLESTEM)</td>
<td>The stem variable is not usable.</td>
</tr>
<tr>
<td>-19</td>
<td>(FCAI_RESULT_REXXERROR)</td>
<td>An error occurred when storing, fetching, or dropping a REXX variable.</td>
</tr>
<tr>
<td>≥0</td>
<td></td>
<td>The call completed successfully.</td>
</tr>
<tr>
<td>0</td>
<td>(FCAI_RESULT_OK)</td>
<td>No additional information is provided.</td>
</tr>
</tbody>
</table>

Examples:

rc=ftpapi('fcai.', 'get_fcai_map', 'fcaiMap.')
TERM request

Format:

```
ftpapi("stem", "term")
```

**Purpose:** The user program issues the TERM request to terminate this instance of interface use. A TERM request is accepted by the interface at any time.

**Rule:** Each successful invocation of INIT should be terminated with a corresponding invocation of TERM.

**Parameters:**

- `stem`  
  The name of a stem that represents the FTP client environment. The stem must have been passed on a prior successful INIT function call.

- `'term'`  
  Requests the termination of this instance of the FTP interface. The string literal is not case sensitive.

**Results:**

*Table 30. FTP client TERM request return codes*

<table>
<thead>
<tr>
<th>Value</th>
<th>Return codes</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0</td>
<td></td>
<td>The call did not complete successfully. The possible failure return codes are listed in the Return codes column.</td>
</tr>
<tr>
<td>-2</td>
<td>FCAI_RESULT_IE</td>
<td>An interface error occurred. The interface error code is stored in the stem.FCAI_IE stem variable.</td>
</tr>
<tr>
<td>-3</td>
<td>FCAI_RESULT_CEC</td>
<td>A client error occurred. The client error code is stored in the stem.FCAI_CEC field stem variable.</td>
</tr>
<tr>
<td>-17</td>
<td>FCAI_RESULT_UNUSABLESTEM</td>
<td>The stem variable is not usable.</td>
</tr>
<tr>
<td>-18</td>
<td>FCAI_TASK_TASKMISMATCH</td>
<td>The task is not the same as the INIT task.</td>
</tr>
<tr>
<td>-19</td>
<td>FCAI_RESULT_REXXERROR</td>
<td>An error occurred when storing, fetching, or dropping a REXX variable.</td>
</tr>
<tr>
<td>-32</td>
<td>FCAI_TASK_CLIPROCESSKILL</td>
<td>A TERM request issued BPX1KIL to end the client process.</td>
</tr>
<tr>
<td>≥0</td>
<td></td>
<td>The call completed successfully.</td>
</tr>
<tr>
<td>0</td>
<td>FCAI_RESULT_OK</td>
<td>No additional information is provided.</td>
</tr>
</tbody>
</table>

**Examples:**

```
rc = ftpapi('fcai.', 'term')
```
FTP client API for REXX trace return codes

Table 31 describes the return codes that can be stored in the FCAL_TRACE_DATASET_RETCODE variable.

Table 31. FTP client API for REXX trace return codes

<table>
<thead>
<tr>
<th>Code type</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>0</td>
<td>No errors detected with the FTP client API for REXX trace</td>
</tr>
<tr>
<td>Data set allocation error</td>
<td>65537 (X'10001')</td>
<td>Insufficient storage</td>
</tr>
<tr>
<td>Data set allocation error</td>
<td>65538 (X'10002')</td>
<td>DD name not valid</td>
</tr>
<tr>
<td>Data set allocation error</td>
<td>65539 (X'10003')</td>
<td>Data set in use</td>
</tr>
<tr>
<td>Data set allocation error</td>
<td>65540 (X'10004')</td>
<td>Bad DSorg value</td>
</tr>
<tr>
<td>Data set allocation error</td>
<td>65542 (X'10006')</td>
<td>No DD name returned by DYNALLOC</td>
</tr>
<tr>
<td>Data set allocation error</td>
<td>65543 (X'10007')</td>
<td>PDS is not supported</td>
</tr>
<tr>
<td>Data set allocation error</td>
<td>131072–196607 (X'20000'–X'2FFFF')</td>
<td>DYNALLOC failure. The lower 16 bits contain the SVC99ERROR value; see the z/OS MVS Diagnosis Reference</td>
</tr>
<tr>
<td>Open data set error</td>
<td>262145 (X'40001')</td>
<td>Data set characteristics not valid</td>
</tr>
<tr>
<td>Open data set error</td>
<td>262146 (X'40002')</td>
<td>OPEN failure</td>
</tr>
<tr>
<td>Data set deallocation error</td>
<td>524288 (X'80000')</td>
<td>DYNALLOC failure</td>
</tr>
<tr>
<td>Write failure</td>
<td>786433 (X'C0001')</td>
<td>SYNAD error</td>
</tr>
<tr>
<td>Write failure</td>
<td>786434 (X'C0002')</td>
<td>End of data set</td>
</tr>
</tbody>
</table>

Output register information for the FTP client API

For the Cobol, C, assembler, and PL/I programming languages, when control returns to the caller, the general purpose registers (GPRs) contain the following:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Used to contain a return code</td>
</tr>
</tbody>
</table>

Restriction: The REXX programming language does not use output registers to return information to the REXX program. For information about REXX program output, see “FTP client API for REXX function” on page 439.

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-15</td>
<td>Unchanged</td>
</tr>
</tbody>
</table>
If a caller depends on the contents of a register that is to be used by the system, the caller must save the contents of that register before invoking the interface and restore the contents after the system returns control.

FTP client API: Other output that is returned to the application

After a request drives processing in the client, the client sends output to the interface. The interface retrieves this output automatically for INIT and SCMD requests issued in wait mode (or an SCMD QUIT request generated by a TERM request). An SCMD request that is issued in no-wait mode retrieves no output and an SCMD request that fails to complete before FCAI_ReqTimer expires might or might not retrieve partial output. The caller must retrieve the results for an SCMD request that returns a value of FCAI_Status_InProgress with the POLL request. See “FTP client API messages and replies” on page 463 for a discussion of the final message, EZA2121I.

When the output is retrieved, the interface stores it in the interface buffer. This output can be retrieved by the user program with a GETL request from the stem variables set using a GET_FCAI_MAP request (see “GET_FCAI_MAP request” on page 457). For example, for the request FtpApi("fcai.", "get_fcai_map", "fcaimap."), the total number of lines of output are stored in the stem variable FCAI_NumberLines. The output is composed of the following four types of lines:

- Messages generated by the FTP client
- Replies sent from the FTP server
- List data from the FTP server for a DIR or LS subcommand
- Trace data for the FTP client

See “Example of the GETL call instruction” on page 429, which shows message, reply, list, and trace output lines. See “FTP client API messages and replies” on page 463 for additional information about the difference between messages and replies.

The FCAI_PID value is updated after a successful INIT request and is cleared after a TERM request. The following statistics in the FCAI are updated for each complete INIT or SCMD request, and for each POLL request until the prior SCMD request completes. A GETL request does not change the statistics because it retrieves only the output that has already been copied to the interface buffer from the client. A TERM request terminates the interface and frees the interface buffer; it returns no output.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCAI_NumberLines</td>
<td>Total number of lines of output.</td>
</tr>
<tr>
<td>FCAI_LongestLine</td>
<td>Size of the longest line of output. Use this value to acquire or define a space into which a line of output can be copied.</td>
</tr>
<tr>
<td>FCAI_SizeAll</td>
<td>Total size of all lines of output.</td>
</tr>
<tr>
<td>FCAI_SizeMessages</td>
<td>Size of all lines of messages from the client.</td>
</tr>
<tr>
<td>FCAI_SizeReplies</td>
<td>Size of all lines of replies from the server.</td>
</tr>
<tr>
<td>FCAI_SizeList</td>
<td>Size of all lines of data from a DIR or LS subcommand.</td>
</tr>
<tr>
<td>FCAI_SizeTrace</td>
<td>Size of all lines of trace data.</td>
</tr>
</tbody>
</table>
The output data is available for retrieval with a GETL request until the next SCMD or TERM request is processed. The next SCMD request clears all status fields for the new request and reuses the space allocated to the interface buffer. You should use a GETL request with the COPY operation if you want to preserve the output.

Prompts from the client

Some z/OS FTP client subcommands cause the client to prompt the user for more input. For example, processing the USER subcommand causes the client to prompt for a password after sending a USER command to the FTP server. Prompt behavior changes when the z/OS FTP client is invoked from the FTP client API.

As shown in “Parameter values that are returned to the application” on page 424, some prompts affect status codes returned by the interface. See “Interpreting results from an interface request” on page 464 for more information about status codes. The following sections describe the current prompt situations for the FTP client and how they are handled by the FTP client API.

Prompts not used by the FTP client API

- Prompt for IP address if not supplied as a parameter on the INIT request
  The FTP client prompts immediately if an IP address or host name was not supplied at client start. The FTP client API does not pass this prompt to the user program. The user program should use an SCMD request to send an OPEN subcommand as soon as it wants a session with the FTP server.

- Prompt for a user ID after an OPEN subcommand
  The FTP client prompts for a user ID for login after the session is set up with the server. The FTP client API does not pass this prompt to the user program. The user program should use an SCMD request to send a USER subcommand as soon as it wants to log in with the FTP server. The user program can provide the password as well as the user ID as parameters with the USER subcommand.

- Prompt for a subcommand after a PROXY subcommand
  The FTP client prompts for a subcommand if the PROXY subcommand is entered without a subcommand parameter. The FTP client API does not support PROXY without a subcommand. If the client is invoked by the FTP client API and receives PROXY without a subcommand, the request fails with FCAI-Result = FCAI-Result-CEC and FCAI-CEC = FCAI-CEC-PROXY-ERR.

  **Requirement:** Every PROXY subcommand that is issued to the FTP client API must be in the following format: PROXY subcommand <optional parameters>. This requirement includes the PASS subcommand, which must be entered as PROXY PASS password (if the password was not included on the prior USER subcommand, as in PROXY USER userid password). See the PROXY subcommand information in the z/OS Communications Server: IP User's Guide and Commands for an explanation of the PROXY subcommand and its parameters.

- Prompt for confirmation for MGET, MPUT, and MDELETE subcommands
  The FTP client prompts for confirmation for these subcommands if the PROMPT subcommand has toggled to prompting. The FTP client API does not pass this prompt to the user program. The subcommand is executed in no-prompt mode.

Prompts returned in FCAI-Status

- Prompt for password after a USER subcommand
  The FTP client prompts for a password to complete a login if the password was not passed with the USER subcommand. The FTP client API passes this prompt
to the user program by using the status FCAI-Status-PromptPass. The user program should use an SCMD request to send a PASS subcommand as the next subcommand. If any subcommand other than PASS is sent, the request fails with FCAI-Result = FCAI-Result-IE and FCAI-IE = FCAI-PassPromptErr.

- Prompt for accounting information after a PASS or a CD (CWD) subcommand

Some FTP servers prompt the FTP client for accounting information after a PASS or CWD command is processed. The FTP client API passes this prompt to the user program by using the status FCAI-Status-PromptAcct. The user program should use an SCMD request to send an ACCT subcommand as the next subcommand. If any subcommand other than ACCT is sent, the request fails with FCAI-Result = FCAI-Result-IE and FCAI-IE = FCAI-AcctPromptErr.

Tip: When a PASS or ACCT subcommand is expected, the interface refuses any other SCMD request until the prompt is satisfied. The user program can issue a GETL or TERM request without satisfying the prompt. A TERM request generates a QUIT subcommand that is accepted and stops the client process.

**FTP client API command prompt**

Each subcommand that is sent to the FTP client ends with an output message that is a prompt for the next subcommand. This message is the last of the messages that are returned as output; it can be retrieved with the GETL request for the last message. See “FTP client API messages and replies” for an example of EZA2121I, which is the command prompt used by the FTP client API.

**FTP client API messages and replies**

Messages are information statements that are provided by the FTP client. Replies are the responses to commands that are returned from the FTP server to the client. Replies are described in [FTPD reply codes](z/OS Communications Server: IP and SNA Codes).

Messages are composed of a message ID followed by message text. FTP client messages are described in [z/OS Communications Server: IP Messages Volume 1 (EZA)] and [z/OS Communications Server: IP Messages Volume 4 (EZZ, SNM)]. You can use the message ID to look up the message in these volumes. However, the message IDs are not written to output unless the client is executing in verbose mode.

Following is an example that uses the `verbose` subcommand in an interactive environment. The FTP client API processes the subcommand on an SCMD request as previously described. Note that the `verbose` subcommand acts as a toggle.

```
Verbose
EZA2859I Message IDs are displayed when running in z/OS UNIX
EZA1460I Command: verbose
VERBOSE
Message IDs are not displayed when running in z/OS UNIX
Command:
```

EZA2859I and EZA1460I (the first token that is displayed when executing in verbose mode) are message IDs. The phrases that follow the message ID are message text. Notice that after the `verbose` command is executed the second time to toggle verbose off, the message texts appear with no message ID. See the [Verbose subcommand](z/OS Communications Server: IP User’s Guide and Commands) for more information about entering and exiting verbose mode.
The FTP client API does not use the EZA1460I Command: message. Instead, it uses a new message with additional status information about the subcommand that completed. The syntax is:

\texttt{EZA2121I Command (ee-ss-cccc-rrr):}

where:

- \texttt{ee} is the 2-digit decimal client error code for the subcommand (00 if no error)
- \texttt{ss} is the 2-digit decimal subcommand code (the field is blank on INIT when an implicit OPEN was not performed)
- \texttt{cccc} is the final 4-character FTP command sent to the server by the subcommand (all blanks if no command was sent)
- \texttt{rrr} is the last 3-character server reply to the final FTP command (all blanks if no reply was received)

Replies are composed of a 3-digit numeric reply code followed by text. The significance of the reply prefix is described in RFC 959, \textit{File Transfer Protocol} (see Appendix E, “Related protocol specifications,” on page 743 for information about accessing RFCs); replies used by the z/OS FTP server are described in \textit{z/OS Communications Server: IP and SNA Codes}. Unlike message numbers, reply codes are never suppressed. Your program can usually disregard the text and inspect the reply code to determine whether the server processed the FTP command successfully. The one exception is when an FTP client sends the SITE command to a z/OS FTP server. The z/OS FTP server SITE reply is always 200 (implying success), even when one or more errors occurred when processing the SITE command, as in the following example:

\begin{verbatim}
site bogus
>>> SITE bogus
200 -Unrecognized parameter 'bogus' on SITE command.
200 SITE command was accepted
Command(00-34-SITE-200):
\end{verbatim}

The following sample output from the CD subcommand shows messages EZA1701I and EZA2121I from the FTP client and the reply 250 from the FTP server. Note that the CD subcommand causes the client to send the CWD command to the server.

\begin{verbatim}
CD /u/user33/
EZA1701I >>> CWD /u/user33/
250 HFS directory /u/user33/ is the current working directory
EZA2121I Command(00-07-CWD-250):
verbose
Message IDs are not displayed when running in z/OS UNIX
Command(00-71- - ):
CD /u/user33/
>>> CWD /u/user33/
250 HFS directory /u/user33/ is the current working directory
Command(00-07-CWD-250):
\end{verbatim}

**Interpreting results from an interface request**

The results of a request to the FTP Client Application Interface are reported to the user program in the FCAI control block. See “FTP Client Application Interface (FCAI) control block” on page 404.

**FCAI request completion values**

The following list describes how to interpret the contents of FCAI Result and how to use the secondary results fields.
If FCAI_Result contains . . .  Then . . .  Explanation

<table>
<thead>
<tr>
<th>FCAI_Result_OK</th>
<th>no further action is required.</th>
<th>The request is complete with no additional status or errors to report.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCAI_Result_Status</td>
<td>check the FCAI_Status field for additional status information.</td>
<td>The interface uses the status field to report prompts from the client, to report an indication that the last request is still in-progress, and to report an error in the interface trace function.</td>
</tr>
<tr>
<td>FCAI_Result_IE</td>
<td>check the FCAI_IE field for error information.</td>
<td>The request returned an interface error, which indicates that the interface was unable to process the request for some reason. That reason might be a parameter on the call that was not valid, a failure in a service routine (such as GETMAIN), or termination of the client process.</td>
</tr>
<tr>
<td>FCAI_Result_CEC</td>
<td>check the FCAI_CEC field for the error code that was returned by the client. You can optionally issue GETL to retrieve diagnostic information (see &quot;GETL&quot; on page 427).</td>
<td>The request completed with an FTP Client Error Code. For information about Client Error Codes and diagnosing errors in the client and server, see FTP return codes in the z/OS Communications Server: IP User’s Guide and Commands and Diagnosing FTP problems in the z/OS Communications Server: IP Diagnosis Guide.</td>
</tr>
</tbody>
</table>

Other values returned in FCAI_Result are listed along with their descriptions in “FTP Client Application Interface (FCAI) control block” on page 404.

Tips:
1. The value in FCAI_Result is returned to the caller in the return code register. Additionally, callers in COBOL, C, and PL/I can access the return code value in the FCAI_ReturnCode field and the reason code value in the FCAI_ReasonCode field.
2. The result code FCAI_Result_UnusableFCAI is returned only in the return code register.
3. Always verify that the return code register is set to 0 before inspecting the FCAI.

For all requests that communicate with the FTP server, the field FCAI_ReplyCode reports the last reply that was received from the final server command sent to the FTP server by the client for the request. The FCAI_ReplyCode field is a binary field with a length of 2. For example, a 250 reply from the server is recorded in the field as X'00FA'. A value of all zeros indicates that the client did not communicate with the FTP server. The following conditions apply to the FCAI_ReplyCode field.
- GETL and TERM requests do not populate the FCAI_ReplyCode field.
- An INIT request populates this field only if an implicit OPEN was performed.
- SCMD requests for locally processed subcommands such as LOCSITE and LPWD do not communicate with the FTP server. Those subcommands, as well as subcommands that fail before sending a command to the server, do not populate this field.
• A POLL request might or might not populate this field, depending on the prior subcommand that was in-progress.

Requests that initiate a subcommand in the z/OS FTP client return a value in the FCAI_SCMD field. This field is set by all SCMD requests and by the INIT request when start parameters that were passed on the request generate an implicit OPEN subcommand.

FCAI_ReturnCode and FCAI_ReasonCode values are set when certain services fail. See “FTP Client Application Interface (FCAI) control block” on page 404 as well as “Programming notes for the FTP client API” for more information.

Considerations when evaluating request completion values

• The request completion values FCAI_CEC and FCAI_ReplyCode are available when the request completes. An FCAI_SCMD value is available when the subcommand completes but might also be set by a POLL request prior to completion of the subcommand. See “POLL” on page 425 and the MODE parameter definition in “SCMD” on page 422 for more information about the completion of subcommands.

• After a valid PROXY subcommand, the FCAI_SCMD value that is returned and the value in message EZA2121I reflect the client subcommand that was passed as a parameter with PROXY. The server command (if any) that was sent to the PROXY server appears in message EZA2121I. For example, PROXY DIR returns the subcommand code for DIR in FCAI_SCMD. Message EZA2121I displays the DIR subcommand code and the LIST server command.

• If a failure occurs in the interface trace function while processing the request, FCAI_Status_TraceFailed is added to any other status value that is returned in FCAI_Status. FCAI_Result contains FCAI_Result_Status unless there was a concurrent error on the request. See “FCAI_Status_TraceFailed and FCAI_TraceStatus: Reporting failures in the interface trace function” on page 467 for more information.

• When a failure occurs in the interface with the client, all subsequent requests except GETL and TERM return FCAI_IE_CliProcessBroken.

• When the client processes a QUIT subcommand, all subsequent requests except GETL and TERM return FCAI_IE_CliProcessStopped.

• Prior to a successful INIT, any request other than INIT returns FCAI_IE_NoTokenAddr.

• FCAI_Result_CliProcessKill is an informational code returned only by a TERM request. See “TERM” on page 433.

Programming notes for the FTP client API

The following sections contain additional information about the following:

• FCAI_Status_TraceFailed
• FCAI_TraceStatus
• FCAI_IE_LengthInvalid
• FCAI_ReqTimer
• FCAI_PollWait
• FCAI_IE_InternalErr

There is also a discussion of exceptional conditions in the z/OS FTP client process.
FCAI_Status_TraceFailed and FCAI_TraceStatus: Reporting failures in the interface trace function

- A failure in the interface trace function is not treated as a severe error; if possible, processing continues for the request.
- The interface trace function is disabled when it encounters an error and cannot be restarted for this instance of use of the interface.
- The additional information described in Table 13 on page 407 is available only immediately after the request that returns FCAI_Status_TraceFailed. A concurrent error on the request can overwrite the additional information fields.
- When a request returns FCAI_Status_TraceFailed and sets a reason of FCAI_TraceStatus_AllocErr (error in dynamic allocation), the additional information can be interpreted as follows:
  - The return code, upon completion of the call to the interface, does not contain the return code from dynamic allocation. The return code from dynamic allocation is not reported except as noted in the following item.
  - The FCAI_ReturnCode field contains S99ERROR or the number 8.
    - If the FCAI_ReturnCode field contains the value 8, the dynamic allocation return code was 8. This return code means an installation validation routine failed the request and S99ERROR is not available.
  - The FCAI_ReasonCode field contains S99ERSN for DFSMS™ failures. For all other cases, it contains S99INFO.
    - S99ERSN is reported in FCAI_ReasonCode for all S99ERROR values that begin with X'97'.
    - S99INFO can be returned when no error was reported on the allocation. In those cases, the FCAI_ReasonCode field contains the S99INFO value even though no other results fields are set.
    - S99INFO can be (and often is) 0 when an error is reported. Only certain conditions set S99INFO.
- FCAI_Result contains FCAI_Result_Status when the trace encounters a failure unless a concurrent error sets FCAI_Result to a higher value.
- The request that disables the interface trace function adds FCAI_Status_TraceFailed to any other value that is returned in FCAI_Status. When FCAI_Status_TraceFailed is subtracted from FCAI_Status, the result is one of the FCAI_Status values or 0. The result that remains in FCAI_Status is described in Table 15 on page 408.
- FCAI_TraceStatus always reflects the current status of the interface trace function and, if applicable, the reason it was disabled. FCAI_TraceStatus field values are described in Table 13 on page 407.

FCAI_IE_LengthInvalid: Improper lengths passed to the interface

- On an INIT request, a length value less than 0 was passed for the optional start parameters. The value 0 is accepted and bypasses sending start parameters.
- On an INIT request, the list of environment variables contained a negative count word or length.
- On a GETL request, a length value less than or equal to 0 was passed in the buffer vector.
  Tip: Values greater than zero but insufficient to hold the first line of selected output return FCAI_IE_BufferTooSmall.
• On an SCMD request, the subcommand string length value is less than or equal to 0. A subcommand string is required on the request.

**FCAI_ReqTimer: Controlling requests that retrieve results from the spawned z/OS FTP client process**

• FCAI_ReqTimer is used to limit the time the interface attempts to retrieve results from INIT and SCMD requests issued in wait mode, and TERM that automatically generates SCMD QUIT. (POLL reads data from the client but accepts only what has been written and returns immediately; GETL does not read from the client.)
• FCAI_ReqTimer is an unsigned, 1-byte hexadecimal value in the range 1-255 that indicates the number of seconds to wait for the request to complete. The value 0 means not to use a timer on the request (wait until completion).
  – FCAI_ReqTimer is approximate.
  – FCAI_ReqTimer is not related to performance. Using a low value does not improve response time on the request. It is intended only to prevent the interface from polling a non-responsive client process indefinitely.
  – FCAI_ReqTimer is ignored by POLL, GETL, SCMD issued in no-wait mode, and TERM that does not generate a QUIT subcommand.
• If the interface detects that the client process is no longer there, it returns FCAI_IE_CliProcessBroken on any request except TERM requests. When FCAI_ReqTimer expires, the client process is still there but was unable to return all the results in the time limit that was specified. The application must determine how to respond when FCAI_ReqTimer expires.
• Timer expiration during INIT returns FCAI_Result_IE in FCAI_Result and FCAI_IE_ReqTimerExpired in FCAI_IE. INIT is the only request that returns this interface error. The error indicates that the interface failed to initialize.
• Timer expiration during a QUIT generated by TERM causes TERM to return FCAI_Result_CliProcessKill.
• Timer expiration on an SCMD issued in wait mode returns FCAI_Result_Status in FCAI_Result and FCAI_Status_InProgress in FCAI_Status. At that point the behavior of the interface is the same as if the SCMD had been issued in no-wait mode. See “SCMD” on page 423 and “POLL” on page 425 for more information about no-wait processing.

**FCAI_PollWait: Specifying a wait time before POLL**

To assist the application in managing POLL requests, the FTP client API automatically pauses before it reads from the pipe to the client process. This wait suspends the interface and the application and it protects the application from sending POLL requests at a rate that degrades performance.

FCAI_PollWait is used as follows:
• The value 0 (the default setting) in FCAI_PollWait instructs a POLL request to wait 1 second prior to reading the pipe from the client.
• A value greater than 0 enables a progressive wait timer and sets the maximum number of seconds that will be used. The field accepts a value in the range 0–255 seconds (4.25 minutes), but a value of 32 seconds or less is recommended for most subcommands.
• The current timer value is stored internally in the interface and persists between POLL requests that are issued for the same prior subcommand. If a progressive timer has not been enabled, the current timer value is always 1 second. A
progressive wait timer begins at 1 second and doubles after each POLL until the maximum setting is reached or exceeded. If exceeded, the timer value is set to the value that was supplied by the user.

- The application can enable or disable the progressive timer and increase or decrease the maximum value to use on any POLL, regardless of the current timer value. The interface checks prior to each POLL to ensure that the timer does not exceed the specified maximum (the value 0 sets the maximum to 1 second).
- The current timer value resets to 1 second after a POLL receives data from the pipe. This enables the user to retrieve all available output efficiently and removes much of the application's burden of managing a progressive timer. If a progressive timer has been requested, it begins to progress again as no data is returned (unless all output has been retrieved).
- The application is responsible for sending each POLL request. POLL requests are not generated automatically by the interface. The PollWait interval is in addition to any wait done in the application program.

**FCAI_IE_InternalErr: Unanticipated exceptional conditions in the interface**

Examples of conditions that return FCAI_IE_InternalErr are:

- A linked interface buffer part was expected but none was found.
- A length field that was not valid was detected in the interface buffer.
- A computation during buffer navigation unexpectedly resulted in a negative value.

Conditions of this type indicate a logic error, storage overlay, or some other unrecoverable error in the interface. The interface accepts only a TERM request after the error.

To diagnose the error, dump all storage in the application address space as soon as the error code is returned. Contact the IBM support center for assistance if needed.

**Guideline:** If the application remains active after the storage dump, issue TERM to kill the client process and then terminate the application.

**Exceptional conditions in the z/OS FTP client**

- The following failures in the FTP client are considered fatal by the FTP client API.
  - If the FTP client process experiences an abnormal termination, the next request that attempts to read from the client detects the condition and returns FCAI_IE_CliProcessBroken.
  - If an error is encountered while spawning the FTP client, establishing pipes to the client, or communicating with the client, the interface error that describes the condition is returned to the application.

If you need assistance in diagnosing these failures, contact the IBM support center.

- If the FTP client stops responding to the interface during INIT or SCMD processing (including a QUIT subcommand generated by TERM) and FCAI_ReqTimer is 0 (wait until completion), the application waits indefinitely for the client. Dump all the storage in the address spaces for the interface and the spawned FTP client process, cancel the application, and kill the FTP client process if necessary. Contact the IBM support center if you need assistance.
• When a request returns FCAI_Status_InProgress because it exceeded the FCAI_ReqTimer value, the application must ascertain whether this is an error condition and how to proceed.

• If a signal is raised by a service invoked by the FTP client process, it is generally handled within the FTP client and reported to the application by a Client Error Code when appropriate.

• If a signal is raised on behalf of the client (that is, when the child process ends), neither the client nor the FTP client API blocks or handles the signal.

• If the interface must issue BPX1KIL to kill the FTP client process during TERM, FCAI_Result_CliProcessKill is returned to inform the application. No further action is required.

Using the FTP client API trace

The FTP client API trace is used to debug problems in the interface or record activity and data that are returned to the interface that might not otherwise be available to the application (see “TERM” on page 433). The interface trace cannot be used to debug errors in the trace itself or any error that prevents the interface from accessing the trace data set. The trace writes records for interface events, which include requests to the interface, the results of interface requests, and output from the client. Client output includes client messages, server replies, list data, and DEBUG and DUMP trace data that is received from the client.

The FTP client API trace is controlled by a statement that is coded in the FTP.DATA file for the FTP client. The statement is named TRACECAPI and is described in z/OS Communications Server: IP Configuration Reference.

Tip: The FTP client API trace does not include trace records for the FTP client API for REXX function package. See “FTP client API for REXX function” on page 439 for information about trace records for the FTP client API for REXX function package.

The following table lists the settings accepted for TRACECAPI.

<table>
<thead>
<tr>
<th>If the statement is coded with a value of</th>
<th>Then . . .</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>all interface events are traced.</td>
<td>n/a</td>
</tr>
<tr>
<td>NONE</td>
<td>no events are traced.</td>
<td>n/a</td>
</tr>
<tr>
<td>CONDITIONAL</td>
<td>events are traced only when requested by the user program. This is the default.</td>
<td>The user program can request that events be traced by setting the FCAI_TraceIt field. FCAI_Traceit_Yes indicates that events are traced by the interface; FCAI_Traceit_No indicates that events are not traced.</td>
</tr>
</tbody>
</table>

The settings in the following two fields in the FCAI are applicable when the interface trace is initialized (the interface trace is initialized by the first request that requests tracing):

• FCAI_TraceSClass

  The interface trace is written to a spool file of the SYSOUT class supplied in FCAI_TraceSClass. Valid values are in the range A—Z and 0—9. The default value is A. The first request that is traced allocates and opens the spool file.
When the trace file is opened, its ddname is placed in FCAI_TraceName. Once the file is opened, it stays open until a TERM request is processed.

- **FCAI_TraceID**
  
  The user program can request that an ID be placed at the start of each trace record. This is done by putting a 3-byte character ID string in the field FCAI_TraceID. This value is written as the first three characters of each trace record. The trace ID is placed on each record to uniquely identify records from the same process. Trace records from different processes are written to different spool files. The ID ensures that records retain their identity when aggregated. The following example shows the trace records if FCAI_TraceID=TRC:

  TRC INIT>-a never
  TRC INIT<00000000 00000000 00000000 00000000
  TRC SCMD>open 9.42.105.93 6321|W

At the top of the trace output and every 64 lines thereafter, the interface writes a header record to provide information about the trace records being written. The header record contains the FCAI_TraceID, the updated date and time, the decimal value of process ID (pid) of the spawned z/OS client, and the decimal value of the TCB address for the user program’s task. The following is a sample header record:

```
```

The following example uses a sample trace for a very simple session with the FTP client API to show the format of the trace. The lines are numbered for the discussion of the trace; that is, the numbers do not appear in an actual trace. This trace shows requests and request results as well as client messages, server replies, and debug traces entries. The FCAI_TraceID value is 0, so each trace record is preceded by three blank characters (not reflected in this example). Also, the trace header records are not shown in the example.

```
1  PAW INIT>-a never  |CEE_DMPTARG=/etc
2  INIT<00000000 00000000 00000000 00000000
3  SCMD>open 9.42.105.93 6321|W
4  Connecting to: 9.42.105.93 port: 6321.
5  220-FTPDJG1 IBM FTP CS V1R6 at MVS164, 14:44:15 on 2007.
6  220 Connection will not timeout.
7  Command(00-10- -220):
8  SCMD<00000000 0A000000 00000000 00000000
9  SCMD>user user33
10 >>> USER user33
11 331 Send password please.
12 Command(00-19-USER-331):
13 SCMD<01020000 13000000 00000000 00000000
14 SCMD>pass *****
15 >>> PASS
16 230 USER33 is logged on. Working directory is "/u/user33".
17 Command(00-26-PASS-230):
18 SCMD<00000000 1A000000 00000000 00000000
19 SCMD>debug fsc
20 Active client traces - FSC(1)
21 Command(00-11- -):
22 SCMD<00000000 0B000000 00000000 00000000
23 SCMD>get a abc111 (repl|W
24 CG0226 get: F=1 p=FSA ARTWT=00001
25 CG3531 rcvFile: entered
26 MR1278 set_filename: entered with pathname abc111
27 CG1359 hfs_rcvFile: entered
28 MF0750 seq_open_file: recfm is NONE
29 MF1068 seq_open_file: OSBN --> wb,recfm=*,Noseek
   for /u/user33/abc111
30 MF1216 seq_open_file: stream 166412C4 has maxreclen 0
```
Line 1: This is the INIT request to the interface. The character > shows the direction of the flow. The character | is used to separate the start options "-a never" from the environment variable "_CEE_DMPTARG=/etc" that was passed on the spawn. Each parameter passed to the FTP client in the start options is separated from what follows in the trace by a null. Nulls should not be inserted into the start options by the application program (an ending null is permissible). The nulls in the trace are a result of the parsing mechanism that the interface uses to build the argument list for the spawn of the client.

Line 2: This is the result of the request from the interface. The character < shows the direction of flow. The four words of output are the "Request Completion Values" from the FCAI. The values are in hexadecimal and do not display as shown here. The 00 in byte 0 indicates successful initialization of the interface.

Line 3: This is the first subcommand. The subcommand string is displayed. If a mode parameter is entered, it is displayed following the character |. The character | is used to separate parameters in all of the request records. In this example, the user program entered a W for wait mode.

Line 4: This is a client message.

Line 5: This is the first line of the 220 server reply.

Line 6: This is the last line of the 220 server reply.

Line 7: This is the client message that indicates the end of the subcommand. The result is 00 (no errors) for the open (subcommand code decimal 10). The client subcommand caused a connect to the server but no server command actually flowed -- so the command field has four blanks. However, the last reply from the server was the 220 reply.

Line 8: These are the completion results for the open subcommand. The results in byte 0 are 00 (OK). The x'0A' in the second word is the open subcommand code in hexadecimal.

Line 9: Another subcommand. This time the user program did not pass a mode parameter (default mode is wait).

Line 10: This line shows the client message indicating a command to the server.

Line 11: This line shows the server reply which requests a password.

Line 12: This line shows as result of 00 (no errors) for the user subcommand (code 19). The last server command was USER and the last reply was 331.

Line 13: This line contains the completion results for the user subcommand. The x'01' in byte 0 indicates that there is additional status. The x'02' in byte 1 indicates that the user program is prompted...
for a pass subcommand.

Line 14: This is the pass subcommand. ****** is displayed to keep the 
actual password out of the trace.

Line 15: This line shows the client message indicating a command to the 
server.

Line 16: This line shows the server reply.

Line 17: This line shows a result of 00 (OK) for the pass subcommand 
(code 26). The last server command was PASS and the last reply 
was 230 -- the user program is logged in.

Line 18: This line shows the completion results for the pass subcommand.

Line 19: This line shows a subcommand to activate one of the client traces.

Line 24: This line is a client trace entry.

Line 45: This line shows a result of 00 (no errors) for the get subcommand 
(code 16). The last server command was RETR and the last reply 
was 250 -- the file transfer completed successfully.

Line 46: This line shows a quit subcommand that was generated by a request to 
terminate the interface. The user program failed to issue a QUIT to stop 
the client so TERM automatically generated a QUIT on behalf of the 
application.

Line 50: This line shows a request to end the interface to the Client 
API.

Line 51: The interface has ended.

The following example shows a portion of a trace after login and before the 
interface ends; this example shows some of the errors that can be reported by the 
trace.

```bash
1 OOPS>
2 OOPS<02000200 00000000 00000000 00000000
3 SCMD>
4 SCMD<02000300 00000000 00000000 00000000
5 GETL>D
6 GETL<02004000 00000000 00000000 00000000
7 GETL>FIND|J
8 GETL<02004100 00000000 00000000 00000000
9 POLL>
10 POLL<02003000 00000000 00000000 00000000
11 SCMD>get ! abc111 (rep|
12 >>> PORT 9,42,105,93,4,17
13 200 Port request OK.
14 >>> RETR !
15 501 Invalid data set name "!". Use MVS Dsname conventions.
16 Command(02-16-RETR-501):
17 SCMD<03000002 10000000 00000000 00000000
18 SCMD>get a 'user33.aaaaaaaaaaaaaaaaaaaa'
19 Invalid local file identifier
20 Command(18-16- - -):
21 SCMD<03000012 10000000 00000000 00000000
```

Line 1: This is a request type that is unknown to the interface.

Line 2: The results in byte 0 are 02 (interface error) with an explanation 
in byte 2 of 02 (unknown request).

Line 3: The SCMD request has no parameters.

Line 4: The results in byte 0 are 02 (interface error) with an explanation 
in byte 2 of 03 (parameter missing).

Line 5: The GETL request has an unknown operation value.

Line 6: The results in byte 0 are 02 (interface error) with an explanation 
in byte 2 of 40 (decimal 64) (unknown operation)

Line 7: The GETL request has an unknown type value for the Find operation.

Line 8: The results in byte 0 are 02 (interface error) with an explanation 
in byte 2 of 41 (decimal 65) (unknown type)

Line 9: This is a POLL request when a prior subcommand is not in 
progress.

Line 10: The results in byte 0 are 02 (interface error) with an explanation 
in byte 2 of 30 (decimal 48) (request not in progress)

Line 11: This SCMD request has a get with a bad remote file identifier. 
Commands are sent to the server (see lines 12 to 15).
FTP client API sample programs

The following sample programs for the FTP client API are available in the SEZAINST data set:

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EZAFTPAW</td>
<td>Assembler language FTP client API sample program</td>
</tr>
<tr>
<td>EZAFTPAX</td>
<td>COBOL language FTP client API sample program</td>
</tr>
<tr>
<td>EZAFTPAY</td>
<td>PL/I language FTP client API sample program</td>
</tr>
<tr>
<td>EZAFTPIR</td>
<td>FTP client API for REXX sample program</td>
</tr>
</tbody>
</table>

The FTPCAPIC FTP client API for C sample program is found in /usr/lpp/tcpip/samples/ftpcapic.c. The FTPCAPIJ FTP client API for Java sample program is found in /usr/lpp/tcpip/samples/ftpcapij.java.
Chapter 14. Network management interfaces

z/OS Communications Server provides several interfaces that enable network monitor and management applications to obtain information about their network operations, for both TCP/IP and VTAM. These interfaces for z/OS Communications Server TCP/IP provide the following capabilities:

- Programatically obtain copies of TCP/IP packet, OSAENTA, and data trace buffers, in real time, as the traces are collected.
- Format or filter the TCP/IP packet trace, OSAENTA packet trace, or data trace records that are collected.
- Obtain the following information:
  - Activation and deactivation events that are buffered for TCP connections in SMF format
  - Event information that is buffered for the FTP and TN3270 clients and servers in SMF format
  - Event information that is buffered for IP security in SMF format; information is provided from the IKE daemon and from the TCP/IP stack.
  - Detailed information and statistics for IP filtering and IPSec security associations on local TCP/IP stacks
  - Detailed information and statistics for IP filtering and IPSec security associations on remote network security services (NSS) clients when using the NSS server
  - TCP/IP profile information and profile change information, which is buffered; this information is provided in SMF event records.
- Control the following filters and associations:
  - IP filters and IPSec security associations on local TCP/IP stacks
  - IP filters and IPSec security associations on remote NSS clients when using the NSS server; see z/OS Communications Server: IP Configuration Guide for more information about network security services.
- Monitor the following functions using a callable API:
  - TCP connection and UDP endpoint activity
  - TCP/IP storage usage
  - TN3270E Telnet server connection performance
  - TCP/IP sysplex networking data
  - TCP/IP stack profile statement settings
- Drop one or multiple TCP connections or UDP endpoints

The interfaces for z/OS Communications Server VTAM provide the following:
- The ability to collect Enterprise Extender (EE) summary and connection data
- The ability to collect HPR endpoint data
- Communication Storage Manager (CSM) storage statistics

Some of the information provided by these interfaces can be obtained from other types of documented interfaces provided by z/OS Communications Server such as SNMP, SMF, command display output, and VTAM exits. TCP/IP packet trace collection and formatting interfaces provide access to packet trace data that was not previously available through an authorized, real-time z/OS Communications.
Server interface. Some of the event information in SMF format is currently available through traditional SMF services, and can be collected using an SMF user exit to monitor SMF records.

The interfaces that are described in this topic provide an alternative for collecting some of the TCP/IP SMF records and are expected to perform efficiently. Most of the data that is provided by the network management interface for monitoring TCP/UDP endpoints and TCP/IP storage described in “TCP/IP callable NMI (EZBNMIFR)” on page 598 can be collected from supported SNMP MIBs. Storage usage information is available through displays and the VTAM Performance Monitor Interface (PMI). When used correctly, the interfaces documented in this document provide well-defined and efficient APIs to be used for obtaining management information related to the IP and SNA (VTAM) components of z/OS Communications Server. They also allow for easy application migration to subsequent z/OS Communications Server releases. They are targeted for use by responsible network management applications.

The following describe the programming interfaces for these functions in detail, and provide the information required to develop network management applications that use them. These interfaces have the following characteristics:

- Use a client/server model or a called interface
- Require all network management clients to be run locally on the same z/OS image as the Communications Server
- Are provided for C/C++ and assembler, except as otherwise indicated

In this topic, the term TCP/IP is used to represent the IP component of z/OS Communications Server and the term VTAM refers to the SNA component of z/OS Communications Server.

**Local IPSec NMI**

The z/OS Communications Server IKE daemon provides the IPSec network management interface (NMI). The IPSec NMI is an AF_UNIX socket interface through which network management applications can manage IP filtering and IPSec on local TCP/IP stacks. Use this interface for network management applications that expect to maintain agents on each individual z/OS system or use it in any environments where z/OS network security services (NSS) is not enabled. If your applications use a centralized management and monitoring approach, you should consider using the NSS management interface that is described in “Network security services (NSS) network management NMI” on page 531.

This interface enables applications to obtain the following types of data regarding the local TCP/IP stacks and the IKE daemon:

- Information about which TCP/IP stacks are configured for integrated IPSec/VPN
- Summary statistics for IKE, IPSec, and IP filtering activity for a particular TCP/IP stack
- Detailed information about IP filters for a particular TCP/IP stack
- Detailed information about IPSec and IKE security associations (SAs) for a particular TCP/IP stack
- Port translation information for NAT traversal
- Information about which IP interfaces are active for a given TCP/IP stack
- Information about NSS clients that are active in the local IKE daemon
In addition, network management applications can perform the following functions to control IP filtering and IPSec over the same AF_UNIX socket:

- Activate and deactivate manual and dynamic tunnels
- Refresh dynamic tunnels
- Switch between default IP filters and policy-based IP filters

With the IPSec network management interface, a client network management application makes requests and performs management actions by sending messages over an AF_UNIX stream socket connection to the IKE daemon. The requested data is returned to the application directly over the AF_UNIX connection.

**Tip:** If you are processing IPSec SMF records, there are some structures that were designed to be analogous to IPSec NMI structures. If you have code to process these structures, you might not need to write new parsing code. The section names are indicated in the individual SMF records and are described in detail in [SMF type 119 records](z/OS Communications Server: IP Configuration Reference) information in the [z/OS Communications Server: IP Configuration Reference](z/OS Communications Server: IP Configuration Reference).

### Local IPSec NMI: Configuring the interface

The z/OS system administrator can restrict access to the IKE network management interface as follows:

- Access to the stack monitoring functions (those that request information only about specific stacks) within this interface is controlled by defining the SERVAUTH resource name `EZB.NETMGMT.sysname.tcpipname.IPSEC.DISPLAY` in the SERVAUTH class (where the `sysname` value represents the MVS system name where the interface is being invoked, and the `tcpipname` value is the name of the TCP/IP stack).

- Access to the stack control functions (those that take some action on a specific stack) is controlled through the `EZB.NETMGMT.sysname.tcpipname.IPSEC.CONTROL` resource.

- Access to IKE daemon-level monitoring functions (those that request information at the daemon level) is controlled through the `EZB.NETMGMT.sysname.sysname.IKED.DISPLAY` resource.

For applications that use the interface, the MVS user ID should be permitted to the defined resource. If the resource is not defined, then only superusers or users permitted to the `BPX.SUPERUSER` resource in the FACILITY class are permitted to access the interface.

Additionally, permitted client applications must have permission to enter the `/var/sock` directory and to write to the `/var/sock/ipsecmgmt` socket.

**Guideline:** If you are developing a feature for a product that is to be used by other parties, include instructions in your documentation that indicating that administrators must define and give appropriate permission to the given security resource to use that feature; if the resource is not defined, indicating administrators must run your program as superuser.

**Requirements:**

- The IKED OMVS user ID must have write access to the `/var/sock` directory (or else have permission to create this directory).
- z/OS Communications Server IKE daemon and Policy Agent must be active on the system where data is being collected.
Local IPSec NMI: Connecting to the server

For an application to use this interface, it must connect to the AF_UNIX stream socket provided by the IKE daemon for this interface. The socket path name is /var/sock/ipsecmgmt. You can use the Language Environment C/C++ API or the UNIX System Services BPX Callable Assembler services to create AF_UNIX sockets and connect to this service.

When an application connects to the socket, the IKE daemon sends an initialization message to the client application. When the IKE daemon closes a client connection (reasons for doing so include severe errors in the format of data requests sent by the application to the IKE daemon, or IKE daemon termination), the IKE daemon attempts to send a termination message to the client before closing the connection. Both the initialization and termination messages conform to the general response message structure used by the IKE daemon to send data to the application (see “IPSec NMI request/response format”).

The initialization message contains only a message header (see “IPSec NMI initialization and termination messages” on page 528). The version number reported in the message header indicates the maximum version of the interface supported by the IKE daemon. After the initialization message has been received by the client, the client can send requests for IPSec management data to the server.

Result: The IKE daemon does not send an INIT message to the client application until it has successfully connected to the Policy Agent.

The termination message also contains only a message header (see “IPSec NMI initialization and termination messages” on page 528). The message header contains a return code and a reason code that indicates the reason for terminating the connection.

IPSec NMI request/response format

This interface exchanges messages over an AF_UNIX socket using a request-response model. The client application builds and sends an NMI request over the socket. The request specifies the action or the type of information requested and might contain optional input parameters. The IKE daemon then provides a response message over the socket that contains the results of the request, including the requested data if this is a monitoring request. The client application must then read this response data from the socket. A severe formatting error in the client application’s NMI request might result in the IKE daemon sending a termination record and closing the connection.

The IPSec network management interface provides the formatted response data directly to the client application over the AF_UNIX connection.

Rule: All EBCDIC data is right-padded with blank characters and uses codepage IBM-1047 encoding.

IPSec NMI request and response data structures

The network management interface for monitoring IP filtering and IPSec provides data structures for C, C++, and assembler programs to access the interface. The C and C++ header file is contained in the ezbnmsec.h and ezbnmv2.h files, which are located in the /usr/include directory. The assembler macro is located in the EZBNMSEA member of the SEZANMAC data set.
IPSec NMI request and response message format

NMI request and response messages share a common format. An IPSec NMI message consists of a message header followed by zero or more records. The message header is defined by the NMsecMessageHdr structure.

Table 32. NMsecMessageHdr structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsecMIdent</td>
<td>0</td>
<td>4 bytes</td>
<td>EBCDIC</td>
<td>Message header identifier; set to NMsec_MSGIDENT (EBCDIC ‘NM’sM’).</td>
</tr>
<tr>
<td>NMsecMHdrLength</td>
<td>4</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Length of the message header. See the NMsecMMsgLength field for the length of entire message.</td>
</tr>
<tr>
<td>NMsecMVersion</td>
<td>8</td>
<td>2 bytes</td>
<td>Binary</td>
<td>NMI version. Only version 2 is currently supported by this interface (NMsec_VERSION2).</td>
</tr>
<tr>
<td>NMsecMType</td>
<td>10</td>
<td>2 bytes</td>
<td>Binary</td>
<td>Message type. For a request, this indicates the type of request being made. For a response, this indicates the type of response data, and is identical to the request type. See “IPSec NMI request messages” on page 484 for a description of the request types.</td>
</tr>
<tr>
<td>NMsecMCorrelator</td>
<td>12</td>
<td>16 bytes</td>
<td>Binary</td>
<td>User-defined field for correlating NMI requests with responses. The interface echoes the correlator for a given request on the corresponding response.</td>
</tr>
<tr>
<td>NMsecMRsvd1</td>
<td>28</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Reserved; set to 0.</td>
</tr>
<tr>
<td>NMsecMRc</td>
<td>32</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Return code. The client must set this field to 0 in a request message. For a reply, this field is 0 for a successful reply, or a nonzero value for an error (see “Network security services NMI return and reason codes” on page 536).</td>
</tr>
<tr>
<td>NMsecMRsn</td>
<td>36</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Reason code. The client must set this field to 0 in a request message. For a reply, if the NMsecMRc field indicates an error, this field might provide additional information about the error (see “Network security services NMI return and reason codes” on page 536).</td>
</tr>
<tr>
<td>NMsecMMsgLength</td>
<td>40</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Length of entire message, including the message header.</td>
</tr>
<tr>
<td>NMsecMTime</td>
<td>44</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Timestamp. The server ignores this field in a request message. For a reply, this value indicates the UNIX timestamp for the server. This might be correlated with timestamps in result fields.</td>
</tr>
<tr>
<td>NMsecMRsvd2</td>
<td>48</td>
<td>20 bytes</td>
<td>Binary</td>
<td>Reserved; set to 0.</td>
</tr>
</tbody>
</table>
### Table 32. NMsecMessageHdr structure (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsMInRec</td>
<td>68</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Input record descriptor. This field is set by the client application for request messages and describes which records, if any, are present on the request. This descriptor is described by the NMsecInRecDesc structure. See Table 33 on page 481 for details.</td>
</tr>
<tr>
<td>NMsMRsvd3</td>
<td>76</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Reserved; set to 0.</td>
</tr>
<tr>
<td>NMsMOutRec</td>
<td>84</td>
<td>16 bytes</td>
<td>Binary</td>
<td>Output record descriptor. The client application should set this field to 0 on input. The server completes the field with information describing the records that contain the result data. This descriptor is described by the NMsecOutRecDesc structure. See Table 34 on page 481 for details.</td>
</tr>
<tr>
<td>NMsMOutRec2</td>
<td>100</td>
<td>16 bytes</td>
<td>Binary</td>
<td>Secondary output record descriptor. This descriptor identifies zero or more secondary result records for a given request. For example, a request might provide a single record containing global configuration information. The client application should set this field to 0 on input. The server completes this field with information describing the secondary result records. Secondary result records are provided only for certain requests; such requests describe the layout of the corresponding secondary result records. This field is described by the NMsecOutRecDesc structure. See Table 34 on page 481 for details.</td>
</tr>
<tr>
<td>NMsMTarget</td>
<td>116</td>
<td>24 bytes</td>
<td>EBCDIC</td>
<td>The target for routing the request. Most request types apply to a single TCP/IP stack. The target field must contain the job name for that TCP/IP stack, right-padded with blanks. <strong>Rule:</strong> If the request applies to all stacks (this is valid only for the NMsec_GET_STACKINFO request), then this field must be filled with blanks. <strong>Result:</strong> The server echoes the client’s target string on a reply message.</td>
</tr>
</tbody>
</table>
Input record descriptor

Table 33. Input record descriptor

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsiROffset</td>
<td>0</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Offset to the first input record, measured in bytes from the start of the message.</td>
</tr>
<tr>
<td>NMsiRRsvd1</td>
<td>4</td>
<td>2 bytes</td>
<td>Binary</td>
<td>Reserved; set to 0.</td>
</tr>
<tr>
<td>NMsiINumber</td>
<td>6</td>
<td>2 bytes</td>
<td>Binary</td>
<td>Number of input records present in message.</td>
</tr>
</tbody>
</table>

Result: These fields are set to 0 on a reply message sent by the server.

Output record descriptor

Table 34. Output record descriptor

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsoROffset</td>
<td>0</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Offset to first output record, measured in bytes from the start of the message.</td>
</tr>
<tr>
<td>NMsoRTotal</td>
<td>4</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Number of output records that would have been generated in the absence of input filters. If the request did not have input filters, or if input filters were not applicable for the request, the value of this field is the same as the NMsoRNumber field value.</td>
</tr>
<tr>
<td>NMsoRNumber</td>
<td>8</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Number of output records present in message.</td>
</tr>
<tr>
<td>NMsoRRsvd1</td>
<td>12</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Reserved; set to 0.</td>
</tr>
</tbody>
</table>

The message header is followed by zero or more records. Records can vary in length. Each record consists of a record header, followed by one or more section descriptors that describe the sections within the record, followed by one or more sections that contain the actual record data. Conceptually, the layout of a message and its records is as shown in Figure 9 on page 482.
The record header is described by the NMsecRecordHdr structure.

**Table 35. NMsecRecordHdr structure**

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsecRIdent</td>
<td>0</td>
<td>4 bytes</td>
<td>EBCDIC</td>
<td>Record header identifier; set to NMsec_RECIDENT (EBCDIC 'NMsecR').</td>
</tr>
<tr>
<td>NMsecRLength</td>
<td>4</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Total record length, including record header, section descriptors and data.</td>
</tr>
<tr>
<td>NMsecRNumCascadeSecDesc</td>
<td>8</td>
<td>2 bytes</td>
<td>Binary</td>
<td>Number of cascading section descriptors present in this record.</td>
</tr>
<tr>
<td>NMsecRNumSecDesc</td>
<td>10</td>
<td>2 bytes</td>
<td>Binary</td>
<td>Number of section descriptors present in this record.</td>
</tr>
</tbody>
</table>

A record’s section descriptors immediately follow the record header. Standard section descriptors are described by the NMsecSecDesc structure.

**Table 36. NMsecSecDesc structure**

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsecSOffset</td>
<td>0</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Offset from the start of record to the first section referenced by this section descriptor.</td>
</tr>
<tr>
<td>NMsecSLength</td>
<td>4</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Length of each section referenced by this section descriptor.</td>
</tr>
<tr>
<td>NMsecSNumber</td>
<td>8</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Number of sections referenced by this section descriptor (can be 0).</td>
</tr>
</tbody>
</table>

Each standard section descriptor describes a set of sections present in the record. Each descriptor indicates the offset to the first such section from the start of the record (not from the start of the message), the length of each section in the set, and the number of sections in the set. The number of sections can be zero. If there is
more than one section, then the sections identified by a given descriptor are
uniform in length. In Figure 9 on page 482, sections 1A and 1B are both described
by section descriptor 1 and have the same length, and section 2 does not need to
be present if the length or count of section 2 is 0.

A special kind of section descriptor called a cascading section descriptor indicates
the offset and length of a section that contains a set of records of a different type.
A section that contains these kinds of records is called a cascading record container
section. The NMscRecords field of a cascading section descriptor indicates how
many records are contained within the cascading record container section. These
cascading constructs enable records of one type to be nested within another record.
Cascading section descriptors appear after all standard descriptors. The record type
determines the number of each type of descriptor. Cascading descriptors are
described by the NMsecCascadingSecDesc structure.

Table 37. NMsecCascadingSecDesc structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMscOffset</td>
<td>0</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Offset from the start of record to the cascading record container section referenced by this cascading section descriptor</td>
</tr>
<tr>
<td>NMscSLen</td>
<td>4</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Length of the cascading record container section referenced by this cascading section descriptor (can be 0)</td>
</tr>
<tr>
<td>NMscSRec</td>
<td>8</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Number of records within the referenced cascading record container section (can be 0)</td>
</tr>
</tbody>
</table>

Following the section descriptors are the record’s sections, whose location and
number are described by the set of section descriptors. The format of the sections
is determined by the message type for the message. For example, the first section
descriptor might identify a single section containing statistical data, while the
second section descriptor might identify a section containing a variable-length IKE
identity. In Figure 9 on page 482, sections 1A and 1B might always have the same
length, but the length of section 2 in one record can differ from the length of
section 2 in another record. Section 2 is not always present in every record, but
section descriptor 2 is always present.

The records for a message can differ in length because some data is present or
absent, or because there is variable-length data. However, all records in a message
have the same type and format for the data that is present in those records. In
other words, for a given message, all records have the same number of section
descriptors, and the sections referenced by each descriptor have the same semantic
content. However, data for any given section in each record within a message
might or might not be present (data not present would be indicated by a section
count value 0 in the associated section descriptor).

The size of any given structure that is contained in a section can increase from one
release to the next, but the format of the data from the earlier release does not
change. If new data is added to a section for a given release, it is added at the end
of the section so that existing data mappings continue to resolve correctly without
recompiling applications. To ensure that applications are compatible with future
releases, if applications check the validity of a section’s length, they should always
test for a length that is greater than or equal to the expected length.
Result: If a message contains records (described by the NMsMOutRec field) and secondary records (described by the NMsMOutRec2 field), then the records and secondary records are not necessarily of the same type and format. See "IPSec NMI request messages" for details about the format the records and secondary records for each request.

IPSec NMI request messages

Client applications send request messages to the server. Request records contain the input parameters for the request. Input records for monitoring requests are called filter records or input filters. Control requests have a variety of input record formats. The following message types are supported by the server.

- Monitoring requests.
  Access to each of these functions is controlled using the EZB.NETMGMT.sysname.tcpipname.IPSEC.DISPLAY resource definition in the SERVAUTH class, unless otherwise noted.

  Each number in parentheses represents the value of the given request type constant, which is to be stored in the request message’s NMsMType field.

  - NMsec_GET_STACKINFO (2)—Obtain IP security and defensive filtering configuration information for a given TCP/IP stack, or optionally obtain this information for all active TCP/IP stacks.

    Rule: To obtain configuration information for a specific TCP/IP stack, set the NMsMTarget field in the request message header to be the same as the value for the stack’s job name. To obtain configuration information for all TCP/IP stacks, set the NMsMTarget field in the request message header to blanks.

  - NMsec_GET_SUMMARY (3)—Retrieve summary IKE, IPSec, and IP filtering data from and for a particular stack.

  - NMsec_GET_IPFLTCURR (4)—Retrieve detailed information from a particular stack about the currently active IP filters. These filters can be either the default IP security filters (filters that originate from the TCP/IP profile) or the policy IP security filters (filters that originate from Policy Agent). Any defensive filters that are installed are also included.

  - NMsec_GET_IPFLTDEFAULT (5)—Retrieve detailed information from a particular stack about the default IP security filters (filters that originate from the TCP/IP profile).

    Result: The default IP security filters are returned, regardless of whether they comprise the currently active filter set that is in use by the stack.

  - NMsec_GET_IPFLT POLICY (6)—Retrieve detailed information from a particular stack about the policy IP security filters (filters that originate from Policy Agent).

    Results:
    - The policy IP security filters are returned regardless of whether they are the currently active filter set in use by the stack.
    - If Policy Agent has not installed IP security filters in the stack, then a message that contains no filters is returned.

  - NMsec_GET_PORTTRAN (7)—Retrieve IPv4 NAT traversal port translation information from a particular stack.

  - NMsec_GET_IPTUNMANUAL (8)—Retrieve detailed information about manual tunnels from a particular stack.

  - NMsec_GET_IPTUNDYNSTACK (9)—Retrieve detailed information about dynamic tunnels (phase 2 tunnels) from a particular stack.
- NMsec_GET_IPTUNDYN (10)—Retrieve detailed IKE-related information about dynamic tunnels (phase 2 tunnels) for a particular stack.
- NMsec_GET_IKETUN (11)—Retrieve detailed information about IKE tunnels (phase 1 tunnels) for a particular stack.
- NMsec_GET_IKETUNCASCADE (12)—Retrieve detailed information about IKE tunnels for a particular stack, along with information about the associated dynamic tunnels (phase 2 tunnels) for each IKE tunnel.
- NMsec_GET_IPINTERFACES (13)—Retrieve the list of IP interfaces that belong to a particular stack.
- NMsec_GET_IKENSINFO (14)—Retrieve network security services information for the IKE daemon.

Rules:
- Access to this function is controlled using the EZB.NETMGMT.sysname.sysname.IKED.DISPLAY resource definition in the SERVAUTH class.
- Set the NMsmTarget field in the request message header to blanks for this request.

- Control requests.
  Access to each of these functions is controlled using the EZB.NETMGMT.sysname.tcpipname.IPSEC.CONTROL resource definition in the SERVAUTH class
  - NMsec_ACTIVATE_IPTUNMANUAL (1001)—Activate a manual tunnel.
  - NMsec_ACTIVATE_IPTUN (1002)—Activate a dynamic IPSec tunnel.
  - NMsec_DEACTIVATE_IPTUNMANUAL (1003)—Deactivate a manual tunnel.
  - NMsec_DEACTIVATE_IPTUN (1004)—Deactivate a dynamic IPSec tunnel.
  - NMsec_DEACTIVATE_IKETUN (1005)—Deactivate an IKE tunnel.
  - NMsec_REFRESH_IPTUN (1006)—Refresh a dynamic IPSec tunnel.
  - NMsec_REFRESH_IKETUN (1007)—Refresh an IKE tunnel.
  - NMsec_LOAD_POLICY (1008)—Switch between default IP filters and policy-based IP filters.

**IPSec NMI monitoring request format**

<table>
<thead>
<tr>
<th>NMsecMessageHdr</th>
<th>NMsecRecordHdr</th>
<th>NMsecSecDesc</th>
<th>NMsecInFilter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Input filter records (0-20)

*Figure 10. NMI monitoring request format*

Monitoring requests that allow request records (not all of them do) call their request records filter records or input filters. If no input filters are provided, then all applicable data is returned over the interface. However, if input filters are provided, then the returned data is limited based on the input filters. Some requests do not support input filters (see Table 38 on page 487). Each input filter is specified by the client application in a separate record in the request message. Up to twenty input filter elements can be specified. Each input filter specifies one or more attribute to be restricted in the results returned over the interface. The attributes filtered by a single filter are combined with a logical AND; that is, all of
the attributes must match a response record in order for that record to be returned over the interface. Multiple filters are combined with a logical OR; a response record needs to match only one input filter for that record to be returned over the interface.

Table 38 on page 487 shows which input filter specifications are valid for each request type. The filter specifications are described in detail in subsequent sections.
### Table 38. Valid input filter specifications for request types

<table>
<thead>
<tr>
<th>Filter specification</th>
<th>STACK INFO</th>
<th>SUMMARY</th>
<th>IPFLTCURR, IPFLTDEFAULT, IPFLTTYPE</th>
<th>PORT TRAN</th>
<th>IPTUN MANUAL</th>
<th>IPTUN DYNSTACK</th>
<th>IPTUN DYNIKE</th>
<th>IKETUN, IKETUN CASCADE</th>
<th>IP INTERFACES</th>
<th>IKENS INFO</th>
<th>CLIENT INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMrFltSrcAddr4</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMrFltSrcAddr6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMrFltDestAddr4</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMrFltDestAddr6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMrFltProtocol</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMrFltSrcPort</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMrFltDestPort</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMrFltLclEndpt4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMrFltLclEndpt6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMrFltRmtEndpt4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMrFltRmtEndpt6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMrFltTunnelID</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>NMrFltObjName</td>
<td>x (filter name)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x (VPN action)</td>
<td>x (VPN action)</td>
<td>x (VPN action)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMrFltObjGroupName</td>
<td>x (filter group name)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMrFltAssocName</td>
<td>x (VPN action)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMrFltFlagIPFltType</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMrFltSASState</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMrFltSWSAShadow</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NMrFltFlagDiscipline</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Restriction: The NMsecGetFlagDiscipline filter, which is part of the network security services NMI, can be used only with the NMsec_GET_CLIENTINFO request to the NSS server. See “Network security services NMI response messages” on page 533.

The client should provide each input filter element in a record in the request message. Each record should contain a record header, a single section descriptor identifying the input filter, and the input filter structure. The input filter is described by the NMsecInFilter structure. The input filter consists of a bit mask that indicates which filtering attributes are specified, followed by fields that specify the actual attribute values that are to be filtered. The layout of the structure is shown in Table 39, where bit 0 represents the high-order bit of a byte.

### Table 39. NMsecInFilter structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsecFltFlagIPv6</td>
<td>0, bit 0</td>
<td>1 bit</td>
<td>Binary</td>
<td>IPv6 indicator. If set, addresses in input filter are IPv6, otherwise, addresses are IPv4.</td>
</tr>
<tr>
<td>NMsecFltFlagSrcAddr</td>
<td>0, bit 1</td>
<td>1 bit</td>
<td>Binary</td>
<td>Source address indicator. If set, the input filter specifies a source address for filtering.</td>
</tr>
<tr>
<td>NMsecFltFlagDstAddr</td>
<td>0, bit 2</td>
<td>1 bit</td>
<td>Binary</td>
<td>Destination address indicator. If set, the input filter specifies a destination address for filtering.</td>
</tr>
<tr>
<td>NMsecFltFlagProto</td>
<td>0, bit 3</td>
<td>1 bit</td>
<td>Binary</td>
<td>Protocol indicator. If set, the input filter specifies an IP protocol number for filtering.</td>
</tr>
<tr>
<td>NMsecFltFlagSrcPort</td>
<td>0, bit 4</td>
<td>1 bit</td>
<td>Binary</td>
<td>Source port indicator. If set, the input filter specifies a source port number for filtering.</td>
</tr>
<tr>
<td>NMsecFltFlagDstPort</td>
<td>0, bit 5</td>
<td>1 bit</td>
<td>Binary</td>
<td>Destination port indicator. If set, the input filter specifies a destination port number for filtering.</td>
</tr>
<tr>
<td>NMsecFltFlagLclEndpt</td>
<td>0, bit 6</td>
<td>1 bit</td>
<td>Binary</td>
<td>Local security endpoint indicator. If set, the input filter specifies a local security endpoint address for filtering.</td>
</tr>
<tr>
<td>NMsecFltFlagRmtEndpt</td>
<td>0, bit 7</td>
<td>1 bit</td>
<td>Binary</td>
<td>Remote security endpoint indicator. If set, the input filter specifies a remote security endpoint address for filtering.</td>
</tr>
<tr>
<td>NMsecFltFlagTunnelID</td>
<td>1, bit 0</td>
<td>1 bit</td>
<td>Binary</td>
<td>Tunnel ID indicator. If set, the input filter specifies a tunnel ID for filtering.</td>
</tr>
<tr>
<td>NMsecFltFlagObjName</td>
<td>1, bit 1</td>
<td>1 bit</td>
<td>Binary</td>
<td>Object name indicator. If set, the input filter specifies an object name for filtering.</td>
</tr>
<tr>
<td>NMsecFltFlagObjGrpName</td>
<td>1, bit 2</td>
<td>1 bit</td>
<td>Binary</td>
<td>Object group name indicator. If set, the input filter specifies an object group name for filtering.</td>
</tr>
<tr>
<td>NMsecFltFlagAssocName</td>
<td>1, bit 3</td>
<td>1 bit</td>
<td>Binary</td>
<td>Associated object name indicator. If set, the input filter specifies an associated object name for filtering.</td>
</tr>
<tr>
<td>NMsecFltFlagSASState</td>
<td>1, bit 4</td>
<td>1 bit</td>
<td>Binary</td>
<td>Security association state indicator. If set, the input filter specifies a security association state for filtering.</td>
</tr>
<tr>
<td>NMsecFltFlagShadow</td>
<td>1, bit 5</td>
<td>1 bit</td>
<td>Binary</td>
<td>Shadow indicator. If set, the input filter specifies a SWSA shadow disposition for filtering.</td>
</tr>
</tbody>
</table>
### Table 39. NMsSecInFilter structure (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsFltFlagIPFltType</td>
<td>1, bit</td>
<td>1 bit</td>
<td>Binary</td>
<td>IP filter type mask. If set for an IP filter request, the NMsIPFltType.xxx bits indicate the types of IP filters that match the input filter. If not set, the NMsIPFltType.xxx bits are ignored and IP filters of any type match. Details about IP filter mask types are listed in this table.</td>
</tr>
<tr>
<td>NMsFltFlagDiscipline</td>
<td>1, bit</td>
<td>1 bit</td>
<td>Binary</td>
<td>Discipline indicator. If set, the input filter specifies a discipline for filtering.</td>
</tr>
<tr>
<td>NMsFltRsvd1</td>
<td>1, bit</td>
<td>16 bits</td>
<td>Binary</td>
<td>Reserved bits. Must be set to 0.</td>
</tr>
<tr>
<td>NMsFltSrcAddr4</td>
<td>4</td>
<td>4 bytes</td>
<td>Binary</td>
<td>IPv4 or IPv6 source address selector. For an IP filter request the following apply:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• If the IP filter represents outbound traffic, this input filter matches the IP filter if the IP filter contains this address within its source IP address specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• If the IP filter represents inbound traffic, this input filter matches the IP filter if the IP filter contains this address within its destination IP address specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>For a dynamic IP tunnel request, this input filter matches the dynamic IP tunnel if the dynamic tunnel contains this address within its source IP address specification for tunnel data.</td>
</tr>
<tr>
<td>NMsFltSrcAddr6</td>
<td>4</td>
<td>16 bytes</td>
<td>Binary</td>
<td>IPv4 or IPv6 source address selector. For an IP filter request the following apply:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• If the IP filter represents outbound traffic, this input filter matches the IP filter if the IP filter contains this address within its destination IP address specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• If the IP filter represents inbound traffic, this input filter matches the IP filter if the IP filter contains this address within its source IP address specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>For a dynamic IP tunnel request, this input filter matches the dynamic IP tunnel if the dynamic tunnel contains this address within its destination IP address specification for tunnel data.</td>
</tr>
<tr>
<td>NMsFltDstAddr4</td>
<td>20</td>
<td>4 bytes</td>
<td>Binary</td>
<td>IPv4 or IPv6 destination address selector. For an IP filter request the following apply:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• If the IP filter represents outbound traffic, this input filter matches the IP filter if the IP filter contains this address within its destination IP address specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• If the IP filter represents inbound traffic, this input filter matches the IP filter if the IP filter contains this address within its source IP address specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>For a dynamic IP tunnel request, this input filter matches the dynamic IP tunnel if the dynamic tunnel contains this address within its destination IP address specification for tunnel data.</td>
</tr>
<tr>
<td>NMsFltDstAddr6</td>
<td>20</td>
<td>16 bytes</td>
<td>Binary</td>
<td>IPv4 or IPv6 destination address selector. For an IP filter request the following apply:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• If the IP filter represents outbound traffic, this input filter matches the IP filter if the IP filter contains this address within its destination IP address specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• If the IP filter represents inbound traffic, this input filter matches the IP filter if the IP filter contains this address within its source IP address specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>For a dynamic IP tunnel request, this input filter matches the dynamic IP tunnel if the dynamic tunnel contains this address within its destination IP address specification for tunnel data.</td>
</tr>
<tr>
<td>NMsFltProtocol</td>
<td>36</td>
<td>1 byte</td>
<td>Binary</td>
<td>Protocol selector. For IP filter requests, port translation requests, and dynamic IP tunnel requests, this field limits the results based on the IP protocol number (corresponding to the IP protocol number in the IPv4 or IPv6 header). This input filter matches the result data if the result data contains this protocol within its IP protocol specification.</td>
</tr>
<tr>
<td>NMsFltRsvd2</td>
<td>37</td>
<td>1 byte</td>
<td>Binary</td>
<td>Reserved field. Must be set to 0.</td>
</tr>
</tbody>
</table>
Table 39. NMsecInFilter structure (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsFltSrcPort</td>
<td>38</td>
<td>2 bytes</td>
<td>Binary</td>
<td>Source port selector. For an IP filter request the following apply:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• If the IP filter represents outbound traffic, this input filter matches the IP filter if the IP filter contains this port within its source port specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• If the IP filter represents inbound traffic, this input filter matches the IP filter if the IP filter contains this port within its destination port specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>For a dynamic IP tunnel request, this input filter matches the dynamic IP tunnel if the dynamic tunnel contains this port within its source port specification for tunnel data.</td>
</tr>
<tr>
<td>NMsFltDstPort</td>
<td>40</td>
<td>2 bytes</td>
<td>Binary</td>
<td>Destination port selector. For an IP filter request the following apply:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• If the IP filter represents outbound traffic, this input filter matches the IP filter if the IP filter contains this port within its destination port specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• If the IP filter represents inbound traffic, this input filter matches the IP filter if the IP filter contains this port within its source port specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>For a dynamic IP tunnel request, this input filter matches the dynamic IP tunnel if the dynamic tunnel contains this port within its destination port specification for tunnel data. For a port translation request, this input filter matches the port translation entry if the translated source port matches this port.</td>
</tr>
<tr>
<td>NMsFltRsvd3</td>
<td>42</td>
<td>2 bytes</td>
<td>Binary</td>
<td>Reserved field. Must be set to 0.</td>
</tr>
<tr>
<td>NMsFltLclEndpt4</td>
<td>44</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Local security endpoint selector. For all IP and IKE tunnels, manual or dynamic, this input filter matches the tunnel if the tunnel’s local security endpoint IP address is the same as this address.</td>
</tr>
<tr>
<td>NMsFltLclEndpt6</td>
<td>44</td>
<td>16 bytes</td>
<td>Binary</td>
<td></td>
</tr>
<tr>
<td>NMsFltRmtEndpt4</td>
<td>60</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Remote security endpoint selector. For all IP and IKE tunnels, manual or dynamic, this input filter matches the tunnel if the tunnel’s remote security endpoint IP address is the same as this address.</td>
</tr>
<tr>
<td>NMsFltRmtEndpt6</td>
<td>60</td>
<td>16 bytes</td>
<td>Binary</td>
<td></td>
</tr>
<tr>
<td>NMsFltTunnelID</td>
<td>76</td>
<td>48 bytes</td>
<td>EBCDIC</td>
<td>Tunnel ID selector. For all IKE and IP tunnels, manual or dynamic, this input filter matches the tunnel if the tunnel’s tunnel ID matches this EBCDIC string. For IP filter requests, this input filter matches any IP filter associated with a manual or dynamic IP tunnel that has this tunnel ID.</td>
</tr>
</tbody>
</table>
Table 39. NMsecInFilter structure (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsFltObjName</td>
<td>124</td>
<td>48</td>
<td>EBCDIC</td>
<td>Object name selector. For an IP filter request, this field limits the results based on the filter name. For IKE tunnels, this field limits the results based on the KeyExchangeRule name. For IP tunnels, this field limits the results based on the IPDynVpnAction or IPManVpnAction name.</td>
</tr>
<tr>
<td>NMsFltObjGroupName</td>
<td>172</td>
<td>48</td>
<td>EBCDIC</td>
<td>Group name selector. For an IP filter request, this field limits the results based on the filter group name.</td>
</tr>
<tr>
<td>NMsFltAssocName</td>
<td>220</td>
<td>48</td>
<td>EBCDIC</td>
<td>Associated object name selector. For an IP filter request, this field limits the results based on the IPDynVpnAction or IPManVpnAction name. For IP tunnels, this field limits the results based on the LocalDynVpnRule name.</td>
</tr>
<tr>
<td>NMsFltIPFltTypeGeneric</td>
<td>268, bit 0</td>
<td>1 bit</td>
<td>Binary</td>
<td>Generic IP filter mask. If set for an IP filter request, this input filter matches generic PERMIT and DENY IP security filters and defensive filters. If not set, generic IP security filters and defensive filters are not matched.</td>
</tr>
<tr>
<td>NMsFltIPFltTypeManual</td>
<td>268, bit 1</td>
<td>1 bit</td>
<td>Binary</td>
<td>Manual IP filter mask. If set for an IP filter request, this input filter matches IP filters referencing manual IP tunnels. If not set, manual IP filters are not matched.</td>
</tr>
<tr>
<td>NMsFltIPFltTypeDynAnchor</td>
<td>268, bit 2</td>
<td>1 bit</td>
<td>Binary</td>
<td>Dynamic anchor IP filter mask. If set for an IP filter request, this input filter matches IP filters that serve as anchors for dynamic IP tunnels. If not set, dynamic anchor IP filters are not matched.</td>
</tr>
<tr>
<td>NMsFltIPFltTypeDynamic</td>
<td>268, bit 3</td>
<td>1 bit</td>
<td>Binary</td>
<td>Dynamic IP filter mask. If set for an IP filter request, this input filter matches dynamic IP filters for dynamic IP tunnels. If not set, dynamic IP filters are not matched.</td>
</tr>
<tr>
<td>NMsFltIPFltTypeNATTAnchor</td>
<td>268, bit 4</td>
<td>1 bit</td>
<td>Binary</td>
<td>NATT anchor IP filter mask. If set for an IP filter request, this input filter matches IP filters that serve as anchors for NAT traversal IP tunnels. If not set, NATT anchor IP filters are not matched.</td>
</tr>
<tr>
<td>NMsFltIPFltTypeNATTDyn</td>
<td>268, bit 5</td>
<td>1 bit</td>
<td>Binary</td>
<td>NATT dynamic IP filter mask. If set for an IP filter request, this input filter matches dynamic IP filters for NAT traversal IP tunnels. If not set, NATT dynamic IP filters are not matched.</td>
</tr>
<tr>
<td>NMsFltIPFltTypeNRF</td>
<td>268, bit 6</td>
<td>1 bit</td>
<td>Binary</td>
<td>NATT resolution filter mask. If set for an IP filter request, this input filter matches NATT resolution filters for NAT traversal IP tunnels. If not set, NATT resolution filters are not matched.</td>
</tr>
<tr>
<td>NMsFltRsvd4</td>
<td>268, bit 7</td>
<td>1 bit</td>
<td>Binary</td>
<td>Reserved bit. Must be set to 0.</td>
</tr>
</tbody>
</table>

Chapter 14. Network management interfaces 491
<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsFltDisciplineIPSec</td>
<td>269, bit 0</td>
<td>1 bit</td>
<td>Binary</td>
<td>Discipline filter mask. If set for an NSS client info request, this input filter matches NSS clients that are registered for the IPSec discipline.</td>
</tr>
<tr>
<td>NMsFltDisciplineXMLApp</td>
<td>269, bit 1</td>
<td>1 bit</td>
<td>Binary</td>
<td>Discipline filter mask. If set for an NSS client info request, this input filter matches NSS clients that are registered for the XMLAppliance discipline.</td>
</tr>
<tr>
<td>NMsFltDisciplineRsvd</td>
<td>269, bit 2</td>
<td>6 bits</td>
<td>Binary</td>
<td>Reserved bits. Must be set to 0.</td>
</tr>
<tr>
<td>NMsFltSAState</td>
<td>270</td>
<td>1 byte</td>
<td>Binary</td>
<td>SA state selector. For an IP or IKE tunnel request, this field limits the results based on the security association (SA) state. Valid state values are as follows:</td>
</tr>
</tbody>
</table>
|                        |        |        |        | **NMsec_SASTATE_INACTIVE** (1)  
|                        |        |        |        | Tunnel is inactive                                                                   |
|                        |        |        |        | **NMsec_SASTATE_PENDING** (2)  
|                        |        |        |        | Tunnel is awaiting negotiation                                                      |
|                        |        |        |        | **NMsec_SASTATE_INCOMPLETE** (3)  
|                        |        |        |        | Tunnel is in negotiation                                                            |
|                        |        |        |        | **NMsec_SASTATE_ACTIVE** (4)  
|                        |        |        |        | Tunnel is active                                                                    |
|                        |        |        |        | **NMsec_SASTATE_EXPIRED** (5)  
|                        |        |        |        | Tunnel is expired                                                                   |
| NMsFltSWSAShadow      | 271    | 1 byte | Binary | SWSA shadow indicator. This is applicable for IP filters and for dynamic IP tunnels. Valid values are as follows:                            |
|                        |        |        |        | **NMsec_SHADOW** (1)  
|                        |        |        |        | Match objects that are SWSA shadow objects originating from a remote distributor only.                                               |
|                        |        |        |        | **NMsec_NONSHADOW** (0)  
|                        |        |        |        | Match objects that are not SWSA shadow objects only.                                |

**IPSec NMI control request formats**

Control request record formats vary with each request type.
- The following section, NMsecTunnel, is used across several record types.
- All EBCDIC fields are blank-padded and they are not NUL-terminated.
Table 40. NMsecTunnel field descriptions

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsecTunnelName</td>
<td>0</td>
<td>48 bytes</td>
<td>EBCDIC</td>
<td>The name that is associated with the tunnel. This name comes from a Policy Agent configuration file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• For manual tunnels, this is an IpManVpnActionName name.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• For dynamic IPSec tunnels, this is a LocalDynVpnRuleName name.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• For IKE tunnels, this is a KeyExchangeRuleName name.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This field must be set to blanks when a tunnel name is not specified.</td>
</tr>
<tr>
<td>NMsecTunnelID</td>
<td>48</td>
<td>48 bytes</td>
<td>EBCDIC</td>
<td>The tunnel ID that is associated with this tunnel.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This field is used for any refresh and deactivation requests. This field must be set to blanks when a tunnel ID is not specified.</td>
</tr>
<tr>
<td>NMsecStatus</td>
<td>96</td>
<td>1 byte</td>
<td>Binary</td>
<td>Tunnel Status. This field is set to 0 on a request message. On a response this field is set to the status of the tunnel’s state change. Valid state values are as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>NMsec_TUNSTATUS_NOTFOUND</strong> (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The requested tunnel was not found.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>NMsec_TUNSTATUS_STATEUPDATED</strong> (2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The tunnel’s state was updated.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>NMsec_TUNSTATUS_STATEALREADYSET</strong> (3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The tunnel’s state was already set to the state requested.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>NMsec_TUNSTATUS_NOKER</strong> (5)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Applicable only to dynamic tunnel activation, this status indicates that there is no KeyExchangeRule rule corresponding to the requested LocalDynVpnRule rule.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>NMsec_TUNSTATUS_NOFILTER</strong> (6)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Applicable only to dynamic tunnel activation, this status indicates that there is no dynamic IPSec IpFilterRule rule corresponding to the requested LocalDynVpnRule rule.</td>
</tr>
<tr>
<td>NMsecTunnelRsvd1</td>
<td>97</td>
<td>3 bytes</td>
<td>Binary</td>
<td>Reserved. Must be set to 0.</td>
</tr>
</tbody>
</table>

**NMsec_ACTIVATE_IPTUNMANUAL**

**NMsec_ACTIVATE_IPTUNMANUAL** request format

<table>
<thead>
<tr>
<th>NMsecMessageType hdr</th>
<th>NMsecRecordHdr</th>
<th>NMsecSecDesc</th>
<th>NMsecTunnel (1-1n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request records (0 or 1)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Figure 11. NMsec_ACTIVATE_IPTUNMANUAL request form*

Activates one or more manual tunnels. The request format contains zero or one record with one fixed-length section that contains one or more NMsecTunnel
instances (described in Table 40 on page 493). Each NMsecTunnel instance identifies a manual tunnel to activate. If the Request Record is not present then all manual tunnels are activated.

**Restriction:** Manual tunnel activation requests for multiple tunnels must contain uniform tunnel specifications, either tunnel IDs or tunnel names.

**NMsec_ACTIVATE_IPTUNDYN**

Activates a dynamic IPSec tunnel. Each record has one section, NMsecTunnel (described in Table 40 on page 493). The NMsecTunnel section identifies the dynamic tunnel that is to be activated.

**NMsec_DEACTIVATE_IPTUNMANUAL**

Deactivates one or more manual tunnels. The request format contains zero or one record with one fixed-length section that contains one or more NMsecTunnel instances (described in Table 40 on page 493). Each NMsecTunnel instance identifies a manual tunnel to deactivate. If the request record is not present then all manual tunnels are deactivated.

**Restriction:** Manual tunnel deactivation requests for multiple tunnels must contain uniform tunnel specifications, either tunnel IDs or tunnel names.

**NMsec_DEACTIVATE_IPTUNDYN**

Deactivates one or all dynamic tunnels. The request format contains zero or one record with one section, NMsecTunnel (described in Table 40 on page 493). The NMsecTunnel section identifies the dynamic tunnel to be deactivated. If the Request Record is not present then all dynamic tunnels are deactivated.
**NMsec_DEACTIVATE_IKETUN**

Deactivates one or all IKE tunnels. The request format contains zero or one record with one section, NMsecTunnel, (described in Table 40 on page 493). The NMsecTunnel section identifies the IKE tunnel to be deactivated. If the Request Record is not present then all IKE tunnels are deactivated.

**NMsec_REFRESH_IPTUNDYN**

Refreshes a dynamic IPSec tunnel. Contains a single record that has one section, NMsecTunnel (described in Table 40 on page 493). The NMsecTunnel section identifies the dynamic tunnel to be refreshed.

**NMsec_REFRESH_IKETUN**

Refreshes an IKE tunnel. Contains a single record that has one section, NMsecTunnel (described in Table 40 on page 493). The NMsecTunnel section identifies the IKE tunnel to be refreshed.

**NMsec_LOAD_POLICY**

Switches between default IP filters and policy-based IP filters. The call indicates whether the default policy or configured policy should be loaded. After this call completes, the client will have initiated the policy load operation.

Selecting the NMsec_FLT_DEFAULT option causes the stack to use the default IP filter rules. Default IP filter rules consist of the IP filter rules that are specified by the TCPIP profile, if any, and an implicit DENY-ALL rule. While the profile IP filters are in effect, manual, dynamic, and IKE tunnels still exist, but they are not used. These tunnels might expire or be deactivated. Tunnel refreshes might not occur and new dynamic tunnels might not be activated.
Switching between default and configured policy is useful when there is a need to quickly restrict system access to a very small subset of allowable traffic. This might occur when a system is under some sort of security attack or just before going into a maintenance state.

Selecting the NMsec_FLT_POLICY option causes the stack to use the policy IP filter rules as supplied from a policy configuration file or server. If no policy IP filters were previously defined to the stack, the stack continues to use the default IP filter rules until the policy configuration file is installed by the Policy Agent. If policy IP filter rules were previously defined to the stack, those policy IP filters become effective again. Tunnel activity can resume, including refreshes and new activations. The IKE daemon attempts to perform all configured autoactivations.

The active policy definitions (default or configured) are remembered across activations of the stack and system IPLs.

Each record has one section, NMsecPolicySource, which contains the following data.

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsPolSrcSource</td>
<td>0</td>
<td>1 byte</td>
<td>Binary</td>
<td>Indicates which policy should be loaded or reloaded. One of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NMsec_FLT_POLICY (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NMsec_FLT_DEFAULT (0)</td>
</tr>
<tr>
<td>NMsPolSrcRsvd1</td>
<td>1</td>
<td>3 bytes</td>
<td>Binary</td>
<td>Reserved. Set to zeroes.</td>
</tr>
</tbody>
</table>

**IPSec NMI response messages**

Response messages contain zero or more response records. The layout of each record depends on the message type. The fields in the response records for each request type are described in the following sections. Some section layouts are shared between several record types.

All EBCDIC fields are blank-padded and are not NUL-terminated.

**NMsec_GET_STACKINFO**

<table>
<thead>
<tr>
<th>NMsec_GET_STACKINFO response format</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsecMessageHdr</td>
</tr>
<tr>
<td>Response records (1 per stack)</td>
</tr>
</tbody>
</table>

Each record returned identifies a single stack that is active on the system. Each record has the following sections:

- One section, NMsecStack, describes attributes of the stack.

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsStackIPSecurity</td>
<td>0, bit 0</td>
<td>1 bit</td>
<td>Binary</td>
<td>If set, IP security is enabled for this stack.</td>
</tr>
<tr>
<td>Field</td>
<td>Offset</td>
<td>Length</td>
<td>Format</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------</td>
<td>--------</td>
<td>--------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NMsStackIPv6Security</td>
<td>0, bit 1</td>
<td>1 bit</td>
<td>Binary</td>
<td>If set, IPv6 IP security is enabled for this stack.</td>
</tr>
<tr>
<td>NMsStackDVIPSec</td>
<td>0, bit 2</td>
<td>1 bit</td>
<td>Binary</td>
<td>If set, sysplex-wide security associations (DVIPSEC) is enabled for this stack.</td>
</tr>
<tr>
<td>NMsStackLogging</td>
<td>0, bit 3</td>
<td>1 bit</td>
<td>Binary</td>
<td>If set, filter logging is enabled for this stack.</td>
</tr>
<tr>
<td>NMsStackPreDecap</td>
<td>0, bit 4</td>
<td>1 bit</td>
<td>Binary</td>
<td>If set, pre-decapsulation filtering is enabled for this stack.</td>
</tr>
<tr>
<td>NMsStackFilterSet</td>
<td>0, bit 5</td>
<td>1 bit</td>
<td>Binary</td>
<td>Current filter set indicator. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>NMsec_FLT_DEFAULT (0)</strong>&lt;br&gt;Default filters are currently in effect. The default filters originate from the TCP/IP profile.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>NMsec_FLT_POLICY (1)</strong>&lt;br&gt;Policy filters are currently in effect. The policy filters originate in the Policy Agent configuration.</td>
</tr>
<tr>
<td>NMsStackRsvd1</td>
<td>0, bit 6</td>
<td>26 bits</td>
<td>Binary</td>
<td>Reserved bits.</td>
</tr>
<tr>
<td>NMsStackName</td>
<td>4</td>
<td>24 bytes</td>
<td>EBCDIC</td>
<td>The job name of the TCP/IP stack.</td>
</tr>
<tr>
<td>NMsStackNATKeepAlive</td>
<td>28</td>
<td>4 bytes</td>
<td>Binary</td>
<td>NAT keepalive interval, in seconds, used to regulate sending NAT keepalive messages for a NAT traversal tunnel when a NAT device is detected in front of the local host.</td>
</tr>
<tr>
<td>NMsStackFilterCount</td>
<td>32</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Number of configured filters in the current filter set. This number does not include any dynamic filters.</td>
</tr>
<tr>
<td>NMsStackDefFltCount</td>
<td>36</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Number of defensive filters that are currently installed in the TCP/IP stack.</td>
</tr>
</tbody>
</table>
Table 42. NMsecStack structure (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsecStackDefFltMode</td>
<td>40</td>
<td>1 byte</td>
<td>Binary</td>
<td>Defensive filtering mode. Possible values are:</td>
</tr>
<tr>
<td>NMsStackDefFltMode</td>
<td></td>
<td></td>
<td></td>
<td>- <strong>NMsec.DEFFLT_INACTIVE</strong> (0) Defensive filtering is inactive for the stack.</td>
</tr>
<tr>
<td>NMsStackDefFltMode</td>
<td></td>
<td></td>
<td></td>
<td>- <strong>NMsec.DEFFLT_ACTIVE</strong> (1) Defensive filtering is active for the stack.</td>
</tr>
<tr>
<td>NMsStackDefFltMode</td>
<td></td>
<td></td>
<td></td>
<td>- <strong>NMsec.DEFFLT_SIMULATE</strong> (2) Defensive filtering is active for the stack.</td>
</tr>
<tr>
<td>NMsStackRsvd2</td>
<td>41</td>
<td>3 bytes</td>
<td>Binary</td>
<td>Reserved bytes.</td>
</tr>
</tbody>
</table>

- Zero to ten NMsecStackExclAddr sections that contain the defensive filtering exclusion list.

Table 43. NMsecStackExclAddr structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsStackExclAddrFlagIsSingle</td>
<td>0, bit 0</td>
<td>1 bit</td>
<td>Binary</td>
<td>Single exclusion address indicator. If set, the exclusion address is indicated by the NMsStackExclAddr4 or NMsStackExclAddr6 field.</td>
</tr>
<tr>
<td>NMsStackExclAddrFlagIsPrefix</td>
<td>0, bit 1</td>
<td>1 bit</td>
<td>Binary</td>
<td>Prefixed exclusion address indicator. If set, the exclusion address is indicated by the NMsStackExclAddr4 or NMsStackExclAddr6 field, and the exclusion address prefix length is indicated by the NMsStackExclAddrPrefix field.</td>
</tr>
<tr>
<td>NMsStackExclAddrFlagIPv6</td>
<td>0, bit 2</td>
<td>1 bit</td>
<td>Binary</td>
<td>IPv6 indicator. If set, exclusion addresses are IPv6; otherwise they are IPv4.</td>
</tr>
<tr>
<td>NMsStackExclAddrRsvd1</td>
<td>0, bit 3</td>
<td>5 bits</td>
<td>Binary</td>
<td>Reserved bits.</td>
</tr>
<tr>
<td>NMsStackExclAddrRsvd2</td>
<td>1</td>
<td>2 bytes</td>
<td>Binary</td>
<td>Reserved.</td>
</tr>
<tr>
<td>NMsStackExclAddrPrefix</td>
<td>3</td>
<td>1 byte</td>
<td>Binary</td>
<td>If the NMsStackExclAddrIsPrefix field is set, this value is the length of the defensive filter exclusion address prefix, in bits.</td>
</tr>
<tr>
<td>NMsStackExclAddr4</td>
<td>4</td>
<td>4 bytes</td>
<td>Binary</td>
<td>If the NMsStackExclAddrFlagIsSingle field is set, this value is a defensive filter IPv4 exclusion address. If the NMsStackExclAddrFlagIsPrefix field is set, this value is a defensive filter IPv6 exclusion address base.</td>
</tr>
</tbody>
</table>
NMsec_GET_SUMMARY

NMsec_GET_SUMMARY response format

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsStatP1Active</td>
<td>0</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Current number of active IKE tunnels.</td>
</tr>
<tr>
<td>NMsStatP1InProgress</td>
<td>4</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Current number of IKE tunnels in-progress, either pending or in negotiation.</td>
</tr>
<tr>
<td>NMsStatP1Expired</td>
<td>8</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Current number of expired IKE tunnels. This is a current count (not cumulative). Expired IKE tunnels are retained until all associated dynamic tunnels have expired.</td>
</tr>
<tr>
<td>NMsStatP1LclActSuccess</td>
<td>12</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Cumulative number of successful IKE tunnel activations that were initiated locally for this stack over the life of the IKE daemon. This data is cumulative even across stack restarts.</td>
</tr>
<tr>
<td>NMsStatP1RmtActSuccess</td>
<td>20</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Cumulative number of successful IKE tunnel activations that were initiated remotely for this stack over the life of the IKE daemon. This data is cumulative even across stack restarts.</td>
</tr>
<tr>
<td>NMsStatP1LclActFailure</td>
<td>28</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Cumulative number of failed IKE tunnel activations that were initiated locally for this stack over the life of the IKE daemon. This data is cumulative even across stack restarts.</td>
</tr>
<tr>
<td>NMsStatP1RmtActFailure</td>
<td>36</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Cumulative number of failed IKE tunnel activations that were initiated remotely for this stack over the life of the IKE daemon. This data is cumulative even across stack restarts.</td>
</tr>
</tbody>
</table>

Figure 19. NMsec_GET_SUMMARY response format

For the requested stack, one record is returned, which indicates statistical data. This record has a single section, NMsecStatistics, that contains the following data.

Table 44. NMsecStatistics structure
Table 44. NMsecStatistics structure (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMStatP1Retranmit</td>
<td>44</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Cumulative number of retransmitted key exchange (phase 1) messages sent for this stack over the life of the IKE daemon.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This data is cumulative even across stack restarts.</td>
</tr>
<tr>
<td>NMStatP1Replay</td>
<td>52</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Cumulative number of replayed key exchange (phase 1) messages received for this stack over the life of the IKE daemon.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This data is cumulative even across stack restarts.</td>
</tr>
<tr>
<td>NMStatP1Invalid</td>
<td>60</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Cumulative number of key exchange (phase 1) messages that are not valid that have been received for this stack over the life of the IKE daemon.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This number does not include message authentication failures.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This data is cumulative even across stack restarts.</td>
</tr>
<tr>
<td>NMStatP1AuthFail</td>
<td>68</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Cumulative number of key exchange (phase 1) message authentication failures for this stack over the life of the IKE daemon.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This data is cumulative even across stack restarts.</td>
</tr>
<tr>
<td>NMStatP2Active</td>
<td>76</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Current number of active dynamic tunnels known to the TCP/IP stack.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This number does not include SWSA shadow tunnels or manual tunnels.</td>
</tr>
<tr>
<td>NMStatP2ActiveShadow</td>
<td>80</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Current number of active dynamic SWSA shadow tunnels known to the TCP/IP stack.</td>
</tr>
<tr>
<td>NMStatP2InProgress</td>
<td>84</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Current number of dynamic tunnels in progress, either pending or in negotiation.</td>
</tr>
<tr>
<td>NMStatP2Expired</td>
<td>88</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Current number of expired dynamic tunnels known to the TCP/IP stack.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This includes both non-shadow and shadow tunnels.</td>
</tr>
<tr>
<td>NMStatP2ActSuccess</td>
<td>92</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Cumulative number of successful dynamic tunnel activations for this stack over the life of the IKE daemon.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This data is cumulative even across stack restarts.</td>
</tr>
<tr>
<td>Field</td>
<td>Offset</td>
<td>Length</td>
<td>Format</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NMsStatP2ActFailure</td>
<td>100</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Cumulative number of failed dynamic tunnel activations for this stack over the life of the IKE daemon. This data is cumulative even across stack restarts.</td>
</tr>
<tr>
<td>NMsStatP2Retransmit</td>
<td>108</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Cumulative number of retransmitted QUICKMODE (phase 2) messages sent for this stack over the life of the IKE daemon. This data is cumulative even across stack restarts.</td>
</tr>
<tr>
<td>NMsStatP2Replay</td>
<td>116</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Cumulative number of replayed QUICKMODE (phase 2) messages received for this stack over the life of the IKE daemon. This data is cumulative even across stack restarts.</td>
</tr>
<tr>
<td>NMsStatP2Invalid</td>
<td>124</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Cumulative number of QUICKMODE (phase 2) messages that were not valid that were received for this stack over the life of the IKE daemon. This data is cumulative even across stack restarts.</td>
</tr>
<tr>
<td>NMsStatP1BytesOut</td>
<td>132</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Cumulative number of outbound bytes of IKE traffic protected by IKE tunnels for this stack over the life of the IKE daemon. This data is cumulative even across stack restarts.</td>
</tr>
<tr>
<td>NMsStatP1BytesIn</td>
<td>140</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Cumulative number of inbound bytes of IKE traffic protected by IKE tunnels for this stack over the life of the IKE daemon. This data is cumulative even across stack restarts.</td>
</tr>
<tr>
<td>NMsStatP2BytesOut</td>
<td>148</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Cumulative number of outbound bytes of IP traffic protected by dynamic tunnels for this stack over the life of the TCP/IP stack.</td>
</tr>
<tr>
<td>NMsStatP2BytesIn</td>
<td>156</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Cumulative number of inbound bytes of IP traffic protected by dynamic tunnels for this stack over the life of the TCP/IP stack.</td>
</tr>
<tr>
<td>NMsStatFilterDeny</td>
<td>164</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Cumulative number of packets denied as the result of IP filter action DENY for this stack over the life of the TCP/IP stack.</td>
</tr>
</tbody>
</table>
Table 44. NMsecStatistics structure (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsecStatFilterMismatch</td>
<td>172</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Cumulative number of packets denied as the result of mismatch with filter action for this stack over the life of the TCP/IP stack.</td>
</tr>
<tr>
<td>NMsecStatFilterMatch</td>
<td>180</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Cumulative number of packets matching an IP filter over the life of the TCP/IP stack. This includes generic (permit and deny) filters, IPSec filters, and defensive filters.</td>
</tr>
</tbody>
</table>

**NMsec_GET_IPFLTLCURR, NMsec_GET_IPFLTDEFAULT, and NMsec_GET_IPFLTPOILICY**

NMsec_GET_IPFLTLCURR, NMsec_GET_IPFLTDEFAULT, NMsec_GET_IPFLTPOILICY response format

![NMsec_GET_IPFLTLCURR, NMsec_GET_IPFLTDEFAULT, NMsec_GET_IPFLTPOILICY response format](image)

For the requested stack, zero or more records, which represent IP filters, are returned. Each record that is returned identifies a single IP filter and contains two sections that describe the data. Filters are presented in an ordered sequence. Generic IP filters (permit or deny), manual tunnel filters, and dynamic anchor filters are presented in the order in which they are configured. Dynamic anchor filters are presented immediately before the dynamic and NATT anchor filters that are associated with them. NATT anchor filters are presented immediately before the NATT dynamic filters that are associated with them. NAT traversal resolution filters (NRFs) for a NATT anchor filter are presented immediately after the NATT dynamic filters for that NATT anchor filter. Defensive filters are presented based on the order in which they were added to the stack, most recent to least recent. Some IP filters might be absent from the result list because of input filters provided on the request message.

- One section, NMsecIPFilter, describes the basic properties of an IP filter. This section contains the following data.

Table 45. NMsecIPFilter structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsecIPFltName</td>
<td>0</td>
<td>48 bytes</td>
<td>EBCDIC</td>
<td>Filter rule name. Bytes 41-48 of the filter rule name consist of the filter rule name extension, which is a numeric extension used to distinguish between distinct filter rules that result from the same configured filter rule.</td>
</tr>
<tr>
<td>NMsecIPFltGroupName</td>
<td>48</td>
<td>48 bytes</td>
<td>EBCDIC</td>
<td>Filter rule group name or blank if there is no filter group.</td>
</tr>
<tr>
<td>NMsecIPFltLSASName</td>
<td>96</td>
<td>48 bytes</td>
<td>EBCDIC</td>
<td>Local start action name or blank if there is no local start action.</td>
</tr>
<tr>
<td>NMsecIPFltVPNActionName</td>
<td>144</td>
<td>48 bytes</td>
<td>EBCDIC</td>
<td>VPN action name or blank if there is no VPN action.</td>
</tr>
<tr>
<td>NMsecIPFltTunnelID</td>
<td>192</td>
<td>48 bytes</td>
<td>EBCDIC</td>
<td>Associated tunnel ID or blank if there is no associated tunnel.</td>
</tr>
<tr>
<td>Field</td>
<td>Offset</td>
<td>Length</td>
<td>Format</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>--------</td>
<td>--------</td>
<td>---------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NMsIPFltFlagIPv6</td>
<td>240, bit 0</td>
<td>1 bit</td>
<td>Binary</td>
<td>IPv6 indicator. If set, IP addresses for traffic and security endpoints are IPv6; otherwise they are IPv4.</td>
</tr>
<tr>
<td>NMsIPFltFlagOnDemand</td>
<td>240, bit 1</td>
<td>1 bit</td>
<td>Binary</td>
<td>On-demand indicator. If set for a dynamic anchor filter, a dynamic filter, a NAT traversal anchor filter, or a NAT traversal dynamic filter, this value indicates that on-demand activations are permitted for this traffic specification.</td>
</tr>
<tr>
<td>NMsIPFltFlagShadow</td>
<td>240, bit 2</td>
<td>1 bit</td>
<td>Binary</td>
<td>SWSA shadow indicator. If set for a dynamic filter, this value indicates that the filter originated from a distributing stack.</td>
</tr>
<tr>
<td>NMsIPFltFlagSrcIsSingle</td>
<td>240, bit 3</td>
<td>1 bit</td>
<td>Binary</td>
<td>Single source address indicator. If set, the source address is indicated by the NMsIPFltSrcAddr4 or NMsIPFltSrcAddr6 field.</td>
</tr>
<tr>
<td>NMsIPFltFlagSrcIsPrefix</td>
<td>240, bit 4</td>
<td>1 bit</td>
<td>Binary</td>
<td>Prefixed source address indicator. If set, the source address is indicated by the NMsIPFltSrcAddr4 or NMsIPFltSrcAddr6 field, and the source address prefix is indicated by the NMsIPFltSrcAddrPrefix field.</td>
</tr>
<tr>
<td>NMsIPFltFlagSrcIsRange</td>
<td>240, bit 5</td>
<td>1 bit</td>
<td>Binary</td>
<td>Ranged source address indicator. If set, the source address range is indicated by the NMsIPFltSrcAddr4 and NMsIPFltSrcAddrRange4 fields, or the NMsIPFltSrcAddr6 and NMsIPFltSrcAddrRange6 fields.</td>
</tr>
<tr>
<td>NMsIPFltFlagDstIsSingle</td>
<td>240, bit 6</td>
<td>1 bit</td>
<td>Binary</td>
<td>Single destination address indicator. If set, the destination address is indicated by the NMsIPFltDstAddr4 or NMsIPFltDstAddr6 field.</td>
</tr>
<tr>
<td>NMsIPFltFlagDstIsPrefix</td>
<td>240, bit 7</td>
<td>1 bit</td>
<td>Binary</td>
<td>Prefixed destination address indicator. If set, the destination address is indicated by the NMsIPFltDstAddr4 or NMsIPFltDstAddr6 field, and the destination address prefix is indicated by the NMsIPFltDstAddrPrefix field.</td>
</tr>
<tr>
<td>NMsIPFltFlagDstIsRange</td>
<td>241, bit 0</td>
<td>1 bit</td>
<td>Binary</td>
<td>Ranged destination address indicator. If set, the destination address range is indicated by the NMsIPFltDstAddr4 and NMsIPFltDstAddrRange4 fields, or the NMsIPFltDstAddr6 and NMsIPFltDstAddrRange6 fields.</td>
</tr>
<tr>
<td>NMsIPFltFlagProtoDef</td>
<td>241, bit 1</td>
<td>1 bit</td>
<td>Binary</td>
<td>Protocol indicator. If set, the filter protocol is indicated by the NMsIPFltProtocol field, otherwise, the filter applies to all protocols.</td>
</tr>
<tr>
<td>NMsIPFltFlagSrcPortDef</td>
<td>241, bit 2</td>
<td>1 bit</td>
<td>Binary</td>
<td>Source port indicator. If set, the source port range is indicated by the NMsIPFltSrcPort and NMsIPFltSrcPortRange fields; otherwise, the filter applies to all source ports. This indicator is not valid and has the value 0 if the filter protocol is not TCP or UDP.</td>
</tr>
<tr>
<td>NMsIPFltFlagDstPortDef</td>
<td>241, bit 3</td>
<td>1 bit</td>
<td>Binary</td>
<td>Destination port indicator. If set, the destination port range is indicated by the NMsIPFltDstPort and NMsIPFltDstPortRange fields; otherwise, the filter applies to all destination ports. This indicator is not valid and has the value 0 if the filter protocol is not TCP or UDP.</td>
</tr>
<tr>
<td>NMsIPFltFlagICMPTypeDef</td>
<td>241, bit 4</td>
<td>1 bit</td>
<td>Binary</td>
<td>ICMP type indicator. If set, the ICMP type is indicated by the NMsIPFltICMPType field; otherwise, the filter applies to all ICMP types. This indicator is not valid and has the value 0 if the filter protocol is not ICMP or ICMPv6.</td>
</tr>
<tr>
<td>NMsIPFltFlagICMPCodeDef</td>
<td>241, bit 5</td>
<td>1 bit</td>
<td>Binary</td>
<td>ICMP code indicator. If set, the ICMP code is indicated by the NMsIPFltICMPCode field; otherwise, the filter applies to all ICMP codes. This indicator is not valid and has the value 0 if the filter protocol is not ICMP or ICMPv6.</td>
</tr>
<tr>
<td>Field</td>
<td>Offset</td>
<td>Length</td>
<td>Format</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NMIPFiltFlagOSPFTypeDef</td>
<td>241, bit 6</td>
<td>1 bit</td>
<td>Binary</td>
<td>OSPF type indicator. If set, the OSPF type is indicated by the NMIPFiltOSPFTypeDef field; otherwise, the filter applies to all OSPF types. This indicator is not valid and has the value 0 if the filter protocol is not OSPF.</td>
</tr>
<tr>
<td>NMIPFiltFlagSecAddrPktGran</td>
<td>241, bit 7</td>
<td>1 bit</td>
<td>Binary</td>
<td>Source address granularity indicator. If set for a dynamic anchor filter, on-demand activations use the packet source address; otherwise, they use the filter source address specification.</td>
</tr>
<tr>
<td>NMIPFiltFlagDstAddrPktGran</td>
<td>242, bit 0</td>
<td>1 bit</td>
<td>Binary</td>
<td>Destination address granularity indicator. If set for a dynamic anchor filter, on-demand activations use the packet destination address; otherwise, they use the filter destination address specification.</td>
</tr>
<tr>
<td>NMIPFiltFlagProtoPktGran</td>
<td>242, bit 1</td>
<td>1 bit</td>
<td>Binary</td>
<td>Protocol granularity indicator. If set for a dynamic anchor filter, on-demand activations use packet protocol; otherwise, they use the filter protocol.</td>
</tr>
<tr>
<td>NMIPFiltFlagSecPortPktGran</td>
<td>242, bit 2</td>
<td>1 bit</td>
<td>Binary</td>
<td>Source port granularity indicator. If set for a dynamic anchor filter, on-demand activations use a packet source port; otherwise, they use the filter source port specification, when possible.</td>
</tr>
<tr>
<td>NMIPFiltFlagDstPortPktGran</td>
<td>242, bit 3</td>
<td>1 bit</td>
<td>Binary</td>
<td>Destination port granularity indicator. If set for a dynamic anchor filter, on-demand activations use packet destination port; otherwise, they use the filter destination port specification, when possible.</td>
</tr>
<tr>
<td>NMIPFiltFlagNATDetect</td>
<td>242, bit 4</td>
<td>1 bit</td>
<td>Binary</td>
<td>NAT indicator. If set for a dynamic filter, a NAT has been detected in front of the IPSec peer.</td>
</tr>
<tr>
<td>NMIPFiltFlagNAPTDetect</td>
<td>242, bit 5</td>
<td>1 bit</td>
<td>Binary</td>
<td>NAPT indicator. If set for a dynamic filter, a NAPT has been detected in front of the IPSec peer. It is possible that a NAPT exists but that it is detected only as a NAT.</td>
</tr>
<tr>
<td>NMIPFiltFlagGWDetect</td>
<td>242, bit 6</td>
<td>1 bit</td>
<td>Binary</td>
<td>NAT traversal gateway indicator. If set for a dynamic filter, the tunnel uses UDP encapsulation and the peer is acting as an IPSec gateway.</td>
</tr>
<tr>
<td>NMIPFiltFlagLogPermit</td>
<td>242, bit 7</td>
<td>1 bit</td>
<td>Binary</td>
<td>LogPermit indicator. If set, permitted packets that match this filter are logged.</td>
</tr>
<tr>
<td>NMIPFiltFlagLogDeny</td>
<td>243, bit 0</td>
<td>1 bit</td>
<td>Binary</td>
<td>LogDeny indicator. If set, denied packets that match this filter are logged.</td>
</tr>
<tr>
<td>NMIPFiltFlagMIPv6TypeDef</td>
<td>243, bit 1</td>
<td>1 bit</td>
<td>Binary</td>
<td>MIPv6 type indicator. If set, MIPv6 type is indicated by NMIPFiltMIPv6Type; otherwise, the filter applies to all MIPv6 types. This indicator is not valid and has the value 0 if the filter protocol is not MIPv6.</td>
</tr>
<tr>
<td>NMIPFiltFlagProtoOpaque</td>
<td>243, bit 2</td>
<td>1 bit</td>
<td>Binary</td>
<td>Opaque protocol indicator. If set, the filter matches packets that have an indeterminate protocol.</td>
</tr>
<tr>
<td>NMIPFiltFlagDiscardICMP</td>
<td>243, bit 3</td>
<td>1 bit</td>
<td>Binary</td>
<td>ICMP error indicator. If set and packets are discarded as a result of this filter rule, ICMP or ICMPv6 destination unreachable messages are sent to the packet origin, which indicates that the packet was administratively prohibited.</td>
</tr>
<tr>
<td>NMIPFiltFlagFragmentsOnly</td>
<td>243, bit 4</td>
<td>1 bit</td>
<td>Binary</td>
<td>Fragment indicator. If set, the filter matches fragmented packets. If clear, the filter matches both fragmented and non-fragmented packets.</td>
</tr>
<tr>
<td>NMIPFiltDefensiveGlobal</td>
<td>243, bit 5</td>
<td>1 bit</td>
<td>Binary</td>
<td>Defensive global indicator. If set for a defensive filter, the filter has a global scope. Not set for non-defensive filters.</td>
</tr>
<tr>
<td>NMIPFiltTransOpaque</td>
<td>243, bit 6</td>
<td>1 bit</td>
<td>Binary</td>
<td>Opaque transport selector indicator. If set, the filter matches packets that have indeterminate transport layer selectors (for example, port, type, or code).</td>
</tr>
<tr>
<td>Field</td>
<td>Offset</td>
<td>Length</td>
<td>Format</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>NMsecIPFilterMIPv6TypePktGran</td>
<td>243, bit 7</td>
<td>1 bit</td>
<td>Binary</td>
<td>MIPv6 type granularity indicator. If set for a dynamic anchor filter, on-demand activations use packet MIPv6 type value; otherwise, they use the filter MIPv6 type specification, when possible.</td>
</tr>
</tbody>
</table>
| NMsecIPFilterType | 244 | 1 byte | Binary | IP filter type. One of the following:  
  - NMsec_IPFLT_GENERIC (1)  
  - NMsec_IPFLT_MANUAL (2)  
  - NMsec_IPFLT_DYNANCHOR (3)  
  - NMsec_IPFLT_DYNAMIC (4)  
  - NMsec_IPFLT_NATTANCHOR (5)  
  - NMsec_IPFLT_NATTDYN (6)  
  - NMsec_IPFLT_NRF (7)  
  - NMsec_IPFLT_DEFENSIVE (8) |
| NMsecIPFilterState | 245 | 1 byte | Binary | IP filter state. One of the following:  
  - NMsec_IPFLT_INACTIVE (0)  
  - Filter is inactive as a result of a time condition.  
  - NMsec_IPFLT_ACTIVE (1)  
  - Filter is active. |
| NMsecIPFilterAction | 246 | 1 byte | Binary | IP filter action. One of the following:  
  - NMsec_IPFLT_PERMIT (1)  
  - NMsec_IPFLT_DENY (2)  
  - NMsec_IPFLT_IPSEC (3)  
  - NMsec_IPFLT_DEFENSIVE_SIMULATE (4) |
| NMsecIPFilterScope | 247 | 1 byte | Binary | IP filter scope. One of the following:  
  - NMsec_IPFLT_LOCAL (1)  
  - NMsec_IPFLT_ROUTED (2)  
  - NMsec_IPFLT_SCOPEALL (3) |
| NMsecIPFilterDirection | 248 | 1 byte | Binary | IP filter direction. One of the following:  
  - NMsec_IPFLT_INBOUND (1)  
  - NMsec_IPFLT_OUTBOUND (2) |
| NMsecIPFilterSecurityClass | 249 | 1 byte | Binary | IP filter security class. Valid values are in the range 0 - 255. The value 0 matches all security classes. |
| NMsecIPFilterTCPConnect | 250 | 1 byte | Binary | TCP connect qualifier. One of the following:  
  - NMsec_IPFLT_CONNECT_NONE (0)  
  - NMsec_IPFLT_CONNECT_INBOUND (1)  
  - NMsec_IPFLT_CONNECT_OUTBOUND (2) |
| NMsecIPFilterICMPTypePktGran | 251, bit 0 | 1 bit | Binary | ICMP type granularity indicator. If set for a dynamic anchor filter, on-demand activations use packet ICMP type value; otherwise, they use the filter ICMP type specification, when possible. |
| NMsecIPFilterICMPCodePktGran | 251, bit 1 | 1 bit | Binary | ICMP code granularity indicator. If set for a dynamic anchor filter, on-demand activations use packet ICMP code value; otherwise, they use the filter ICMP code specification, when possible. |
| NMsecIPFilterICMPv6TypePktGran | 251, bit 2 | 1 bit | Binary | ICMPv6 type granularity indicator. If set for a dynamic anchor filter, on-demand activations use packet ICMPv6 type value; otherwise, they use the filter ICMPv6 type specification, when possible. |
| NMsecIPFilterICMPv6CodePktGran | 251, bit 3 | 1 bit | Binary | ICMPv6 code granularity indicator. If set for a dynamic anchor filter, on-demand activations use packet ICMPv6 code value; otherwise, they use the filter ICMPv6 code specification, when possible. |
### Table 45. NMsecIPFilter structure (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMdIPFltRsvd2</td>
<td>251, bit 4</td>
<td>4 bits</td>
<td>Binary</td>
<td>Reserved</td>
</tr>
<tr>
<td>NMdIPFltProtocol</td>
<td>252</td>
<td>1 byte</td>
<td>Binary</td>
<td>IP filter protocol number, if the NMdIPFltFlagProtoDef field is set. This value corresponds to the IP protocol number in the IPv4 or IPv6 header.</td>
</tr>
<tr>
<td>NMdIPFltICMPType</td>
<td>253</td>
<td>1 byte</td>
<td>Binary</td>
<td>ICMP type, if the NMdIPFltFlagICMPTypeDef field is set.</td>
</tr>
<tr>
<td>NMdIPFltICMPCode</td>
<td>254</td>
<td>1 byte</td>
<td>Binary</td>
<td>ICMP code, if the NMdIPFltFlagICMPCodeDef field is set.</td>
</tr>
<tr>
<td>NMdIPFltOSPFType</td>
<td>255</td>
<td>1 byte</td>
<td>Binary</td>
<td>OSPF type, if the NMdIPFltFlagOSPFTypeDef field is set.</td>
</tr>
<tr>
<td>NMdIPFltSrcPort</td>
<td>256</td>
<td>2 bytes</td>
<td>Binary</td>
<td>Lower end of IP filter source port range, if the NMdIPFltFlagSrcPortDef field is set.</td>
</tr>
<tr>
<td>NMdIPFltSrcPortRange</td>
<td>258</td>
<td>2 bytes</td>
<td>Binary</td>
<td>Upper end of IP filter source port range, if the NMdIPFltFlagSrcPortDef field is set.</td>
</tr>
<tr>
<td>NMdIPFltDstPort</td>
<td>260</td>
<td>2 bytes</td>
<td>Binary</td>
<td>Lower end of IP filter destination port range, if the NMdIPFltFlagDstPortDef field is set.</td>
</tr>
<tr>
<td>NMdIPFltDstPortRange</td>
<td>262</td>
<td>2 bytes</td>
<td>Binary</td>
<td>Upper end of IP filter destination port range, if the NMdIPFltFlagDstPortDef field is set.</td>
</tr>
</tbody>
</table>
| NMdIPFltSrcAddr4 | 264    | 4 bytes | Binary     | One of the following:
  • If the NMdIPFltFlagSrcIsSingle field is set, the filter’s IPv4 or IPv6 source address
  • If the NMdIPFltFlagSrcIsPrefix field is set, the filter’s IPv4 or IPv6 source address base
  • If the NMdIPFltFlagSrcIsRange field is set, the low end of the filter’s IPv4 or IPv6 source address range
| NMdIPFltSrcAddr6 | 264    | 16 bytes | Binary     | If the NMdIPFltFlagSrcIsPrefix field is set, the length of the filter’s source address prefix, in bits. |
| NMdIPFltSrcAddrRange4 | 280 | 4 bytes | Binary     | If the NMdIPFltFlagSrcIsRange field is set, the high end of the filter’s IPv4 or IPv6 source address range. |
| NMdIPFltSrcAddrRange6 | 280 | 16 bytes | Binary     | If the NMdIPFltFlagSrcIsRange field is set, the high end of the filter’s IPv4 or IPv6 source address range. |
| NMdIPFltDstAddr4 | 296    | 4 bytes | Binary     | One of the following:
  • If the NMdIPFltFlagDstIsSingle field is set, the filter’s IPv4 or IPv6 destination address
  • If the NMdIPFltFlagDstIsPrefix field is set, the filter’s IPv4 or IPv6 destination address base
  • If the NMdIPFltFlagDstIsRange field is set, the low end of the filter’s IPv4 or IPv6 destination address range
| NMdIPFltDstAddr6 | 296    | 16 bytes | Binary     | If the NMdIPFltFlagDstIsPrefix field is set, the length of the filter’s destination address prefix, in bits. |
| NMdIPFltDstAddrRange4 | 312 | 4 bytes | Binary     | If the NMdIPFltFlagDstIsRange field is set, the high end of the filter’s IPv4 or IPv6 destination address range. |
| NMdIPFltDstAddrRange6 | 312 | 16 bytes | Binary     | If the NMdIPFltFlagDstIsRange field is set, the high end of the filter’s IPv4 or IPv6 destination address range. |
| NMdIPFltSrcAddrPrefix | 328 | 1 byte | Binary     | If the NMdIPFltFlagSrcIsPrefix field is set, the length of the filter’s source address prefix, in bits. |
| NMdIPFltDstAddrPrefix | 329 | 1 byte | Binary     | If the NMdIPFltFlagDstIsPrefix field is set, the length of the filter’s destination address prefix, in bits. |
| NMdIPFltRsvd3    | 330    | 1 byte | Binary     | Reserved.                                                                                       |
Table 45. NMsecIPFilter structure (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsecIPFltNATClientIDType</td>
<td>331</td>
<td>1 byte</td>
<td>Binary</td>
<td>The NATT client ID is present only when the peer is behind a NAT and a gateway, and the peer supplied a client ID. One of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>NMsec_IPFLT_IDNONE (0)</strong> No client ID. Either this is not a dynamic filter; the peer for this filter’s tunnel is not behind a NAT and a gateway; or no client ID was provided.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>NMsec_IPFLT_IDIP (1)</strong> Client ID is an IPv4 address.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>NMsec_IPFLT_IDRANGE (2)</strong> Client ID is an IPv4 address range.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>NMsec_IPFLT_IDPREFIX (3)</strong> Client ID is an IPv4 address prefix.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>NMsec_IPFLT_IDOTHER (4)</strong> Client ID is another type, represented as an MD5 hash of the ID data.</td>
</tr>
<tr>
<td>NMsecIPFltNATClientIDIP</td>
<td>332</td>
<td>4 bytes</td>
<td>Binary</td>
<td>If NATT client ID type is NMsec_IPFLT_IDIP, NMsec_IPFLT_IDRANGE, or NMsec_IPFLT_IDPREFIX, this field is the base IPv4 address for the client ID.</td>
</tr>
<tr>
<td>NMsecIPFltNATClientIDHash</td>
<td>332</td>
<td>16 bytes</td>
<td>Binary</td>
<td>If NATT client ID type is NMsec_IPFLT_IDRANGE, this field is the MD5 hash of the client’s ID.</td>
</tr>
<tr>
<td>NMsecIPFltNATClientIDIP2</td>
<td>348</td>
<td>4 bytes</td>
<td>Binary</td>
<td>If NATT client ID type is NMsec_IPFLT_IDRANGE, this field is the high end of the IPv4 client ID address range.</td>
</tr>
<tr>
<td>NMsecIPFltNATClientIDPrefix</td>
<td>348</td>
<td>4 bytes</td>
<td>Binary</td>
<td>If NATT client ID type is NMsec_IPFLT_IDPREFIX, this field is the prefix length of the IPv4 client ID, in bits.</td>
</tr>
<tr>
<td>NMsecIPFltRsvd4</td>
<td>348</td>
<td>16 bytes</td>
<td>Binary</td>
<td>Reserved.</td>
</tr>
<tr>
<td>NMsecIPFltNATPeerPort</td>
<td>364</td>
<td>2 bytes</td>
<td>Binary</td>
<td>If this is a dynamic filter for UDP-encapsulated NAT-traversal traffic, this field is the UDP port for the IKE peer; otherwise the value is 0.</td>
</tr>
<tr>
<td>NMsecIPFltNATNRFForigPort</td>
<td>366</td>
<td>2 bytes</td>
<td>Binary</td>
<td>If this is a NAT traversal resolution filter, this field is the original remote port for the TCP or UDP traffic; otherwise the value is 0.</td>
</tr>
<tr>
<td>NMsecIPFltMismatch</td>
<td>368</td>
<td>8 bytes</td>
<td>Binary</td>
<td>The cumulative number of packets denied as a result of a mismatch with this filter’s condition and action over the life of the TCP/IP stack.</td>
</tr>
<tr>
<td>NMsecIPFltMatch</td>
<td>376</td>
<td>8 bytes</td>
<td>Binary</td>
<td>The cumulative number of packets that matched this filter’s condition and action over the life of the TCP/IP stack.</td>
</tr>
<tr>
<td>NMsecIPFltCreateTime</td>
<td>384</td>
<td>4 bytes</td>
<td>Binary</td>
<td>For a statically defined filter that originates from the Policy Agent configuration, this field contains the UNIX time stamp that indicates when the filter was first defined to the current instance of the TCP/IP stack. For a filter that originates from the TCP/IP profile, this field contains the UNIX time stamp that indicates when the profile filter configuration was last replaced. For all dynamically defined filters, the value in this field is 0. For a defensive filter, this field contains the UNIX time stamp that indicates when the defensive filter was created.</td>
</tr>
</tbody>
</table>
Table 45. NMsecIPFilter structure (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsIPFiltUpdateTime</td>
<td>388</td>
<td>4 bytes</td>
<td>Binary</td>
<td>For a statically defined filter that originates from the Policy Agent configuration, this field contains the UNIX time stamp that indicates when the filter’s attributes were last updated in the current instance of the TCP/IP stack. For a filter that originates from the TCP/IP profile, this field contains the UNIX time stamp that indicates when the profile filter configuration was last replaced. For all dynamically defined filters, the value in this field is 0. For a defensive filter, this field contains the UNIX time stamp that indicates when the defensive filter’s attributes were last updated.</td>
</tr>
<tr>
<td>NMsIPFiltMIPv6Type</td>
<td>392</td>
<td>1 byte</td>
<td>Binary</td>
<td>MIPv6 type, if NMsIPFiltFlagMIPv6TypeDef is set.</td>
</tr>
<tr>
<td>NMsIPFiltTypeRange</td>
<td>393</td>
<td>1 byte</td>
<td>Binary</td>
<td>High end of ICMP, ICMPv6, or MIPv6 type range, if the corresponding flag is set.</td>
</tr>
<tr>
<td>NMsIPFiltCodeRange</td>
<td>394</td>
<td>1 byte</td>
<td>Binary</td>
<td>High end of ICMP or ICMPv6 code range, if the corresponding flag is set.</td>
</tr>
<tr>
<td>NMsIPFiltRemoteldType</td>
<td>395</td>
<td>1 byte</td>
<td>Binary</td>
<td>ISAKMP identity type for the remote security endpoint identity, as defined in RFC 2407. ISAKMP peers exchange and verify their identities as part of the IKE tunnel (phase 1) negotiation. These identities can be associated with anchor filters, dynamic filters, or NATT dynamic filters, and are used for filtering purposes. This field has the value 0 if the remote IKE identity is not present or if it is not applicable.</td>
</tr>
<tr>
<td>NMsIPFiltLifetimeExpire</td>
<td>396</td>
<td>8 bytes</td>
<td>Binary</td>
<td>For a defensive filter, this field indicates the time at which the filter expires, in UNIX format; otherwise this field has the value 0 for all non-defensive filters.</td>
</tr>
</tbody>
</table>

- One variable-length section contains the contents of the filter’s remote IKE identity. Regardless of the identity’s type, it is expressed as an EBCDIC string (an IP address is returned in printable form). For a dynamic anchor filter, this represents the identity or wildcarded identities that are permitted for remote communication on this filter. For a dynamic or NATT dynamic filter, this represents the actual remote IKE identity if remote identity filtering is in use. For all other filters, this section is empty. This section is also empty for SWSA shadow filters.

Each of the IP filter requests also returns a single secondary output record (described by the NMsMOutRec2 output record descriptor). This record describes global IP filtering configuration information that is currently in effect for the TCP/IP stack. This global result record contains a single section. This section consists of an NMsecStack structure, which is described in the "NMsec_GET_STACKINFO" on page 496 for the NMsec_GET_STACKINFO request.
NMsec_GET_PORTTRAN

For the requested stack, zero or more records are returned representing NAT traversal port translation entries. Each record that is returned contains a single section, NMsecPortTrans, which contains the following data.

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsecPortTransRemoteAddr</td>
<td>0</td>
<td>4 bytes</td>
<td>Binary</td>
<td>IPv4 public remote address for peer</td>
</tr>
<tr>
<td>NMsecPortTransRemoteInner</td>
<td>4</td>
<td>4 bytes</td>
<td>Binary</td>
<td>IPv4 private remote address for peer</td>
</tr>
<tr>
<td>NMsecPortTransProtocol</td>
<td>8</td>
<td>1 byte</td>
<td>Binary</td>
<td>Protocol for port translation entry, either IPPROTO_TCP or IPPROTO_UDP</td>
</tr>
<tr>
<td>NMsecPortTransRsvd1</td>
<td>9</td>
<td>24 bits</td>
<td>Binary</td>
<td>Reserved bits</td>
</tr>
<tr>
<td>NMsecPortTransOrigPort</td>
<td>12</td>
<td>2 bytes</td>
<td>Binary</td>
<td>Original remote port for connection</td>
</tr>
<tr>
<td>NMsecPortTransNewPort</td>
<td>14</td>
<td>2 bytes</td>
<td>Binary</td>
<td>Translated remote port; the port by which the connection is now known</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>to this TCP/IP stack</td>
</tr>
</tbody>
</table>

Figure 21. NMsec_GET_PORTTRAN response format

NMsec_GET_IPTUNMANUAL

For the requested stack, zero or more records are returned representing manual IP tunnels. Tunnels are presented in an unordered sequence. Each record returned contains two sections.

- One section, NMsecIPTunnel, describes the basic properties of an IP tunnel. This section contains the following data.

Note: This structure is reused for dynamic tunnels, so some possible field values are applicable only to dynamic tunnels.

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsecIPTunID</td>
<td>0</td>
<td>48 bytes</td>
<td>EBCDIC</td>
<td>Tunnel ID</td>
</tr>
<tr>
<td>NMsecIPTunVPNAction</td>
<td>48</td>
<td>48 bytes</td>
<td>EBCDIC</td>
<td>Tunnel VPN action name</td>
</tr>
</tbody>
</table>

Figure 22. NMsec_GET_IPTUNMANUAL response format

Table 46. NMsec_GET_PORTTRAN structure

Table 47. NMsecIPTunnel structure
<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsecIPTunFlagIPv6</td>
<td>96, bit 0</td>
<td>1 bit</td>
<td>Binary</td>
<td>IPv6 indicator. If set, security endpoint and data endpoint addresses are IPv6; otherwise they are IPv4</td>
</tr>
<tr>
<td>NMsecIPTunRsvd1</td>
<td>96, bit 1</td>
<td>31 bits</td>
<td>Binary</td>
<td>Reserved bits.</td>
</tr>
<tr>
<td>NMsecIPTunType</td>
<td>100</td>
<td>1 byte</td>
<td>Binary</td>
<td>Tunnel type. One of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NMsec_IPTUN_MANUAL (1) Manual IP tunnel</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NMsec_IPTUN_STACK (2) Dynamic IP tunnel, as known to the TCP/IP stack</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NMsec_IPTUN_IKE (3) Dynamic IP tunnel, as known to IKE</td>
</tr>
<tr>
<td>NMsecIPTunState</td>
<td>101</td>
<td>1 byte</td>
<td>Binary</td>
<td>Tunnel state. One of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NMsec_SASTATE_INACTIVE (1) Manual tunnel inactive</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NMsec_SASTATE_PENDING (2) Dynamic tunnel is awaiting negotiation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NMsec_SASTATE_INCOMPLETE (3) Dynamic tunnel is in negotiation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NMsec_SASTATE_ACTIVE (4) Manual or dynamic tunnel is active</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NMsec_SASTATE_EXPIRED (5) Dynamic tunnel is expired</td>
</tr>
<tr>
<td>NMsecIPTunRsvd2</td>
<td>102</td>
<td>2 bytes</td>
<td>Binary</td>
<td>Reserved</td>
</tr>
<tr>
<td>NMsecIPTunLcEndpt4</td>
<td>104</td>
<td>4 bytes</td>
<td>Binary</td>
<td>If this is an IPv4 tunnel, this field is the local security endpoint address</td>
</tr>
<tr>
<td>NMsecIPTunLcEndpt6</td>
<td>104</td>
<td>16 bytes</td>
<td>Binary</td>
<td>If this is an IPv6 tunnel, this field is the local security endpoint address</td>
</tr>
<tr>
<td>NMsecIPTunRmtEndpt4</td>
<td>120</td>
<td>4 bytes</td>
<td>Binary</td>
<td>If this is an IPv4 tunnel, this field is the remote security endpoint address</td>
</tr>
<tr>
<td>NMsecIPTunRmtEndpt6</td>
<td>120</td>
<td>16 bytes</td>
<td>Binary</td>
<td>If this is an IPv6 tunnel, this field is the remote security endpoint address</td>
</tr>
<tr>
<td>NMsecIPTunEncapMode</td>
<td>136</td>
<td>1 byte</td>
<td>Binary</td>
<td>Tunnel encapsulation mode. One of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NMsec_IPTUN_TUNNELMODE (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NMsec_IPTUN_TRANSPORTMODE (2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This field is not defined if the tunnel state is NMsec_SASTATE_PENDING or NMsec_SASTATE_INCOMPLETE.</td>
</tr>
<tr>
<td>NMsecIPTunAuthProto</td>
<td>137</td>
<td>1 byte</td>
<td>Binary</td>
<td>Tunnel authentication protocol. One of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IPPROTO_AH (51)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IPPROTO_ESP (50)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This field is not defined if the tunnel state is NMsec_SASTATE_PENDING or NMsec_SASTATE_INCOMPLETE.</td>
</tr>
</tbody>
</table>
Table 47. NMsecIPTunnel structure  (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsecIPTunAuthAlg</td>
<td>138</td>
<td>1 byte</td>
<td>Binary</td>
<td>Tunnel authentication algorithm. One of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NMsec_AUTH_HMAC_MD5 (38)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NMsec_AUTH_HMAC_SHA1 (39)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This field is not defined if the tunnel state is NMsec_SASTATE_PENDING or</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NMsec_SASTATE_INCOMPLETE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>Result:</strong> Although IP tunnels use these authentication algorithms by</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>truncating to 96 bits, the NMsecIPTunAuthAlg field explicitly uses the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>constant names and values that are indicated above to indicate the use of</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>HMAC-MD5-96 and HMAC-SHA1-96.</td>
</tr>
<tr>
<td>NMsecIPTunEncryptAlg</td>
<td>139</td>
<td>1 byte</td>
<td>Binary</td>
<td>Tunnel encryption algorithm. One of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NMsec_ENCR_NONE (0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NMsec_ENCR_NULL (11)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NMsec_ENCR_DES (18)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NMsec_ENCR_3DES (3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NMsec_ENCR_AES (12)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This field is not defined if the tunnel state is NMsec_SASTATE_PENDING or</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NMsec_SASTATE_INCOMPLETE.</td>
</tr>
<tr>
<td>NMsecIPTunInbAuthSPI</td>
<td>140</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Tunnel inbound authentication SPI.</td>
</tr>
<tr>
<td>NMsecIPTunOutbAuthSPI</td>
<td>144</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Tunnel outbound authentication SPI.</td>
</tr>
<tr>
<td>NMsecIPTunInbEncryptSPI</td>
<td>148</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Tunnel inbound encryption SPI.</td>
</tr>
<tr>
<td>NMsecIPTunOutbEncryptSPI</td>
<td>152</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Tunnel outbound encryption SPI.</td>
</tr>
<tr>
<td>NMsecIPTunStartTime</td>
<td>156</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Tunnel start time.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Indicates the time at which the tunnel was activated or refreshed, in UNIX</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>format.</td>
</tr>
</tbody>
</table>

- One section, NMsecIPManualTunnel, describes the attributes that are specific to a manual IP tunnel. This section contains the following data.
<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMslIPManTunOutPkt</td>
<td>0</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Outbound packet count for this tunnel</td>
</tr>
<tr>
<td>NMslIPManTunInPkt</td>
<td>8</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Inbound packet count for this tunnel</td>
</tr>
<tr>
<td>NMslIPManTunOutBytes</td>
<td>16</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Outbound byte count for this tunnel, representing the number of outbound data bytes protected by the tunnel</td>
</tr>
<tr>
<td>NMslIPManTunInBytes</td>
<td>24</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Inbound byte count for this tunnel, representing the number of inbound data bytes protected by the tunnel</td>
</tr>
<tr>
<td>NMslIPManTunClearDF</td>
<td>32, bit 0</td>
<td>1 bit</td>
<td>Binary</td>
<td>Don’t-fragment bit clear indicator. If this bit is set, the IPv4 tunnel mode tunnel clears the DF bit in the outer IP header. If neither the NMslIPManTunClearDF or NMslIPManTunSetDF value is set, the IPv4 tunnel mode tunnel passes through the DF bit from the inner IP header to the outer IP header. This field is not applicable and is always 0 for IPv6 or transport mode tunnels.</td>
</tr>
<tr>
<td>NMslIPManTunSetDF</td>
<td>32, bit 1</td>
<td>1 bit</td>
<td>Binary</td>
<td>Don’t-fragment bit set indicator. If this bit is set, IPv4 the tunnel mode tunnel sets the DF bit in the outer IP header. If neither the NMslIPManTunClearDF or NMslIPManTunSetDF value is set, the IPv4 tunnel mode tunnel passes the DF bit through from the inner IP header to the outer IP header. This field is not applicable and is always 0 for IPv6 or transport mode tunnels.</td>
</tr>
<tr>
<td>NMslIPManTunClearDSCP</td>
<td>32, bit 2</td>
<td>1 bit</td>
<td>Binary</td>
<td>DSCP clear indicator. If this bit is set, tunnel mode tunnel clears the DSCP bit in the outer IP header. If the value of this bit is 0, the tunnel mode tunnel copies the DSCP field from the inner IP header to the outer IP header. This field is not applicable is always 0 for transport mode tunnels.</td>
</tr>
<tr>
<td>NMslIPManTunRsvd1</td>
<td>32, bit 3</td>
<td>29 bits</td>
<td>Binary</td>
<td>Reserved bits</td>
</tr>
</tbody>
</table>
NMsec_GET_IPTUNDYNSTACK

For the requested stack, zero or more records are returned representing dynamic IP tunnels known to the TCP/IP stack. Depending on the input filters provided on the request, the tunnels can include SWSA shadow tunnels. SWSA shadow tunnels originate from a distributing stack and not from the local stack. Tunnels are presented in an unordered sequence, except that instances of a particular tunnel family (all sharing the same tunnel ID) are ordered from most recently activated to least recently activated.

Each record contains the following sections:

One section, NMsecIPTunnel, describes the basic properties of an IP tunnel. The layout of this section is described in Table 47 on page 509.

One section, NMsecIPDynTunnel, describes the basic properties of a dynamic IP tunnel. This section contains the following data.

Table 49. NMsecIPDynTunnel structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsIPDynUDPEncap</td>
<td>0, bit 0</td>
<td>1 bit</td>
<td>Binary</td>
<td>UDP encapsulation indicator. If set, the tunnel uses UDP encapsulation mode.</td>
</tr>
<tr>
<td>NMsIPDynLclNAT</td>
<td>0, bit 1</td>
<td>1 bit</td>
<td>Binary</td>
<td>Local NAT indicator. If set, a NAT has been detected in front of the local security endpoint.</td>
</tr>
<tr>
<td>NMsIPDynRmtNAT</td>
<td>0, bit 2</td>
<td>1 bit</td>
<td>Binary</td>
<td>Remote NAT indicator. If set, a NAT has been detected in front of the remote security endpoint.</td>
</tr>
<tr>
<td>NMsIPDynRmtNAPT</td>
<td>0, bit 3</td>
<td>1 bit</td>
<td>Binary</td>
<td>Remote NAPT indicator. If set, a NAPT has been detected in front of the remote security endpoint. It is possible that a NAPT might exist but might be detected only as a NAT.</td>
</tr>
<tr>
<td>NMsIPDynRmtGW</td>
<td>0, bit 4</td>
<td>1 bit</td>
<td>Binary</td>
<td>Remote NAT traversal gateway indicator. If set, the tunnel uses UDP encapsulation and the remote security endpoint is acting as an IPSec gateway.</td>
</tr>
<tr>
<td>NMsIPDynRmtZOS</td>
<td>0, bit 5</td>
<td>1 bit</td>
<td>Binary</td>
<td>Remote z/OS indicator. If set, the remote peer has been detected to be z/OS. It is possible that the remote peer might be running z/OS but not detected as such, if NAT traversal is not enabled.</td>
</tr>
<tr>
<td>NMsIPDynCanInitP2</td>
<td>0, bit 6</td>
<td>1 bit</td>
<td>Binary</td>
<td>Dynamic tunnel (phase 2) initiation indicator. If set, the local security endpoint can initiate dynamic tunnel negotiations with the remote security endpoint. Otherwise, the remote security endpoint must initiate dynamic tunnel negotiations. Either side can initiate refreshes.</td>
</tr>
</tbody>
</table>

Figure 23. NMsec_GET_IPTUNDYNSTACK response format
Table 49. NMsecIPDynTunnel structure (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsIPDynSrcIsSingle</td>
<td>0, bit 7</td>
<td>1 bit</td>
<td>Binary</td>
<td>Single source address indicator. If set, traffic source address is indicated by the NMsIPDynSrcAddr4 or NMsIPDynSrcAddr6 fields.</td>
</tr>
<tr>
<td>NMsIPDynSrcIsPrefix</td>
<td>1, bit 0</td>
<td>1 bit</td>
<td>Binary</td>
<td>Prefixed source address indicator. If set, traffic source address is indicated by the NMsIPDynSrcAddr4 or NMsIPDynSrcAddr6, fields and the source address prefix is indicated by the NMsIPDynSrcAddrPrefix field.</td>
</tr>
<tr>
<td>NMsIPDynSrcIsRange</td>
<td>1, bit 1</td>
<td>1 bit</td>
<td>Binary</td>
<td>Ranged source address indicator. If set, traffic source address range is indicated by the NMsIPDynSrcAddr4 and NMsIPDynSrcAddrRange4 fields, or by the NMsIPDynSrcAddr6 and NMsIPDynSrcAddrRange6 fields.</td>
</tr>
<tr>
<td>NMsIPDynDstIsSingle</td>
<td>1, bit 2</td>
<td>1 bit</td>
<td>Binary</td>
<td>Single destination address indicator. If set, traffic destination address is indicated by the NMsIPDynDstAddr4 or NMsIPDynDstAddr6 fields.</td>
</tr>
<tr>
<td>NMsIPDynDstIsPrefix</td>
<td>1, bit 3</td>
<td>1 bit</td>
<td>Binary</td>
<td>Prefixed destination address indicator. If set, traffic destination address is indicated by the NMsIPDynDstAddr4 or NMsIPDynDstAddr6 fields, and destination address prefix is indicated by the NMsIPDynDstAddrPrefix field.</td>
</tr>
<tr>
<td>NMsIPDynDstIsRange</td>
<td>1, bit 4</td>
<td>1 bit</td>
<td>Binary</td>
<td>Ranged destination address indicator. If set, traffic destination address range is indicated by the NMsIPDynDstAddr4 and NMsIPDynDstAddrRange4 fields, or by the NMsIPDynDstAddr6 and NMsIPDynDstAddrRange6 fields.</td>
</tr>
<tr>
<td>NMsIPDynTransportOpaque</td>
<td>1, bit 5</td>
<td>1 bit</td>
<td>Binary</td>
<td>Opaque transport selector indicator. If set, the dynamic tunnel is protecting data traffic in which the upper layer selectors, source and destination ports, ICMP or ICMPv6 type, and code or IPv6 Mobility header type are not available as a result of fragmentation.</td>
</tr>
<tr>
<td>NMsIPDynRsvd1</td>
<td>1, bit 6</td>
<td>18 bits</td>
<td>Binary</td>
<td>Reserved bits.</td>
</tr>
<tr>
<td>NMsIPDynVPNRule</td>
<td>4</td>
<td>48 bytes</td>
<td>EBCDIC</td>
<td>Dynamic VPN rule name for this tunnel; otherwise, blank if there is no local dynamic VPN rule.</td>
</tr>
<tr>
<td>NMsIPDynPTTunnelID</td>
<td>52</td>
<td>48 bytes</td>
<td>EBCDIC</td>
<td>Tunnel ID for this tunnel’s parent IKE (phase 1) tunnel. As a result of refreshes, this tunnel ID might represent multiple related IKE tunnels.</td>
</tr>
<tr>
<td>NMsIPDynLifesize</td>
<td>100</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Tunnel lifesize. If not 0, indicates the negotiated lifesize value limit for the tunnel, in bytes.</td>
</tr>
<tr>
<td>NMsIPDynLifesizeRefresh</td>
<td>108</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Tunnel lifesize refresh. If not 0, indicates the lifesize value at which the tunnel is refreshed, in bytes.</td>
</tr>
<tr>
<td>NMsIPDynLifetimeExpire</td>
<td>116</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Tunnel lifetime. Indicates the negotiated time at which the tunnel expires, in UNIX format.</td>
</tr>
</tbody>
</table>
Table 49. NMsecIPDynTunnel structure (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMslIPDynLifetimeRefresh</td>
<td>120</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Tunnel lifetime refresh. Indicates the time at which the tunnel is refreshed, in UNIX format.</td>
</tr>
<tr>
<td>NMslIPDynVPNLifeExpire</td>
<td>124</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Tunnel VPN lifetime expire. If not 0, indicates the time at which the tunnel family ceases to be refreshed, in UNIX format.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This field retains its original value for a refreshed tunnel</td>
</tr>
<tr>
<td>NMslIPDynActMethod</td>
<td>128</td>
<td>1 byte</td>
<td>Binary</td>
<td>Tunnel activation method. One of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NMsec_DYNTUN_USER (1) User activation (from the command line).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NMsec_DYNTUN_REMOTE (2) Remote activation from IPSec peer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NMsec_DYNTUN_ONDEMAND (3) On-demand activation caused by IP traffic.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NMsec_DYNTUN.Takeover (5) SWSA activation as a result of a DVIPA takeover.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NMsec_DYNTUN.AUTOACT (6) Auto-activation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This field retains its original value for a refreshed tunnel</td>
</tr>
<tr>
<td>NMslIPDynRsvd2</td>
<td>129</td>
<td>24 bits</td>
<td>Binary</td>
<td>Reserved bits.</td>
</tr>
<tr>
<td>NMslIPDynRmtUDPPort</td>
<td>132</td>
<td>2 bytes</td>
<td>Binary</td>
<td>If the tunnel uses UDP-encapsulation mode, the IKE UDP port of the remote security endpoint; otherwise, 0.</td>
</tr>
<tr>
<td>NMslIPDynRsvd3</td>
<td>134</td>
<td>2 bytes</td>
<td>Binary</td>
<td>Reserved bits.</td>
</tr>
<tr>
<td>NMslIPDynSrcNATOA</td>
<td>136</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Source NAT-OA payload. NAT-OA payloads are exchanged only for certain UDP-encapsulated tunnels. During NAT traversal negotiations, the IKE peer sends the source IP address that it is aware of.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>If NAT traversal negotiation did not occur or if a peer did not send a source NAT-OA payload, the value of this field is 0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>Restriction:</strong> An IKE peer at a NAT traversal support level that is prior to RFC3947 is not required to send a source NAT-OA payload.</td>
</tr>
<tr>
<td>Field</td>
<td>Offset</td>
<td>Length</td>
<td>Format</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NMsecIPDynDstNATOA</td>
<td>140</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Destination NAT-OA payload. NAT-OA payloads are exchanged only for certain UDP-encapsulated tunnels. During NAT traversal negotiations, the IKE peer sends the destination IP address that it is aware of. If NAT traversal negotiation did not occur or if a peer did not send a destination NAT-OA payload, the value of this field is 0. <strong>Restriction</strong>: An IKE peer at a NAT traversal support level that is prior to RFC3947 will not send a destination NAT-OA payload.</td>
</tr>
<tr>
<td>NMsecIPDynProtocol</td>
<td>144</td>
<td>1 byte</td>
<td>Binary</td>
<td>Protocol for tunnel data. If the value is 0, the tunnel covers all protocols.</td>
</tr>
<tr>
<td>NMsecIPDynRsvd4</td>
<td>145</td>
<td>24 bits</td>
<td>Binary</td>
<td>Reserved bits.</td>
</tr>
<tr>
<td>NMsecIPDynSrcPort</td>
<td>148</td>
<td>2 bytes</td>
<td>Binary</td>
<td>Low end of source port range for tunnel data, or 0 if the tunnel is not limited to TCP or UDP.</td>
</tr>
<tr>
<td>NMsecIPDynDstPort</td>
<td>150</td>
<td>2 bytes</td>
<td>Binary</td>
<td>Low end of destination port range for tunnel data, or 0 if the tunnel is not limited to TCP or UDP.</td>
</tr>
<tr>
<td>NMsecIPDynSrcAddr4</td>
<td>152</td>
<td>4 bytes</td>
<td>Binary</td>
<td>• If the NMsecIPDynSrcIsSingle field is set, this field is the IPv4 or IPv6 source address for tunnel data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• If the NMsecIPDynSrcIsPrefix field is set, this field is the IPv4 or IPv6 source address base for tunnel data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• If the NMsecIPDynSrcIsRange field is set, this field is the low end of the IPv4 or IPv6 source address range for tunnel data</td>
</tr>
<tr>
<td>NMsecIPDynSrcAddr6</td>
<td>156</td>
<td>16 bytes</td>
<td>Binary</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• If the NMsecIPDynSrcIsPrefix field is set, this field is the length of the tunnel data source address prefix, in bits.</td>
</tr>
<tr>
<td>NMsecIPDynSrcAddrRange4</td>
<td>168</td>
<td>4 bytes</td>
<td>Binary</td>
<td>If the NMsecIPDynSrcIsRange field is set, this field is the high end of the IPv4 or IPv6 source address range for tunnel data.</td>
</tr>
<tr>
<td>NMsecIPDynSrcAddrRange6</td>
<td>168</td>
<td>16 bytes</td>
<td>Binary</td>
<td>If the NMsecIPDynSrcIsRange field is set, this field is the high end of the IPv4 or IPv6 source address range for tunnel data.</td>
</tr>
<tr>
<td>NMsecIPDynDstAddr4</td>
<td>184</td>
<td>4 bytes</td>
<td>Binary</td>
<td>• If the NMsecIPDynDstIsSingle field is set, this field is the IPv4 or IPv6 destination address for tunnel data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• If the NMsecIPDynDstIsPrefix field is set, this field is the IPv4 or IPv6 destination address base for tunnel data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• If the NMsecIPDynDstIsRange field is set, this field is the low end of the IPv4 or IPv6 destination address range for tunnel data</td>
</tr>
<tr>
<td>NMsecIPDynDstAddr6</td>
<td>184</td>
<td>16 bytes</td>
<td>Binary</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• If the NMsecIPDynDstIsPrefix field is set, this field is the length of the tunnel data destination address prefix, in bits.</td>
</tr>
<tr>
<td>NMsecIPDynDstAddrRange4</td>
<td>200</td>
<td>4 bytes</td>
<td>Binary</td>
<td>If the NMsecIPDynDstIsRange field is set, this field is the high end of the IPv4 or IPv6 destination address range for tunnel data.</td>
</tr>
<tr>
<td>NMsecIPDynDstAddrRange6</td>
<td>200</td>
<td>16 bytes</td>
<td>Binary</td>
<td>If the NMsecIPDynDstIsRange field is set, this field is the high end of the IPv4 or IPv6 destination address range for tunnel data.</td>
</tr>
<tr>
<td>NMsecIPDynSrcAddrPrefix</td>
<td>216</td>
<td>1 byte</td>
<td>Binary</td>
<td>If the NMsecIPDynSrcIsPrefix field is set, this field is the length of the tunnel data source address prefix, in bits.</td>
</tr>
<tr>
<td>NMsecIPDynDstAddrPrefix</td>
<td>217</td>
<td>1 byte</td>
<td>Binary</td>
<td>If the NMsecIPDynDstIsPrefix field is set, this field is the length of the tunnel data destination address prefix, in bits.</td>
</tr>
</tbody>
</table>
### Table 49. NMsecIPDynTunnel structure (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsIPDynMajorVer</td>
<td>218</td>
<td>1 byte</td>
<td>Binary</td>
<td>Major version of the IKE protocol that is in use. Only the low-order 4 bits are used.</td>
</tr>
<tr>
<td>NMsIPDynMinorVer</td>
<td>219</td>
<td>1 byte</td>
<td>Binary</td>
<td>Minor version of the IKE protocol that is in use. Only the low-order 4 bits are used.</td>
</tr>
<tr>
<td>NMsIPDynType</td>
<td>220</td>
<td>1 byte</td>
<td>Binary</td>
<td>Low end of the ICMP, ICMPv6, or MIPv6 type range for tunnel data, or 0 if the tunnel is not limited to ICMP, ICMPv6, or MIPv6.</td>
</tr>
<tr>
<td>NMsIPDynTypeRange</td>
<td>221</td>
<td>1 byte</td>
<td>Binary</td>
<td>High end of the ICMP, ICMPv6, or MIPv6 type range for tunnel data, or 0 if the tunnel is not limited to ICMP, ICMPv6, or MIPv6. A tunnel that applies to all type values is indicated as the range 0 - 255.</td>
</tr>
<tr>
<td>NMsIPDynCode</td>
<td>222</td>
<td>1 byte</td>
<td>Binary</td>
<td>Low end of ICMP or ICMPv6 code range for tunnel data, or 0 if the tunnel is not limited to ICMP or ICMPv6.</td>
</tr>
<tr>
<td>NMsIPDynCodeRange</td>
<td>223</td>
<td>1 byte</td>
<td>Binary</td>
<td>High end of ICMP or ICMPv6 code range for tunnel data, or 0 if the tunnel is not limited to ICMP or ICMPv6. A tunnel that applies to all code values is indicated as the range 0 - 255.</td>
</tr>
<tr>
<td>NMsIPDynSrcPortRange</td>
<td>224</td>
<td>2 bytes</td>
<td>Binary</td>
<td>High end of source port range for tunnel data, or 0 if the tunnel is not limited to TCP or UDP. A tunnel that applies to all source port values is indicated as the range 0 - 65535.</td>
</tr>
<tr>
<td>NMsIPDynDstPortRange</td>
<td>226</td>
<td>2 bytes</td>
<td>Binary</td>
<td>High end of destination port range for tunnel data, or 0 if the tunnel is not limited to TCP or UDP. A tunnel that applies to all destination port values is indicated as the range 0 - 65535.</td>
</tr>
<tr>
<td>NMsIPDynGeneration</td>
<td>228</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Tunnel generation number. The first dynamic tunnel that has a particular tunnel ID is generation 1. Subsequent refreshes of this dynamic tunnel have the same tunnel ID but have higher generation numbers.</td>
</tr>
</tbody>
</table>

One section, NMsecIPDynamicStack, describes the properties of a dynamic IP tunnel that are specific to the TCP/IP stack. This section contains the following data.

### Table 50. NMsecIPDynamicStack structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsIPDynStackShadow</td>
<td>0, bit 0</td>
<td>1 bit</td>
<td>Binary</td>
<td>SWSA shadow indicator. If set, the tunnel is an SWSA shadow tunnel originating from a distributing stack.</td>
</tr>
<tr>
<td>NMsIPDynStackClearDF</td>
<td>0, bit 1</td>
<td>1 bit</td>
<td>Binary</td>
<td>Don't-fragment bit clear indicator. If this bit is set, the IPv4 tunnel mode tunnel clears the DF bit in the outer IP header. If neither the NMsIPDynStackClearDF nor the NMsIPDynStackSetDF value is set, the IPv4 tunnel mode tunnel passes the DF bit from the inner IP header to the outer IP header. This field is not applicable and is always 0 for IPv6 or transport mode tunnels.</td>
</tr>
<tr>
<td>Field</td>
<td>Offset</td>
<td>Length</td>
<td>Format</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------</td>
<td>--------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NMsecIPDynStackSetDF</td>
<td>0, bit 2</td>
<td>1 bit</td>
<td>Binary</td>
<td>Don’t-fragment bit set indicator. If this bit is set, the IPv4 tunnel mode tunnel sets the DF bit in the outer IP header. If neither the NMsecIPDynStackClearDF or the NMsecIPDynStackSetDF value is set, the IPv4 tunnel mode tunnel passes the DF bit from the inner IP header to the outer IP header. This field is not applicable and is always 0 for IPv6 or transport mode tunnels.</td>
</tr>
<tr>
<td>NMsecIPDynStackClearDSCP</td>
<td>0, bit 3</td>
<td>1 bit</td>
<td>Binary</td>
<td>DSCP clear indicator. If this bit is set, the tunnel mode tunnel clears the DSCP bit in the outer IP header. If this bit has the value 0, the tunnel mode tunnel copies the DSCP field from the inner IP header to the outer IP header. This field is not applicable and is always 0 for transport mode tunnels.</td>
</tr>
<tr>
<td>NMsecIPDynStackRsvd1</td>
<td>0, bit 4</td>
<td>28 bits</td>
<td>Binary</td>
<td>Reserved bits.</td>
</tr>
<tr>
<td>NMsecIPDynStackLifesizeCur</td>
<td>4</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Current lifesize value. If the tunnel lifesize value has been negotiated, this represents the current value of the lifesize counter.</td>
</tr>
<tr>
<td>NMsecIPDynStackOutPkt</td>
<td>12</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Outbound packet count for this tunnel. For SWSA tunnels, this represents this tunnel’s outbound packet count only for this particular TCP/IP stack.</td>
</tr>
<tr>
<td>NMsecIPDynStackInPkt</td>
<td>20</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Inbound packet count for this tunnel. For SWSA tunnels, this represents this tunnel’s inbound packet count only for this particular TCP/IP stack.</td>
</tr>
<tr>
<td>NMsecIPDynStackOutBytes</td>
<td>28</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Outbound byte count for this tunnel, representing the number of outbound data bytes protected by the tunnel. For SWSA tunnels, this represents this tunnel’s outbound byte count only for this particular TCP/IP stack.</td>
</tr>
<tr>
<td>NMsecIPDynStackInBytes</td>
<td>36</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Inbound byte count for this tunnel, representing the number of inbound data bytes protected by the tunnel. For SWSA tunnels, this represents this tunnel’s inbound byte count only for this particular TCP/IP stack.</td>
</tr>
</tbody>
</table>

**NMsec_GET_IPTUNDYNENIKE**

<table>
<thead>
<tr>
<th>NMsecMessageHdr</th>
<th>NMsecRecordHdr</th>
<th>NMsecSecDesc (5)</th>
<th>NMsecIP Tunnel</th>
<th>NMsecIPDynTunnel</th>
<th>NMsecIPDynamicIKE</th>
</tr>
</thead>
</table>

Response records (1 for each tunnel)

**Figure 24. NMsec_GET_IPTUNDYNENIKE response format**

For the requested stack, zero or more records are returned representing dynamic IP tunnels known to the IKE daemon. Tunnels are presented in an unordered sequence, except that instances of a particular tunnel family (all sharing the same
tunnel ID) are ordered from most recently activated to least recently activated. Each record contains the following sections:

- One section, NMsecIP Tunnel, describes the basic properties of an IP tunnel. The layout of this section is described in [Table 47 on page 509](#).
- One section, NMsecIPDynTunnel, describes the basic properties of a dynamic IP tunnel. The layout of this section is described in [Table 49 on page 513](#).
- One section, NMsecIPDynamicIKE, describes the properties of a dynamic IP tunnel that are specific to the IKE daemon. This section contains the following data.

### Table 51. NMsecIPDynamicIKE structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsIPDynIKEsPendingNew</td>
<td>0, bit</td>
<td>1</td>
<td>Binary</td>
<td>Pending new activation indicator. If set, this dynamic IP tunnel is in pending state and it represents a new activation rather than a refresh. If not set, the tunnel is either not in pending state or is not a new activation.</td>
</tr>
<tr>
<td>NMsIPDynIKERevd1</td>
<td>0, bit</td>
<td>31 bits</td>
<td>Binary</td>
<td>Reserved bits.</td>
</tr>
<tr>
<td>NMsIPDynIKEFilter</td>
<td>4</td>
<td>48 bytes</td>
<td>EBCDIC</td>
<td>Filter name for the IP filter related to this dynamic tunnel.</td>
</tr>
<tr>
<td>NMsIPDynIKEDHGroup</td>
<td>52</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Diffie-Hellman group used for perfect forward secrecy (PFS) for this dynamic tunnel, or 0 if phase 2 PFS is not configured.</td>
</tr>
<tr>
<td>NMsIPDynIKEclIDType</td>
<td>56</td>
<td>1 byte</td>
<td>Binary</td>
<td>ISAKMP identity type for the local client ID, as defined in RFC 2407. Client identities can be exchanged during negotiation to limit or define the scope of data protected by the tunnel. If client identities are not exchanged, then the scope of data protection is defined to cover the peers' tunnel endpoint addresses. If client identities were not exchanged during negotiation, this field is 0.</td>
</tr>
<tr>
<td>NMsIPDynIKErmIDType</td>
<td>57</td>
<td>1 byte</td>
<td>Binary</td>
<td>ISAKMP identity type for the remote client ID, as defined in RFC 2407. Client identities might be exchanged during negotiation to limit or define the scope of data protected by the tunnel. If client identities are not exchanged, then the scope of data protection is defined to cover the peers' tunnel endpoint addresses. If client identities were not exchanged during negotiation, this field is 0.</td>
</tr>
</tbody>
</table>
Table 51. NMsecIPDynamicIKE structure (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsecIPDynIKEExtState</td>
<td>58</td>
<td>2 bytes</td>
<td>Binary</td>
<td>Extended tunnel state information. One of the following:</td>
</tr>
<tr>
<td>NMsec_P2STATE_INIT</td>
<td>(0)</td>
<td></td>
<td></td>
<td>No key exchange messages have been initiated.</td>
</tr>
<tr>
<td>NMsec_P2STATE_IN_KEP</td>
<td>(1)</td>
<td></td>
<td></td>
<td>Key exchange messages are being processed, but the full exchange has not completed.</td>
</tr>
<tr>
<td>NMsec_P2STATE_DONE</td>
<td>(2)</td>
<td></td>
<td></td>
<td>All key exchange messages have been completed and the tunnel is usable for traffic.</td>
</tr>
<tr>
<td>NMsec_P2STATE_PENDING_NOTIFY</td>
<td>(3)</td>
<td></td>
<td></td>
<td>Key exchange messages have been completed, but until a connection notification is received from the tunnel endpoint, the tunnel is not done.</td>
</tr>
<tr>
<td>NMsec_P2STATE_PENDING_START</td>
<td>(4)</td>
<td></td>
<td></td>
<td>Tunnel is awaiting the activation of an IKE tunnel to allow it to begin. See the description of the NMsecIPTunState field in Table 47 on page 509 for more succinct state information.</td>
</tr>
</tbody>
</table>

- One variable-length section containing the local client ID for this tunnel’s phase 2 negotiation. Regardless of the identity’s type, the ID is expressed as an EBCDIC string (an IP address is returned in printable form). The length of this section is 0 if no client IDs were exchanged.
- One variable-length section containing the remote client ID for this tunnel’s phase 2 negotiation. Regardless of the identity’s type, it is expressed as an EBCDIC string (an IP address is returned in printable form). The length of this section is 0 if no client IDs were exchanged.

**NMsec_GET_IKETUN**

**NMsec_GET_IKETUN response format**

Figure 25. NMsec_GET_IKETUN response format

For the requested stack, zero or more records are returned representing IKE security associations (IKE tunnels) used by IKE to negotiate IPSec security associations (dynamic tunnels) for the given TCP/IP stack. Tunnels are presented in an unordered sequence, except that instances of a particular tunnel family (all sharing the same tunnel ID) are ordered from the most recently activated to the least recently activated. Each record contains the following sections:

- One section, NMsecIKETunnel, describes attributes of the IKE security association. This section contains the following data.
### Table 52. NMsecIKE Tunnel structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsIKETunIPv6</td>
<td>0, bit 0</td>
<td>1 bit</td>
<td>Binary</td>
<td>IPv6 indicator. If set, the IKE tunnel security endpoints are IPv6 addresses, otherwise they are IPv4.</td>
</tr>
<tr>
<td>NMsIKETunNATAllowed</td>
<td>0, bit 1</td>
<td>1 bit</td>
<td>Binary</td>
<td>NAT traversal indicator. If set, the NAT traversal function is enabled for this IKE tunnel.</td>
</tr>
<tr>
<td>NMsIKETunLcNAT</td>
<td>0, bit 2</td>
<td>1 bit</td>
<td>Binary</td>
<td>Local NAT indicator. If set, a NAT has been detected in front of the local security endpoint.</td>
</tr>
<tr>
<td>NMsIKETunRmtNAT</td>
<td>0, bit 3</td>
<td>1 bit</td>
<td>Binary</td>
<td>Remote NAT indicator. If set, a NAT has been detected in front of the remote security endpoint.</td>
</tr>
<tr>
<td>NMsIKETunRmtNAPT</td>
<td>0, bit 4</td>
<td>1 bit</td>
<td>Binary</td>
<td>Remote NAPT indicator. If set, an NAPT has been detected in front of the remote security endpoint. It is possible that an NAPT might exist but that it is detected only as a NAT.</td>
</tr>
<tr>
<td>NMsIKETunCanInitP1</td>
<td>0, bit 5</td>
<td>1 bit</td>
<td>Binary</td>
<td>IKE tunnel (P1) initiation indicator. If set, the local security endpoint can initiate IKE tunnel negotiations with the remote security endpoint. Otherwise, the remote security endpoint must initiate IKE tunnel negotiations. Either side can initiate refreshes.</td>
</tr>
<tr>
<td>NMsIKETunRsvd1</td>
<td>0, bit 6</td>
<td>26 bits</td>
<td>Binary</td>
<td>Reserved bits.</td>
</tr>
<tr>
<td>NMsIKETunID</td>
<td>4</td>
<td>48 bytes</td>
<td>EBCDIC</td>
<td>Tunnel ID for this IKE tunnel.</td>
</tr>
<tr>
<td>NMsIKETunKeyExchRule</td>
<td>52</td>
<td>48 bytes</td>
<td>EBCDIC</td>
<td>Key exchange rule name for this IKE tunnel.</td>
</tr>
<tr>
<td>NMsIKETunKeyExchAction</td>
<td>100</td>
<td>48 bytes</td>
<td>EBCDIC</td>
<td>Key exchange action name for this IKE tunnel.</td>
</tr>
<tr>
<td>NMsIKETunLcEndpt4</td>
<td>148</td>
<td>4 bytes</td>
<td>Binary</td>
<td>IPv4 or IPv6 local security endpoint for this IKE tunnel.</td>
</tr>
<tr>
<td>NMsIKETunLcEndpt6</td>
<td>148</td>
<td>16 bytes</td>
<td>Binary</td>
<td>Tunnel exchange mode. One of the following:</td>
</tr>
<tr>
<td>NMsIKETunRmtEndpt4</td>
<td>164</td>
<td>4 bytes</td>
<td>Binary</td>
<td>IPv4 or IPv6 remote security endpoint for this IKE tunnel.</td>
</tr>
<tr>
<td>NMsIKETunRmtEndpt6</td>
<td>164</td>
<td>16 bytes</td>
<td>Binary</td>
<td>Tunnel state. One of the following:</td>
</tr>
<tr>
<td>NMsIKETunICookie</td>
<td>180</td>
<td>8 bytes</td>
<td>Binary</td>
<td>The icookie for this IKE tunnel.</td>
</tr>
<tr>
<td>NMsIKETunRCookie</td>
<td>188</td>
<td>8 bytes</td>
<td>Binary</td>
<td>The rcookie for this IKE tunnel.</td>
</tr>
<tr>
<td>NMsIKETunExchangeMode</td>
<td>196</td>
<td>1 byte</td>
<td>Binary</td>
<td>Tunnel state. One of the following:</td>
</tr>
<tr>
<td>NMsIKETunAuthAlg</td>
<td>198</td>
<td>1 byte</td>
<td>Binary</td>
<td>Tunnel authentication algorithm. One of the following:</td>
</tr>
<tr>
<td>NMsIKETunEncryptAlg</td>
<td>199</td>
<td>1 byte</td>
<td>Binary</td>
<td>Tunnel encryption algorithm. One of the following:</td>
</tr>
<tr>
<td>NMsIKETunDHGroup</td>
<td>200</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Diffie-Hellman group used to generate keying material for this IKE tunnel.</td>
</tr>
<tr>
<td>NMsIKETunPeerAuthMethod</td>
<td>204</td>
<td>1 byte</td>
<td>Binary</td>
<td>Tunnel peer authentication method. One of the following:</td>
</tr>
</tbody>
</table>
Table 52. NMsecIKETunnel structure (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsecIKETunRole</td>
<td>205</td>
<td>1 byte</td>
<td>Binary</td>
<td>Tunnel role. One of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- NMsec_IKETUN_INITIATOR (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- NMsec_IKETUN_RESPONDER (2)</td>
</tr>
<tr>
<td>NMsecIKETunNATTLevel</td>
<td>206</td>
<td>1 byte</td>
<td>Binary</td>
<td>NAT traversal support level. One of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- NMsec_IKETUN_NATTNONE (0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- No NAT traversal support, either not configured or not negotiated.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- NMsec_IKETUN_NATTRFCD2 (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- RFC 3947 draft 2 support.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- NMsec_IKETUN_NATTRFCD3 (3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- RFC 3947 draft 3 support.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- NMsec_IKETUN_NATTRFC (4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- RFC 3947 support with non-z/OS peer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- NMsec_IKETUN_NATTZOS (5)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- RFC 3947 support with z/OS peer.</td>
</tr>
<tr>
<td>NMsecIKETunExtState</td>
<td>207</td>
<td>1 byte</td>
<td>Binary</td>
<td>Extended tunnel state information. One of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- NMsec_P1STATE_INIT (0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- No key exchange messages have been initiated.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- NMsec_P1STATE_WAIT_SA (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- The first key exchange message has been sent and the endpoint is waiting</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- for a response.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- NMsec_P1STATE_IN_KE (2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- A key exchange response has been sent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- NMsec_P1STATE_WAIT_KE (3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- A key exchange message has been sent and the endpoint is waiting for a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- response.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- NMsec_P1STATE_DONE (4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- All key exchange messages have been completed and the tunnel is available</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- for data traffic.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- NMsec_P1STATE_EXPIRED (5)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- Tunnel has exceeded its lifetime or lifesize and is not available for</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- data traffic.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>See NMsecIKETunState for more succinct state information.</td>
</tr>
<tr>
<td>NMsecIKETunLifesize</td>
<td>208</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Tunnel lifesize. If not 0, indicates the negotiated lifesize limit for the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>tunnel, in bytes.</td>
</tr>
<tr>
<td>NMsecIKETunLifetime</td>
<td>216</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Negotiated tunnel lifetime. Indicates the total number of seconds the tunnel</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>remains active.</td>
</tr>
<tr>
<td>NMsecIKETunLifetimeRefresh</td>
<td>220</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Tunnel lifetime refresh. Indicates the time at which the tunnel is refreshed,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>in UNIX format.</td>
</tr>
<tr>
<td>NMsecIKETunLifetimeExpire</td>
<td>224</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Tunnel lifesize expiration. Indicates the time at which the tunnel expires,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>in UNIX format.</td>
</tr>
<tr>
<td>NMsecIKETunRmtUDPPort</td>
<td>228</td>
<td>2 bytes</td>
<td>Binary</td>
<td>Remote UDP port used for IKE negotiations.</td>
</tr>
<tr>
<td>NMsecIKETunLIDType</td>
<td>230</td>
<td>1 byte</td>
<td>Binary</td>
<td>ISAKMP identity type for the local security endpoint identity, as defined in</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RFC 2407.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ISAKMP peers exchange and verify each others’ identities as part of the IKE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>tunnel (phase 1) negotiation.</td>
</tr>
</tbody>
</table>
Table 52. NMsecIKETunnel structure (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsecIKETunRIDType</td>
<td>231</td>
<td>1 byte</td>
<td>Binary</td>
<td>ISAKMP identity type for the remote security endpoint identity, as defined in RFC 2407. ISAKMP peers exchange and verify each others' identities as part of the IKE tunnel (phase 1) negotiation.</td>
</tr>
<tr>
<td>NMsecIKETunStartTime</td>
<td>232</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Tunnel start time. Indicates the time at which the tunnel was activated or refreshed, in UNIX format.</td>
</tr>
<tr>
<td>NMsecIKETunMajorVer</td>
<td>236</td>
<td>1 byte</td>
<td>Binary</td>
<td>Major version of the IKE protocol that is in use. Only the low-order 4 bits are used.</td>
</tr>
<tr>
<td>NMsecIKETunMinorVer</td>
<td>237</td>
<td>1 byte</td>
<td>Binary</td>
<td>Minor version of the IKE protocol that is in use. Only the low-order 4 bits are used.</td>
</tr>
<tr>
<td>NMsecIKETunPseudoRandomFunc</td>
<td>238</td>
<td>1 byte</td>
<td>Binary</td>
<td>Pseudo-random function that is used to seed keying material. One of the following: NMsec_AUTH_HMAC_MD5 (38) NMsec_AUTH_HMAC_SHA1 (39)</td>
</tr>
<tr>
<td>NMsecIKETunLocalAuthMethod</td>
<td>239</td>
<td>1 byte</td>
<td>Binary</td>
<td>The authentication method for the local endpoint. One of the following: NMsec_IKETUN_PRESHAREDKEY (3) NMsec_IKETUN_RSASIGNATURE (2)</td>
</tr>
<tr>
<td>NMsecIKETunReauthInterval</td>
<td>240</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Re-authentication interval. Indicates the number of seconds between re-authentication operations.</td>
</tr>
<tr>
<td>NMsecIKETunReauthTime</td>
<td>244</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Tunnel re-authentication time. Indicates the time at which the tunnel is re-authenticated, in UNIX format.</td>
</tr>
<tr>
<td>NMsecIKETunGeneration</td>
<td>248</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Tunnel generation number. The first IKE tunnel that has a particular tunnel ID is generation 1. Subsequent refreshes of this IKE tunnel will have the same tunnel ID but will have higher generation numbers.</td>
</tr>
</tbody>
</table>

- One section, NMsecIKETunStats, indicates various counters and statistics for the IKE tunnel. This section contains the following data.

Table 53. IKE tunnel statistics

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsecIKETunP2Current</td>
<td>0</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Current count of active dynamic tunnels that are associated with this IKE tunnel.</td>
</tr>
<tr>
<td>NMsecIKETunP2InProgress</td>
<td>4</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Current count of pending or in-progress dynamic tunnels that are associated with this IKE tunnel.</td>
</tr>
<tr>
<td>NMsecIKETunP2LclActSuccess</td>
<td>8</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Cumulative count of successful dynamic tunnel activations that were initiated locally for this IKE tunnel.</td>
</tr>
<tr>
<td>NMsecIKETunP2RmtActSuccess</td>
<td>12</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Cumulative count of successful dynamic tunnel activations that were initiated remotely for this IKE tunnel.</td>
</tr>
<tr>
<td>NMsecIKETunP2LclActFailure</td>
<td>16</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Cumulative count of failed dynamic tunnel activations that were initiated locally for this IKE tunnel.</td>
</tr>
<tr>
<td>NMsecIKETunP2RmtActFailure</td>
<td>20</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Cumulative count of failed dynamic tunnel activations that were initiated remotely for this IKE tunnel.</td>
</tr>
<tr>
<td>NMsecIKETunBytes</td>
<td>24</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Cumulative number of bytes that were protected by this IKE tunnel.</td>
</tr>
</tbody>
</table>
Table 53. IKE tunnel statistics (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsecIKETunP1Rexmit</td>
<td>32</td>
<td>8</td>
<td>Binary</td>
<td>Cumulative number of retransmitted key exchange (phase 1) messages sent for</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>this tunnel over the life of the IKE daemon. This data is cumulative even</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>across TCP/IP restarts.</td>
</tr>
<tr>
<td>NMsecIKETunP1Replay</td>
<td>40</td>
<td>8</td>
<td>Binary</td>
<td>Cumulative number of replayed key exchange (phase 1) messages received for</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>this tunnel over the life of the IKE daemon. This data is cumulative even</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>across TCP/IP restarts.</td>
</tr>
<tr>
<td>NMsecIPIKEStatsP2Rexmit</td>
<td>48</td>
<td>8</td>
<td>Binary</td>
<td>Cumulative number of retransmitted QUICKMODE (phase 2) messages sent for</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>this tunnel over the life of the IKE daemon. This data is cumulative even</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>across TCP/IP restarts.</td>
</tr>
<tr>
<td>NMsecIPIKEStatsP2Replay</td>
<td>56</td>
<td>8</td>
<td>Binary</td>
<td>Cumulative number of replayed QUICKMODE (phase 2) messages received for this</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>tunnel over the life of the IKE daemon. This data is cumulative even across</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TCP/IP restarts.</td>
</tr>
</tbody>
</table>

- One variable-length section contains the contents of the local identity used to negotiate the IKE tunnel. Regardless of the identity’s type, it is expressed as an EBCDIC string (an IP address is returned in printable form).
- One variable-length section contains the contents of the remote identity used to negotiate the IKE tunnel. Regardless of the identity’s type, it is expressed as an EBCDIC string (an IP address is returned in printable form).

**NMsec_GET_IKETUNCASCADE**

**NMsec_GET_IKETUNCASCADE response format**

For the requested stack, zero or more records are returned representing IKE security associations (phase 1 tunnels) used by IKE to negotiate IPSec security associations (phase 2 tunnels) for the given TCP/IP stack. The results are similar to the NMsec_GET_IKETUN request, except that cascaded phase 2 tunnel information is also included in the response. Each phase 2 IP tunnel associated with the given phase 1 IKE tunnel is reported in the result record. Each record contains the following sections:

- One section, NMsecIKETunnel, describes attributes of the IKE SA. The layout of this section is described in "NMsec_GET_IKETUN" on page 520.
- One section, NMsecIKETunnelStats, describes various counters and statistics for the IKE tunnel. The layout of this section is described in Table 53 on page 523.
• One variable-length section contains the contents of the local identity used to negotiate the IKE tunnel.
• One variable-length section contains the contents of the remote identity used to negotiate the IKE tunnel.
• One or zero cascaded record containers with a set of dynamic IPSec tunnel records, identified by a single cascading record descriptor in the record header. The records in this section describe the basic tunnel properties of each IPSec security association associated with this IKE tunnel. The format of these cascaded records is described in “NMsec_GET_IPTUNDYNIKE” on page 518.

NMsec_GET_IPINTERFACES

NMsec_GET_IPINTERFACES response format

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsecInterfaceName</td>
<td>0</td>
<td>16</td>
<td>EBCDIC</td>
<td>Interface name</td>
</tr>
<tr>
<td>NMsecInterfaceAddr</td>
<td>16</td>
<td>16</td>
<td>Binary</td>
<td>IP address. If this is an IPv4 address, the last 4 bytes contains the address and the first 12 bytes contain zeroes.</td>
</tr>
<tr>
<td>NMsecInterfaceSecClass</td>
<td>32</td>
<td>1</td>
<td>Binary</td>
<td>Security class</td>
</tr>
<tr>
<td>NMsecIPv6</td>
<td>33, bit 1</td>
<td>1</td>
<td>Binary</td>
<td>IP addressing mode. If set to 1, the interface is using an IPv6 address; otherwise, it is using IPv4.</td>
</tr>
<tr>
<td>NMsecInterfaceActive</td>
<td>33, bit 2</td>
<td>1</td>
<td>Binary</td>
<td>State indicator. If set to 1, the interface is active.</td>
</tr>
<tr>
<td>NMsecInterfaceDVIPA</td>
<td>33, bit 3</td>
<td>1</td>
<td>Binary</td>
<td>DVIPA indicator. If set to 1, the interface is a DVIPA.</td>
</tr>
<tr>
<td>NMsecInterfaceRsvd1</td>
<td>33, bit 4</td>
<td>21</td>
<td>Binary</td>
<td>Reserved. Must be set to 0.</td>
</tr>
</tbody>
</table>

NMsec_GET_IKENSINFO

NMsec_GET_IKENSINFO response format

Each record returned describes the network security services (NSS) attributes for a given stack. Each record has a single section, NMsecNSS, that describes the NSS attributes of the stack. This section contains the following data described in Table 55 on page 526.
Table 55. NMsec_GET_IKENSINFO structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsIKENSClientName</td>
<td>0</td>
<td>24</td>
<td>bytes</td>
<td>The stack's NSS client name. The default format is <code>sysname_stackname</code>, where <code>sysname</code> is the client's MVS system name, and <code>stackname</code> is the TCP/IP job name of the stack that the client represents. However, clients can override this default form with any valid 24-character string. If the client is not configured to use an NSS server, this field contains blanks.</td>
</tr>
<tr>
<td>NMsIKENSStackName</td>
<td>24</td>
<td>8</td>
<td>bytes</td>
<td>TCP/IP job name of the stack that the client represents.</td>
</tr>
<tr>
<td>NMsIKENSServerSysName</td>
<td>40</td>
<td>8</td>
<td>bytes</td>
<td>MVS system name of the system on which the NSS server is running. If the client is not configured to use an NSS server, this field contains blanks.</td>
</tr>
<tr>
<td>NMsIKENSUserid</td>
<td>52</td>
<td>8</td>
<td>bytes</td>
<td>User ID under which the client is registered with server. If the client is not configured to use an NSS server, this field contains blanks.</td>
</tr>
<tr>
<td>NMsIKENSSysName</td>
<td>60</td>
<td>8</td>
<td>bytes</td>
<td>The IPv4 or IPv6 source address of the client connection to the server. If this is an IPv4 address, the destination address is the last 4 bytes of this field, with the first 12 bytes containing zeros.</td>
</tr>
<tr>
<td>NMsIKENSServerAddr</td>
<td>68</td>
<td>16</td>
<td>bytes</td>
<td>The IPv4 or IPv6 destination address of the client connection to the server. If this is an IPv4 address, the destination address is the last 4 bytes of this field, with the first 12 bytes containing zeros.</td>
</tr>
<tr>
<td>NMsIKENSClientPort</td>
<td>84</td>
<td>2</td>
<td>bytes</td>
<td>The TCP source port of the client connection to the server.</td>
</tr>
<tr>
<td>NMsIKENSServerPort</td>
<td>86</td>
<td>2</td>
<td>bytes</td>
<td>The TCP destination port of the client connection to the server.</td>
</tr>
<tr>
<td>NMsIKENSConnTime</td>
<td>88</td>
<td>4</td>
<td>bytes</td>
<td>UNIX-format timestamp indicating when the client connected to the server.</td>
</tr>
<tr>
<td>NMsIKENSLastMsgTime</td>
<td>92</td>
<td>4</td>
<td>bytes</td>
<td>UNIX-format timestamp indicating when the last message was sent to the server.</td>
</tr>
<tr>
<td>NMsIKENSNumFailedRqs</td>
<td>96</td>
<td>8</td>
<td>bytes</td>
<td>Number of failed requests to server.</td>
</tr>
<tr>
<td>NMsIKENSClientAPIVersion</td>
<td>104</td>
<td>1</td>
<td>bytes</td>
<td>The version of the NSS client API that the NSS client is using. The level of NSS support that is available in z/OS version V1R9 and later.</td>
</tr>
</tbody>
</table>
Table 55. NMsec_GET_IKENSINFO structure (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsecIKENSServerAPIVersion</td>
<td>105</td>
<td>1 byte</td>
<td>Binary</td>
<td>The version of the NSS client API that the connected NSS server supports.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• The value 0 indicates that this information is not currently available.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• NMsec_NSS_API_VERSION1 (1) - The level of NSS support that is available in z/OS version V1R9 and later.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• NMsec_NSS_API_VERSION2 (2) - The level of NSS support that is available in z/OS version V1R10 and later.</td>
</tr>
<tr>
<td>NMsecIKENSClientRsvd2</td>
<td>106</td>
<td>2 bytes</td>
<td>Binary</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**NMsec_LOAD_POLICY**

This control request does not contain response records. Rather, the return code and reason code fields in the response message header contain the final status of the request. Because this control request causes IKED to manipulate the file system, a z/OS UNIX System Services I/O error might occur, which causes the return code to contain the error number value EIO. In this case, the reason code contains the error number value of the error condition. For further error diagnosis see z/OS UNIX System Services Messages and Codes.

**NMsec_ACTIVATE_IPTUNMANUAL, NMsec_ACTIVATE_IPTUNDYN, NMsec_DEACTIVATE_IPTUNMANUAL, NMsec_DEACTIVATE_IPTUNDYN, NMsec_REFRESH_IPTUNDYN, NMsec_REFRESH_IKETUN**

One record is returned that indicates the response of the tunnel action request. A successful response from a tunnel control request indicates that the requested operation has been successfully initiated. Because IPSec tunnel activation, deactivation, and refresh requires an exchange of messages between IPSec peers, the final status of the operation can be determined later through a subsequent NMI request that returns the filter or tunnel data.

### Tunnel Control response format

<table>
<thead>
<tr>
<th>NMsecMessageHdr</th>
<th>NMsecRecordHdr</th>
<th>NMsecSecDesc (2)</th>
<th>NMsecCntlResponse</th>
<th>NMsecTunnel (0-n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response records (exactly 1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 29. Tunnel control response format**

For a request to activate or deactivate all tunnels of a given type, the response record contains the NMsecTunCntlResponse section, which is described in Table 56. For requests that indicate a specific tunnel, this section does not exist.

Table 56. NMsecTunCntlResponse structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsecTunCRCount</td>
<td>0</td>
<td>4 bytes</td>
<td>Binary</td>
<td>The number of tunnels processed. When the request is for all tunnels, this value is the number of the tunnels found.</td>
</tr>
</tbody>
</table>

When specific tunnels are requested, the response record contains one NMsecTunnel section for each of the tunnels. The sections are returned in the same order as specified in the request. For requests that operate on all tunnels of a given type, this section does not exist.
IPSec NMI initialization and termination messages

When a client successfully connects to the interface, the server sends an initialization message (a message with type NMsec_INIT) to the client. This message contains no records, but the return code and reason code are 0 to indicate successful connection completion. When the server closes the connection (this might be the result of error, IKED termination, or the client’s closing of the socket), the server attempts to send a termination message (a message with type NMsec_TERM) to the client. This message contains no records, but the return code and reason code indicate the cause of the connection’s termination.

IPSec NMI return and reason codes

When sending a request, the client application should set the message header fields NMsMRC (return code) and NMsMRSn (reason code) to 0. Upon return, the server sets these fields as follows to indicate the status of the request. This service uses the errno values defined by z/OS UNIX System Services.

<table>
<thead>
<tr>
<th>Return code (NMsMRC)</th>
<th>Reason code (NMsMRSn)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>No error</td>
</tr>
</tbody>
</table>
| ENXIO                | 0                     | Requested TCP/IP stack does not exist or is not active.  
                     |                       | System Action: Request is failed but connection remains open.  
                     |                       | Response: Send requests only for active TCP/IP stacks. |
| EOPNOTSUPP           | 0                     | Requested TCP/IP stack is not configured for IP security.  
                     |                       | System Action: Request is failed but connection remains open.  
                     |                       | Response: Send requests only for TCP/IP stacks configured for IP security. |
| EINVAL               | NMsRsnBadIdent (1)    | Invalid message or record identifier supplied in message.  
                     |                       | System Action: Connection is closed.  
                     |                       | Response: Reissue the connection and send a correctly formatted message. |
| EINVAL               | NMsRsnBadVersion (2)  | Invalid version supplied in message header.  
                     |                       | System Action: Request is failed but connection remains open.  
                     |                       | Response: Send a correctly formatted message. |
| EINVAL               | NMsRsnBadType (3)     | Unsupported or unknown message type supplied in message header.  
                     |                       | System Action: Request is failed but connection remains open.  
<pre><code>                 |                       | Response: Send a supported message type. |
</code></pre>
<table>
<thead>
<tr>
<th>Return code (NMsMRc)</th>
<th>Reason code (NMsMRsn)</th>
<th>Description</th>
</tr>
</thead>
</table>
| EINVAL              | NMsRsnExcessiveSize (4) | Excessive message size.  
  System Action: Connection is closed.  
  Response: Reissue the connection and send a correctly formatted message. |
| EINVAL              | NMsRsnHdrSize (5)     | Message header size is not valid.  
  System Action: Request is failed but connection remains open.  
  Response: Send a message with the header size field set to the correct value. |
| EINVAL              | NMsRsnMsgSize (6)     | Message size is not valid. For example, the message might be too short, or the message size might be greater than the sum of its parts.  
  System Action: Connection is closed.  
  Response: Send a correctly formatted message. |
| EINVAL              | NMsRsnReservedNonzero (7) | Reserved data in message header, record header, or record data is a nonzero value. Reserved fields must be set to 0 for compatibility with future versions of the interface.  
  System Action: Request is failed but connection remains open.  
  Response: Send a message with reserved fields set to 0. |
| EINVAL              | NMsRsnRecordLength (8) | Unrecognized record length supplied in message. Length does not correspond to known record data.  
  System Action: Request is failed but connection remains open.  
  Response: Send a message with input filters of the correct length. |
| EINVAL              | NMsRsnRecordCount (9) | Unsupported record count supplied in message. NMI requests currently support a maximum of twenty input filters.  
  System Action: Request is failed but connection remains open.  
  Response: Send a message with the correct number of input filters. |
| EINVAL              | NMsRsnSectionLength (10) | Unrecognized section length supplied in record. Length does not correspond to known section data.  
  System Action: Request is failed but connection remains open.  
  Response: Send a message with correct input filters. |
| EINVAL              | NMsRsnSectionCount (11) | Unrecognized section count supplied in record. NMI requests currently allow one section in an input filter record.  
  System Action: Request is failed but connection remains open.  
  Response: Send a message with correct input filters. |
<table>
<thead>
<tr>
<th>Return code (NMsMr)</th>
<th>Reason code (NMsMrSn)</th>
<th>Description</th>
</tr>
</thead>
</table>
| EINVAL              | NMsRsnFilterSpec (12) | The input filter specification indicates a combination of filter values that is unsupported for the message's request type.  
System Action: Request is failed but connection remains open.  
Response: Send a message with a valid input filter specification for the message type. |
| EINVAL              | NMsRsnFilterValue (13) | The input filter specification contains a value that is out of range.  
System Action: Request is failed but connection remains open.  
Response: Send a message with correct input filter values. |
| EINVAL              | NMsRsnManTypeConflict (14) | Manual tunnel activation and deactivation requests for multiple tunnels must contain uniform tunnel specifications: either tunnel IDs or tunnel names. The request contained a mixture of tunnel names and tunnel IDs.  
System Action: Request is failed but connection remains open.  
Response: Separate manual tunnel names and tunnel IDs into different requests. |
| EINVAL              | NMsRsnPolicySource (15) | The policy source value in the policy load request is not valid.  
System Action: No action is required.  
Response: Send a message with a valid NMsecPolSrcSource value. |
| EACCES              | 0 | Access denied to the requested resource.  
System Action: Request is failed but connection remains open.  
Administrator Response: Permit user to security resource. |
| ENOMEM              | 0 | Insufficient storage available in the server to process the request.  
System Action: Request is failed but connection remains open.  
Response: Increase the REGION size for the IKE daemon, or send a message with a narrower set of input filters to limit the response. |
| ENOMEM              | NMsRsnTooManyConns (1) | The NMI thread is already using its maximum number of 50 connections and cannot accept any more.  
System Action: Connection is not opened and the request is failed.  
Response: Try the request again later. |
<table>
<thead>
<tr>
<th>Return code (NMsMrC)</th>
<th>Reason code (NMsMRsn)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EIO</td>
<td>(z/OS UNIX System Services Erno)</td>
<td>A file system I/O error occurred. The reason code contains the value of the errno that describes the error. System Action: Request is failed but the connection remains open. Response: Diagnose the z/OS UNIX System Services Erno using <a href="#">z/OS UNIX System Services Messages and Codes</a>.</td>
</tr>
</tbody>
</table>

**Network security services (NSS) network management NMI**

*z/OS Communications Server network security services (NSS) server provides an AF_UNIX socket interface through which network management applications can manage IP filtering and IPSec on remote NSS IPSec clients, or monitor NSS clients that are connected to the local NSS server. This interface is available only through the NSS server and should be used by network management applications that monitor and control multiple systems through a single point of control. Applications can perform the following functions using this interface:

- Issue monitoring or control requests through the NSS server to specified NSS IPSec clients. The NSS server routes all monitoring and control requests (described in “Local IPSec NMI” on page 476) to NSS IPSec clients, with the exception of the NMsec_GET_STACKINFO and NMsec_GET_IKENSINFO requests. Routing occurs only if the NSS IPSec client is connected to the NSS server at the time the request is made.
- Request information about one or all of the NSS clients that are currently connected to the NSS server, either for a specified discipline or for all disciplines.

A client network management application requests information and initiates control operations by sending specific requests over an AF_UNIX stream socket connection to the NSS server. If necessary, the request is then redirected to the specified NSS IPSec client, which later responds with the requested data or the results of the requested operation. The response information is then returned to the application directly over the AF_UNIX connection. For most control requests, a successful response indicates that the operation was successfully initiated, but that it is still in progress. You can determine the final status of the control operation later by issuing a subsequent monitoring request for the effected object.

**Network security services NMI: Configuring the interface**

Access to the network security services (NSS) server’s network management interface is controlled through RACF (or an equivalent external security manager product) resource definitions in the SERVAUTH class. Most of these resource names contain the NSS client’s name. The client name is defined by the client.

- For an NSS IPSec client, the default value of a client name takes the form `sysname_stackname`, where the `sysname` value is the MVS system name of the client, and the `stackname` value is the TCP/IP stack name that it represents. You can override the `clientname` value in the client’s IKE daemon configuration file on the NssStackConfig statement or in the IBM Configuration Assistant NSS Perspective on the Client IPSec Settings tab.
For an NSS XMLAppliance client, the default value of a client name is left up to the client application’s implementation.

**Tip:** When you override the `clientname` value for an NSS IPSec client, ensure that the name you define does not match the name of an existing NSS client on the NSS server system. If the names match, users with authority to manage IP security on that system also gain authority to remotely manage the NSS client, because the SERVAUTH resource names are identical.

The z/OS system administrator can restrict access to NSS network management interfaces as follows:

- Access to remote NSS IPSec client monitoring functions (those that request information only from an NSS IPSec client through the NSS server) within this interface can be restricted by defining a RACF (or equivalent external security manager product) resource `EZB.NETMGMT.sysname.clientname.IPSEC.DISPLAY` in the SERVAUTH class (where `sysname` represents the MVS system name where the interface is being invoked, and `clientname` is the name of the NSS IPSec client).

- Access to the remote NSS IPSec client control functions (those that take some action) is controlled through the `EZB.NETMGMT.sysname.clientname.IPSEC.CONTROL` resource (where `sysname` represents the MVS system name where the interface is being invoked, and `clientname` is the name of the NSS IPSec client).

- Access to NSS server monitoring functions (those that request information only about the server itself) is controlled through the resource `EZB.NETMGMT.sysname.sysnameNSS.DISPLAY` in the SERVAUTH class (where the `sysname` value represents the MVS system name where the interface is being invoked).

**Requirement:** For applications that use the interface, the MVS user ID must be permitted to the defined resource. Additionally, permitted client applications must have permission to enter the `/var/sock` directory and to write to the `/var/sock/nss` socket. Ensure that the NSSD OMVS user ID has write access to the `/var/sock/directory (or ensure that it has permission to create this directory).

**Guideline:** If you are developing a feature for a product to be used by other parties, include instructions in your documentation indicating that administrators must define and give appropriate permission to the given security resource to use that feature.

**Network security services NMI: Connecting to the server**

For an application to use this interface, it must connect to the AF_UNIX stream socket that is provided by the NSS server for this interface. The socket path name is `/var/sock/nss`. Either the Language Environment C/C++ API or the UNIX System Services BPX services can be used to create AF_UNIX sockets and connect to this service.

When an application connects to the socket, the server sends an initialization message to the client application. When the NSS server closes a client connection (reasons for doing so include severe errors in the format of data requests sent by the application to the server, or NSS server termination), the NSS server attempts to send a termination message to the client before closing the connection. Both the initialization and termination messages match those used by the IKE daemon (see "IPSec NMI request/response format" on page 478).
Network security services NMI request and response format

The NSS server supports a message format that is almost identical to that used by the IKE daemon for local IPSec monitoring and control (see “Local IPSec NMI” on page 476). Like the local monitoring/control interface, these messages are exchanged over an AF_UNIX socket using a request-response model.

The only difference between the NSS and IPSec NMI message format is that when an NMI message is sent to the NSS server, the NMzMTarget string in the message header identifies the remote NSS client to which the request is directed. Use the clientname field of the target NSS client in the NMzMTarget string, padded on the right with blanks. You can obtain the clientname values of each client connected to the NSS server by issuing the NMsec_GET_CLIENTINFO request. The NMzMTarget field can be set to blanks for an NMsec_GET_CLIENTINFO request. If this field is set to blanks for any other request, the request is rejected with an appropriate error code in the reply header.

Restriction: An NMI request is redirected to an NSS client only if that client has selected the remote management service and is enabled for that service.

Network security services NMI request messages

The NSS server supports all of the request messages described for the IKE daemon except for the NMsec_GET_STACKINFO and NMsec_GET_IKENSINFO requests (see “Local IPSec NMI” on page 476). In addition, the NSS server also supports the NMsec_GET_CLIENTINFO request message.

The NMsec_GET_CLIENTINFO request is a monitoring request that obtains a list of NSS clients that are currently connected to the NSS server as well as summary information about each client. This request allows zero or more input filtering records that specify client discipline type. If the NMzMTarget field in the message header is blank, then information for all of the currently connected clients is returned. If a client name is specified in the NMzMTarget field, then information for only that client is returned as long as the client is connected. If the specified client is not connected, the request fails with an ENXIO return code. Access to this function is controlled through the EZB.NETMGMT.sysname.sysname.NSS.DISPLAY resource definition in the SERVAUTH class.

Network security services NMI response messages

The NSS server supports all of the response messages described for the IKE daemon in “Local IPSec NMI” on page 476 except for the NMsec_GET_STACKINFO and NMsec_GET_IKENSINFO responses. In addition, the NSS server also supports the NMsec_GET_CLIENTINFO response message:

NMsec_GET_CLIENTINFO

NMsec_GET_CLIENTINFO response format

<table>
<thead>
<tr>
<th>NMsecMessageHdr</th>
<th>NMsecRecordHdr</th>
<th>NMsecSecDesc</th>
<th>NMsecNSClient</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response records (1 for each connected client)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 30. NMsec_GET_CLIENTINFO response format

Each record returned identifies and describes a single NSS client. Each record has a single section, NMsecNSClient, which contains the following data.
<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsNSClientName</td>
<td>0</td>
<td>24</td>
<td>EBCDIC</td>
<td>The name of the NSS client. For z/OS clients, the default format is sysname_stackname, where sysname is the z/OS system name, and stackname is the TCP/IP job name. z/OS clients can override this default form with any valid 24-character string. For non z/OS clients, the format of the client name is determined by the client.</td>
</tr>
<tr>
<td>NMsNSClientSysName</td>
<td>24</td>
<td>8</td>
<td>EBCDIC</td>
<td>This field contains the NSS client’s system name. Each NSS client provides this value to the NSS server at connection time. For z/OS NSS clients, this field contains the client’s z/OS system name. For other clients, this field can contain a system identifier such as the first 8 bytes of their system’s host name.</td>
</tr>
<tr>
<td>NMsNSClientStackName</td>
<td>32</td>
<td>8</td>
<td>EBCDIC</td>
<td>This field contains the NSS client’s stack name. Each NSS client provides this value to the NSS server at connection time.</td>
</tr>
<tr>
<td>NMsNSClientUserid</td>
<td>40</td>
<td>8</td>
<td>EBCDIC</td>
<td>The user ID under which the client is registered.</td>
</tr>
<tr>
<td>NMsNSClientIpAddress</td>
<td>48</td>
<td>16</td>
<td>Binary</td>
<td>The IPv4 or IPv6 address from which the client is connected. If this is an IPv4 address, it resides in the last 4 bytes of this field, with the first 12 containing zeroes</td>
</tr>
<tr>
<td>NMsNSClientServerAddress</td>
<td>64</td>
<td>16</td>
<td>Binary</td>
<td>The IPv4 or IPv6 address on which the NSS server is communicating with the client. If this is an IPv4 address, it resides in the last 4 bytes of this field, with the first 12 containing zeroes.</td>
</tr>
<tr>
<td>NMsNSClientClientPort</td>
<td>80</td>
<td>2</td>
<td>Binary</td>
<td>The client’s local TCP port over which the client is communicating with the server.</td>
</tr>
<tr>
<td>NMsNSClientServerPort</td>
<td>82</td>
<td>2</td>
<td>Binary</td>
<td>The TCP port over which the NSS server is communicating with the NSS client.</td>
</tr>
<tr>
<td>NMsNSClientSvcSelCert</td>
<td>84, bit 0</td>
<td>1</td>
<td>Binary</td>
<td>Used in conjunction with the NMsNSClientDiscipline field to indicate which type of certificate service, if any, is selected. If the NMsNSClientDiscipline field is set to NMsec_DISCIPLINE_IPSEC, this field indicates whether the IPSec certificate service is selected. If the NMsNSClientDiscipline field is set to NMsec_DISCIPLINE_XMLAPP, this field indicates whether the XMLAppliance certificate service is selected.</td>
</tr>
<tr>
<td>NMsNSClientSvcEnblCert</td>
<td>84, bit 1</td>
<td>1</td>
<td>Binary</td>
<td>Used in conjunction with the NMsNSClientDiscipline field to indicate which type of certificate service, if any, is enabled. If the NMsNSClientDiscipline field is set to NMsec_DISCIPLINE_IPSEC, this field indicates whether the IPSec certificate service is enabled. When the NMsNSClientDiscipline field is set to NMsec_DISCIPLINE_XMLAPP, this field indicates whether the XMLAppliance certificate service is enabled.</td>
</tr>
<tr>
<td>NMsNSClientSvcSelMgmt</td>
<td>84, bit 2</td>
<td>1</td>
<td>Binary</td>
<td>The IPSec network management service is selected.</td>
</tr>
<tr>
<td>NMsNSClientSvcEnblMgmt</td>
<td>84, bit 3</td>
<td>1</td>
<td>Binary</td>
<td>The IPSec network management service is enabled.</td>
</tr>
<tr>
<td>NMsNSClientIPv6</td>
<td>84, bit 4</td>
<td>1</td>
<td>Binary</td>
<td>IP addressing mode. If set to 1, the client and server are using IPv6 addresses; otherwise, they are using IPv4.</td>
</tr>
<tr>
<td>NMsNSClientSvcSelPrivKey</td>
<td>84, bit 5</td>
<td>1</td>
<td>Binary</td>
<td>XMLAppliance private key service is selected.</td>
</tr>
<tr>
<td>NMsNSClientSvcEnblPrivKey</td>
<td>84, bit 6</td>
<td>1</td>
<td>Binary</td>
<td>XMLAppliance private key service is enabled.</td>
</tr>
<tr>
<td>NMsNSClientSvcSelSAF</td>
<td>84, bit 7</td>
<td>1</td>
<td>Binary</td>
<td>XMLAppliance SAF access service is selected.</td>
</tr>
<tr>
<td>NMsNSClientSvcEnblSAF</td>
<td>84, bit 8</td>
<td>1</td>
<td>Binary</td>
<td>XMLAppliance SAF access service is enabled.</td>
</tr>
<tr>
<td>NMsNSClientRsvd1</td>
<td>84, bit 9</td>
<td>7</td>
<td>Binary</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
Table 58. NMsecNSClient structure (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsecNSClientConnectState</td>
<td>86</td>
<td>2 bytes</td>
<td>Binary</td>
<td>Client connection state. One of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>NMsec_CLIENTSTATUS_CONNECTPENDING (0x0001)</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Connection is pending. The initial TCP connection has been completed, but the ConnectClientReqToSrv request has not yet been</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>successfully processed. The client's name and access rights cannot be determined until after the ConnectClientReqToSrv request has been</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>processed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>NMsec_CLIENTSTATUS_CONNECTED (0x0002)</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The client is connected to the server.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>NMsec_CLIENTSTATUS_DISCONNECTPENDING (0x0003)</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Disconnect is pending. The client is still in the NSS server's tables, but no more requests from that client are processed. The server</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>is in the process of cleaning up after the client and removing the data from the server tables. The application enters this state under</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>one of the following conditions:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• The client’s user ID authorization fails during the processing of a ConnectClientReqToSrv request.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• A DisconnectClientReqToSrv request is received from the client.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• The TCP/IP connection to the client was terminated or was lost.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>NMsec_CLIENTSTATUS_UPDATEPENDING (0x0004)</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Update is pending. The client authorization information or selected services have been reconfigured at the client but the UpdateClientInfoReqToSrv request has not completed processing.</td>
</tr>
<tr>
<td>NMsecNSClientConnTime</td>
<td>88</td>
<td>4 bytes</td>
<td>Binary</td>
<td>UNIX-format timestamp indicating when the client connected to the server.</td>
</tr>
<tr>
<td>NMsecNSClientLastMsgTime</td>
<td>92</td>
<td>4 bytes</td>
<td>Binary</td>
<td>UNIX-format timestamp indicating when the last message was received at the server from the client.</td>
</tr>
<tr>
<td>NMsecNSClientNumreqSigCreate</td>
<td>96</td>
<td>8 bytes</td>
<td>Binary</td>
<td>For NSS IPSec clients, the number of create signature requests that have been received from the client. For NSS XMLAppliance clients, this number is 0.</td>
</tr>
<tr>
<td>NMsecNSClientNumreqSigVerify</td>
<td>104</td>
<td>8 bytes</td>
<td>Binary</td>
<td>For NSS IPSec clients, the number of verify signature requests that have been received from the client. For NSS XMLAppliance clients, this number is 0.</td>
</tr>
<tr>
<td>NMsecNSClientNumreqCACacheRefresh</td>
<td>112</td>
<td>8 bytes</td>
<td>Binary</td>
<td>For NSS IPSec clients, the number of CA cache refreshes that have been requested by the client. For NSS XMLAppliance clients, this number is 0.</td>
</tr>
<tr>
<td>NMsecNSClientNumNMIForward</td>
<td>120</td>
<td>8 bytes</td>
<td>Binary</td>
<td>For NSS IPSec clients, the number of NMI requests that have been forwarded to the client by the server. For NSS XMLAppliance clients, this number is 0.</td>
</tr>
<tr>
<td>NMsecNSClientNumreqCertSvc</td>
<td>128</td>
<td>8 bytes</td>
<td>Binary</td>
<td>The number of XMLAppliance certificate service requests made by the client.</td>
</tr>
<tr>
<td>NMsecNSClientNumreqPrivKeySvc</td>
<td>136</td>
<td>8 bytes</td>
<td>Binary</td>
<td>The number of XMLAppliance private key service requests made by the client.</td>
</tr>
<tr>
<td>NMsecNSClientNumreqSAFSvc</td>
<td>144</td>
<td>8 bytes</td>
<td>Binary</td>
<td>The number of XMLAppliance SAF access service requests made by the client.</td>
</tr>
</tbody>
</table>
Table 58. NMsecNSClient structure (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMsNSClientAPIVersion</td>
<td>152</td>
<td>1 byte</td>
<td>Binary</td>
<td>The version of the NSS client API that the NSS client is using.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- NMsec_NSS_API_VERSION1 (0x01) - The level of NSS support that is available in z/OS version V1R9 and later.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- NMsec_NSS_API_VERSION2 (0x02) - The level of NSS support that is available in z/OS version V1R10 and later.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- NMsec_NSS_API_VERSION3 (0x03) - The level of NSS support that is available in z/OS version V1R11 and later.</td>
</tr>
<tr>
<td>NMsNSClientDiscipline</td>
<td>153</td>
<td>1 byte</td>
<td>Binary</td>
<td>NSS discipline:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- NMsec_DISCIPLINE_IPSEC (0x01) - Indicates an NSS IPSec client.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- NMsec_DISCIPLINE_XMLAPP (0x02) - Indicates an NSS XMLAppliance client.</td>
</tr>
<tr>
<td>NMsNSClientRsvd2</td>
<td>154</td>
<td>2 bytes</td>
<td>Binary</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Network security services NMI initialization and termination messages**

When a network management application successfully connects to the interface, the server sends an initialization message (a message with type NMsec_INIT) to the client. This message contains no records, but the return code and reason code are 0 to indicate successful connection completion.

When the server closes the connection (the connection can close as a result of an error, NSSD terminating, or the application’s closing of the socket), the server attempts to send a termination message (a message with type NMsec_TERM) to the client. This message contains no records, but the return code and reason code indicate the cause of the connection’s termination.

**Network security services NMI return and reason codes**

When sending a request, the client application should set the message header fields NMsMRc (return code) and NMsMRsn (reason code) to 0. Upon return, the server sets these fields as follows to indicate the status of the request.

Table 59. Request return and reason codes

<table>
<thead>
<tr>
<th>Return code (NMsMRc)</th>
<th>Reason code (NMsMRsn)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>No error.</td>
</tr>
<tr>
<td>ENXIO (138)</td>
<td>0</td>
<td>The requested NSS client is not connected.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>System Action: Request is failed but the connection remains open.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Response: Send requests for active NSS clients only.</td>
</tr>
<tr>
<td>Return code (NMzMrc)</td>
<td>Reason code (NMzMrsn)</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>EOPNOTSUPP (1112)</td>
<td>0</td>
<td>The requested NSS client is not enabled for remote monitoring. System Action: Request is failed but connection remains open. Response: Send requests only for NSS clients that are enabled for remote monitoring. Otherwise, configure and permit the given NSS client for remote monitoring.</td>
</tr>
<tr>
<td>EINVAL (121)</td>
<td>NMsRsnBadIdent (1)</td>
<td>Invalid message or record identifier supplied in message. System Action: Connection is closed. Response: Reissue the connection and send a correctly formatted message.</td>
</tr>
<tr>
<td>EINVAL (121)</td>
<td>NMsRsnBadVersion (2)</td>
<td>Invalid version supplied in message header. System Action: Request is failed but connection remains open. Response: Send a correctly formatted message.</td>
</tr>
<tr>
<td>EINVAL (121)</td>
<td>NMsRsnBadType (3)</td>
<td>Unsupported or unknown message type supplied in message header. System Action: Request is failed but connection remains open. Response: Send a supported message type.</td>
</tr>
<tr>
<td>EINVAL (121)</td>
<td>NMsRsnExcessiveSize (4)</td>
<td>Excessive message size. System Action: Connection is closed. Response: Reissue the connection and send a correctly formatted message.</td>
</tr>
<tr>
<td>EINVAL (121)</td>
<td>NMsRsnHdrSize (5)</td>
<td>Message header size is invalid. System Action: Request is failed but connection remains open. Response: Send a message with the header size field set to the correct value.</td>
</tr>
<tr>
<td>EINVAL (121)</td>
<td>NMsRsnMsgSize (6)</td>
<td>Message size is invalid. For example, the message might be too short, or the message size might be greater than the sum of its parts. System Action: Request is failed but connection remains open. Response: Send a correctly formatted message.</td>
</tr>
<tr>
<td>EINVAL (121)</td>
<td>NMsRsnReservedNonzero (7)</td>
<td>Reserved data in message header, record header, or record data specifies a nonzero value. Reserved fields must be set to 0 for compatibility with future versions of the interface. System Action: Request is failed but connection remains open. Response: Send a message with reserved fields set to 0.</td>
</tr>
<tr>
<td>Return code (NMsMrc)</td>
<td>Reason code (NMsMRsn)</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| EINVAL (121) | NMsRsnRecordLength (8) | Unrecognized record length supplied in message. Length does not correspond to known record data.  
System Action: Request is failed but connection remains open.  
Response: Send a message with input filters of the correct length. |
| EINVAL (121) | NMsRsnRecordCount (9) | Unsupported record count supplied in message. NMI requests currently support a maximum of 20 input filters.  
System Action: Request is failed but connection remains open.  
Response: Send a message with the correct number of input filters. |
| EINVAL (121) | NMsRsnSectionLength (10) | Unrecognized section length supplied in record. Length does not correspond to known section data.  
System Action: Request is failed but connection remains open.  
Response: Send a message with correct input filters. |
| EINVAL (121) | NMsRsnSectionCount (11) | Unrecognized section count supplied in record. NMI requests currently allow one section in an input filter record.  
System Action: Request is failed but connection remains open.  
Response: Send a message with correct input filters. |
| EINVAL (121) | NMsRsnFilterSpec (12) | The input filter specification indicates a combination of filter values that is unsupported for the message’s request type.  
System Action: Request is failed but connection remains open.  
Response: Send a message with a valid input filter specification for the message type. |
| EINVAL (121) | NMsRsnFilterValue (13) | The input filter specification contains a value that is out of range.  
System Action: Request is failed but connection remains open.  
Response: Send a message with correct input filter values. |
<table>
<thead>
<tr>
<th>Return code (NMsMrC)</th>
<th>Reason code (NMsMrSn)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EINVAL (121)</td>
<td>NMsMrManTypeConflicts(14)</td>
<td>Manual tunnel activation and deactivation requests for multiple tunnels must contain uniform tunnel specifications: either tunnel IDs or tunnel names. The request contained a mixture of tunnel names and tunnel IDs. System Action: Request is failed but connection remains open. Response: Separate manual tunnel names and tunnel IDs into different requests.</td>
</tr>
<tr>
<td>EINVAL (121)</td>
<td>NMsMrPolicySource (15)</td>
<td>The policy source value in the policy load request is invalid. System Action: NO action is required. Response: Send a message with a valid NMsecPolSrcSource value.</td>
</tr>
<tr>
<td>EACCESS (111)</td>
<td>0</td>
<td>Access denied to the requested resource. System Action: Request is failed but connection remains open. Administrator Response: Permit the user to the security resource.</td>
</tr>
<tr>
<td>EACCESS (111)</td>
<td>0</td>
<td>Insufficient storage available in the server to process the request. System Action: Request is failed but connection remains open. Response: Increase the REGION size for the IKE daemon, or send a message with a narrower set of input filters to limit the response.</td>
</tr>
<tr>
<td>ENOMEM (132)</td>
<td>NMsMrTooManyConns (1)</td>
<td>The NMI thread is already using its maximum number of 50 connections and cannot accept any more. System Action: Connection is not opened and the request is failed. Response: Try the request again later.</td>
</tr>
<tr>
<td>ENOMEM (132)</td>
<td>NMsMrNSClient (2)</td>
<td>Insufficient storage available in the NSS client to process the request. System Action: Request is failed but connection remains open. Response: Increase the REGION size for the NSS client, or send a message with a narrower set of input filters to limit the response.</td>
</tr>
<tr>
<td>ETIMEDOUT (1127)</td>
<td>NMsMrNSClient (2)</td>
<td>Response message was not received from the NSS client in sufficient time. System Action: Request is failed but connection remains open. Response: Resend the request message to the server.</td>
</tr>
</tbody>
</table>
Table 59. Request return and reason codes (continued)

<table>
<thead>
<tr>
<th>Return code (NMsMRc)</th>
<th>Reason code (NMsMRsn)</th>
<th>Description</th>
</tr>
</thead>
</table>
| EIO (122)            | (z/OS UNIX System Services Errno) | A file system I/O error occurred. The reason code contains the errno value that describes the error.  
System Action: Request is failed but connection remains open.  
Response: Diagnose the z/OS UNIX System Services Errno using [z/OS UNIX System Services Messages and Codes](#). |
| EMVSERR (157)        | 0                     | A call to an MVS service failed or an internal NSSD error has occurred.  
System Action: Request fails but connection remains open. A message appears in the MVS system log with additional diagnostic information.  
Response: Contact IBM service. |

### Real-time TCP/IP network monitoring NMI

Table 60 shows the z/OS Communications Server network management interfaces that are described in this section.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTCPDA</td>
<td>Real-time TCP/IP packet trace data NMI</td>
</tr>
<tr>
<td>SYSTCPCN</td>
<td>Real-time TCP connection SMF data NMI</td>
</tr>
<tr>
<td>SYSTCPOT</td>
<td>Real-time OSAENTA packet trace data NMI</td>
</tr>
<tr>
<td>SYSTCPMS</td>
<td>Real-time SMF data NMI</td>
</tr>
</tbody>
</table>

These interfaces allow network management applications to programmatically obtain data in real time. Details for invoking these interfaces and the data provided from them are documented in the following sections. Programmers understand how to parse the data retrieved from these interfaces, and the data structures required to perform this function. Instructions for compiling and linking applications are also provided.

Each of the interfaces described in this section provides a unique type of data to be processed by the end user, but the general interface by which the data is obtained is essentially the same. The records are retrieved using a common data layout, although the records themselves might differ in format depending on the interface.

**Tip:** New SMF 119 records might be added with new releases. If you write an application that processes the SMF 119 records from these NMIs, design the application to receive SMF 119 records that it might not recognize.
The information provided by each interface is as follows:

Table 61. Interface descriptions

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real-time TCP/IP packet trace data NMI (SYSTCPDA)</td>
<td>Using this interface, applications can obtain a copy of network packets (for example, packet trace records) or data trace records that are buffered by the TCP/IP stack’s packet or data trace functions. The packet trace function, data trace function, or both must be enabled with the VARY TCPIP,,PKTTRACE command or VARY TCPIP,,DATTRACE command. See the z/OS Communications Server: IP System Administrator’s Commands for more information about using the Vary command.</td>
</tr>
<tr>
<td>Real-time TCP connection SMF data NMI (SYSTCPCN)</td>
<td>Using this interface, applications can be notified when TCP connections are established or terminated in a near real-time fashion. SYSTCPCN provides applications with a copy of records indicating a TCP connection initiation or termination. These records are presented in the same format as SMF type 119 TCP connection initiation and termination records (for example, subtype 1 and 2 records). The interface can also be used to provide records describing existing TCP connections. This interface does not require TCP/IP SMF recording to be active.</td>
</tr>
<tr>
<td>Real-time TCP/IP OSAENTA trace data NMI (SYSTCPOT)</td>
<td>Using this interface, applications can obtain copies of network packets and records that are buffered by the TCP/IP OSAENTA trace functions. The OSAENTA Trace function must be enabled using the VARY TCPIP,OSAENTA command. See z/OS Communications Server: IP System Administrator’s Commands for more information about using the Vary command.</td>
</tr>
</tbody>
</table>
Table 61. Interface descriptions (continued)

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real-time SMF data NMI (SYSTCPSM)</td>
<td>The records provided through the interface are type 119 SMF records. The specific subtypes that are provided are:</td>
</tr>
<tr>
<td></td>
<td>• TCP/IP profile event record (subtype 4)</td>
</tr>
<tr>
<td></td>
<td>• TN3270E Telnet server session initiation and termination records (subtypes 20 and 21)</td>
</tr>
<tr>
<td></td>
<td>• TSO Telnet client connection initiation and termination records (subtypes 22 and 23)</td>
</tr>
<tr>
<td></td>
<td>• FTP server transfer completion records (subtype 70)</td>
</tr>
<tr>
<td></td>
<td>• FTP server logon failure records (subtype 72)</td>
</tr>
<tr>
<td></td>
<td>• FTP client transfer completion records (subtype 3)</td>
</tr>
<tr>
<td></td>
<td>• IKE tunnel and dynamic tunnel event records (subtypes 73 - 78)</td>
</tr>
<tr>
<td></td>
<td>• Manual tunnel activation and deactivation records (subtypes 79 and 80)</td>
</tr>
<tr>
<td></td>
<td>Except for the MVS SMF header, these records are identical in format to SMF records created by TCP/IP. Some fields in the MVS SMF header are not set.</td>
</tr>
<tr>
<td></td>
<td>These records offer several key advantages over SMF records:</td>
</tr>
<tr>
<td></td>
<td>• They do not require that TCP/IP SMF record capturing is activated.</td>
</tr>
<tr>
<td></td>
<td>• They are presented to the application in a buffered format (for example, when several SMF records are created within a short time interval, they are collected and passed to the application as a group of records instead of individual records).</td>
</tr>
<tr>
<td></td>
<td>In addition to these records, more records are available across this interface that are not currently available from TCP/IP SMF records processing:</td>
</tr>
<tr>
<td></td>
<td>• FTP server transfer initiation records (subtype 100)</td>
</tr>
<tr>
<td></td>
<td>• FTP client transfer initiation records (subtype 101)</td>
</tr>
<tr>
<td></td>
<td>• FTP client login failure records (subtype 102)</td>
</tr>
<tr>
<td></td>
<td>• FTP client session records (subtype 103)</td>
</tr>
<tr>
<td></td>
<td>• FTP server session records (subtype 104)</td>
</tr>
</tbody>
</table>

See "Real-time SMF data NMI (SYSTCPSM) record formats" on page 558 for the structures and mappings of records 100 through 104.

All of these interfaces provide the same two-step process for accessing the data:

1. The Communications Server TCP/IP stack provides an AF_UNIX stream socket for each of the interfaces (see Table 61 on page 541) that allows one or more applications to receive notifications for the data that is being collected. The TCP/IP stack is acting as the server for these AF_UNIX stream sockets, performing the listen() and waiting for incoming connection requests. To use this interface, applications connect to the server’s listening AF_UNIX stream socket. Each of the interfaces has its own, distinct AF_UNIX path name that uniquely identifies the socket to be used by the interface. After SYSTCPDA, SYSTCPOT, or SYSTCPSM is connected, the application immediately starts receiving applicable data. After SYSTCPCN, is connected, the application must send a record to the server to indicate the type of data it desires, only after which does it start receiving applicable data.
2. Each notification record received by the application over the socket represents a buffer being stored by the TCP/IP stack. The actual SYSTCPDA, SYSTCPDN, SYSTCPOT, and SYSTCPSMTP data is not part of this notification record. After the application receives the entire notification record from the AF_UNIX socket, it must then pass this record along with a user-allocated storage buffer to the EZBTMIC1 API routine provided. EZBTMIC1 populates the provided storage buffer with the output records related to the interface that the input notification record defines. After the notification is received over the AF_UNIX socket, the application must invoke EZBTMIC1 or EZBTMIC4 (or TMI_CopyBuffer) immediately because the buffers are stored in a circular queue by the TCP/IP stack, and might eventually be overwritten and invalidated. The network management application also needs to execute at a relatively high priority to ensure that it gets dispatched by the system reasonably quickly so that it can obtain the data before those buffers are overwritten.

The buffer that is copied using the EZBTMIC1 or EZBTMIC4 API call contains the actual data of interest to the application. The format of these buffers, and the records contained therein, are described in "Understanding the common buffer output of the TMI copy buffer service" on page 553.

Guideline: EZBTMIC1 is the API that is used by AMODE 31 callers; EZBTMIC4 is the API that is used by AMODE 64 callers.

In summary, the application connects to open an AF_UNIX socket path name that is defined for the network management interface for which it is to collect information (for example, SYSTCPDN, SYSTCPDA, SYSTCPOT, SYSTCPSMTP) and receives notification records. It passes these records to EZBTMIC1 to copy the actual data of interest into the application’s storage. The application then parses the records in the returned buffer to obtain the actual packet trace or SMF-type records. It is possible for the network management application to connect to one or more of these interfaces from the same application. The application passes these records to an API call to copy the actual data of interest into the application’s storage.

Real-time TCP/IP network monitoring NMI: Configuration and enablement

In order for the TCP/IP stack to collect the data for these interfaces and accept connections over the AF_UNIX socket from clients that want to connect, you must first enable them within the TCP/IP configuration using the NETMONITOR statement in the TCP/IP profile. See the NETMONITOR statement information in the z/OS Communications Server: IP Configuration Reference for details. If you are developing a feature for a product to be used by other parties, you should include in your documentation instructions indicating that administrators must make these configuration changes in order to use that feature.

The z/OS system administrator might restrict access to each of these interfaces by defining the SERVAUTH class EZB.NETMGMT.sysname.tcpprocname.interface profile with UACC of NONE in RACF (or the equivalent security product), and permitting only certain management applications or users to access that interface.

Guidelines:
1. The user ID referenced for this authorization check is the user ID associated with the task and MVS address space that issues the connect() call for the AF_UNIX stream socket.
2. sysname represents the MVS system name where the interface is being invoked.
3. *tcpprocname* represents the job name associated with a TCP/IP started task procedure.

4. *interface* represents SYSTCPDA, SYSTCPCN, SYSTCPOG, or SYSTCPGM.

Information about the SERVAUTH classes:

*EZB.NETMGMT.sysname.tcpprocname.SYSTCPDA*
*EZB.NETMGMT.sysname.tcpprocname.SYSTCPCN*
*EZB.NETMGMT.sysname.tcpprocname.SYSTCPOG* and
*EZB.NETMGMT.sysname.tcpprocname.SYSTCPGM* can be found in *z/OS Communications Server: IP Configuration Guide*.

If the RACF profile is not defined for the interface, then only superusers (users with an OMVS UID of 0 or users permitted to access the BPX.SUPERUSER resource in the FACILITY class) are permitted to use the interface. If you are developing a feature for a product to be used by other parties, include in your documentation instructions indicating that administrators must either define and give appropriate permission to the given security resource for use of that feature, or must run your program as superuser.

**Real-time TCP/IP network monitoring NMI: Connecting to the server**

The application that is to use one of the interfaces must connect to the appropriate AF_UNIX stream socket provided by TCP/IP, which acts as the server. The socket path names for each of these interfaces are as follows. For each of the following, *tcpipprocname* is the procedure name used to start TCP/IP.

- Network monitor interface for capturing data packets (SYSTCPDA)
  
  `/var/sock/SYSTCPDA.tcpipprocname`

- Network monitor interface for obtaining TCP connection information (SYSTCPCN)
  
  `/var/sock/SYSTCPCN.tcpipprocname`

- Network monitor interface for capturing OSAENTA packets (SYSTCPOG)
  
  `/var/sock/SYSTCPOG.tcpipprocname`

- Network monitor interface for obtaining real-time SMF data (SYSTCPGM)
  
  `/var/sock/SYSTCPGM.tcpipprocname`

Use either the Language Environment C/C++ API or the UNIX System Services BPX callable services to open AF_UNIX sockets and connect to the given service.

**Real-time TCP/IP network monitoring NMI: Interacting with the servers**

In the case of the TCP connection information service, after connecting to the SYSTCPCN server over AF_UNIX socket, `/var/sock/SYSTCPCN.tcpipprocname`, the application must then send a connection request record to the server over the connected socket (see the tmi_conn_request record in "Real-time TCP/IP network monitoring NMI: Requests sent by the client to the server" on page 546). For the other three services, the application does not need to take action.

After the client connects to the desired server (or, in the case of the SYSTCPCN service, after sending a connection request record), the server sends an initial record to the client, identifies the server (see the tmi_init record in "Records sent by the server to the client: Initialization record" on page 547). After that record is received, the client is sent tmi_token records that represent data buffers. A record is
sent for each data buffer that is filled in by TCP/IP. Records for partial buffers are
sent if there has been no activity for a brief period. In case there is no activity, the
client should be prepared to wait indefinitely for incoming tokens.

When the server needs to terminate the connection, it attempts to send a special
termination record (see the tmi_term record in “Records sent by the server to the
client: Termination record” on page 547) over the socket to the connected
application, after which it closes the socket. This termination record describes the
reason for closure. In some cases, the server might be unable to send such a record,
and will close the socket. The application should be prepared to handle either case.

Particularly for the SYSTCPDA, SYSTCPO, and SYSTCPCN interfaces, large
amounts of data can be generated. For SYSTCPDA, do not activate a packet trace
filter option that is too broad, in order to avoid recording unnecessary data; see the
VARY TCPIP,DATTRACE or VARY TCPIP,PKTRACE information in the z/OS
Communications Server: IP System Administrator’s Commands and the packet trace
(SYSTCPDA) for TCP/IP stacks information in z/OS Communications Server: IP
Diagnosis Guide for details. For SYSTCPCN, the NETMONITOR MINLIFETIME
TCP/IP profile configuration option can be used to restrict the collection of
short-lived connections; see the NETMONITOR statement information in z/OS
Communications Server: IP Configuration Reference for details. For SYSTCPO, see the
VARY TCPIP,OSAENTA information in z/OS Communications Server: IP System
Administrator’s Commands.

Restriction: Except in the case of sending a connection request record for the
SYSTCPCN service, the client application must never send data to the server. If
data is unexpectedly received by the server, the server sends a termination record
with tmit_termcode = EPIPE to the client, and closes the connection.

Real-time TCP/IP network monitoring NMI common record
header

All data sent over the AF_UNIX socket by the client and the server is prefixed
with a common header indicating the length of the entire record (this length
includes the header) and the type of data contained within the record. The format
for the header is as follows, as defined in ezbytmih.h (an assembler mapping for
this structure is in EZBYTMIA):

```
struct tmi_header
{
    int TmiHr_len;       /* Length of this record */
    int TmiHr_Id;        /* Identifier for this record */
    int TmiHr_Ver;       /* Version identifier for this */
    int TmiHr_resv;      /* reserved */
};
#define TmiHr_CnRqst 0xC3D5D9D8 /* Constant("CNRQ") */
#define TmiHr_Init 0xC9D5C9E3 /* Constant("INIT") */
#define TmiHr_Term 0xE3C5D9D4 /* Constant("TERM") */
#define TmiHr_SmfTok 0xE2D4E3D2 /* Constant("SMTK") */
#define TmiHr_PktTok 0xE2D7D3E2 /* Constant("TPKT") */
#define TmiHr_Version1 1    /* Version number */
```

Chapter 14. Network management interfaces 545
Real-time TCP/IP network monitoring NMI: Requests sent by the client to the server

For the SYSTCPCN service only, the client must send a request record to the server after connecting to the server’s AF_UNIX socket. This request record is in the following format, defined in ezbytmih.h (an assembler mapping for this structure is in EZBYTMIA):

```c
struct tmi_conn_request /* Conn info server request */
{
    struct tmi_header tmicnrq_hdr; /* Header; id=TMI_ID_CNRQST */
    unsigned int tmicnrq_list :1; /* Requests connection list */
    unsigned int tmicnrq_smf :1; /* Requests init/term SMFrcd*/
    unsigned int tmicnrq_rsvd1 :30; /* Reserved, set to 0 */
    char tmicnrq_rsvd2[12]; /* Reserved, set to 0 */
};
```

The client should initialize the fields of this request structure as follows:

- Initialize `tmicnrq_hdr` using the length of `tmi_conn_request`, the appropriate record ID (TMIHr_CnRqst), and the correct version (TMIHr_Version1).
- Initialize the `tmicnrq_list` and `tmicnrq_smf` fields as described in the following list.
- Initialize all remaining fields to 0.

The two fields `tmicnrq_list` and `tmicnrq_smf` control the data that the SYSTCPCN server sends to the client. These fields should be set as follows:

- `tmicnrq_list`
  
  If set, the server sends the client zero or more tokens that represent data buffers that contain a list of all established TCP connections at the time the client connected. These connections are represented as type 119 TCP connection initiation SMF records. If this field is set to 0, no such list is sent to the client.

- `tmicnrq_smf`
  
  If set, the server sends tokens to the client. These tokens represent data buffers that contain type 119 TCP connection initiation and termination SMF records, representing TCP connections that are established and closed on the TCP/IP stack. If this field is set to 0, the server does not send any tokens, representing ongoing connection establishment and closure.

The SYSTCPCN server waits until it has received this entire record from the client before it starts processing connection information on the client’s behalf. If the client does not send a complete record, then the server never reports data to the client, since the client has not completed initialization. If the server receives a record with an unrecognized version, a bad length, or a bad eyecatcher, then it sends a termination record (see “Records sent by the server to the client: Termination record” on page 547) with `tmit_termcode = EINVAL` to the client, and closes the connection.

Real-time TCP/IP network monitoring NMI: Records sent by the server to the client

For each of the three interfaces, the server sends three types of records to the client:

- Initialization records
- Termination records
- Token records

Each record is described in the sections that follow.
Records sent by the server to the client: Initialization record

After the client connects to the server, the server sends an initialization record to the client. The initialization record can be recognized as having a \textit{TmiHr\_Id} equal to \textit{TmiHr\_Init}. This record contains miscellaneous information about the server and the stack that the client can choose to use or ignore. This record has the following format, defined in ezbytmih.h (an assembler mapping for this structure is in EZBYTMIA):

\begin{verbatim}
struct tmi_init /* Connection startup record */
{
    struct tmi_header tmii_hdr; /* Record header */
    char tmii_sysn[8]; /* System name (EBCDIC) */
    char tmii_comp[8]; /* Component name (EBCDIC) */
    char tmii_sub[8]; /* TCPIP job name (EBCDIC) */
    char tmii_time[8]; /* Time TCPIP started (STCK) */
    int tmii_bufsz; /* Maximum size of buffer */
    char tmii_rsvd[12]; /* Reserved */
};
\end{verbatim}

- The component name, \textit{tmii\_comp}, represents the server that the client is connected to. This is one of SYSTCPDA, SYSTCPCN, SYSTCPO\_T, or SYSTCPS\_M, depending on the server that is being accessed.
- The \textit{tmii\_bufsz} value is the minimum size of the buffer required to be provided on the EZBTMIC1 call. If the value is 0, a maximum of a 64 KB buffer is copied.

Records sent by the server to the client: Termination record

The termination record is sent when the server closes the connection. The termination record can be recognized as having a \textit{TmiHr\_Id} equal to \textit{TmiHr\_Term}. The connection might be closed as part of normal operation (for example the service is being disabled or the stack is terminating), or it might be closed as the result of some error. A termination code in the record indicates the termination reason.

This record is the last data sent by the server before close; after sending the termination record, the server closes the connection. The stack attempts to send the termination record before it closes the socket. However, under certain abnormal stack termination conditions, it might be unsuccessful; furthermore, if the client's receive buffer is full, it might also be unsuccessful. In such cases the connection is closed.

The format of this record is as follows, as defined in ezbytmih.h (an assembler mapping for this structure is in EZBYTMIA):

\begin{verbatim}
struct tmi_term /* Termination notification rcd */
{
    struct tmi_header tmit_hdr; /* Record header */
    unsigned int tmit_termcode; /* Termination code */
    char tmit_tstamp[8]; /* Termination timestamp */
    char tmit_rsvd[12]; /* Reserved */
};
\end{verbatim}

The possible values for \textit{tmit\_termcode} and their explanations are as follows, as defined in errno.h:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error; planned termination. Either this function is being disabled or the TCP/IP stack is ending.</td>
</tr>
<tr>
<td>EACCES  (111)</td>
<td>The client is not permitted to connect to the server.</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EINVAL (121)</td>
<td>The client has sent data that is not valid to the server.</td>
</tr>
<tr>
<td>ENOMEM (132)</td>
<td>The server was unable to allocate necessary storage.</td>
</tr>
<tr>
<td>EPIPE (140)</td>
<td>The client has erroneously sent data to the server when the server was not expecting data.</td>
</tr>
<tr>
<td>EWOULDBLOCK (1102)</td>
<td>The server could not write to the client socket because the client’s receive buffer is full.</td>
</tr>
</tbody>
</table>

See z/OS UNIX System Services Messages and Codes for more detail.

The `tmst_tstamp` field contains an 8-byte MVS TOD clock value for the time of termination of the connection.

The client should expect to receive no more data on the connection following this record; the connection is closed by the server.

**Records sent by the server to the client: Token record**

The server sends the `tmi_token` record when a buffer has been filled with records for the given service. The token record can be recognized as having a `TmiHr_Id` value that is equal to the `TmiHr_PktTok` value (in the case of SYSTCPDA and SYSTCPOT) or the `TmiHr_SmfTok` value (in the case of SYSTCPCN and SYSTCPMSM). In addition, each of the servers will, after a brief period of inactivity, flush a partially filled buffer, sending a token for that partial buffer and advancing to the next internal buffer.

The format of this record is as follows, as defined in ezbytmih.h (an assembler mapping for this structure is in EZBYTMIA):

```c
struct tmi_token
{
    struct tmi_header    tmik_hdr;    /* Record header */
    char                  tmik_token[32]; /* Token representing buffer */
};
```

The `tmik_token` record contains a token describing the data buffer. The client’s actions upon receiving this record are discussed in "Copying the trace buffer."

**Copying the trace buffer**

Use the EZBMLC or EZBMLCI service to copy the data buffer to the client application’s storage. You can invoke EZBMLC or EZBMLCI through a C function, TML_CopyBuffer, which calls the callable service. EZBMLC and EZBMLCI use the `tmik_token` record that was recently read from the AF_UNIX socket as input to locate and copy the data buffer to the buffer that is provided by the user.
EZBTMIC1 or EZBTMIC4: Copy TCP/IP Management Interface
Data Buffer

The EZBTMIC1 and EZBTMIC4 callable services use a token that is provided over a TCP/IP management interface to copy a data buffer into application storage. These services are also referred to as the TMI copy buffer services.

Guideline: EZBTMIC1 is the API that is used by AMODE 31 callers, EZBTMIC4 is the API that is used by AMODE 64 callers. References to the EZBTMIC1 API also apply to the EZBTMIC4 API.

EZBTMIC1 requirements

<table>
<thead>
<tr>
<th>Authorization:</th>
<th>Supervisor state or problem state, any PSW key; Caller must be APF authorized.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>31-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary 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 must be in the primary address space.</td>
</tr>
</tbody>
</table>

EZBTMIC4 requirements

<table>
<thead>
<tr>
<th>Authorization:</th>
<th>Supervisor state or problem state, any PSW key; Caller must be APF authorized.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN = HASN</td>
</tr>
<tr>
<td>AMODE:</td>
<td>64-bit</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary 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 must be in the primary address space.</td>
</tr>
</tbody>
</table>

EZBTMIC1 format

CALL EZBTMIC1,(Token,
    Bufptr,
    Return_value,
    Return_code,
    Reason_code)

EZBTMIC4 format

CALL EZBTMIC1,(Token,
    Bufptr,
    Return_value,
    Return_code,
    Reason_code)
**EZBTMIC1 or EZBTMIC4 parameters**

**Token**  The name of a record containing a token describing a TCP/IP management interface data buffer.

- **Type:** Structure
- **Length:** Size of buffer token record

**Bufptr**  The address of a buffer into which the TCP/IP management data buffer is copied.

- **Type:** Structure
- **Length:** 12

The `bufptr` parameter is a 12-byte structure describing the address of the buffer:

```assembly
Bufptr DS 0F  /* Buffer pointer */
Buf_alet DC F'0'  /* Buffer ALET, or 0 */
Buf_addr_hi DC F'0'  /* Highword of 64bit bufptr */
Buf_addr DC A(0)  /* Lowword of 64bit bufptr */
```

- If the buffer is in a data space, then `Buf_alet` is the ALET of the data space; otherwise it is 0. If the buffer is in 64-bit storage, then `Buf_addr_hi` and `Buf_addr` contain the 64-bit address of the buffer. If the buffer is in 24-bit or 31-bit storage, then `Buf_addr_hi` contains zeros and the buffer address in `Buf_addr`. To improve performance, place the buffer on a page boundary.

This buffer can represent the following:

- When the token is a `TmiHr_PktTok` token, the data buffer contains the unformatted packet trace data records (SYSTCPDA or SYSTCPOF).
- When the token is a `TmiHr_SmfTok` token, the data buffer will contain SMF records (SYSTCPCCN or SYSTCPCHM).

**Return_value**  Returned parameter. The name of a fullword in which the TMI buffer copy service returns the results of the request:

- **Type:** Integer
- **Length:** Fullword

- >0: The data buffer has been successfully copied into the application buffer. The return value is the number of bytes of data that has been copied into the buffer. This length does not include the trailing halfword of zeros in the buffer.
- −1: The system could not complete the request, for reasons such as the data buffer being no longer valid. See `Return_code` and `Reason_code` for more details.

**Return_code**  Returned parameter. The name of a fullword in which the TMI buffer copy service stores the return code. The TMI buffer copy service returns `Return_code` only if `Return_value` is −1. The TMI buffer copy service can return one of the following values in the `Return_code` parameter:

<table>
<thead>
<tr>
<th>Return_value</th>
<th>Return_code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;0</td>
<td>0</td>
<td>The request was successful.</td>
</tr>
<tr>
<td>Return_value</td>
<td>Return_code</td>
<td>Meaning</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>−1</td>
<td>EACCES</td>
<td>The application is not APF authorized.</td>
</tr>
<tr>
<td>−1</td>
<td>EBADF</td>
<td>The token provided to locate a buffer is not a valid token.</td>
</tr>
<tr>
<td>−1</td>
<td>EFAULT</td>
<td>The address is incorrect.</td>
</tr>
<tr>
<td>−1</td>
<td>EINVAL</td>
<td>The token provided to locate a buffer does not specify a valid data buffer.</td>
</tr>
<tr>
<td>−1</td>
<td>EILSEQ</td>
<td>The data buffer described by token has been overwritten and is no longer available.</td>
</tr>
</tbody>
</table>

**Reason_code**

The name of a fullword in which the TMI buffer copy service stores the reason code.

Type: Integer  
Length: Fullword

The TMI buffer copy service returns *Reason_code* only if *Return_value* is −1. The reason code contains diagnostic information and is described in [z/OS UNIX System Services Messages and Codes](https://www.ibm.com/support/knowledgecenter/SSEG9H_2.2.0/com.ibm.zos.hlposalog.pdf).

**EZBTMIC1 or EZBTMIC4 usage notes**

- Compiling and linking
  - Assembler mappings for the various records that flow over the AF_UNIX socket are in macro EZBTMIA.
  - EZBTMIC1 and EZBTMIC4 are defined as callable stubs in SYS1.CSSLIB.

**TMI_CopyBuffer: Copy TCP/IP Management Interface Data Buffer**

The `TMI_CopyBuffer()` function copies the TMI data buffer described in *token* to the application-provided buffer pointed to by *bufptr*. `Ezbytmih.h` contains this definition.

**TMI_CopyBuffer format**

```c
void Tmi_CopyBuffer (struct tmi_header *token,
                     struct bufptr_t *bufptr,
                     int *retval,
                     int *retcode,
                     int *rsncode);
```

**TMI_CopyBuffer parameters**

- **token**  
  The pointer to the token record read from the TCP/IP management interface service. The record contains a token used to locate a data buffer to be copied.

- **bufptr**  
  A pointer to a `tmi_bufptr` structure describing a 64 KB buffer provided by the user. The indicated buffer is overwritten with the contents of the TMI data buffer if the call is successful.

  The `tmi_bufptr` structure is a 12-byte structure that describes the address of the buffer for AMODE 31 callers.
struct tmi_bufptr /* Buffer pointer */
{
  int buf_alet; /* Buffer ALET, or 0 */
  int buf_addr_hi; /* Highword of 64bit bufptr */
  void *buf_addr; /* Lowword of 64bit bufptr */
};

When _LP64 is defined, the tmi_bufptr structure is a 12-byte structure that
describes the address of the buffer for AMODE 64 callers.

struct tmi_bufptr /* Buffer pointer */
{
  int buf_alet; /* Buffer ALET, or 0 */
  void *buf_addr; /* Pointer to 64bit bufptr */
};

retval The returned value. If successful, TMI_CopyBuffer() returns the number of
bytes copied in retval. If unsuccessful, TMI_CopyBuffer() returns −1 in
retval and returns retcode as described in the following definition.

retcode
A pointer to a fullword in which the TMI buffer copy service stores the
return code. The TMI buffer copy service returns retcode only if retval is −1.
The TMI buffer copy service can return one of the following values in the
retcode parameter.

<table>
<thead>
<tr>
<th>Return_code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EACCES</td>
<td>The application is not APF authorized.</td>
</tr>
<tr>
<td>EBADF</td>
<td>The token provided to locate a buffer is not a valid token.</td>
</tr>
<tr>
<td>EFAULT</td>
<td>Using the buffer parameter as specified would result in an attempt to access storage outside the caller’s address space.</td>
</tr>
<tr>
<td>EINVAL</td>
<td>The token provided to locate a buffer does not specify a valid data buffer.</td>
</tr>
<tr>
<td>EILSEQ</td>
<td>The data buffer described by token has been overwritten and no longer available.</td>
</tr>
</tbody>
</table>

rsncode
The address of a fullword in which the TMI buffer copy service stores the
reason code. The TMI buffer copy service returns rsncode only if retval is
−1. The reason code contains diagnostic information and is described in
z/OS UNIX System Services Messages and Codes.

TMI_CopyBuffer usage notes
- Character data
  Some of the data contained in the TMI data buffer might be system data, such as
  job names. Such data is encoded in EBCDIC and the application should be
  prepared to process it appropriately.
- Compiling and linking
  The callable service routine that provides this service is provided as a callable
  stub located in SYS1.CSSLIB.
Understanding the common buffer output of the TMI copy buffer service

Upon successful completion of the EZBTMIC1 or EZBTMIC4 call of the TMI_CopyBuffer(), the user-supplied buffer is filled with cte records, which contain the data provided by the service being used.

The data records for the server are stored sequentially within individual data buffers. The cte describes the length of the data record. The data record is immediately followed by a cteplg (cte epilogue) structure. The first cte structure begins at the beginning of the buffer. The last cteplg is followed by a cte whose ctelenp field is 0; this signifies the end of the data in the buffer. The layout of the buffer is as follows:

| cte | data | cte_plogue | cte | data | cte_plogue | ... | cte | data | cte_plogue | binary 0 |

The cte is a 16-byte descriptor whose format is as follows (as defined in ezbytmih.h, and in ITTCTE in SYS1.MACLIB):

```c
struct cte {
    unsigned short cteplenp; /* Length of CTE and cte_epilogue. */
    short cteoff; /* Offset from start of CTE */
    unsigned long ctefmtid; /* Format ID of record */
    unsigned long long cteetime; /* STCK timestamp of record creation */
};
```

ctelenp holds the total length of the record, including the cte, the data record, and the cte_plogue. cteoff is the offset to the data record from the start of the cte. The ctefmtid is a format ID specific to each service; it is described in "Format of service-specific data." The cteetime is an 8-byte STCK timestamp of the time the record was written.

The format of the 2-byte cteplg is as follows (as defined in ezbytmih.h, and in ITTCTE in SYS1.MACLIB):

```c
struct cteplg {
    unsigned short ctelen; /* Length of CTE, data, and cte_plogue. */
};
```

The field ctelen holds the same value as the ctelenp field in the cte.

Format of service-specific data

The following information describes how to process CTE records for SYSTCPDA, SYSTCPCN, and SYSTCPSM.

Processing the CTE records for SYSTCPDA and SYSTCPOT

The following ctefmtid values are supported for the SYSTCPDA and SYSTCPOT interfaces:

<table>
<thead>
<tr>
<th>ctefmtid</th>
<th>Data area</th>
<th>Description</th>
<th>Command to start</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (TRCIDPKT)</td>
<td>Described by the GtCntl structure</td>
<td>IPv4 packet trace record</td>
<td>VARY TCP,,PKTTRACE</td>
</tr>
</tbody>
</table>

See note
<table>
<thead>
<tr>
<th>ctefmtid</th>
<th>Data area</th>
<th>Description</th>
<th>Command to start</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 (TRCIDX25)</td>
<td>Described by the GtCntl structure</td>
<td>IPv4 packet trace record</td>
<td>N/A</td>
</tr>
<tr>
<td>See note</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 (TRCIDDAT)</td>
<td>Described by the GtCntl structure</td>
<td>IPv4 data trace record</td>
<td>VARY TCPIP,,DATTRACE</td>
</tr>
<tr>
<td>See note</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 (PTHIdPkt)</td>
<td>Described by the PTHDR_T structure</td>
<td>IPv4 or IPv6 packet trace record</td>
<td>VARY TCPIP,,PKTTRACE</td>
</tr>
<tr>
<td>5 (PTHIdDat)</td>
<td>Described by the PTHDR_T structure</td>
<td>IPv4 or IPv6 data trace record</td>
<td>VARY TCPIP,,DATTRACE</td>
</tr>
<tr>
<td>6 (PTHIdEE)</td>
<td>Described by the PTHDR_T structure</td>
<td>IPv4 or IPv6 EE trace record</td>
<td>VARY TCPIP,,PKTTRACE, SRCP =12000</td>
</tr>
<tr>
<td>7 (PTHidNTA)</td>
<td>Described by the PTHDR_T structure</td>
<td>IPv4 and IPv6 OSAENTA trace record</td>
<td>VARY TCPIP,,OSAENTA</td>
</tr>
</tbody>
</table>

Notes:
1. As of the V1R6 release, type 1, 2, and 3 records are no longer written.
2. In V1R11 and later releases, the GtCntl description is not included in EZBCTHDR.

If tracing for the TCP/IP data trace and the TCP/IP packet trace is active, the trace buffer will contain both types of records. The client must handle this condition.

The PTHDR_T is defined in EZBYPTHQA and contains the following information:

- **pth_len** Length of the PTHDR_T structure
- **pth_seqnum** Sequence number of this packet trace record
- **pth_flag** Flag indicators
  - PTH_Cftxt 0x40 Confidential was not recorded in data trace record
  - PTH_Seg_Offload 0x10 TCP Segmentation Offload
  - PTH_Pdu 0x08 Data from multiple PDUs
  - PTH_Adj 0x04 Record size was adjusted by +1 (reflected in the ctelen and ctelenp). The data length was odd and a single pad byte was added.
  - PTH_Abrv 0x02 ABBREV parameter was used on the trace command
  - PTH_Out 0x01 IP packet was sent = 1 rcvd = 0
- **pth_devty** The type of device represented by the interface being traced.

554 z/OS V1R11.0 Comm Svrv: IP Programmer’s Guide and Reference
PTHQIDIO  43  - iQdio
PTH6loopb  51  - IPv6 loopback
PTH6vipa   52  - fipv6ipa
PTH6ipaqnet 53  - ifp6ipaqnet
PTH6ipaqtr  54  - ifp6ipaqtr
PTH6mpc    55  - ifp6mpc
PTH6ipaqidio 56  - ifp6ipaqidio

pth_tlen    Portion of the payload that is actually traced. If ABBREV was not specified on the trace command then this will be the name as pth_plen. If ABBREV was specified, then it will be this value.

pth_infname Name of the interface the packet was traced on in EBCDIC character format

pth_jobname The jobname from the data trace record

pth_DtDomain Socket domain (AF_INET or AF_INET6)

pth_DtType Socket type (Sock#_Stream, Sock#_Dgram, Sock#_Raw)

pth_DtProto Socket protocol number

pth_time    Stored time of day clock when packet was processed by the trace

pth_src     Hexadecimal source IP address of this packet (IPv6 or IPv4)

pth_dst     Hexadecimal destination IP address of this packet (IPv6 or IPv4)

pth_sport   Hexadecimal source IP port number

pth_dport   Hexadecimal destination IP port

pth_trcnt   Total count of records traced

pth_tcb     Task control block address of the sender of the outbound. On inbound, this will usually be task associated with the TCP/IP stack

pth_asid    Asbasid of the sender of the outbound packet. On inbound, this will usually be the asid of the TCP/IP stack

PTH_SeqNr   OSAENTA trace sequence number

Pth_Vlan    OSAENTA Vlan id field

Pth_VlanPri Vlan priority (0-7)

Pth_VlanId  Vlan Id (0-4095)

PTH_NtaFlag OSAENTA Flags

Pth_OutB    1-Outbound, 0-Inbound

Pth_Lpar    1=Lpar_to_Lpar

Pth_DD      Data Device is valid

Pth_gVlan   VLAN frame

PTH_LS      Large Send

Pth_QHdr    Qdio header present

Pth_Exhdr   1=Extension header

Pth_Layer2  1=Layer2 0=Layer3

pth_lost    Total lost record count

pth_plen    Payload length (If segmentation is offloaded, then this is the total data length of all segments being offloaded plus the length of one set of headers.)

PTH_InfIx   Interface index (PKTT)

PTH_Cid    Communication Id(PKTD)

PTH_DevId  Device Id  (PKTL)

PTH_DropRsn Packet discard reason code. pth_DropRsn can be compared with the discard reason code that is provided by EZBCODE in sys1.sezanmac. A packet can be traced twice, once at the lower level IP layer, and again as a discarded packet in an upper level protocol layer of TCPIP. This value is 0 if the packet was not discarded.

PTH_OffSegLen Length of each of the first N-1 segments being offloaded, not including headers - i.e. the MSS (meaningful only when the PTH_Seg_Offload flag is on)

IPv4 addresses in pth_src and pth_dst are prefixed with x'000000000000', x'00000000FFFF' or x'000000000000FFFF'.

**Processing trace records in a buffer:** The EZBTMIC1 call or the TMI_CopyBuffer() service is used to receive a buffer of trace records defined by a starting CTE structure and ending with a 2-byte ctelen field, which has the same value as the ctelen. The PTHDR_T structure follows the CTE and has many fields for use when processing the trace records. The pth_tlen field is the IP packet...
payload length, although this field could reflect the ABBREV parameter on the
PKTTRACE command. When TCP segmentation is being offloaded (indicated by
flag pth_seg_offload), then the pth_plen field represents the data length of all the
segments being offloaded. When the pth_seg_offload flag is set, the pth_numsegs field
indicates the number of offloaded segments and the pth_offseglen field indicates the
length of each segment (the MSS). In some cases, to obtain the entire IP packet,
multiple trace records must be processed. These trace records could span multiple
buffers and will probably not be contiguous. In this case, several fields must be
examined. See the example of packet records in "Example of split buffers for IP
packet X." The cteplen will be less than the pth_tlen. The pth_seqnum fields must be
used to determine the ordered chain of records that make up the IP packet. The
first record in the sequence will have pth_seqnum=0 and will contain the IP protocol
headers. The pth_len and pth_time is the same for each record in the sequence.

Example of split buffers for IP packet X: First TMI_CopyBuffer() issued; a complete
buffer received:

<table>
<thead>
<tr>
<th>Trace Records</th>
<th>PTHDR_T structure</th>
<th>trace data</th>
<th>ctelemp=1 KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record1 for IP packet X</td>
<td>pth_seqnum=0</td>
<td>(IP packet X)</td>
<td></td>
</tr>
<tr>
<td>CTE structure</td>
<td>pth_tlen=64 KB</td>
<td>contains IP</td>
<td></td>
</tr>
<tr>
<td>ctelemp=1 KB</td>
<td>pth_time=Time X</td>
<td>headers</td>
<td></td>
</tr>
</tbody>
</table>

Second TMI_CopyBuffer issued; a complete buffer received:

<table>
<thead>
<tr>
<th>Trace Records</th>
<th>PTHDR_T structure</th>
<th>trace data</th>
<th>ctelemp=1 KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record1 for IP packet Y</td>
<td>pth_seqnum=0</td>
<td>(IP packet Y)</td>
<td></td>
</tr>
<tr>
<td>CTE structure</td>
<td>pth_tlen=ip</td>
<td>contains IP</td>
<td></td>
</tr>
<tr>
<td>ctelemp=1 KB</td>
<td>payload length (less than 1 KB)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Record2 for IP packet X
CTE structure
ctelemp=n

<table>
<thead>
<tr>
<th>Trace Records</th>
<th>PTHDR_T structure</th>
<th>trace data</th>
<th>ctelemp=nmn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record2 for IP packet X</td>
<td>pth_seqnum=1</td>
<td>(IP packet X continued)</td>
<td></td>
</tr>
<tr>
<td>CTE structure</td>
<td>pth_tlen=64 KB</td>
<td>no headers</td>
<td></td>
</tr>
<tr>
<td>ctelemp=nmn</td>
<td>pth_time=Time X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Last TMI_CopyBuffer() issued; a partial buffer received:

<table>
<thead>
<tr>
<th>Trace Records</th>
<th>PTHDR_T structure</th>
<th>trace data</th>
<th>ctelemp=nmn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record n for IP packet X</td>
<td>pth_seqnum=n-1</td>
<td>(IP packet X continued)</td>
<td></td>
</tr>
<tr>
<td>CTE structure</td>
<td>pth_tlen=64 KB</td>
<td>No headers</td>
<td></td>
</tr>
<tr>
<td>ctelemp=nmn</td>
<td>pth_time=Time X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Processing the CTE records for SYSTCPCN
The TCP connection information server (SYSTCPCN) presents information about
the establishment and closing of TCP connections as they occur. Type 119 SMF TCP
connection initiation and termination records (subtypes 1 and 2) are stored in the
data buffer to reflect this activity. Each record in the data buffer is a complete type
119 SMF record, of subtype 1 or 2.

Additionally, if requested, the server fills one or more buffers with the list of
currently active connections. This list is provided as type 119 TCP connection
initiation records (subtype 1), so that entries in the list are indistinguishable from
newly established connections (except that the connection establishment timestamp
is in the past). This set of records is sent only once per new connection, after the
initialization.

For the TCP connection information server, the ctefmtid for the CTE is always equal
to the subtype of the SMF record (either 1 or 2) following the CTE in the data
buffer.

Applications can use this interface to dynamically maintain a list of active TCP
connections. As a result of timing issues, it is possible that an application will
receive two initiation records for a given connection (if the connection is
established around the time the client connects, its initiation record will be sent, as
will a record identifying it as a preexisting established connection). It is also
possible that an application will receive a termination record for a connection for
which it has not received an initiation record. Client applications should be
prepared to handle both of these possibilities.

SMF recording for TCP connection initiation and termination records does not need
to be active for this service to function. Moreover, activating this service does not
cause TCP connection initiation and termination SMF records to be recorded into
the SMF data sets if they are not already enabled.

C structures for mapping the SMF type 119 records can be found in ezasmf.h.
Assembler mappings for the structures can be found in EZASMF77 in
SYS1.MACLIB.

**Processing the CTE records for SYSTCPSM**
The real-time SMF data server (SYSTCPSM) reports type 119 SMF event records for
TCP/IP applications. Each record in the data buffer is a complete type 119 SMF
record. The records reported, and their subtypes, are as follows:

- FTP client transfer initialization (subtype 101)
- FTP client transfer completion (subtype 3)
- FTP client login failure records (subtype 102)
- FTP client session records (subtype 103)
- FTP server transfer initialization (subtype 100)
- FTP server session records (subtype 104)
- FTP server transfer completion (subtype 70)
- FTP server logon failure (subtype 72)
- TN3270E Telnet server session initialization (subtype 20)
- TN3270E Telnet server session termination (subtype 21)
- TSO Telnet client connection initialization (subtype 22)
- TSO Telnet client connection termination (subtype 23)
- IPSec IKE Tunnel Activation/Refresh (subtype 73)
- IPSec IKE Tunnel Deactivation/Expire (subtype 74)
- IPSec Dynamic Tunnel Activation/Refresh (subtype 75)
• IPSec Dynamic Tunnel Deactivation (subtype 76)
• IPSec Dynamic Tunnel Added (subtype 77)
• IPSec Dynamic Tunnel Removed (subtype 78)
• IPSec Manual Tunnel Activation (subtype 79)
• IPSec Manual Tunnel Deactivation (subtype 80)
• TCP/IP profile event (subtype 4)

For the real-time SMF data server, the `ctefmtid` for the CTE is always equal to the subtype of the SMF record (one of the values listed above) following the CTE in the data buffer. Table 62 lists the structures and macros for mapping the SMF 119 record subtypes that are delivered by these interfaces.

**Table 62. SMF 119 record subtypes**

<table>
<thead>
<tr>
<th>Subtype</th>
<th>C/C++</th>
<th>Assembler macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>3, 20, 21, 22, 23, 70, 72, 73, 74, 75, 76, 77, 78, 79, 80</td>
<td>SEZANMAC(EZASMF) /usr/include/ezasmf.h</td>
<td>SYS1.MACLIB(EZASMF77)</td>
</tr>
<tr>
<td>4</td>
<td>SEZANMAC(EZBNMMPC) /usr/include/ezbnmmpc.h</td>
<td>SEZANMAC(EZBNMMPA)</td>
</tr>
<tr>
<td>100, 101, 102, 103, 104</td>
<td>SEZANMAC(EZANMFTC)</td>
<td>SEZANMAC(EZANMFTA)</td>
</tr>
</tbody>
</table>

The FTP type 119 records of subtypes 100 through 104 are available only across the SYSTCPSM real-time interface.

See SMF type 119 records information in the `z/OS Communications Server: IP Configuration Reference` for the formats of SMF type 119 records.

**Real-time SMF data NMI (SYSTCPSM) record formats**

The following record formats are used by the Real-time SMF data NMI (SYSTCPSM).

**SYSTCPSM record formats: TCP/IP identification section**

Every record starts with a common section followed by the TCP/IP identification section. The self-defining sections vary, as do the sections that follow the TCP/IP identification section. The TCP/IP identification section is identical for all records except one field, SMF119TI_Comp, at offset 32 (X'20') within the record. This field indicates whether the record was written by the FTP client or the FTP server.

Table 63 shows the format of the TCP/IP identification section.

**Table 63. TCP/IP identification section format**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'0')</td>
<td>SMF119TI_SYSName</td>
<td>8</td>
<td>System name from SYSNAME in IEASYSxx</td>
</tr>
<tr>
<td>8 (X'8')</td>
<td>SMF119TI_SysplexName</td>
<td>8</td>
<td>Sysplex name from SYSPLEX in COUPLExx</td>
</tr>
<tr>
<td>16 (X'10')</td>
<td>SMF119TI_Stack</td>
<td>8</td>
<td>TCP/IP stack name for z/OS Communications Server 390</td>
</tr>
<tr>
<td>24 (X'18')</td>
<td>SMF119TI_ReleaseID</td>
<td>8</td>
<td>TCP/IP release identifier</td>
</tr>
</tbody>
</table>
Table 63. TCP/IP identification section format (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 (X'20')</td>
<td>SMF119TI_Comp</td>
<td>8</td>
<td>TCP/IP subcomponent that created the record (right padded with blanks):</td>
</tr>
<tr>
<td></td>
<td>v FTPC: FTP client</td>
<td></td>
<td>• FTPC: FTP client</td>
</tr>
<tr>
<td></td>
<td>v FTPS: FTP server</td>
<td></td>
<td>• FTPS: FTP server</td>
</tr>
<tr>
<td></td>
<td>v IP: IP layer</td>
<td></td>
<td>• IP: IP layer</td>
</tr>
<tr>
<td></td>
<td>v IKE: IKE daemon</td>
<td></td>
<td>• IKE: IKE daemon</td>
</tr>
<tr>
<td></td>
<td>v STACK: Entire TCP/IP stack</td>
<td></td>
<td>• STACK: Entire TCP/IP stack</td>
</tr>
<tr>
<td></td>
<td>v TCP: TCP layer</td>
<td></td>
<td>• TCP: TCP layer</td>
</tr>
<tr>
<td></td>
<td>v TN3270C: TN3270 client</td>
<td></td>
<td>• TN3270C: TN3270 client</td>
</tr>
<tr>
<td></td>
<td>v TN3270S: TN3270 server</td>
<td></td>
<td>• TN3270S: TN3270 server</td>
</tr>
<tr>
<td></td>
<td>v UDP: UDP layer</td>
<td></td>
<td>• UDP: UDP layer</td>
</tr>
<tr>
<td>40 (X'28')</td>
<td>SMF119TI_ASName</td>
<td>8</td>
<td>Started-task qualifier or address-space name of address space that writes this SMF record</td>
</tr>
<tr>
<td>48 (X'30')</td>
<td>SMF119TI_UserID</td>
<td>8</td>
<td>User ID of security context under which this SMF record is written</td>
</tr>
<tr>
<td>56 (X'38')</td>
<td>SMF119TI_ASID</td>
<td>4</td>
<td>ASID of address space that writes this SMF record</td>
</tr>
<tr>
<td>60 (X'3C')</td>
<td>SMF119TI_Reason</td>
<td>4</td>
<td>Reason for writing this SMF record:</td>
</tr>
<tr>
<td></td>
<td>v X'48': Event record, more records follow</td>
<td></td>
<td>• X'48': Event record, more records follow</td>
</tr>
<tr>
<td></td>
<td>v X'08': Event record, last record in set</td>
<td></td>
<td>• X'08': Event record, last record in set</td>
</tr>
<tr>
<td></td>
<td>v X'C0': Interval record, more records follow</td>
<td></td>
<td>• X'C0': Interval record, more records follow</td>
</tr>
<tr>
<td></td>
<td>v X'80': Interval record, last record in set</td>
<td></td>
<td>• X'80': Interval record, last record in set</td>
</tr>
<tr>
<td></td>
<td>v X'60': End-of-statistics record, more records follow</td>
<td></td>
<td>• X'60': End-of-statistics record, more records follow</td>
</tr>
<tr>
<td></td>
<td>v X'20': End-of-statistics record, last record in set</td>
<td></td>
<td>• X'20': End-of-statistics record, last record in set</td>
</tr>
<tr>
<td></td>
<td>v X'50': Shutdown starts record, more records follow</td>
<td></td>
<td>• X'50': Shutdown starts record, more records follow</td>
</tr>
<tr>
<td></td>
<td>v X'10': Shutdown starts record, last record in set</td>
<td></td>
<td>• X'10': Shutdown starts record, last record in set</td>
</tr>
</tbody>
</table>

SYSTCPMSM record formats: FTP server transfer initialization record

Table 64 shows the FTP server transfer initialization self-defining section of SMF record.

Table 64. FTP server transfer initialization self-defining section

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'0')</td>
<td>Standard SMF header</td>
<td>24</td>
<td>N/A</td>
<td>Standard SMF header; subtype is 100 (X'64)</td>
</tr>
</tbody>
</table>

Self-defining section
Table 64. FTP server transfer initialization self-defining section (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>24 (X'18')</td>
<td>SMFI19SD_TRN</td>
<td>2</td>
<td>Binary</td>
<td>Number of triplets in this record (V1R4: 5, V1R5: 6)</td>
</tr>
<tr>
<td>26 (X'1A')</td>
<td></td>
<td>2</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>28 (X'1C')</td>
<td>SMFI19IDOff</td>
<td>4</td>
<td>Binary</td>
<td>Offset to TCP/IP identification section.</td>
</tr>
<tr>
<td>32 (X'20')</td>
<td>SMFI19IDLen</td>
<td>2</td>
<td>Binary</td>
<td>Length of TCP/IP identification section.</td>
</tr>
<tr>
<td>34 (X'22')</td>
<td>SMFI19IDNum</td>
<td>2</td>
<td>Binary</td>
<td>Number of TCP/IP identification sections.</td>
</tr>
<tr>
<td>36 (X'24')</td>
<td>SMFI19S1Off</td>
<td>4</td>
<td>Binary</td>
<td>Offset to FTP server transfer initialization section</td>
</tr>
<tr>
<td>40 (X'28')</td>
<td>SMFI19S1Len</td>
<td>2</td>
<td>Binary</td>
<td>Length of FTP server transfer initialization section</td>
</tr>
<tr>
<td>42 (X'2A')</td>
<td>SMFI19S1Num</td>
<td>2</td>
<td>Binary</td>
<td>Number of FTP server transfer initialization sections</td>
</tr>
<tr>
<td>44 (X'2C')</td>
<td>SMFI19S2Off</td>
<td>4</td>
<td>Binary</td>
<td>Offset to FTP server hostname section</td>
</tr>
<tr>
<td>48 (X'30')</td>
<td>SMFI19S2Len</td>
<td>2</td>
<td>Binary</td>
<td>Length of FTP server hostname section</td>
</tr>
<tr>
<td>50 (X'32')</td>
<td>SMFI19S2Num</td>
<td>2</td>
<td>Binary</td>
<td>Number of FTP server hostname sections</td>
</tr>
<tr>
<td>52 (X'34')</td>
<td>SMFI19S3Off</td>
<td>4</td>
<td>Binary</td>
<td>Offset to FTP server first associated data set name section</td>
</tr>
<tr>
<td>56 (X'38')</td>
<td>SMFI19S3Len</td>
<td>2</td>
<td>Binary</td>
<td>Length of FTP server first associated data set name section</td>
</tr>
<tr>
<td>58 (X'3A')</td>
<td>SMFI19S3Num</td>
<td>2</td>
<td>Binary</td>
<td>Number of FTP server first associated data set name sections</td>
</tr>
<tr>
<td>60 (X'3C')</td>
<td>SMFI19S4Off</td>
<td>4</td>
<td>Binary</td>
<td>Offset to FTP server second associated data set name section</td>
</tr>
<tr>
<td>64 (X'40')</td>
<td>SMFI19S4Len</td>
<td>2</td>
<td>Binary</td>
<td>Length of FTP server second associated data set name section</td>
</tr>
<tr>
<td>66 (X'42')</td>
<td>SMFI19S4Num</td>
<td>2</td>
<td>Binary</td>
<td>Number of FTP server second associated data set name sections</td>
</tr>
<tr>
<td>68 (X'44')</td>
<td>SMFI19S5Off</td>
<td>4</td>
<td>Binary</td>
<td>Offset to FTP server security section (V1R5)</td>
</tr>
<tr>
<td>72 (X'48')</td>
<td>SMFI19S5Len</td>
<td>2</td>
<td>Binary</td>
<td>Length of FTP server security section (V1R5)</td>
</tr>
<tr>
<td>74 (X'4A')</td>
<td>SMFI19S5Num</td>
<td>2</td>
<td>Binary</td>
<td>Number of FTP server security sections (V1R5)</td>
</tr>
</tbody>
</table>

See Table 63 on page 558 for the contents of the TCP/IP stack identification section.

Table 65 on page 561 shows the FTP server transfer initialization record section (located physically after the TCP/IP identification section in the record). This section is slightly different from the one in the transfer completion record and the field names are therefore different from the completion record. The mapping of this record section is in EZANMFTA (assembler macro) for assembler code and in EZANMFTC (a C header) for C code.
Table 65. FTP server transfer initialization record section

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
</table>
| 0      | SMF119FT_FSIOPer    | 1      | FTP Operation according to SMF77 subtype classification (this is really redundant information, the same information can be found in SMF119FT_FSICmd).  
  - X'01': Append  
  - X'02': Delete  
  - X'03': Rename  
  - X'04': Retrieve  
  - X'05': Store  
  - X'06': Store Unique |
| 1      | SMF119FT_FSIActPas  | 1      | Passive or active mode data connection:  
  - X'00' active using default ip and port  
  - X'01' active using PORT  
  - X'02' active using EPRT  
  - X'03' passive using PASV  
  - X'04' passive using EPSV |
| 2      |                     | 2      | Reserved                                                     |
| 4      | SMF119FT_FSICmd     | 4      | FTP command (according to RFC 959+; see Appendix E, “Related protocol specifications,” on page 743 for information about accessing RFCs) |
| 8      | SMF119FT_FSIFileType| 4      | File type (SEQ, JES, or SQL)                                 |
| 12     | SMF119FT_FSIDRIP    | 16     | Remote IP address (data connection)                          |
| 28     | SMF119FT_FSIDLIP    | 16     | Local IP address (data connection)                           |
| 44     | SMF119FT_FSIDRPort  | 2      | Remote port number (data connection)                         |
| 46     | SMF119FT_FSIDLPort  | 2      | Local port number (data connection - server)                 |
| 48     | SMF119FT_FSICRIP    | 16     | Remote IP address (control connection)                       |
| 64     | SMF119FT_FSICLIP    | 16     | Local IP address (control connection)                        |
| 80     | SMF119FT_FSICRPort  | 2      | Remote port number (control connection - client)             |
| 82     | SMF119FT_FSICLPort  | 2      | Local port number (control connection - server)              |
| 84     | SMF119FT_FSISUser   | 8      | Client user ID on server                                     |
| 92     | SMF119FT_FSIFileType| 1      | Data type  
  - A: ASCII  
  - E: EBCDIC  
  - I: Image  
  - B: Double-byte  
  - U: UCS-2 |
| 93     | SMF119FT_FSMMode    | 1      | Transmission mode  
  - B: Block  
  - C: Compressed  
  - S: Stream |
Table 65. FTP server transfer initialization record section (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>94</td>
<td>SMF119FT_FSIStruct</td>
<td>1</td>
<td>Data structure</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• F: File</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• R: Record</td>
</tr>
<tr>
<td>95</td>
<td>SMF119FT_FSIDsType</td>
<td>1</td>
<td>Data set type</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• S: SEQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• P: PDS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• H: z/OS UNIX file</td>
</tr>
<tr>
<td>96</td>
<td>SMF119FT_FSIStartTime</td>
<td>4</td>
<td>Data connection start time, formatted in 1/100 seconds since midnight [using Coordinated Universal Time (UTC)]</td>
</tr>
<tr>
<td>100</td>
<td>SMF119FT_FSIDSDate</td>
<td>4</td>
<td>Data connection start date (format: 0cyydddF). If the start date is not available, the value specified is X'0000000F'.</td>
</tr>
<tr>
<td>104</td>
<td>SMF119FT_FSIControlSTime</td>
<td>4</td>
<td>Control connection start time in 1/100 seconds since midnight [using Coordinated Universal Time (UTC)] (FTP session start time)</td>
</tr>
<tr>
<td>108</td>
<td>SMF119FT_FSIControlSDate</td>
<td>4</td>
<td>Control connection start date (format: 0cyydddF). If the end date is not available, the value specified is X'0000000F' (FTP sessions start date)</td>
</tr>
<tr>
<td>112</td>
<td>SMF119FT_FSIM1</td>
<td>8</td>
<td>PDS Member name</td>
</tr>
<tr>
<td>120</td>
<td>SMF119FT_FSIM2</td>
<td>8</td>
<td>Second PDS member name (if rename operation)</td>
</tr>
<tr>
<td>128</td>
<td>SMF119FT_FSIConnID</td>
<td>4</td>
<td>TCP connection ID of FTP control connection (z/OS version V1R8 and later)</td>
</tr>
<tr>
<td>132</td>
<td>SMF119FT_FSIDConnID</td>
<td>4</td>
<td>TCP connection ID of FTP data connection, or zero (z/OS version V1R8 and later)</td>
</tr>
<tr>
<td>136</td>
<td>SMF119FT_FSIsessionID</td>
<td>15</td>
<td>FTP activity logging session ID (z/OS version V1R8 and later)</td>
</tr>
<tr>
<td>151</td>
<td>Reserved</td>
<td>1</td>
<td>Reserved (z/OS version V1R8 and later)</td>
</tr>
</tbody>
</table>

Table 66 shows the FTP server hostname section, physically located after the FTP server transfer initialization section. This section is optional and is identical to the one present in the transfer completion record, and is present only if a gethostbyaddr operation was performed for the Local IP address.

Table 66. FTP server hostname section

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SMF119FT_Hostname</td>
<td>n</td>
<td>Hostname</td>
</tr>
</tbody>
</table>

Table 67 on page 563 shows the FTP server MVS data set name section, physically located after the FTP server hostname section (if present) or the FTP server transfer initialization section. This section represents the MVS data set names associated with the file transfer and is identical to the one present in the completion record. A second instance of the section is included for Rename File Transfer operations.
Table 67. FTP server MVS data set name section

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SMF119FT_MVSDataSet</td>
<td>44</td>
<td>MVS Data set name</td>
</tr>
</tbody>
</table>

Table 68 shows the FTP server z/OS UNIX file name section, physically located after the FTP server MVS data set name section. It is identical to the one present in the completion record. One or two names might be included in this section.

Table 68. FTP server z/OS UNIX file name section

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SMF119FT_HFSLen1</td>
<td>2</td>
<td>Length of first z/OS UNIX file name</td>
</tr>
<tr>
<td>2</td>
<td>SMF119FT_HFSName1</td>
<td>n</td>
<td>z/OS UNIX file name</td>
</tr>
<tr>
<td>2+n</td>
<td>SMF119FT_HFSLen2</td>
<td>2</td>
<td>Length of second z/OS UNIX file name (0 if only one z/OS UNIX file name is being reported)</td>
</tr>
<tr>
<td>4+n</td>
<td>SMF119FT_HFSName2</td>
<td>m</td>
<td>z/OS UNIX file name</td>
</tr>
</tbody>
</table>

Table 69 displays the FTP server security section.

Table 69. FTP server security section

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'0')</td>
<td>SMF119FT_FSMechanism</td>
<td>1</td>
<td>EBCDIC</td>
<td>Protection Mechanism</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• N: None</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• T: TLS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• G: GSSAPI</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• A: AT-TLS</td>
</tr>
<tr>
<td>1 (X'1')</td>
<td>SMF119FT_FSCProtect</td>
<td>1</td>
<td>EBCDIC</td>
<td>Control Connection Protection Level</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• N: None</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• C: Clear</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• S: Safe</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• P: Private</td>
</tr>
<tr>
<td>2 (X'2')</td>
<td>SMF119FT_FSDProtect</td>
<td>1</td>
<td>EBCDIC</td>
<td>Data Connection Protection Level</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• N: None</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• C: Clear</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• S: Safe</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• P: Private</td>
</tr>
<tr>
<td>3 (X'3')</td>
<td>SMF119FT_FSLoginMech</td>
<td>1</td>
<td>EBCDIC</td>
<td>Login Method</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• P: Password</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• C: Certificate</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• T: Kerberos ticket</td>
</tr>
</tbody>
</table>
Table 69. FTP server security section (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 (X'4')</td>
<td>SMF119FT_FSProtoLevel</td>
<td>8</td>
<td>EBCDIC</td>
<td>Protocol level (present only if Protocol Mechanism is TLS or AT-TLS).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• SSLV2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• SSLV3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• TLSV1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• TLSV1.1</td>
</tr>
<tr>
<td>12 (X'C')</td>
<td>SMF119FT_FSCipherSpec</td>
<td>20</td>
<td>EBCDIC</td>
<td>Cipher Specification (present only if Protocol Mechanism is TLS or AT-TLS).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Possible values when Protocol Level is SSLV2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• RC4 US</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• RC4 Export</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• RC2 US</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• RC2 Export</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• DES 56-bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Triple-DES US</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Possible values when Protocol Level is SSLV3, TLSV1, or TLSV1.1:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• SSL_NULL_MD5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• SSL_NULL_SHA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• SSL_RC4_MD5_EX</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• SSL_RC4_MD5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• SSL_RC4_SHA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• SSL_RC2_MD5_EX</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• SSL_DES_SHA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• SSL_3DES_SHA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• SSL_AES_128_SHA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• SSL_AES_256_SHA</td>
</tr>
<tr>
<td>32 (X'20')</td>
<td>SMF119FT_FSProtobufSize</td>
<td>4</td>
<td>Binary</td>
<td>Negotiated protection buffer size</td>
</tr>
<tr>
<td>36 (X'24')</td>
<td>SMF119FT_FSCipher</td>
<td>2</td>
<td>EBCDIC</td>
<td>Hexadecimal value of Cipher Specification (present only if Protocol Mechanism is TLS or AT-TLS).</td>
</tr>
<tr>
<td>38 (X'26')</td>
<td>SMF119FT_FSfips140</td>
<td>1</td>
<td>Binary</td>
<td>FIPS 140 Status</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• X'00': FIPS 140 off</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• X'01': FIPS 140 on</td>
</tr>
</tbody>
</table>
### SYSTCPMSM record formats: FTP server session record

Table 70 describes the FTP server session record.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'0')</td>
<td>Standard SMF header</td>
<td>24</td>
<td>N/A</td>
<td>Standard SMF header; subtype is 104 (X'68')</td>
</tr>
</tbody>
</table>

**Self-defining section**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>24 (X'18')</td>
<td>SMF119SD_TRN</td>
<td>2</td>
<td>Binary</td>
<td>Number of triplets in this record</td>
</tr>
<tr>
<td>26 (X'1A')</td>
<td>SMF119IDOff</td>
<td>4</td>
<td>Binary</td>
<td>Offset to TCP/IP identification section. *</td>
</tr>
<tr>
<td>32 (X'20')</td>
<td>SMF119IDLen</td>
<td>2</td>
<td>Binary</td>
<td>Length of TCP/IP identification section. *</td>
</tr>
<tr>
<td>34 (X'22')</td>
<td>SMF119IDNum</td>
<td>2</td>
<td>Binary</td>
<td>Number of TCP/IP identification sections. *</td>
</tr>
<tr>
<td>36 (X'24')</td>
<td>SMF119S1Off</td>
<td>4</td>
<td>Binary</td>
<td>Offset to FTP server session section</td>
</tr>
<tr>
<td>40 (X'28')</td>
<td>SMF119S1Len</td>
<td>2</td>
<td>Binary</td>
<td>Length of FTP server session section</td>
</tr>
<tr>
<td>42 (X'2A')</td>
<td>SMF119S1Num</td>
<td>2</td>
<td>Binary</td>
<td>Number of FTP server session sections</td>
</tr>
<tr>
<td>44 (X'2C')</td>
<td>SMF119S2Off</td>
<td>4</td>
<td>Binary</td>
<td>Offset to FTP server security section</td>
</tr>
<tr>
<td>48 (X'30')</td>
<td>SMF119S2Len</td>
<td>2</td>
<td>Binary</td>
<td>Length of FTP server security section</td>
</tr>
<tr>
<td>50 (X'32')</td>
<td>SMF119S2Num</td>
<td>2</td>
<td>Binary</td>
<td>Number of FTP server security sections</td>
</tr>
</tbody>
</table>

* See Table 63 on page 558 for the contents of the TCP/IP identification section.

Table 71 describes the server session section.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'0')</td>
<td>SMF119FT_FSRRIP</td>
<td>16</td>
<td>Binary</td>
<td>Remote IP address</td>
</tr>
<tr>
<td>16 (X'10')</td>
<td>SMF119FT_FSRLIP</td>
<td>16</td>
<td>Binary</td>
<td>Local IP address</td>
</tr>
<tr>
<td>32 (X'20')</td>
<td>SMF119FT_FSRRPort</td>
<td>2</td>
<td>Binary</td>
<td>Remote port number (client)</td>
</tr>
<tr>
<td>34 (X'22')</td>
<td>SMF119FT_FSRLPort</td>
<td>2</td>
<td>Binary</td>
<td>Local port number (server)</td>
</tr>
<tr>
<td>36 (X'24')</td>
<td>SMF119FT_FSRUserID</td>
<td>8</td>
<td>Binary</td>
<td>Client user ID</td>
</tr>
</tbody>
</table>
Table 71. Server session section (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>44 (X'2C')</td>
<td>SMF119FT_FSRReason</td>
<td>4</td>
<td>Binary</td>
<td>Session end reason.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• X'00': Normal session end; QUIT or REIN command received</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• X'01': Security authentication or negotiation failed; incorrect specification of security keywords; possible security handshake deadlock</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• X'02': Control connection socket error; network error</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• X'03': Control connection closed prematurely</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• X'04': Sequence received on control connection was not valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This field is valid only when the value of the SMF119FT_FSREvent field is T.</td>
</tr>
<tr>
<td>48 (X'30')</td>
<td>SMF119FT_FSREvent</td>
<td>1</td>
<td>Binary</td>
<td>Session event</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• I: Session start; client is logged into server</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• T: Session has ended</td>
</tr>
<tr>
<td>49 (X'31')</td>
<td>Reserved</td>
<td>3</td>
<td>Binary</td>
<td>Reserved</td>
</tr>
<tr>
<td>52 (X'34')</td>
<td>SMF119FT_FSRTime</td>
<td>4</td>
<td>Binary</td>
<td>Session start time in hundreths of a second since midnight [using Coordinated Universal Time (UTC)].</td>
</tr>
<tr>
<td>56 (X'38')</td>
<td>SMF119FT_FSRSDate</td>
<td>4</td>
<td>Binary</td>
<td>Session start date (format: 0cyydddF). If the date is not available, the value specified is X'0000000F'. (FTP sessions start date).</td>
</tr>
<tr>
<td>60 (X'3C')</td>
<td>SMF119FT_FSRETime</td>
<td>4</td>
<td>Binary</td>
<td>Session end time in hundreths of a second since midnight [using Coordinated Universal Time (UTC)]. This field is defined only when the value of the SMF119FT_FSREvent field is T.</td>
</tr>
<tr>
<td>64 (X'40')</td>
<td>SMF119FT_FSREDate</td>
<td>4</td>
<td>Binary</td>
<td>Session end date (format: 0cyydddF). If the date is not available, the value specified is X'0000000F'. This field is defined only when the value of the SMF119FT_FSREvent field is T.</td>
</tr>
<tr>
<td>68 (X'44')</td>
<td>SMF119FT_FSRCConnID</td>
<td>4</td>
<td>Binary</td>
<td>TCP connection ID of FTP control connection</td>
</tr>
<tr>
<td>72 (X'48')</td>
<td>SMF119FT_FSRSessionID</td>
<td>15</td>
<td>EBCDIC</td>
<td>FTP activity logging session ID</td>
</tr>
<tr>
<td>87 (X'57')</td>
<td>Reserved</td>
<td>1</td>
<td>N/A</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
Table 72 describes the FTP server security section.

**Table 72. FTP server security section**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
</table>
| 0 (X'0') | SMF119FT_FSNMechanism | 1 | EBCDIC | Possible values are:  
|        |                     |        |        | • N: None  
|        |                     |        |        | • T: TLS  
|        |                     |        |        | • G: GSSAPI  
|        |                     |        |        | • A: AT-TLS  |
| 1 (X'1') | SMF119FT_FSNCProtect | 1 | EBCDIC | Control Connection Protection Level. Possible values are:  
|        |                     |        |        | • N: None  
|        |                     |        |        | • C: Clear  
|        |                     |        |        | • S: Safe  
|        |                     |        |        | • P: Private  |
| 2 (X'2') | SMF119FT_FSNDProtect | 1 | EBCDIC | Data Connection Protection Level. Possible values are:  
|        |                     |        |        | • N: None  
|        |                     |        |        | • C: Clear  
|        |                     |        |        | • S: Safe  
|        |                     |        |        | • P: Private  |
| 3 (X'3') | SMF119FT_FSNLoginMech | 1 | EBCDIC | Login method:  
|        |                     |        |        | • P: Password  
|        |                     |        |        | • C: Certificate  
|        |                     |        |        | • T: Kerberos ticket  |
| 4 (X'4') | SMF119FT_FSNProtoLevel | 8 | EBCDIC | Protocol level (present only if Protocol Mechanism is TLS or AT-TLS). Possible values are:  
|        |                     |        |        | • SSLV2  
|        |                     |        |        | • SSLV3  
|        |                     |        |        | • TLSV1  
|        |                     |        |        | • TLSV1.1  |
### Table 72. FTP server security section (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>12 (X'C')</td>
<td>SMF119FT_FSNCipherSpec</td>
<td>20</td>
<td>EBCDIC</td>
<td>Cipher Specification (present only if the value of Protocol Mechanism is TLS or AT-TLS). Possible values when Protocol Level is SSLV2: • RC4 US • RC4 Export • RC2 US • RC2 Export • DES 56-Bit • Triple DES US Possible values when Protocol Level is SSLV3, TLSV1, or TLSV1.1: • SSL_NULL_MD5 • SSL_NULL_SHA • SSL_RC4_MD5 • SSL_RC4_SHA • SSL_RC2_MD5 • SSL_3DES_SHA • SSL_AES_128_SHA • SSL_AES_256_SHA</td>
</tr>
<tr>
<td>32 (X'20')</td>
<td>SMF119FT_FSNProtoBufSize</td>
<td>4</td>
<td>Binary</td>
<td>Negotiated protection buffer size</td>
</tr>
<tr>
<td>36 (X'24')</td>
<td>SMF119FT_FSNCipher</td>
<td>2</td>
<td>EBCDIC</td>
<td>Hexadecimal value of Cipher Specification (present only if Protocol Mechanism is TLS or AT-TLS).</td>
</tr>
<tr>
<td>38 (X'26')</td>
<td>SMF119FT_FSNFips140</td>
<td>1</td>
<td>Binary</td>
<td>FIPS 140 Status • X'00': FIPS 140 off • X'01': FIPS 140 on</td>
</tr>
</tbody>
</table>

### SYSTCPSSM record formats: FTP client transfer initialization record

The following table shows the FTP client transfer initialization self-defining section of the SMF record.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (x'0')</td>
<td>Standard SMF header</td>
<td>24</td>
<td>N/A</td>
<td>Standard SMF header; subtype is 101 (X'65')</td>
</tr>
</tbody>
</table>

**Self-defining section**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>24 (x'18')</td>
<td>SMF119SD_TRN</td>
<td>2</td>
<td>Binary</td>
<td>Number of triplets in this record (V1R4: 4, V1R5: 5, V1R8: 6)</td>
</tr>
<tr>
<td>26 (x'1A')</td>
<td></td>
<td>2</td>
<td></td>
<td>Reserved</td>
</tr>
</tbody>
</table>
Table 73 on page 570 describes the FTP client transfer initialization record section, which is physically located after the TCP/IP identification section. This section is slightly different from the one in the transfer completion record and the field names are therefore different from the completion record. The mapping of this record section is in EZANMFTA (assembler macro) for assembler code and in EZANMFTC (a C header) for C code.
### Table 73. FTP client transfer initialization record section

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SMF119FT_FCICmd</td>
<td>4</td>
<td>FTP subcommand (according to RFC 959; see Appendix E, “Related protocol specifications,” on page 743 for information about accessing RFCs)</td>
</tr>
<tr>
<td>4</td>
<td>SMF119FT_FCIFType</td>
<td>4</td>
<td>Local file type (SEQ or SQL)</td>
</tr>
<tr>
<td>8</td>
<td>SMF119FT_FCIDRIP</td>
<td>16</td>
<td>Remote IP address (data connection)</td>
</tr>
<tr>
<td>24</td>
<td>SMF119FT_FCIDLIP</td>
<td>16</td>
<td>Local IP address (data connection)</td>
</tr>
<tr>
<td>40</td>
<td>SMF119FT_FCIDRPort</td>
<td>2</td>
<td>Remote port number (data connection)</td>
</tr>
<tr>
<td>42</td>
<td>SMF119FT_FCIDLPort</td>
<td>2</td>
<td>Local port number (data connection)</td>
</tr>
<tr>
<td>44</td>
<td>SMF119FT_FCICRIP</td>
<td>16</td>
<td>Remote IP address (control connection)</td>
</tr>
<tr>
<td>60</td>
<td>SMF119FT_FCICLIP</td>
<td>16</td>
<td>Local IP address (control connection)</td>
</tr>
<tr>
<td>76</td>
<td>SMF119FT_FCICRPort</td>
<td>2</td>
<td>Remote port number (control connection)</td>
</tr>
<tr>
<td>78</td>
<td>SMF119FT_FCICLPort</td>
<td>2</td>
<td>Local port number (control connection)</td>
</tr>
<tr>
<td>80</td>
<td>SMF119FT_FCIRUser</td>
<td>8</td>
<td>User ID (login name) on server</td>
</tr>
<tr>
<td>88</td>
<td>SMF119FT_FCILUser</td>
<td>8</td>
<td>Local User ID</td>
</tr>
<tr>
<td>96</td>
<td>SMF119FT_FCIType</td>
<td>1</td>
<td>Data format</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• A: ASCII</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• E: EBCDIC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• I: Image</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• B: Double-byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• U: UCS-2</td>
</tr>
<tr>
<td>97</td>
<td>SMF119FT_FCIMode</td>
<td>1</td>
<td>Transfer mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• B: Block</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• C: Compressed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• S: Stream</td>
</tr>
<tr>
<td>98</td>
<td>SMF119FT_FCIStruct</td>
<td>1</td>
<td>Structure</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• F: File</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• R: Record</td>
</tr>
<tr>
<td>99</td>
<td>SMF119FT_FCIDSType</td>
<td>1</td>
<td>Data set type</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• S: SEQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• P: PDS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• H: z/OS UNIX file system</td>
</tr>
<tr>
<td>100</td>
<td>SMF119FT_FCSTime</td>
<td>4</td>
<td>Start time of data connection, in a hundredth of a second, since midnight [using Coordinated Universal Time (UTC)]</td>
</tr>
<tr>
<td>104</td>
<td>SMF119FT_FCSIDate</td>
<td>4</td>
<td>Start date of data connection (format: 0cgadddf). If the start date is not available, the value specified is X’0000000F’.</td>
</tr>
<tr>
<td>108</td>
<td>SMF119FT_FCICSTime</td>
<td>4</td>
<td>Start time of control connection, in a hundredth of a second, since midnight [using Coordinated Universal Time (UTC)]. FTP session start time.</td>
</tr>
</tbody>
</table>
Table 73. FTP client transfer initialization record section (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>112</td>
<td>SMF119FT_FCICSSDate</td>
<td>4</td>
<td>Start date of the control connection (format 0cyydddF). If the start date is not available, the value specified is X'0000000F'. FTP session start date.</td>
</tr>
<tr>
<td>116</td>
<td>SMF119FT_FCIM1</td>
<td>8</td>
<td>PDS member name</td>
</tr>
<tr>
<td>124</td>
<td>SMF119FT_FCIActPas</td>
<td>1</td>
<td>Passive or active mode data connection. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• X'00': Active using default IP and port</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• X'01': Active using PORT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• X'03': Passive using PASV</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• X'04': Passive using EPSV</td>
</tr>
<tr>
<td>125</td>
<td>Reserved</td>
<td>3</td>
<td>Reserved</td>
</tr>
<tr>
<td>128</td>
<td>SMF119FT_FCICConnID</td>
<td>4</td>
<td>TCP connection ID of FTP control connection (z/OS version V1R8 and later)</td>
</tr>
<tr>
<td>132</td>
<td>SMF119FT_FCIDConnID</td>
<td>4</td>
<td>TCP connection ID of FTP data connection (z/OS version V1R8 and later)</td>
</tr>
</tbody>
</table>

Table 74 describes the FTP client associated data set name section, which is physically located after the FTP client transfer initialization section. This section is identical to the one present in the transfer completion record.

Table 74. FTP client associated data set name section

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SMF119FT_FCFileName</td>
<td>n</td>
<td>MVS data set or z/OS UNIX file name associated with a file transfer operation. Use the data set type field information in the FTP client transfer initialization section to determine the type of file name that is represented by this value.</td>
</tr>
</tbody>
</table>

Table 75 describes the FTP client SOCKS section, which is present only if the connection passes through a SOCKS server.

Table 75. FTP client SOCKS section

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SMF119FT_FCCIP</td>
<td>16</td>
<td>SOCKS server IP address</td>
</tr>
<tr>
<td>16</td>
<td>SMF119FT_FCCPort</td>
<td>2</td>
<td>SOCKS Server port number</td>
</tr>
<tr>
<td>18</td>
<td>SMF119FT_FCCProt</td>
<td>1</td>
<td>SOCKS protocol version. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• X'01': SOCKS Version 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• X'02': SOCKS Version 5</td>
</tr>
</tbody>
</table>
Table 76 describes the FTP client security section.

### Table 76. FTP client security section

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
</table>
| 0 (X'0') | SMF119FT_FCMechanism | 1      | EBCDIC | Protection Mechanism. Possible values are:  
  - N: None  
  - T: TLS  
  - G: GSSAPI  
  - A: AT-TLS |
| 1 (X'1') | SMF119FT_FCCProtect | 1      | EBCDIC | Control Connection Protection Level. Possible values are:  
  - N: None  
  - C: Clear  
  - S: Safe  
  - P: Private |
| 2 (X'2') | SMF119FT_FCDProtect | 1      | EBCDIC | Data Connection Protection Level. Possible values are:  
  - N: None  
  - C: Clear  
  - S: Safe  
  - P: Private |
| 3 (X'3') | SMF119FT_FCLLoginMech | 1      | EBCDIC | Login Method. Possible values are:  
  - U: Undefined; the login method is not defined for the client.  
  - P: Password  
  - C: Certificate |
| 4 (X'4') | SMF119FT_FCProtoLevel | 8      | EBCDIC | Protocol level (present only if Protocol Mechanism is TLS or AT-TLS). Possible values are:  
  - SSLV2  
  - SSLV3  
  - TLSV1  
  - TLSV1.1 |
Table 76. FTP client security section (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
</table>
| 12 (X'C') | SMF119FT_FCCipherSpec   | 20     | EBCDIC | Cipher Specification (present only if Protocol Mechanism is TLS or AT-TLS). Possible values when Protocol Level is SSLv2 are:  
  - RC4 US  
  - RC4 Export  
  - RC2 US  
  - RC2 Export  
  - DES 56-bit  
  - Triple-DES US  
  Possible values when Protocol Level is SSLv3, TLSv1, or TLSv1.1 are:  
  - SSL_NULL_MD5  
  - SSL_NULL_SHA  
  - SSL_RC4_MD5_EX  
  - SSL_RC4_MD5  
  - SSL_RC4_SHA  
  - SSL_RC2_MD5_EX  
  - SSL_DES_SHA  
  - SSL_3DES_SHA  
  - SSL_AES_128_SHA  
  - SSL_AES_256_SHA |
| 32 (X'20') | SMF119FT_FCProtBuffSize | 4      | Binary | Negotiated protection buffer size                    |
| 36 (X'24') | SMF119FT_FCCipher       | 2      | EBCDIC | Hexadecimal value of the Cipher Specification (present only if the Protocol Mechanism value is TLS or AT-TLS). |
| 38 (X'26') | SMF119FT_FCFips140     | 1      | Binary | FIPS 140 Status  
  - X'00': FIPS 140 off  
  - X'01': FIPS 140 on |

Table 77 describes the FTP client user name section.

Table 77. FTP client user name section

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(X'0')</td>
<td>SMF119FT_FCUserID</td>
<td>n</td>
<td>EBCDIC</td>
<td>User name or user ID used to log into the FTP server.</td>
</tr>
</tbody>
</table>

SYSTCPMSM record formats: FTP client login failure record

Table 78 on page 574 describes the FTP client login failure self-defining section of the SMF record.
Table 78. FTP client login failure self-defining section

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (x'0')</td>
<td>Standard SMF header</td>
<td>24</td>
<td>N/A</td>
<td>Standard SMF header; subtype is 102 (X'66')</td>
</tr>
</tbody>
</table>

**Self-defining section**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>24 (x'18')</td>
<td>SMF119SD_TRN</td>
<td>2</td>
<td>Binary</td>
<td>Number of triplets in this record</td>
</tr>
<tr>
<td>26 (x'1A')</td>
<td>Reserved</td>
<td>2</td>
<td>N/A</td>
<td>Reserved</td>
</tr>
<tr>
<td>28 (x'1C')</td>
<td>SMF119IDOOff</td>
<td>4</td>
<td>Binary</td>
<td>Offset to TCP/IP identification section *</td>
</tr>
<tr>
<td>32 (x'20')</td>
<td>SMF119IDLen</td>
<td>2</td>
<td>Binary</td>
<td>Length of TCP/IP identification section *</td>
</tr>
<tr>
<td>34 (x'22')</td>
<td>SMF119IDNum</td>
<td>2</td>
<td>Binary</td>
<td>Number of TCP/IP identification sections *</td>
</tr>
<tr>
<td>36 (x'24')</td>
<td>SMF119S1Off</td>
<td>4</td>
<td>Binary</td>
<td>Offset to FTP client login failure section</td>
</tr>
<tr>
<td>40 (x'28')</td>
<td>SMF119S1Len</td>
<td>2</td>
<td>Binary</td>
<td>Length of FTP client login failure section</td>
</tr>
<tr>
<td>42 (x'2A')</td>
<td>SMF119S1Num</td>
<td>2</td>
<td>Binary</td>
<td>Number of FTP client login failure sections</td>
</tr>
<tr>
<td>44 (x'2C')</td>
<td>SMF119S2Off</td>
<td>4</td>
<td>Binary</td>
<td>Offset to FTP client SOCKS section</td>
</tr>
<tr>
<td>48 (x'30')</td>
<td>SMF119S2Len</td>
<td>2</td>
<td>Binary</td>
<td>Length of FTP client SOCKS section</td>
</tr>
<tr>
<td>50 (x'32')</td>
<td>SMF119S2Num</td>
<td>2</td>
<td>Binary</td>
<td>Number of FTP client SOCKS sections</td>
</tr>
<tr>
<td>52 (x'34')</td>
<td>SMF119S3Off</td>
<td>4</td>
<td>Binary</td>
<td>Offset to FTP client Security section</td>
</tr>
<tr>
<td>56 (x'38')</td>
<td>SMF119S3Len</td>
<td>2</td>
<td>Binary</td>
<td>Length of FTP client Security section</td>
</tr>
<tr>
<td>58 (x'3A')</td>
<td>SMF119S3Num</td>
<td>2</td>
<td>Binary</td>
<td>Number of FTP client Security sections</td>
</tr>
<tr>
<td>60 (x'3C')</td>
<td>SMF119S4Off</td>
<td>4</td>
<td>Binary</td>
<td>Offset to FTP client user name section (z/OS version V1R8 and later)</td>
</tr>
<tr>
<td>64 (x'40')</td>
<td>SMF119S4Len</td>
<td>2</td>
<td>Binary</td>
<td>Length of FTP client user name section (z/OS version V1R8 and later)</td>
</tr>
<tr>
<td>66 (x'42')</td>
<td>SMF119S4Num</td>
<td>2</td>
<td>Binary</td>
<td>Number of FTP client user name sections (z/OS version V1R8 and later)</td>
</tr>
</tbody>
</table>

* See Table 63 on page 558 for the contents of the TCP/IP identification section.

Table 79 shows the client login failure session section, which follows the TCP/IP identification section.

Table 79. Client login failure session section

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(x'0')</td>
<td>SMF119FT_FCLRIP</td>
<td>16</td>
<td>Binary</td>
<td>Remote IP address (server)</td>
</tr>
<tr>
<td>16(x'10')</td>
<td>SMF119FT_FCLLIP</td>
<td>16</td>
<td>Binary</td>
<td>Local IP address (client)</td>
</tr>
<tr>
<td>32(x'20')</td>
<td>SMF119FT_FCLRPort</td>
<td>2</td>
<td>Binary</td>
<td>Remote port number (server)</td>
</tr>
<tr>
<td>34(x'22')</td>
<td>SMF119FT_FCLLPort</td>
<td>2</td>
<td>Binary</td>
<td>Local port number (client)</td>
</tr>
</tbody>
</table>
Table 79. Client login failure session section (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>36(x’24’)</td>
<td>SMF119FT_FCLUserID</td>
<td>8</td>
<td>EBCDIC</td>
<td>Local user ID</td>
</tr>
<tr>
<td>44(x’2C’)</td>
<td>SMF119FT_FCLReason</td>
<td>4</td>
<td>Binary</td>
<td>Login failure reason. The reason is a Client Error Code as documented in FTP Client Error Codes in z/OS Communications Server: IP User's Guide and Commands. Following are the client error codes most likely for login failures.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X’0A’ FTP_SESSION_ERROR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Socket, send, or receive error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X’0B’ FTP_LOGIN_FAILED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>User ID, password, or account information is not valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X’11’ FTP_AUTHENTICATION</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Security authentication or negotiation failed; incorrect specification of security keywords.</td>
</tr>
<tr>
<td>48 (x’30’)</td>
<td>SMF119FT_FCLConnID</td>
<td>4</td>
<td>Binary</td>
<td>TCP connection ID of FTP control connection</td>
</tr>
</tbody>
</table>

Table 80 describes the FTP client SOCKS section, which is present only if the connection passes through a SOCKS server.

Table 80. FTP client SOCKS section

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SMF119FT_FCLIP</td>
<td>16</td>
<td>SOCKS server IP address</td>
</tr>
<tr>
<td>16</td>
<td>SMF119FT_FCLPort</td>
<td>2</td>
<td>SOCKS server port number</td>
</tr>
<tr>
<td>18</td>
<td>SMF119FT_FCLProt</td>
<td>1</td>
<td>SOCKS protocol version. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X’01’: SOCKS Version 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X’02’: SOCKS Version 5</td>
</tr>
</tbody>
</table>

Table 81 defines the FTP client login failure security section.

Table 81. FTP client login failure security section

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X’0’)</td>
<td>SMF119FT_FCLMechanism</td>
<td>1</td>
<td>EBCDIC</td>
<td>Protection Mechanism. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>N: None</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>T: TLS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>G: GSSAPI</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A: AT-TLS</td>
</tr>
<tr>
<td>Offset</td>
<td>Name</td>
<td>Length</td>
<td>Format</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------</td>
<td>--------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1 (X'1')</td>
<td>SMF119FT_FCLCProtect</td>
<td>1</td>
<td>EBCDIC</td>
<td>Control Connection Protection Level. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• N: None</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• C: Clear</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• S: Safe</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• P: Private</td>
</tr>
<tr>
<td>2 (X'2')</td>
<td>SMF119FT_FCLDProtect</td>
<td>1</td>
<td>EBCDIC</td>
<td>Data Connection Protection Level. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• N: None</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• C: Clear</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• S: Safe</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• P: Private</td>
</tr>
<tr>
<td>3 (X'3')</td>
<td>SMF119FT_FCLLoginMech</td>
<td>1</td>
<td>EBCDIC</td>
<td>Login Method. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• U: Undefined; the login method is not defined for the client.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• P: Password</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• C: Certificate</td>
</tr>
<tr>
<td>4 (X'4')</td>
<td>SMF119FT_FCLProtoLevel</td>
<td>8</td>
<td>EBCDIC</td>
<td>Protocol level (present only if Protocol Mechanism is TLS or AT-TLS). Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• SSLV2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• SSLV3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• TLSV1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• TLSV1.1</td>
</tr>
</tbody>
</table>
Table 81. FTP client login failure security section (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
</table>
| 12 (X'C') | SMF119FT_FCLCipherSpec     | 20     | EBCDIC | Cipher specification (present only if Protocol Mechanism is TLS or AT-TLS). Possible values when Protocol Level is SSLV2 are:  
  - RC4 US  
  - RC4 Export  
  - RC2 US  
  - RC2 Export  
  - DES 56-bit  
  - Triple-DES US  
  Possible values when Protocol Level is SSLV3, TLSV1, or TLSV1.1 are:  
  - SSL_NULL_MD5  
  - SSL_NULL_SHA  
  - SSL_RC4_MD5  
  - SSL_RC4_MD5_EX  
  - SSL_RC4_SHA  
  - SSL_RC2_MD5_EX  
  - SSL_DES_SHA  
  - SSL_3DES_SHA  
  - SSL_AES_128_SHA  
  - SSL_AES_256_SHA |
| 32 (X'20') | SMF119FT_FCLProtBuffSize        | 4      | Binary | Negotiated protection buffer size |
| 36(x'24') | SMF119FT_FCLCipher          | 2      | EBCDIC | Hexadecimal value of Cipher Specification (present only if Protocol Mechanism is TLS or AT-TLS). |
| 38 (X'26') | SMF119FT_FCLFips140        | 1      | Binary | FIPS 140 Status  
  - X'00': FIPS 140 off  
  - X'01': FIPS 140 on |

Table 82 shows the FTP client user name section.

Table 82. FTP client user name section

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(X'0')</td>
<td>SMF119FT_FCLUserID</td>
<td>n</td>
<td>EBCDIC</td>
<td>User name or user ID used to log on to the FTP server</td>
</tr>
</tbody>
</table>
SYSTCPSM record formats: FTP client session record

Table 83 shows the FTP client session record self-defining section.

**Table 83. FTP client session record self-defining section**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (x'0')</td>
<td>Standard SMF header</td>
<td>24</td>
<td>N/A</td>
<td>Standard SMF header; subtype is 103 (x'67')</td>
</tr>
<tr>
<td>24 (x'18')</td>
<td>SMF119SD_TRN</td>
<td>2</td>
<td>Binary</td>
<td>Number of triplets in this record</td>
</tr>
<tr>
<td>26 (x'1A')</td>
<td>Reserved</td>
<td>2</td>
<td>N/A</td>
<td>Reserved</td>
</tr>
<tr>
<td>28 (x'1C')</td>
<td>SMF119IDOff</td>
<td>4</td>
<td>Binary</td>
<td>Offset to TCP/IP identification section *</td>
</tr>
<tr>
<td>32 (x'20')</td>
<td>SMF119IDLen</td>
<td>2</td>
<td>Binary</td>
<td>Length of TCP/IP identification section *</td>
</tr>
<tr>
<td>34 (x'22')</td>
<td>SMF119IDNum</td>
<td>2</td>
<td>Binary</td>
<td>Number of TCP/IP identification sections *</td>
</tr>
<tr>
<td>36 (x'24')</td>
<td>SMF119S1Off</td>
<td>4</td>
<td>Binary</td>
<td>Offset to FTP client session section</td>
</tr>
<tr>
<td>40 (x'28')</td>
<td>SMF119S1Len</td>
<td>2</td>
<td>Binary</td>
<td>Length of FTP client session section</td>
</tr>
<tr>
<td>42 (x'2A')</td>
<td>SMF119S1Num</td>
<td>2</td>
<td>Binary</td>
<td>Number of FTP client session sections</td>
</tr>
<tr>
<td>44 (x'2C')</td>
<td>SMF119S2Off</td>
<td>4</td>
<td>Binary</td>
<td>Offset to FTP client SOCKS section</td>
</tr>
<tr>
<td>48 (X'32')</td>
<td>SMF119S2Len</td>
<td>2</td>
<td>Binary</td>
<td>Length of FTP client SOCKS section</td>
</tr>
<tr>
<td>50 (X'34')</td>
<td>SMF119S2Num</td>
<td>2</td>
<td>Binary</td>
<td>Number of FTP client SOCKS sections</td>
</tr>
<tr>
<td>52 (X'34')</td>
<td>SMF119S3Off</td>
<td>4</td>
<td>Binary</td>
<td>Offset to FTP client security section</td>
</tr>
<tr>
<td>56 (X'38')</td>
<td>SMF119S3Len</td>
<td>2</td>
<td>Binary</td>
<td>Length of FTP client security section</td>
</tr>
<tr>
<td>58 (X'3A')</td>
<td>SMF119S3Num</td>
<td>2</td>
<td>Binary</td>
<td>Number of FTP client security sections</td>
</tr>
<tr>
<td>60 (X'3C')</td>
<td>SMF119S4Off</td>
<td>4</td>
<td>Binary</td>
<td>Offset to FTP client session user name section</td>
</tr>
<tr>
<td>64 (X'40')</td>
<td>SMF119S4Len</td>
<td>2</td>
<td>Binary</td>
<td>Length of FTP client session user name section</td>
</tr>
<tr>
<td>66 (X'42')</td>
<td>SMF119S4Num</td>
<td>2</td>
<td>Binary</td>
<td>Number of FTP client session user name section</td>
</tr>
</tbody>
</table>

*See Table 63 on page 558 for the contents of the TCP/IP identification section.*

Table 84 shows the FTP client session section.

**Table 84. FTP client session section**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'0')</td>
<td>SMF119FT_FCNRIP</td>
<td>16</td>
<td>Binary</td>
<td>Remote IP address (server)</td>
</tr>
<tr>
<td>16 (X'10')</td>
<td>SMF119FT_FCNLIP</td>
<td>16</td>
<td>Binary</td>
<td>Local IP address (client)</td>
</tr>
<tr>
<td>32 (X'20')</td>
<td>SMF119FT_FCNRPort</td>
<td>2</td>
<td>Binary</td>
<td>Remote port number (server)</td>
</tr>
<tr>
<td>34 (X'22')</td>
<td>SMF119FT_FCNLPort</td>
<td>2</td>
<td>Binary</td>
<td>Local port number (Client)</td>
</tr>
<tr>
<td>36 (X'24')</td>
<td>SMF119FT_FCNUserID</td>
<td>8</td>
<td>EBCDIC</td>
<td>Local User ID</td>
</tr>
</tbody>
</table>
Table 84. FTP client session section (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>44 (X'2C')</td>
<td>SMF119FT_FCNReason</td>
<td>4</td>
<td>Binary</td>
<td>Session end reason. The reason is a client error code as documented in FTP Client Error Codes in z/OS Communications Server: IP User’s Guide and Commands. If no error occurred, the value of this field is 0. This field is defined only when the value of the SMF119FT_FCNEvent field is T.</td>
</tr>
</tbody>
</table>
| 48 (X'30') | SMF119FT_FCNEvent | 1      | EBCDIC | • I: Session has started; client is logged into the server  
• T: Session has ended |
| 49 (X'31') | Reserved          | 3      | Binary | Reserved |
| 52 (X'34') | SMF119FT_FCNSTime | 4      | Binary | Session start time, in one hundredths of a second, since midnight [using Coordinated Universal Time (UTC)]. |
| 56 (X'38') | SMF119FT_FCNSDate | 4      | Binary | Session start date (format: 0cyydddF). If the date is not available, the value specified is X'0000000F'. |
| 60 (X'3C') | SMF119FT_FCNETime | 4      | Binary | Session end time, in one hundredths of a second, since midnight [using Coordinated Universal Time (UTC)].  
This field is defined only when the value of SMF119FT_FCNEvent is T. |
| 64 (X'40') | SMF119FT_FNEDate | 4      | Binary | Session end date (format: 0cyydddF). If the date is not available, the value specified is X'0000000F'.  
This field is defined only when the value of SMF119FT_FCNEvent is T. |
| 68 (X'44') | SMF119FT_FCNConnID | 4 | Binary | TCP connection ID of FTP control connection |

Table 85 shows the FTP client SOCKS section, which is present only if the connection passes through a SOCKS server.

Table 85. FTP client SOCKS section

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SMF119FT_FCNIP</td>
<td>16</td>
<td>SOCKS server IP address</td>
</tr>
<tr>
<td>16</td>
<td>SMF119FT_FCNPort</td>
<td>2</td>
<td>SOCKS server port number</td>
</tr>
</tbody>
</table>
Table 85. FTP client SOCKS section (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>SMF119FT_FCNProt</td>
<td>1</td>
<td>SOCKS protocol version. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• X'01' SOCKS Version 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• X'02' SOCKS Version 5</td>
</tr>
</tbody>
</table>

Table 86. FTP client security section

Table 86. FTP client security section

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (X'0')</td>
<td>SMF119FT_FCNMechanism</td>
<td>1</td>
<td>EBCDIC</td>
<td>Protection Mechanism. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• N: None</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• T: TLS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• G: GSSAPI</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• A: AT-TLS</td>
</tr>
<tr>
<td>1 (X'1')</td>
<td>SMF119FT_FCNCProtect</td>
<td>1</td>
<td>EBCDIC</td>
<td>Control Connection Protection Level. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• N: None</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• C: Clear</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• S: Safe</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• P: Private</td>
</tr>
<tr>
<td>2 (X'2')</td>
<td>SMF119FT_FCNDProtect</td>
<td>1</td>
<td>EBCDIC</td>
<td>Data Connection Protection Level. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• N: None</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• C: Clear</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• S: Safe</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• P: Private</td>
</tr>
<tr>
<td>3 (X'3')</td>
<td>SMF119FT_FCNLoginMech</td>
<td>1</td>
<td>EBCDIC</td>
<td>Login Method. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• P: Password</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• C: Certificate</td>
</tr>
<tr>
<td>4 (X'4')</td>
<td>SMF119FT_FCNProtoLevel</td>
<td>8</td>
<td>EBCDIC</td>
<td>Protocol level (present only if Protocol Mechanism value is TLS or AT-TLS). Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• SSLV2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• SSLV3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• TLSV1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• TLSV1.1</td>
</tr>
</tbody>
</table>
Table 86. FTP client security section (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
</table>
| 12 (X'C') | SMF119FT_FCNCipherSpec | 20     | EBCDIC | Cipher specification (present only if Protocol Mechanism value is TLS or AT-TLS).
|        |                           |        |        | Possible values when Protocol Level is SSLv2 are: |
|        |                           |        |        | • RC4 US |
|        |                           |        |        | • RC4 Export |
|        |                           |        |        | • RC2 US |
|        |                           |        |        | • RC2 Export |
|        |                           |        |        | • DES 56-bit |
|        |                           |        |        | • Triple-DES US |
|        |                           |        |        | Possible values when Protocol Level value is SSLv3, TLSv1, or TLSv1.1 are: |
|        |                           |        |        | • SSL_NULL_MD5 |
|        |                           |        |        | • SSL_NULL_SHA |
|        |                           |        |        | • SSL_RC4_MD5_EX |
|        |                           |        |        | • SSL_RC4_MD5 |
|        |                           |        |        | • SSL_RC4_SHA |
|        |                           |        |        | • SSL_RC2_MD5_EX |
|        |                           |        |        | • SSL_DES_SHA |
|        |                           |        |        | • SSL_3DES_SHA |
|        |                           |        |        | • SSL_AES_128_SHA |
|        |                           |        |        | • SSL_AES_256_SHA |

| 32 (X'20') | SMF119FT_FCNPctBufSize | 4      | Binary | Negotiated protection buffer size |
| 36 (X'24') | SMF119FT_FCNCipher     | 2      | EBCDIC | Hexadecimal value of Cipher Specification (present only if Protocol Mechanism is TLS or AT-TLS). |
| 38 (X'26') | SMF119FT_FCNFips140    | 1      | Binary | FIPS 140 Status |
|           |                           |        |        | • X'00': FIPS 140 off |
|           |                           |        |        | • X'01': FIPS 140 on |

Table 87 shows the FTP client session user name section.

Table 87. FTP client session user name section

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(x'0')</td>
<td>SMF119FT_FCNUserID</td>
<td>n</td>
<td>EBCDIC</td>
<td>User name or user ID used to log into the FTP server.</td>
</tr>
</tbody>
</table>

Packet and data trace formatting NMI

Records collected from the SYSTCPDA and SYSTCPOT interface described in “Real-time TCP/IP network monitoring NMI” on page 540 can be formatted programmatically with the EZBCTAPI macro. This section describes how the EZBCTAPI interface can be used.
The interface to the formatter described in this topic provides a means for network applications to format packet and data trace records. An application program can capture a copy of the packet and data trace buffers using the network management interface for TCP/IP real-time packet and data tracing (SYSTCPDA) or OSAENTA packets (SYSTCPOT), which is described in "Real-time TCP/IP network monitoring NMI" on page 540.

Trace records are laid out in the trace buffer as a series of Component Trace Entries (CTEs). Each CTE contains one trace record. The format identification field (CteFmtId) describes the layout of data in the trace record. Types 1, 2 and 3 contain a header (GTCNTL) that is described by the EZBCTHDR macro (or the EZBYCTHH header). Types 4, 5, and 6 contain a header (PTHDR_T) described by the EZBYPTHA macro (or the EZBYPTHH header). The following table depicts the layout of the various records.

<table>
<thead>
<tr>
<th>CteFmtId</th>
<th>Description</th>
<th>Header</th>
<th>IP Header</th>
<th>Protocol</th>
<th>Data</th>
<th>V1R7</th>
<th>V1R8</th>
<th>V1R9+</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Packet Trace</td>
<td>GTCNTL</td>
<td>IPv4</td>
<td>variable</td>
<td>variable</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>2</td>
<td>X25 Trace</td>
<td>GTCNTL</td>
<td>IPv4</td>
<td>variable</td>
<td>variable</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>3</td>
<td>Data Trace</td>
<td>GTCNTL</td>
<td>N/A</td>
<td>N/A</td>
<td>variable</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>4</td>
<td>Packet Trace</td>
<td>PTHDR_T</td>
<td>IPv4</td>
<td>variable</td>
<td>variable</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>5</td>
<td>Packet Trace</td>
<td>PTHDR_T</td>
<td>IPv6</td>
<td>variable</td>
<td>variable</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>6</td>
<td>Data Trace</td>
<td>PTHDR_T</td>
<td>N/A</td>
<td>N/A</td>
<td>variable</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>7</td>
<td>EE Trace¹</td>
<td>PTHDR_T</td>
<td>N/A</td>
<td>N/A</td>
<td>variable</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

¹ EE stands for Enterprise Extender. Read about Enterprise Extender in the Enterprise Extender connections information in the z/OS Communications Server: SNA Network Implementation Guide.

Note: Record types 1, 2, and 3 are no longer created by TCP/IP.

The ABBREV value of the PKTTRACE, DATTRACE, and OSAENTA commands determines the amount of data that is available. The layout of CTEs in the 64 KB buffer is shown in the following figure.
Packet and data trace formatting NMI: Configuration and enablement

There is no formal configuration required to enable this interface.

EZBCTAPI network management interface for formatting packet trace records

The EZBCTAPI macro accepts parameters to format component trace records from the TCP/IP packet trace, OSAENTA, and data trace. The data is formatted in the same fashion as is done using the IBM-provided packet trace and data trace formatters that are available with the IPCS CTRACE command. Note however that this interface does not require an IPCS environment to be active.

Requirement: High-level assembler language, Version 1 Release 5 or later is required to use this macro.

The EZBCTAPI macro enables users to pass component trace records to the format routine for processing and capture the formatted output text. There are several functions performed by the macro:

- SETUP - Define the formatting environment with the various parameters.
- FORMAT - Pass a record to the formatting interface.
- TERM - Delete the formatting environment allowing final output to be shown.
- QUIT - Delete the formatting environment without any final output. Summary and statistical reports created at the end of SYSTCPDA processing will not be formatted. This request should be used for quick termination of the interface when no further output is desired.
The header files and macros are described in the following table.

<table>
<thead>
<tr>
<th>Header files for C/C++ programs</th>
<th>Macros for assembler programs</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>EZBCTAPI</td>
<td>Used to format the records created by the SYSTCPDA interfaces.</td>
</tr>
<tr>
<td>EZBYCTHH</td>
<td>EZBCTHDR</td>
<td>Packet trace header describing the TCP/IP packet for types 1, 2, and 3 trace records.</td>
</tr>
<tr>
<td>EZBYPTHH</td>
<td>EZBYPTHA</td>
<td>Packet trace header describing the TCP/IP packets for types 4, 5, and 6 trace records.</td>
</tr>
<tr>
<td>N/A</td>
<td>EZBYPTO</td>
<td>Describes packet trace options for the formatter.</td>
</tr>
</tbody>
</table>

These header files and macros are included in the SEZANMAC data set. This data set must be available in the concatenation when compiling or assembling a part that makes use of these definitions.

**EZBCTAPI NMI requirements**

Minimum authorization: Problem state and any PSW key  
Dispatchable unit mode: Task  
Cross memory mode: PASN=HASN=SASN  
AMODE: 31-bit  
ASC mode: Primary  
Interrupt status: Enabled for I/O and external interrupts  
Locks: No locks held  
Control parameters: Must be addressable in the primary address space and have a storage key that matches the PSW key

**EZBCTAPI NMI format**

name name: Symbol. Begin name in column 1.  
One or more blanks must precede EZBCTAPI.  
EZBCTAPI  
One or more blanks must follow EZBCTAPI.  
SETUP  
FORMAT  
TERM  
QUIT  

,WORKAREA=workarea workarea: RX-type address or register (2) - (12).  
,API=epaaddr epaaddr: RX-type address or register (2) - (12).  
,COMP=name name: RX-type address or register (2) - (12).  
,CTE=record record: RX-type address or register (2) - (12).  
,ENTRYID=entryId_list entryId_list: Rx-type address or register (2) - (12).  
,LDTO=stcktime stcktime: RX-type address or register (2) - (12).  
,LSO=stcktime stcktime: RX-type address or register (2) - (12).
EZBCTAPI NMI parameters
The parameters are explained in this section. First, select one of the four required
parameters that define the function that the interface is to perform (SETUP,
FORMAT, TERM, QUIT). Next, select the optional parameters that you need.

The required parameters are as follows:

**SETUP**
Initialize the interface by allocating and initializing control blocks and
loading the component trace format table. Most of the other keywords can
be specified to define the processing options.

**FORMAT**
Locate the specific entry in the format table and call the format routine.
The CTE keyword identifies the record to be formatted.

**TERM**
End the interface by calling the filter routine one last time to issue any
final reports and release all the allocated resources.

**QUIT**
End the interface by calling the filter routine one last time to release all the
allocated resources acquired by the formatter.

The optional parameters are as follows:
.API=epaaddr
  Specifies the location of a word that contains the location of the EZBCTAPI routine. Use this keyword in the SETUP call to pass the entry pointer address to the interface. This might be useful to avoid the overhead of loading and deleting this reentrant interface module. If the API keyword is not used, then the EZBCTAPI routine is loaded by the SETUP function and deleted by the TERM or QUIT function.

.COMP=name
  Specifies the location of an 8-byte character field containing the name of the CTRACE component. If not specified, the component name of SYSTCPDA is used.

.CTE=record
  Specifies the location of a component trace record. Used with the FORMAT function.

.ENTRYID=entryId_list
  Specifies a list of entry identifiers used to to select a subset of CTE entries. The format of the list is fullword count, followed by a list of two fullword pairs. The fullword count contains the number of fullword pairs that follow. The first word of the pair contains the low value of the entry ID and second word contains the high value of the entry ID. For example, to format only type 5 data trace records, use the following:

  DC F'1,5,5

  The count is one pair of words, and the low and high values are both 5.

.LDTO=stcktime
  Specifies the location of an 8-byte store clock field. This field is in units of STCK timer units. It contains the local date time offset. This field is used to convert STCK time stamps in the component trace records to local time. If not specified, the field CVTLDTO is used as the default.

.LSO=stcktime
  Specifies the location of an 8-byte store clock field. This field is in units of STCK timer units. It contains the leap seconds time offset. This field is used to convert STCK time stamps in the component trace records to GMT time and local time. If not specified, the field CVTLSO is used as the default.

.MAXLINE=number
  Specifies the location of a word than contains the maximum line width for formatted output. The minimum value is 60 and the maximum value is 250. The default value is 80.

.NMCTF=epaaddr
  Specifies the location of a word that contains the location of the EZBNMCTF stub routine. This might be useful to avoid the overhead of loading and deleting this reentrant interface module. This keyword should be used on each invocation that will invoke the interface (MF=(E)). If the NMCTF keyword is not specified, then the EZBNMCTF routine is called by the macro as an external reference and EZBNMCTF must be link-edited with the application program.

.OBTAIN=epaaddr
  Specifies the location of a word that contains an entry point location of a routine used by the interface to obtain storage. The default is a routine that uses the STORAGE (OBTAIN) macro to obtain the storage from the
operating system. If the OBTAIN keyword is specified then the RELEASE keyword must be specified. It is passed these pointers in a parameter list addressed by register 1:

- The work area
- The 4-word user token (see the USERTOKEN definition later in this section)
- The word where the location of the obtained storage is returned
- The word with the length of the storage to be obtained

The following return codes are supported:

- 00: The storage was obtained. The location of the storage is returned.
- 04: The storage could not be obtained. The address is null.

Standard calling conventions are used to call the routine in the same environment when the EZBCTAPI interface was called.

\ OPTIONS=options

Specifies the address of options to be passed to the packet trace formatter. These options are described by EZBYPTO data area. See "Passing options to the packet trace formatter" on page 596 for more information.

\ PRTSRV=epaaddr

Specifies the location of a word that contains entry point location of a routine used by the interface and formatter to print lines of text and messages. It is passed these parameters in a parameter list addressed by register 1:

- The BLSUPPR2 parameter list.
- The 4-word user token (see the USERTOKEN definition later in this section).

The following return codes are supported from the print routine:

- 00: The line of text was printed.
- 04: The line was not printed and future output is to be suppressed.

Standard calling conventions are used to call the routine in the same environment when the EZBCTAPI interface was called.

To generate the BLSUPPR2 parameter list use the BLSUPPR2 macro:

PPR2 BLSUPPR2 DSECT=YES

The BLSUPPR2 macro is described in the z/OS MVS Programming: Assembler Services Reference ABE-HSI.

The following fields are defined as:

- PPR2BUF: Location of buffer containing the data to be printed
- PPR2BUFL: Length of data in the buffer to be printed
- PPR2MSG: The buffer contains a message
- PPR2OVIN: Overflow indentation level (0 for the first line, 2 for subsequent lines)

The print buffer is in the EBCDIC code page. The buffer has been translated to change unprintable characters to periods. The new line character (X'15') is located in each data line and the print function is called for each new line. Should the data buffer be larger than the MAXLINE value minus 1, then the print function is called as many times as needed with the rest of the print line with PPR2OVIN set to 2.
`RELEASE=epaaddr` Specifies the location of a word that contains the entry point location of a routine used by the interface to release storage. The default is a routine that uses the STORAGE (RELEASE) macro to release the storage back to the operating system. If the RELEASE keyword is specified, then the OBTAIN keyword must be specified. It is passed these pointers in a parameter list addressed by register 1:

- The work area
- The 4-word user token (see the USERTOKEN definition later in this section)
- The word with the location of the storage to be released
- The word with the length of the storage to be obtained

The following return codes are supported:

- 00: The storage was released.
- 04: The storage could not be released.

Standard calling conventions are used to call the routine in the same environment when the EZBCTAPI interface was called.

`RETCODE=retcode` Specifies the location where the interface return code is stored. The return code is also in general purpose register (GPR) 15.

`REPORT=FULL`  
`REPORT=SHORT`  
`REPORT=SUMMARY`  
`REPORT=TALLY`  

Formats the report.

- FULL Formats the IP protocol headers and packet data. This includes the component mnemonic, entry identifier, date and time, and a description of the trace record. FULL is the default report option.

- SHORT Formats the IP protocol headers. This includes the component mnemonic, entry identifier, date and time, and a description of the trace record.

- SUMMARY Requests two lines per trace record. Key fields from each qualifying trace record are printed following the date, time, and entry description.

- TALLY Requests a list of trace entry definitions for the component and counts how many times each trace entry occurred.

`RSNCODE=rsncode` Specifies the location where the interface reason code is stored. The reason code is also in GPR 0. EZBCTAPI provides a reason code if the return code is other than 0.

`TABLE=name` Specifies the location of the 8-character field that contains the name for the format table (EZBPTFM4) or two words. The first word contains zeros and the second word contains the entry point address of EZBPTFM4. If not
specified or the name is not used, then the EZBPTFM4 table is loaded. This might be useful to avoid the overhead of loading and deleting this format table.

\texttt{TIME=GMT}

\texttt{TIME=LOCAL}

Specifies the conversion of the time field in the component trace records. The default is \texttt{TIME=LOCAL}.

\textbf{GMT} The time is shown as Greenwich Mean Time.

\textbf{LOCAL} The time is shown as local time.

\texttt{USERSTOKEN=token}

Specifies the location of a 4-word field that is copied and passed to the print service routine and the storage functions. The default is four words of zeros.

\texttt{WORKAREA=workarea}

The location of a 16 KB work area used by the interface for its control blocks, work area, and save areas. The work area is cleared by the SETUP function. This work area must remain intact until the TERM or QUIT function is called. The work area cannot be shared across tasks. Specification is optional; if not specified, a 16 KB work area is obtained.

\texttt{MF=(L,list_addr)}

\texttt{MF=(L,list_addr,attr)}

Requests that a EZBCTAPI parameter list be defined. \texttt{list_addr} is the name assigned to the list. \texttt{attr} is an optional attribute used to define the parameter list. The default is 0D. No other keywords can be used with this macro format.

\texttt{MF=G}

Requests that the EZBCTAPI parameter list description be generated. No other keywords can be used with this macro format.

\texttt{MF=(M,list_addr)}

\texttt{MF=(M,list_addr,COMPLETE)}

Request that the EZBCTAPI parameter list be modified. \texttt{COMPLETE} requests that the parameter list be set to binary zeros before any modifications.

\texttt{MF=(E,list_addr)}

\texttt{MF=(E,list_addr,COMPLETE)}

Requests that the EZBCTAPI parameter list be modified. \texttt{COMPLETE} requests that the parameter list be set to binary zeros before any modifications. In addition, for the SETUP function the EZBCTAPI interface program is loaded, and for the TERM and QUIT functions the interface program is deleted (see the API keyword to modify this behavior). The interface program is then called.

\textbf{Restriction: COMPLETE} does not apply to TERM and QUIT functions.

The following table shows supported functions and keyword combinations.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Input/ Output</th>
<th>MF(E) SETUP</th>
<th>MF(E) FORMAT</th>
<th>MF(E) TERM</th>
<th>MF(E) QUIT</th>
<th>MF(M)</th>
<th>MF(L)</th>
<th>MF(G)</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORKAREA</td>
<td>1</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The location of a 72-byte standard save area in the primary address space.

Before issuing the EZBCTAPI macro, the caller does not have to place any information into any access register (AR).

**EZBCTAPI NMI output register information**

When control returns to the caller, the general purpose registers (GPRs) contain:

- **Reason code**, if GPR 15 contains a nonzero return code; otherwise, used as a work register by the system
- **Used as a work register by the system**
- **Unchanged**
- **Used as a work register by the system**
- **Return code**
When control returns to the caller, the access registers (ARs) contain:

**Register contents**

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

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

**EZBCTAPI NMI ABEND codes**

There are no ABEND codes.

**EZBCTAPI NMI return and reason codes**

When control returns from EZBCTAPI, GPR 15 (and retcode, if you coded RETCODE) contains one of the return codes shown in Table 88. GPR 0 (and rsncode, if you coded RSNCODE) might contain one of the reason codes shown in Table 88.

Table 88. EZBCTAPI return and reason codes

<table>
<thead>
<tr>
<th>Hexadecimal return code (CtApi_IRtnCd)</th>
<th>Hexadecimal reason code (CtApi_IRsnCd)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>Function was successful.</td>
</tr>
<tr>
<td>04</td>
<td>See note</td>
<td>The FORMAT function was not successful.</td>
</tr>
<tr>
<td>04</td>
<td>10</td>
<td>The SETUP function was not done or did not complete.</td>
</tr>
<tr>
<td>04</td>
<td>11</td>
<td>The trace record is not the correct format.</td>
</tr>
<tr>
<td>04</td>
<td>18</td>
<td>The trace record could not be identified.</td>
</tr>
<tr>
<td>04</td>
<td>1B</td>
<td>The filter/analysis routine failed.</td>
</tr>
<tr>
<td>08</td>
<td>See note</td>
<td>The SETUP function was not successful.</td>
</tr>
<tr>
<td>08</td>
<td>01</td>
<td>The SETUP function has already initialized the interface.</td>
</tr>
<tr>
<td>08</td>
<td>02</td>
<td>Print callback function was not provided.</td>
</tr>
<tr>
<td>08</td>
<td>03</td>
<td>Unable to load format table.</td>
</tr>
<tr>
<td>08</td>
<td>04</td>
<td>Unable to allocate storage for tables.</td>
</tr>
<tr>
<td>08</td>
<td>05</td>
<td>Unable to load analysis/format exit.</td>
</tr>
<tr>
<td>0C</td>
<td>xx</td>
<td>Unknown function code xx.</td>
</tr>
<tr>
<td>10</td>
<td>See note</td>
<td>Unable to load the function interface.</td>
</tr>
<tr>
<td>10</td>
<td>04</td>
<td>The EZBCTAPI interface routine could not be found.</td>
</tr>
<tr>
<td>10</td>
<td>08</td>
<td>An error occurred loading the EZBCTAPI interface routine.</td>
</tr>
<tr>
<td>14</td>
<td>See note</td>
<td>Unable to obtain storage for a work area.</td>
</tr>
<tr>
<td>14</td>
<td>04</td>
<td>The program was not able to obtain storage for the work area.</td>
</tr>
</tbody>
</table>
Table 88. EZBCTAPI return and reason codes (continued)

<table>
<thead>
<tr>
<th>Hexadecimal return code (CtApi_IRtnCd)</th>
<th>Hexadecimal reason code (CtApi_IRsnCd)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>xxxxxxxx</td>
<td>The interface routine or the analysis routine abended; xxxxxxxx is the abend code.</td>
</tr>
</tbody>
</table>

Note: The first line of a new return code is a generic line about the return code.

Table 89. EZBCTAPI formatter return and reason codes

<table>
<thead>
<tr>
<th>Hexadecimal return code (CtApi_FRtnCd)</th>
<th>Hexadecimal reason code (CtApi_FRsnCd)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>N/A</td>
<td>Normal processing of the entry</td>
</tr>
<tr>
<td>04</td>
<td>N/A</td>
<td>Reread the records from the first</td>
</tr>
<tr>
<td>08</td>
<td>N/A</td>
<td>The current entry is bypassed</td>
</tr>
<tr>
<td>0C</td>
<td>N/A</td>
<td>No further calls to the format/analysis routine</td>
</tr>
<tr>
<td>10</td>
<td>N/A</td>
<td>Ending of the subcommand</td>
</tr>
</tbody>
</table>

These return codes are described in [z/OS MVS IPCS Customization](#) for a CTRACE formatter filter/analysis exit. The packet trace formatter uses only a return code of 0 or 8. The interface return code (CtApi_IRtnCd) is always 0 for formatter return codes of 0, 4, 8, and 12; otherwise, an interface return code of 4 is returned (see interface reason code X'1B').

**EZBCTAPI NMI: Capturing trace records:** This section describes the process of capturing trace records.

**Before you begin:** You need to have done the following:

1. Update the TCP/IP profile to allow trace data to be copied: NETMONITOR PKTTRCService.
2. Grant authority to an application program to capture trace data:
   - Make the program APF authorized, and
   - Define the user with BPX.SUPERUSER, or
   - Permit the user to access EZB.NETMGMT.sysname.tcpipname.SYSTCPDA

Perform the following steps to capture trace records.

1. Start the application program.
   The application program does the following:
   a. Defines the format options in the EZBYPTO control block, passed to EZBCTAPI.
   b. Uses the EZBCTAPI macro to set up the packet trace formatter interface.
   c. Connects an AF_UNIX socket to the SYSTCPDA service (described in "Real-time TCP/IP network monitoring NMI" on page 540).
   d. Allocates a 64 KB buffer.
   e. In a loop, reads a record from the AF_UNIX socket. The first word of each record contains the length of the record. The record contains tokens that describe a TCP/IP trace buffer that contains data to be copied.
f. Calls EZBTMIC1 to copy the TCP/IP trace buffer to the application 64 KB buffer.

g. For a return value of zero or negative, reads the next record from the AF_UNIX socket.

The return value contains the amount of data moved into the buffer. The buffer contains a series of component trace entries (CTE). A CTE is described by the ITTCTE data area.

h. Processes each CTE in the buffer by calling the format function of EZBCTAPI, passing the address of the CTE.

The length of each CTE is the unsigned halfword at the start of each CTE. A CTE with a length of zero indicates the end of the buffer. This last halfword of zeros is not included in the return value of the amount of data moved.

i. Loops to read the next record from the socket.

2. Issue VARY TCPIP,,PKTTRACE or VARY TCPIP,,DATTRACE commands to collect the data of interest.

At termination, the application program frees the 64 KB buffer, closes the socket, and calls the TERM function of EZBCTAPI.

**EZBCTAPI NMI: Performance implications**

There are no performance implications.

**Example of initializing the EZBCTAPI exit environment**

```assembly
COPY EZBCTAPI
EZBCTAPI CSECT
SAVE (14,12),/* SET A BASE REGISTER
LR 12,15
USING EZBCTAPI,12
LA 15,MAINS
LA 15,8(13)
ST 15,8(13)
ST 13,4(15)
LR 13,15

/******************************************************************************/
/* INITIALIZE THE OPTIONS */
/******************************************************************************/
PTO USING EZBYPTO,APTO MAP THE OPTIONS AREA
XC APTO,APTO ZERO THE OPTIONS FLAGS AND PTRS
LA 0,EZBYPTO_SZ SET LENGTH OF OPTIONS AREA
STH 0,PTO.PTO_LENGTH
ST 13,PTO.PTO_OFFSET

* Set Format(DETAIL) Segment REASSEM STATS(DETAIL)
OI PTO.PTO_FORMAT,L'PTO_FORMAT
OI PTO.PTO_FMTDTL,L'PTO_FMTDTL
OI PTO.PTO_STATS,L'PTO_STATS
NI PTO.PTO_STCSUM,255-L'PTO_STCSUM SET STAT(DETAIL)
OI PTO.PREASM,L'PTO_REASM
OI PTO.PTO_SEGMENT,L'PTO_SEGMENT

* OPEN (PRINTDCB,OUTPUT) OPEN THE PRINT FILE
* STORAGE OBTAIN,LENGTH=CTAPI_WKSIZE,ADDR=(8)
* GET STORAGE FOR ABDPL WORK AREA
* INITIALIZE THE EZBCTAPI PARAMETER LIST
```
EZBCTAPI WORKAREA=(8),

COMP=C'CL8'SYSTCPDA',

PRTRSV=A(PRINTSRV),

OPTIONS=APTO,

REPORT=FULL,

TIME=LOCAL,

USERTOKEN=PRINTTKN,

MAXLINE=A(L'PRINTBUF-1),

MF=(M,CTAPIL,COMPLETE)

* GET A BUFFER FOR READING BUFFERS

* STORAGE OBTAIN,LENGTH=64*1024

ST 1,ABUFFER31

* SET UP THE FORMATTER INTERFACE

* EZBCTAPI SETUP,MF=(E,CTAPIL), SET UP THE INTERFACE

RETCODE=RETCDE,RSNCODE=RETRSN

LTR 15,15 DID THIS WORK

BNZ ERROR

* READ IN A TOKEN

* LOOP1 DS 0H

CALL BPX1RED,(SOCKET,

ABUFFER,PRIMARYALET,LBUFTKN,

RETVAL,RETCDE,RETRSN),VL

L 15,RETVAL

LTR 15,15

BNP EOF CLOSE SOCKET AND EXIT

* READ IN DATA BUFFERS

* ST 15,LBUFTKN

CALL EZBTMIC1,(BUFTOKEN,LBUFTKN,RETVAL,RETCDE,RETRSN)

L 15,RETVAL

LTR 15,15

BNP EOF CLOSE SOCKET AND EXIT

* GET ADDRESS THE BUFFER USING CTE,3

* LOOP2 DS 0H

LH 2,CTELENP GET LENGTH OF THIS RECORD

N 2,'X'0000FFFF' ALLOW UP TO 64K RECORDS

LTR 2,2 IS THIS THE END

BNP LOOP1 YES, DO THE NEXT BUFFER

EZBCTAPI FORMAT,CTE=CTE,

MF=(E,CTAPIL)

ALR 3,2 POINT TO THE NEXT CTE

B LOOP2 DO THE NEXT RECORD

* EZBCTAPI TERM,MF=(E,CTAPIL)

STORAGE RELEASE,LENGTH=CTAPI_WKSIZE,ADDR=(8)

* RELEASE STORAGE ABDPL WORK AREA

CLOSE (PRINTDCB)

L 13,4(13)

RETURN (14,12),RC=0

* ERROR DS 0H

* DATA
Chapter 14. Network management interfaces
Passing options to the packet trace formatter

The EZBYPTO macro describes a data area that can be passed using the EZBCTAPI OPTIONS keyword. This data area contains flags, values, and pointers that describe packet trace formatter options. Table 90 shows the option and field settings required to select the option.

These same options are available through the SYSTCPDA CTRACE formatter. You can find a detailed explanation in the packet trace (SYSTCPDA) for TCP/IP stacks and OSAENTA trace (SYSTCPOT) information in z/OS Communications Server: IP Diagnosis Guide.

Table 90. Available EZBYPTO options

<table>
<thead>
<tr>
<th>Option</th>
<th>Field setting</th>
<th>Field format</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII</td>
<td>Pto_Dump=1;Pto_DmpCd=PtoAscii;</td>
<td>Bit flag</td>
</tr>
<tr>
<td>BASIC(DETAIL)</td>
<td>Pto_Basic=1;Pto_BasDtl=1;</td>
<td>Bit flag</td>
</tr>
<tr>
<td>BASIC(SUMMARY)</td>
<td>Pto_Basic=1;Pto_BasDtl=0;</td>
<td>Bit flag</td>
</tr>
<tr>
<td>BOTH</td>
<td>Pto_Dump=1;Pto_DmpCd=PtoBoth;</td>
<td>Bit flag</td>
</tr>
<tr>
<td>CLEANUP(nnmm)</td>
<td>Pto_Cleanup=1;Pto_GcIntvl=nnmm;</td>
<td>Bit flag</td>
</tr>
<tr>
<td>DEVICE(list)</td>
<td>Pto_Deviceaddr=Addr(list);Pto_Device#=nn</td>
<td>List of 32-bit word pairs</td>
</tr>
<tr>
<td>DISCARD(list)</td>
<td>Pto_Discardaddr=Addr(list);Pto_Discard#=nn</td>
<td>List of 16-bit word pairs</td>
</tr>
<tr>
<td>DUMP</td>
<td>Pto_Dump=1;</td>
<td>Bit flag</td>
</tr>
<tr>
<td>DUMP(nnmm)</td>
<td>Pto_Dump=1;Pto_MaxDmp=nnmm;</td>
<td>Bit flag; 31-bit word</td>
</tr>
<tr>
<td>EBCDIC</td>
<td>Pto_Dump=1;Pto_DmpCd=PtoEbcdic;</td>
<td>Bit flag; Value</td>
</tr>
<tr>
<td>ELEMENT(list)</td>
<td>Pto_Elementaddr=Addr(list);Pto_Element=nn</td>
<td>List of 32-bit word pairs</td>
</tr>
<tr>
<td>ETHTYPE(list)</td>
<td>Pto_EthTypeaddr=Addr(list);Pto_EthType#=nn</td>
<td>List of 32-bit word pairs</td>
</tr>
<tr>
<td>FLAGS(flags)</td>
<td>Pto_Flagsaddr=Addr(Pto_Flagss),</td>
<td>16 bytes of bit flag used to select packets</td>
</tr>
<tr>
<td></td>
<td>Pto_Flags#size(Pto_Flagss);</td>
<td></td>
</tr>
<tr>
<td>FLAGS(ANY</td>
<td>ALL)</td>
<td>Pto_FlagAny=1</td>
</tr>
<tr>
<td>FORMAT(DETAIL)</td>
<td>Pto_Format=1;Pto_FmtDtl=1;</td>
<td>Bit flags</td>
</tr>
<tr>
<td>Option</td>
<td>Field setting</td>
<td>Field format</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>FORMAT(SUMMARY)</td>
<td>Pto_Format=1;Pto_FmtDtl=0;</td>
<td>Bit flags</td>
</tr>
<tr>
<td>FULL</td>
<td>Pto_Dump=1;Pto_Format=1;Pto_FmtDtl=1;</td>
<td>Bit flags</td>
</tr>
<tr>
<td>HEX</td>
<td>Pto_Dump=1;Pto_DmpCd=PtoHex;</td>
<td>Bit flags</td>
</tr>
<tr>
<td>INTERFACE</td>
<td>Pto_Links@=Addr(list);Pto_Links#=nn</td>
<td>List of 16-byte character strings</td>
</tr>
<tr>
<td>IPADDR(list)</td>
<td>Pto_Addr@=Addr(list);Pto_Addr#=nn</td>
<td>List of 16-byte byte IPv6 addresses</td>
</tr>
<tr>
<td>MACADDR(list)</td>
<td>Pto_MacAddr@=Addr(list);Pto_MacAddr#=nn</td>
<td>List of 6-byte Mac addresses</td>
</tr>
<tr>
<td>PORT(list)</td>
<td>Pto_Port@=Addr(list);Pto_Port#=nn</td>
<td>List of 16-bit word pairs</td>
</tr>
<tr>
<td>PROTOCOL(list)</td>
<td>Pto_Proto@=Addr(list);Pto_Proto#=nn</td>
<td>List of 32-bit word pairs</td>
</tr>
<tr>
<td>REASSEMBLY(11111)</td>
<td>Pto_ReAsm=1;Pto_MaxRsm=11111111</td>
<td>Bit flags</td>
</tr>
<tr>
<td>REASSEMBLY(DETAIL)</td>
<td>Pto_ReAsm=1;Pto_RsmSum=0</td>
<td>Bit flags</td>
</tr>
<tr>
<td>REASSEMBLY(SUMMARY)</td>
<td>Pto_ReAsm=1;Pto_RsmSum=1</td>
<td>Bit flags</td>
</tr>
<tr>
<td>NOREASSEMBLY</td>
<td>Pto_ReAsm=0;</td>
<td>Bit flag</td>
</tr>
<tr>
<td>SEGMENT</td>
<td>Pto_Segment=1;</td>
<td>Bit flag</td>
</tr>
<tr>
<td>NOSEGMENT</td>
<td>Pto_Segment=0;</td>
<td>Bit flag</td>
</tr>
<tr>
<td>SESSION(DETAIL)</td>
<td>Pto_SesRpt=Pto_SesDetail;Pto_Session=1;</td>
<td>Bit flag</td>
</tr>
<tr>
<td>SESSION(SUMMARY)</td>
<td>Pto_SesRpt=Pto_SesSummary;Pto_Session=1;</td>
<td>Bit flag</td>
</tr>
<tr>
<td>SESSION(STATE)</td>
<td>Pto_SesRpt=Pto_SesState;Pto_Session=1;</td>
<td>Bit flag</td>
</tr>
<tr>
<td>SPEED(local,remote)</td>
<td>Pto_LSpeed=nn;Pto_RSpeed=nn</td>
<td>Two 32-bit words</td>
</tr>
<tr>
<td>STATISTICS(DETAIL)</td>
<td>Pto_Stats=1;Pto_StcSum=0</td>
<td>Bit flag</td>
</tr>
<tr>
<td>STATISTICS(SUMMARY)</td>
<td>Pto_Stats=1;Pto_StcSum=1</td>
<td>Bit flag</td>
</tr>
<tr>
<td>STREAMS(1111)</td>
<td>Pto_Stream=1;Pto_StrmBuf=11111111</td>
<td>Bit flag</td>
</tr>
<tr>
<td>STREAMS(DETAIL)</td>
<td>Pto_Stream=1;Pto_StrmSum=0</td>
<td>Bit flag</td>
</tr>
<tr>
<td>STREAMS(SUMMARY)</td>
<td>Pto_Stream=1;Pto_StrmSum=1</td>
<td>Bit flag</td>
</tr>
<tr>
<td>SUBAREA(list)</td>
<td>Pto_Subarea@=Addr(list);Pto_SubArea#=nn</td>
<td>List of 32-bit word pairs</td>
</tr>
<tr>
<td>SUMMARY</td>
<td>Pto_Summary=1;</td>
<td>Bit flag</td>
</tr>
<tr>
<td>TCID(list)</td>
<td>Pto_Tcid@=Addr(list);Pto_Tcid#=nn</td>
<td>List of 8-byte hex strings</td>
</tr>
<tr>
<td>TH5SA(list)</td>
<td>Pto_Th5SA@=Addr(list);Pto_Th5SA#=nn</td>
<td>List of 8-byte hex strings</td>
</tr>
<tr>
<td>TALLY</td>
<td>Pto_Stats=1;Pto_StcSum=0</td>
<td>Bit flag</td>
</tr>
<tr>
<td>VLANID(list)</td>
<td>Pto_VlanId@=Addr(list);Pto_VlanId#=nn</td>
<td>List of 32-bit word pairs</td>
</tr>
</tbody>
</table>

Notes:

1. A packet might span multiple trace records. When segmented records are encountered, the SEGMENT option re-creates the packet as a single trace record. The packet is not used until the last trace segment record is passed to the formatter. Until that time, the packet is saved in a temporary buffer. Use the NOSEGMENT option to prevent this. The CLEANUP value can be used to free the temporary buffers for segments that will not be completed. The QUIT or TERM function frees all unprocessed segments.

2. When the NOSEGMENT option is used, only the first segment has the IP header and protocol headers.

3. A packet might be fragmented. When you specify the REASSEMBLY option, the formatter saves the fragments in a temporary buffer until all the fragments.
have been processed to recreate the original complete packet. The packet is not used until the last trace record is passed to the formatter. The CLEANUP value frees temporary buffers that have not completed, for reassembly. The QUIT or TERM function frees all unprocessed fragments.

4. Use the NOREASSEMBLY option to prevent this saving of records.

5. If the CLEANUP value is 0, then the temporary buffers are not released until the QUIT or TERM function.

6. You can use the EZBYPTO options control block to request multiple reports.

7. Use of the EZBCTAPI TERM function creates the SESSION, STATISTICS, and STREAMS reports.

8. The EZBYPTO data area is not processed by the SETUP function call. The values in the data area and values pointed from the data area must remain intact until after the first FORMAT, TERM, or QUIT function call.

9. If the first and only discard reason code in the Pto_Discard1(1) field is 65535 (X'FFFF'), then all packets with a nonzero discard reason code are selected. If one of the discard reason codes is 0, then packets that were not discarded are selected.

**Using the packet trace formatter**

There are two ways of passing the formatter truncated records so that trace records contain only headers.

- Use the ABBREV keyword of the PKTTRACE command to truncate traced records. No matter the value of ABBREV, the record will always contain the IP header and protocol header.

- Shorten the data passed to the formatter. Use these steps:

  1. Determine whether the trace record is the first segment of packet. The sequence number field of the header (PTH_SeqNum) is 0. The record contains the IP header and protocol header. Otherwise the record contains only data.

  2. Set the CTELENP field (the first halfword of a trace record) to the smaller of CTELENP or the sum of the size of the CTEFDATA field, the size of the PTH_HDR field, the size of the IP header and the size of the protocol header.

  3. Set the PTO_SEGMENT flag to 0. The length also includes the 2-byte length field CTELENE.

Records passed to the formatter must always contain at least the ITTCTE, PTHDR_t, the IP header and the protocol header.

**TCP/IP callable NMI (EZBNMIFR)**

z/OS Communications Server provides a high-speed low-overhead callable programming interface for network management applications to access data related to the TCP/IP stack. Use the EZBNMIFR network management interface to perform the following functions:

- Monitor TCP or UDP endpoints
- Monitor TCP/IP storage
- Drop one or more TCP connections
- Drop one or more UDP endpoint
- Monitor TN3270E Telnet server performance
- Monitor TCP/IP sysplex networking data
- Monitor TCP/IP stack profile statement settings
This section describes the details for invoking the EZBNMIFR interface with the defined input parameters and for processing the output it provides. The following topics are addressed:

- “EZBNMIFR overview”
- “EZBNMIFR: Configuration and enablement” on page 600
- “Using the EZBNMIFR requests” on page 600
- “TCP/IP NMI request format” on page 603
- “TCP/IP NMI response format” on page 612
- “TCP/IP NMI request and response data structures” on page 615
- “TCP/IP NMI examples” on page 616

**EZBNMIFR overview**

You can invoke the EZBNMIFR interface to perform two types of requests: poll-type requests and action-type requests.

**EZBNMIFR: Poll-type requests**

For poll-type requests, EZBNMIFR is a callable interface that returns data related to the TCP/IP stack at a given point in time. In most cases, the caller can specify filters that limit the returned data to a specific set of information.

Poll-type requests enable you to obtain the following types of information from the TCP/IP stack:

- Active TCP connections
- Active UDP endpoints
- Active TCP listeners
- TCP/IP storage utilization
- TN3270E Telnet server monitor groups
- TN3270E Telnet server connection performance data
- Sysplex XCF data
- Dynamic VIPA addresses
- Dynamic VIPA port distribution
- Dynamic VIPA routes
- Dynamic VIPA connections
- TCP/IP profile statement settings

**EZBNMIFR: Action-type requests**

Requests to drop TCP connections or UDP endpoints are requests for an action. The caller must specify the connection identifier, local IP address, local port, remote IP address and remote port for the TCP connections, or local IP address and local port for UDP endpoints to drop. The remote IP address, remote port, and connection identifier are ignored for UDP endpoints.

The callable interface that drops a TCP connection or UDP endpoint is similar to the Netstat DRop/-D command, which can be invoked from the TSO, z/OS UNIX, and MVS operator environments. The major difference is that the callable interface can drop multiple connections at a time. The caller must specify the connection identifier, local IP address, local port, remote IP address, and remote port for TCP and local IP addresses and local port for UDP endpoints.

**Tip:** When a TCP connection or UDP endpoint is dropped, the associated socket is not closed. The application that owns the associated socket must close the socket.
The following action-type request tells the TCP/IP stack to perform an action:

- Drop TCP connections or UDP endpoints

**EZBNMIFR: Configuration and enablement**

There is no configuration required to enable this interface when it is used as a poll-type interface.

Authorization to drop a TCP connection or UDP endpoint is identical to the TSO, z/OS UNIX, and MVS Operator Netstat Drop commands. An application can use this interface to drop a TCP connection or UDP endpoint only if the MVS.VARY.TCPPIP.DROP security profile in the OPERCMDS class is defined and the user ID associated with the application is permitted to this resource. Therefore, if a user ID is already permitted to issue Netstat DROP/-D, the user ID can use the EZBNMIFR callable interface to drop a TCP connection or UDP endpoint.

**Using the EZBNMIFR requests**

This material describes how to use EZBNMIFR requests with the TCP/IP stack or the TN3270E Telnet server.

**EZBNMIFR requirements**

Minimum authorization: Supervisor state, executing in system key, APF-authorized, or superuser

Dispatchable unit mode: Task or SRB

Cross memory mode: PASN=SASN=HASN

AMODE: 31-bit or 64-bit

ASC mode: Primary

Interrupt status: Enabled for I/O and external interrupts

Locks: Not applicable

Control parameters: Must reside in an addressable area in the primary address space and must be accessible using caller's execution key

**EZBNMIFR format**

Invoke EZBNMIFR, as follows.

For C/C++ callers:

```c
NWMSServices(JobName,
    RequestResponseBuffer,
    &RequestResponseBufferAlet,
    &RequestResponseBufferLength,
    &ReturnValue,
    &ReturnCode,
    &ReasonCode);
```

For assembler callers:

```assembly
CALL EZBNMIFR,(JobName,
    RequestResponseBuffer,
    RequestResponseBufferAlet,
    RequestResponseBufferLength,
    ReturnValue,
    ReturnCode,
    ReasonCode)
```
**EZBNMIFR parameters**

**JobName**
Supplied and returned parameter.

**Type:** Character
**Length:** Doubleword

The name of an 8-character field that contains the EBCDIC job name of the target TCP/IP stack or TN3270E Telnet server. If the first character of the supplied job name is an asterisk (*), the call is made to the first active TCP/IP stack and its job name is returned.

**Tip:** You can use the GetTCPListeners request to find all active TN3270E Telnet servers. The returned listener list indicates which application names are TN3270E Telnet server-type applications. Use the results from all TCP/IP stacks to determine which TN3270E Telnet servers have affinity to a particular stack and which servers have no affinity.

**RequestResponseBuffer**
Supplied parameter

**Type:** Character
**Length:** Variable

The name of the storage area that contains an input request. The input request must be in the format of a request header (NWMHeader) as defined in the EZBNMRFHC header file. On successful completion of the request, the storage will contain an output response in the same format.

**RequestResponseBufferAlet**
Supplied parameter.

**Type:** Integer
**Length:** Fullword

The name of a fullword that contains the ALET of RequestResponseBuffer. If a nonzero ALET is specified, the ALET must represent a valid entry in the caller's dispatchable unit access list (DU-AL).

**RequestResponseBufferLength**
Supplied parameter.

**Type:** Integer
**Length:** Fullword

The name of a fullword that contains the length of the request/response buffer. If the buffer length is too short to contain all of the requested information, the request fails with the return code ENOBUFF. The length that is needed to contain all of the information is provided in the NWMHeader data structure of the response, in the NWMBytesNeeded field. If the buffer length is not the minimum size for the request, the request fails with the return code ENOBUFF, but the value that is needed is not provided in the NWMBytesNeeded field. The minimum size is the combined length of the NWMHeader data structure and any input filters.
**ReturnValue**

Returned parameter.

Type: Integer
Length: Fullword

The name of a fullword in which the EZBNMIFR service returns one of the following:

- 0 or positive, if the request is successful. A value greater than 0 indicates the number of output data bytes copied to the response buffer. See "TCP/IP NMI response format" on page 612 for additional details about processing request completions.
- -1, if the request is not successful.

**ReturnCode**

Returned parameter.

Type: Integer
Length: Fullword

The name of a fullword in which the EZBNMIFR service stores the return code (errno). The EZBNMIFR service returns ReturnCode only if ReturnValue is -1.

**ReasonCode**

Returned parameter.

Type: Integer
Length: Fullword

The name of a fullword in which the EZBNMIFR service stores the reason code (errnojr). The EZBNMIFR service returns ReasonCode only if ReturnValue is -1. ReasonCode further qualifies the ReturnCode value.

The EZBNMIFR service sets the following return codes and reason codes:

<table>
<thead>
<tr>
<th>ReturnValue</th>
<th>ReturnCode</th>
<th>ReasonCode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>The request was successful.</td>
</tr>
<tr>
<td>-1</td>
<td>ENOBUFS</td>
<td>JRBuffTooSmall</td>
<td>The request was not successful. The request/response buffer is too small to contain all of the requested information. Some of the requested information might be returned. If the buffer was large enough for some information to be returned, the NWMHeader NWMBytesNeeded field might contain the buffer size needed to return all of the requested information. See the description of the RequestResponseBufferLength parameter for an explanation of when the NWMBytesNeeded value is provided.</td>
</tr>
<tr>
<td>ReturnValue</td>
<td>ReturnCode</td>
<td>ReasonCode</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------</td>
<td>------------</td>
<td>----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-1</td>
<td>EACCES</td>
<td>JRSAFNotAuthorized</td>
<td>The request was not successful. The caller is not authorized. For the DropConnection request, this might be returned if the user is not permitted to the MVS.VARY.TCPIP:DROP security profile in the OPERCMDS class.</td>
</tr>
<tr>
<td>-1</td>
<td>EAGAIN</td>
<td>JRTCPNOTUP</td>
<td>The request was not successful. The target TCP/IP stack or TN3270E Telnet server was not active.</td>
</tr>
<tr>
<td>-1</td>
<td>EAGAIN</td>
<td>JRMustBeSysplex</td>
<td>The request was not successful. The target TCP/IP stack has not joined a sysplex.</td>
</tr>
<tr>
<td>-1</td>
<td>EFAULT</td>
<td>JRReadUserStorageFailed</td>
<td>The request was not successful. A program check occurred while copying input parameters, or while copying input data from the request/response buffer.</td>
</tr>
<tr>
<td>-1</td>
<td>EFAULT</td>
<td>JRWriteUserStorageFailed</td>
<td>The request was not successful. A program check occurred while copying output parameters, or while copying output data to the request/response buffer.</td>
</tr>
<tr>
<td>-1</td>
<td>EINVAL</td>
<td>JRInvalidValue</td>
<td>The request was not successful. A value that is not valid was specified in the request/response header.</td>
</tr>
<tr>
<td>-1</td>
<td>ETCPIERR</td>
<td>JRTcpError</td>
<td>The request was not successful. An unexpected error occurred.</td>
</tr>
</tbody>
</table>

Network management applications can use any of the following methods to invoke the EZBNMIFR service:

- Issue a LOAD macro to obtain the EZBNMIFR service entry point address, and then CALL that address. The EZBNMIFR load module must reside in a linklist data set (for example, the SEZALOAD load library of TCP/IP), or in the LPA area.
- Issue a LINK macro to invoke the EZBNMIFR service. The EZBNMIFR load module must reside in a linklist data set (for example, the SEZALOAD load library of TCP/IP), or in the LPA area.
- Link-edit EZBNMIFR directly into the application load module, and then CALL the EZBNMIFR service. Include SYS1.CSSLIB(EZBNMIFR) in the application load module link-edit.
- For 64-bit C/C++ applications, link-edit the EZBNMIF4 program directly into the application load module, and then CALL the EZBNMIF4 service. Include SYS1.CSSLIB(EZBNMIF4) in the application load module link-edit.

**TCP/IP NMI request format**

The following section describes the format and details of the poll-type requests provided with EZBNMIFR. A second section describing action-type requests follows.
Format and details for poll-type requests

The following poll-type requests are provided by EZBNMIFR. The request constant, which is specified in the NWMTcpListenType field in the NWMHeader data structure, follows the request name.

- GetTCPListeners (NWMTcpListenType) — obtain information about active TCP listeners
- GetUDPTable (NWMUdpConnType) — obtain information about active UDP sockets
- GetConnectionDetail (NWMTcpConnType) — obtain information about active TCP connections
- GetStorageStatistics (NWMStgStatsType) — obtain information about TCP/IP storage utilization. This request does not support filtering.
- GetTnMonitorGroups (NWMTnMonGrpType) — obtain information about TN3270E Telnet server monitor groups
- GetTnConnectionData (NWMTnConnType) — obtain information about TN3270E Telnet server connection performance data
- GetSysplexXCF (NWMSyXcfType) — obtain information about all TCP/IP stacks in the subplex. This request returns a list of all TCP/IP stacks in the same subplex as the invoked TCP/IP stack. For each TCP/IP stack, the MVS system name and one or more dynamic XCF IP addresses are returned. There are no filters defined for this request. If the invoked TCP/IP stack has not joined a sysplex, then return value -1, return code EAGAIN, and reason code JRMustBeSysplex are returned without any other data.
- GetDVIPAList (NWMDvListType) — obtain information about dynamic virtual IP addresses (DVIPAs). This request returns a list of all IPv4 and IPv6 DVIPAs for the invoked TCP/IP stack. For each DVIPA, the MVS system name, TCP/IP job name, and various status information are returned.
- GetDVIPAPortDist (NWMDvPortDistType) — obtain information about dynamic virtual IP address (DVIPA) port distribution. This request returns a list of IPv4 and IPv6 distributed DVIPAs and ports. For each distributed DVIPA and port pair, one or more entries are returned for each target TCP/IP stack. If the invoked TCP/IP stack has not joined a sysplex, then return value -1, return code EAGAIN, and reason code JRMustBeSysplex are returned without any other data. If the TCP/IP stack is not a distributing stack, then an empty response buffer is returned with a successful return value, return code, and reason code. If the same DVIPA and port pair are affected by more than one QOS Policy, then an entry with the same DVIPA and port is returned for each QOS policy.
- GetDVIPARoute (NWMDvRouteType) — obtain information about dynamic virtual IP address (DVIPA) routes. This request returns a list of information that is defined on VIPAROUTE profile statements. Each entry includes the dynamic XCF address of a target TCP/IP stack and the corresponding target IP address that is used to route connection requests to that TCP/IP stack. Output is returned only by a distributing TCP/IP stack, or by a backup TCP/IP stack for a distributed DVIPA when the backup TCP/IP stack is assuming ownership of the distributed DVIPA. If the invoked TCP/IP stack has not joined a sysplex, then return value -1, return code EAGAIN, and reason code JRMustBeSysplex are returned without any other data. If the invoked TCP/IP stack is neither a distributing stack nor a backup stack, then an empty response buffer is returned with a successful return value, return code, and reason code.
- GetDVIPAConnRTab (NWMDvConnRTabType) — obtain information about dynamic virtual IP addresses (DVIPA) connections. This call returns a list of IPv4 and IPv6 DVIPA TCP connections. Entries are returned for the following:
– All DVIPA interfaces for which MOVEABLE IMMEDIATE or NONDISRUPTIVE was specified
– On a sysplex distributor routing stack, every connection that is being routed through this distributor
– On a stack taking over a DVIPA, every connection to the DVIPA
– On a sysplex distributor target stack or a stack that is in the process of giving up a DVIPA, every connection for which the stack is an endpoint

If none of these apply, then an empty response buffer is returned with a successful reason value, return code, and reason code. If the invoked TCP/IP stack has not joined a sysplex, then return value -1, return code EAGAIN, and reason code JRMustBeSysplex are returned without any other data.

- GetProfile (NWMProfileType) — obtain information about the current TCP/IP profile statement settings. This request does not support filtering. To detect changes to the profile statement settings, callers can use this callable request to obtain an initial set of current profile settings, and then do one of the following:
  - Repeat the request, over a time interval, comparing returned data from a previous response to the returned data from the last response.
  - Obtain the SMF Type 119 subtype 4 TCP/IP profile event records. These records provide information about changes to the profile settings that were made using VARY TCPIP,OBEYFILE command processing. The records are created only if the PROFILE parameter is specified on the SMFCONFIG profile statement. These records are written to the MVS SMF data sets, and can also be obtained from the Real-time TCP/IP network monitoring NMI. For more information about this SMF record, see the TCP/IP profile event record (subtype 4) information in z/OS Communications Server: IP Configuration Reference. The SMF record might be created even if some errors occurred during the VARY TCPIP,OBEYFILE command processing. Application programs that process these records must compare the sections of changed information to the previous profile settings, to determine whether profile changes actually occurred.

The general format of the request consists of the request header and the request section descriptors (triplets), which define the input data. A triplet describes the input filters and contains the offset, in bytes, of the request section relative to the beginning of the request buffer, the number of elements in the request section, and the length of an element in the request section.

Filter request section:

For requests that support filters, you can use filters to limit the data that is returned to data that matches the specified filter values. Not all filters are supported for all requests.

The following request types do not support any filters. If filters are specified for these requests the filters are ignored.
- GetStorageStatistics
- GetSysplexXCF
- GetProfile
The following table describes all possible filters.

**Table 91. Available EZBNMIF poll-type request filters**

<table>
<thead>
<tr>
<th>Filter item</th>
<th>Filter item value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID</td>
<td>A 16-bit address space number of a socket application address space.</td>
</tr>
<tr>
<td>Resource name</td>
<td>An EBCDIC job name, right-padded with blanks if less than 8 characters long, of a socket application address space (Client Name in Netstat displays). A question mark can be used to wildcard a single character, and an asterisk can be used to wildcard zero or more characters. For example, the value A?C* matches all names with a first character A and a third character C, but does not match two-character names or names beginning with B through Z.</td>
</tr>
<tr>
<td>Resource ID</td>
<td>A 32-bit unsigned binary TCP/IP resource identifier (Client ID in Netstat displays).</td>
</tr>
<tr>
<td>Server resource ID</td>
<td>A 32-bit unsigned binary TCP/IP resource identifier of the related server listening connection.</td>
</tr>
<tr>
<td>Local or source IP address</td>
<td>A 32-bit IPv4 address or a 128-bit IPv6 address. The local or source IP address filter value is specified as the IP address field within a sockaddr structure. The sockaddr address family field must be set to indicate whether the local IP address filter value is an IPv4 address or an IPv6 address. For IPv4 connections, the local IP address filter value can be specified as either an IPv4 address (for example, 9.1.2.3) or an IPv4-mapped IPv6 address (for example, ::FFFF:9.1.2.3). For all connections, a null address can be specified as either an IPv4 address (0.0.0.0), an IPv4-mapped IPv6 address (::FFFF:0.0.0.0), or an IPv6 address (::).</td>
</tr>
<tr>
<td>Local or source IP address prefix</td>
<td>A 16-bit signed binary value that specifies the number of local or source IP address bits to use. For example, the value 12 means that the first 12 bits of a local or source IP address are compared to the first 12 bits of the local IP address filter value. The value 0 means that all address bits are compared. A value greater than 32 for an IPv4 address, or greater than 128 for an IPv6 address, means that all address bits are compared.</td>
</tr>
<tr>
<td>Local or source port</td>
<td>A 16-bit unsigned binary port number.</td>
</tr>
<tr>
<td>Remote or destination IP address</td>
<td>A 32-bit IPv4 address or a 128-bit IPv6 address. The remote or destination IP address filter value is specified as the IP address field within a sockaddr structure. The sockaddr address family field must be set to indicate whether the remote IP address filter value is an IPv4 address or an IPv6 address. For IPv4 connections, the remote IP address filter value can be specified as either an IPv4 address (for example, 9.1.2.3) or an IPv4-mapped IPv6 address (for example, ::FFFF:9.1.2.3). For all connections, a null address can be specified as either an IPv4 address (0.0.0.0), an IPv4-mapped IPv6 address (::FFFF:0.0.0.0), or an IPv6 address (:).</td>
</tr>
<tr>
<td>Remote or destination IP address prefix</td>
<td>A 16-bit signed binary value specifying the number of remote or destination IP address bits to use. For example, the value 12 means that the first 12 bits of a remote or destination IP address are compared to the first 12 bits of the remote IP address filter value. The value 0 means that all address bits are compared. A value greater than 32 for an IPv4 address, or greater than 128 for an IPv6 address, means that all address bits are compared.</td>
</tr>
<tr>
<td>Filter item</td>
<td>Filter item value</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Remote or destination</td>
<td>A 16-bit unsigned binary port number.</td>
</tr>
<tr>
<td>port</td>
<td></td>
</tr>
<tr>
<td>Lu name</td>
<td>An EBCDIC LU name (right-padded with blanks if less than 8 characters in length) of the TN3270E LU representing the client. Use a question mark (?) as a wildcard for a single character and an asterisk (<em>) as a wildcard for zero or more characters. For example, the value A?C</em> matches all names with a first character A and a third character C, but does not match 2-character names, names beginning with B through Z, or names with anything other than C in the third position.</td>
</tr>
<tr>
<td>Monitor group identifier</td>
<td>A 32-bit unsigned binary value assigned by the TN3270E Telnet server to identify up to 255 unique monitor groups. Any parameter change within an existing monitor group or a new monitor group causes the TN3270E Telnet server to assign a new identifier. The identifier is reported in the monitor group table and connection data allowing a comparison between monitoring criteria and actual connection performance. The monitor group identifier can be obtained by issuing the GetTnMonitorGroups request.</td>
</tr>
<tr>
<td>Application name</td>
<td>An EBCDIC application name (right-padded with blanks if less than 8 characters in length) of the SNA application name in session with the TN3270E secondary LU representing the client. The application name can have wildcard characters. Use a question mark (?) as a wildcard for a single character, and an asterisk (<em>) as a wildcard for 0 or more characters. For example, the value A?C</em> matches all application names with a first character A and a third character C, but does not match 2-character names, names beginning with B through Z, or names with anything other than C in the third position.</td>
</tr>
<tr>
<td>Application data</td>
<td>An EBCDIC character string (right-padded with blanks if less than 40 characters in length) associated with a TCP socket by the owning application using the SIOCSAPPLDATA IOCTL. The application data filter can have wildcard characters. Use a question mark (?) as a wildcard for a single character and an asterisk (*) as a wildcard for zero or more characters. For z/OS Communications Server applications, see Application data in the z/OS Communications Server: IP Configuration Reference for applications that use the SIOCSAPPLDATA ioctl as a source for information about the content, format, and meaning of the application data that the applications associate with the sockets that they own. For other applications, see the documentation that is supplied by the application.</td>
</tr>
<tr>
<td>Interface name</td>
<td>An EBCDIC interface name (right-padded with blanks if less than 16 characters in length) of an IPv4 or IPv6 interface. The interface name can have wildcard characters. Use a question mark (?) as a wildcard for a single character, and use an asterisk (<em>) as a wildcard for zero or more characters. For example, the value A?C</em> matches all interface names with a first character A and a third character C, but does not match 2-character names, names beginning with B through Z, or names that have anything other than the character C in the third position.</td>
</tr>
<tr>
<td>Filter item</td>
<td>Filter item value</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Dynamic virtual IP address and family</td>
<td>A 32-bit IPv4 address or a 128-bit IPv6 address. The dynamic VIPA address family field must also be set to indicate whether the DVIPA filter value is an IPv4 address or an IPv6 address. For IPv4 addresses, the DVIPA filter value can be specified as either an IPv4 address (for example, 9.1.2.3) or an IPv4-mapped IPv6 address (for example, ::FFFF:9.1.2.3). A null address can be specified as either an IPv4 address (0.0.0.0), an IPv4-mapped IPv6 address (:FFFF:0.0.0.0), or an IPv6 address (::). The dynamic virtual IP address family field must be set to AF_INET for an IPv4 address or AF_INET6 for an IPv6 address.</td>
</tr>
<tr>
<td>Dynamic virtual IP address prefix</td>
<td>A 16-bit signed binary value that specifies the number of dynamic virtual IP address bits to use. For example, the value 12 means that the first 12 bits of a dynamic VIPA are compared to the first 12 bits of the dynamic VIPA filter value. The value 0 means that all address bits are compared. A value greater than 32 for an IPv4 address, or greater than 128 for an IPv6 address, means that all address bits are compared.</td>
</tr>
<tr>
<td>Dynamic virtual IP address port</td>
<td>A 16-bit unsigned binary port number.</td>
</tr>
<tr>
<td>Destination XCF IP address and family</td>
<td>A 32-bit IPv4 address or a 128-bit IPv6 address. The destination XCF IP address family field must also be set to indicate whether the destination XCF IP address filter value is an IPv4 address or an IPv6 address. For IPv4 addresses, the destination XCF IP address filter value can be specified as either an IPv4 address (for example, 9.1.2.3) or an IPv4-mapped IPv6 address (for example, ::FFFF:9.1.2.3). A null address can be specified as either an IPv4 address (0.0.0.0), an IPv4-mapped IPv6 address (:FFFF:0.0.0.0), or an IPv6 address (:). The destination XCF IP address family field must be set to AF_INET for an IPv4 address or AF_INET6 for an IPv6 address.</td>
</tr>
<tr>
<td>Destination XCF IP address prefix</td>
<td>A 16-bit signed binary value that specifies the number of destination XCF IP address bits to use. For example, the value 12 specifies that the first 12 bits of a destination XCF IP address are compared to the first 12 bits of the destination XCF IP address filter value. The value 0 specifies that all address bits are compared. A value greater than 32 for an IPv4 address, or greater than 128 for an IPv6 address, specifies that all address bits are compared.</td>
</tr>
<tr>
<td>Target IP address and family</td>
<td>A 32-bit IPv4 address or a 128-bit IPv6 address. The target IP address family field must also be set to indicate whether the target IP address filter value is an IPv4 address or an IPv6 address. For IPv4 addresses, the destination XCF IP address filter value can be specified as either an IPv4 address (for example, 9.1.2.3) or as an IPv4-mapped IPv6 address (for example, ::FFFF:9.1.2.3). A null address can be specified as either an IPv4 address (0.0.0.0), as an IPv4-mapped IPv6 address (:FFFF:0.0.0.0), or as an IPv6 address (:). The target IP address family field must be set to AF_INET for an IPv4 address or AF_INET6 for an IPv6 address.</td>
</tr>
</tbody>
</table>
### Table 91. Available EZBNMIFR poll-type request filters (continued)

<table>
<thead>
<tr>
<th>Filter item</th>
<th>Filter item value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target IP address prefix</td>
<td>A 16-bit signed binary value that specifies the number of target IP address bits to use. For example, the value 12 means that the first 12 bits of a target IP address are compared to the first 12 bits of the target IP address filter value. The value 0 means that all address bits are compared. A value greater than 32 for an IPv4 address, or greater than 128 for an IPv6 address, means that all address bits are compared.</td>
</tr>
</tbody>
</table>

You can specify 1 – 4 filter elements. Each filter element can contain any combination of the items that are listed in [Table 91 on page 606](#). A filter element that does not have any applicable items matches all data for the request. The data must match all items that are specified in a filter element to pass that filter check; data must pass at least one filter check to be selected.

If you do not specify any filters (triplet offset field is 0, or triplet element count field is 0, or triplet element length field is 0), then the caller is requesting all information that is applicable to that request.

Table 92 and Table 93 on page 610 show which filter items are applicable for each request type that supports filters. If you specify inapplicable filters for a particular request type, they are ignored.

### Table 92. Filter items applicable to each request type (part 1)

<table>
<thead>
<tr>
<th>Filter items</th>
<th>GetTCPListeners</th>
<th>GetUDPTable</th>
<th>GetConnectionDetail</th>
<th>GetTnMonitorGroups</th>
<th>GetTnConnectionData</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Resource name</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Resource ID</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Server resource ID</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Local or source IP address</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Local or source port</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Remote or destination IP address</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Remote or destination IP address prefix</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Remote or destination port</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Lu name</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Monitor group identifier</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Application name</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Application data</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Interface name</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Dynamic virtual IP address and family</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Dynamic virtual IP address prefix</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Dynamic virtual IP address port</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
### Table 92. Filter items applicable to each request type (part 1) (continued)

<table>
<thead>
<tr>
<th>Filter items</th>
<th>GetTCPListeners</th>
<th>GetUDPTable</th>
<th>GetConnectionDetail</th>
<th>GetTnMonitorGroups</th>
<th>GetTnConnectionData</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination XCF IP address and family</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Destination XCF IP address prefix</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Target IP address and family</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Target IP address prefix</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

### Table 93. Filter items applicable to each request type (part 2)

<table>
<thead>
<tr>
<th>Filter items</th>
<th>GetDVIPAList</th>
<th>GetDVIPAPortDist</th>
<th>GetDVIPARoute</th>
<th>GetDVIPAConnRTab</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Resource name</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Resource ID</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Server resource ID</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Local or source IP address</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Local or source IP address prefix</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Local or source port</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Remote or destination IP address</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Remote or destination IP address prefix</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Remote or destination port</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Lu name</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Monitor group identifier</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Application name</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Application data</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Interface name</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Dynamic virtual IP address and family</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Dynamic virtual IP address</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Dynamic virtual IP address prefix</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Destination XCF IP address and family</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Destination XCF IP address prefix</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Target IP address and family</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Target IP address prefix</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

**Filter example**

Two filters are defined:
- Local IP Address = 9.0.0.1, Local Port = 5000
- Resource Name = FTP*

The following TCP connections exist:
• Resource Name = FTP1, Local IP Address = 9.0.0.2, Local Port = 5001
• Resource Name = FTP2, Local IP Address = 9.0.0.1, Local Port = 5000
• Resource Name = USR1, Local IP Address = 9.0.0.1, Local Port = 5002

When a GetConnectionDetail request is made, connection 1 is selected because it matches filter 2, connection 2 is selected because it matches filter 1, and connection 3 is not selected because it does not match either filter.

Format and details for action-type requests
The following section describes the format and details of the action-type requests provided with EZBNMIFR:

**DropConnection**

Drop one or more TCP connections or UDP endpoints.

The general format of the input for this request consists of the request header and the request section descriptors (triplets), which define the input data. In this case, a triplet describes the input and output buffer. It consists of the offset, in bytes, of the request section relative to the beginning of the request buffer, the number of elements in the request section, and the length of an element in the request section.

To drop a connection, the NWMDropConnEntry structure describes the input and output to the DropConnection request. Each element must input a resource ID, local address, local port, remote address, remote port, and protocol. It is possible that for a particular connection or endpoint specification, the drop attempt will fail. For this reason, the NWMDropConnEntry structure contains a return code and reason code to describe the reason for the failure. The following table describes the NWMDropConnEntry structure.

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource ID</td>
<td>Input</td>
<td>A 32-bit unsigned binary TCP/IP resource identifier (Client ID in Netstat displays). This descriptor is required for TCP connections and is ignored for UDP endpoints.</td>
</tr>
<tr>
<td>Local IP address</td>
<td>Input</td>
<td>A 32-bit IPv4 address or a 128-bit IPv6 address. The local IP address value is specified as the IP address field within a sockaddr structure. The sockaddr address family field must be set to indicate whether the local IP address value is an IPv4 address or an IPv6 address. For IPv4 connections, the local IP address value can be specified as either an IPv4 address (for example, 9.1.2.3) or as an IPv4-mapped IPv6 address (for example, ::FFFF:9.1.2.3). For all connections, a null address can be specified as either an IPv4 address (0.0.0.0), as an IPv4-mapped IPv6 address (::FFFF:0.0.0.0), or as an IPv6 address (::). This descriptor is required.</td>
</tr>
<tr>
<td>Local port</td>
<td>Input</td>
<td>A 16-bit unsigned binary port number. The local port value is specified as the port field within the sockaddr structure. This descriptor is required.</td>
</tr>
</tbody>
</table>
Table 94. NWMDropConnEntry description (continued)

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote IP address</td>
<td>Input</td>
<td>A 32-bit IPv4 address or a 128-bit IPv6 address. The remote IP address filter value is specified as the IP address field within a sockaddr structure. The sockaddr address family field must be set to indicate whether the remote IP address value is an IPv4 address or an IPv6 address. For IPv4 connections, the remote IP address value can be specified as either an IPv4 address (for example, 9.1.2.3) or as an IPv4-mapped IPv6 address (for example, ::FFFF:9.1.2.3). For all connections, a null address can be specified as either an IPv4 address (0.0.0.0), an IPv4-mapped IPv6 address (::FFFF:0.0.0.0), or an IPv6 address (:). This descriptor is required for TCP connections and is ignored for UDP endpoints.</td>
</tr>
<tr>
<td>Remote port</td>
<td>Input</td>
<td>A 16-bit unsigned binary port number. The remote port value is specified as the port field within the sockaddr structure. This descriptor is required for TCP connections and is ignored for UDP endpoints.</td>
</tr>
<tr>
<td>Protocol</td>
<td>Input</td>
<td>An 8-bit character representing either IPPROTO_TCP or IPPROTO_UDP.</td>
</tr>
<tr>
<td>Return code</td>
<td>Output</td>
<td>A 4-byte value, NWMDropConnRc. If this value is nonzero, it indicates that the drop attempted for this connection failed. This return code describes the reason for failure.</td>
</tr>
<tr>
<td>Reason code</td>
<td>Output</td>
<td>A 4-byte value, NWMDropConnRs. When the Return Code is set, this value might provide more detailed information about why the drop request failed for this connection.</td>
</tr>
</tbody>
</table>

See "TCP/IP NMI response format" for information about processing the result of a DropConnection request.

**TCP/IP NMI response format**

The general format of the response is:

- The response header, the request section descriptors (triplets), and the response section descriptors (quadruplets). Processing is slightly different for the request types (poll-type and action-type) as described in the following sections.
- The request sections
- The response sections; one of the following types of response sections is returned:
  - TCP connection information
  - TCP listener information
  - UDP connection information
  - TCP/IP storage statistics
  - DropConnection entries
  - TN3270E Telnet server monitor groups
  - TN3270E Telnet server connection performance information
  - Sysplex XCF data
  - Dynamic VIPA addresses
  - Dynamic VIPA port distribution
Tip: Connection elements for TN3270E Telnet server connection performance data are returned only if the connection is being monitored by a MonitorGroup that is mapped to the connection. See Connection monitoring mapping statement in z/OS Communications Server: IP Configuration Guide for details.

Processing poll-type request responses

Processing responses for the following poll-type requests is described in this section:
- GetTCPListeners
- GetUDPTable
- GetConnectionDetail
- GetStorageStatistics
- GetTnMonitorGroups
- GetTnConnectionData
- GetSysplexXCF
- GetDVIPAList
- GetDVIPAPortDist
- GetDVIPARoute
- GetDVIPAConnRTab
- GetProfile

A quadruplet consists of the offset, in bytes, of the response section that is relative to the beginning of the response buffer, the number of elements in the response section, the length of a response section element, and the total number of elements that passed the requested filter checks.

The response header contains the number of bytes required to contain all the requested data. When the return code is ENOBUFF, use this value to allocate a larger request/response buffer and reissue the request.

GetProfile response format

For the GetProfile request, the output is returned as one record. The response section quadruplet contains the following values:
- Offset is the offset, into the response buffer, of a GetProfile record header.
- The length of each element is always 0.
- The number of elements in the response section is always 1 to indicate that only one record was returned.
- The total number of matching elements is always 1, because filters are not supported.

The record header is mapped by the NWMRecHdr structure. The header consists of the following fields:
- An EBCDIC identifier
- The total length of the record
The number of section descriptors (triplets) that are present in this record. There are always 20 section descriptors returned. The section descriptor triplets are mapped by the NWMTriple structure.

The section descriptors (triplets) immediately follow the record header, and the sections immediately follow the section descriptors. If there is no profile information for a section, the section descriptor triplet fields for that section all contain 0.

The section structures in the GetProfile response are identical to the section structures in the TCP/IP profile SMF 119 subtype 4 event records. If you already have an application that processes the SMF record section structures, you can also use it for processing the GetProfile response section structures. See the TCP/IP profile event record (subtype 4) information in z/OS Communications Server: IP Configuration Reference for a layout of this SMF record.

In the GetProfile response, the Profile Information Common and Data Set Name sections primarily contain information about the initial profile, not about the last change to the profile; however, the following fields contain the date and time of the last change to the profile:

- NMTP_PICOChangeTime
- NMTP_PICOChangeDate

Processing action-type request responses

Processing the response for the DropConnection action-type request is described in this section.

For this type of request, the quadruplet contains the offset and number of elements, which is the same as the offset and number of elements in the triplet (output is the same as the input). If the call to EZBNMIFR returns a nonnegative return value, and the value for NWMQMatch returned in the quadruplet section is equal to the number of entries input, NWMQNumber, then all of the connections or endpoints were dropped successfully. If the call to EZBNMIFR returns a nonnegative return value, and if NWMQMatch is less than NWMQNumber, then not all of the connections or endpoints were successfully dropped. In this case, the program should examine the return code that is set in each NWMDropConnEntry field. If the value of the return code is nonzero, then this connection was not dropped; if the value of the return code is 0, then the connection was dropped.

The following describes the codes:

<table>
<thead>
<tr>
<th>NWMDropConnRC</th>
<th>NWMDropConnRSN</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENOENT</td>
<td>JRGGetConnErr</td>
<td>The connection was not in the correct state for retrieving or the connection was not found.</td>
</tr>
<tr>
<td>EMVSERR</td>
<td>JRPATDELErr</td>
<td>Deletion of a restricted port entry failed.</td>
</tr>
<tr>
<td>EACCES</td>
<td>JRPORUACCESSAUTH</td>
<td>User does not have authority to access this port.</td>
</tr>
<tr>
<td>EMVSERR</td>
<td>JRPATFNDErr</td>
<td>Search for a restricted port failed or the connection was not found.</td>
</tr>
</tbody>
</table>
Table 95. Return code values (continued)

<table>
<thead>
<tr>
<th>NWMDropConnRC</th>
<th>NWMDropConnRSN</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENOENT</td>
<td>JRPATFNDErr</td>
<td>Search for a restricted port failed or the connection was not found.</td>
</tr>
<tr>
<td>ENOENT</td>
<td>JRGETCONNERR</td>
<td>The connection was not in the correct state for retrieving.</td>
</tr>
<tr>
<td>EAGAIN</td>
<td>JRUDPNOTUP</td>
<td>TCP/IP was not initialized</td>
</tr>
<tr>
<td>EAGAIN</td>
<td>JRTPCPNOTUP</td>
<td>The request was not successful. The target TCP/IP stack was not active.</td>
</tr>
<tr>
<td>EINVAL</td>
<td>JRINVALIDVALUE</td>
<td>The request was not successful. A value that is not valid was specified in the request/response header.</td>
</tr>
</tbody>
</table>

**Guideline:** Input to the DropConnection request will most likely be from the output result of a GetUDPTable or GetConnectionDetail request where the filtered connection information might return connections that are not intended for termination. Applications that support the DropConnection request should be coded to ensure that the connections input for termination have been examined carefully by programming logic that selects connections that meet a specific criteria, such as state.

**Example:** One NWMDropConnEntry is submitted:

Resource ID = 003A, Local IP Address = 9.0.0.1, Local Port = 5003, Remote IP Address = 9.0.0.5, Remote Port = 3000, Protocol = TCP

The following TCP connections exist:

- Resource Name = FTP1, Resource ID = 001A, Local IP Address = 9.0.0.2, Local Port = 5000, Remote IP Address = 9.0.0.5, Remote Port = 3001
- Resource Name = FTP2, Resource ID = 002A, Local IP Address = 9.0.0.1, Local Port = 5001, Remote IP Address = 9.0.0.5, Remote Port = 3002
- Resource Name = USR1, Resource ID = 004F, Local IP Address = 9.0.0.1, Local Port = 5002, Remote IP Address = 9.0.0.5, Remote Port = 3003
- Resource Name = USR7, Resource ID = 003A, Local IP Address = 9.0.0.1, Local Port = 5003, Remote IP Address = 9.0.0.5, Remote Port = 3000

When a DropConnection request is made, connection 4 is dropped because it matches the five required items.

**TCP/IP NMI request and response data structures**

The NMI request and response data structures for C/C++ and assembler programs are located as follows:

<table>
<thead>
<tr>
<th>Header file for C/C++ programs</th>
<th>Macros for assembler programs</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>EZBNMRRHC</td>
<td>EZBNMRHA</td>
<td>The NMI request and response data structure definitions.</td>
</tr>
<tr>
<td>EZBNMMPC</td>
<td>EZBNMMPA</td>
<td>The GetProfile request data structure definitions for the sections of profile information in the response.</td>
</tr>
</tbody>
</table>
These header files and macros are included in the SEZANMAC data set and the header files are also included in the z/OS UNIX file system directory, /usr/include. When you compile or assemble a program in an MVS batch job, the SEZANMAC data set must be available in the MVS batch job concatenation. For an example of the mappings of the request and response data structures, see sample EZBNMIFR mappings.

TCP/IP NMI examples

Example 1: The following C/C++ code fragment shows how to format a request to obtain TCP connection information using the filters in the filter definition example (see “Filter example” on page 610):

```c
/********************************************************************
/* */
/* NMI data definitions */
/* */
/********************************************************************

typedef struct {
  NWMHeader NMIheader;
  NWMFilter NMIfilter[2];
} NMIbuftype;
NMIbuftype *NMIbuffer;
unsigned int NMIalet;
int NMIlength;
int RV;
int RC;
unsigned int RSN;
#define NMIBUFSIZE 8192
NMIbuffer=malloc(NMIBUFSIZE);
NMIalet=0;
NMIlength=NMIBUFSIZE;
/********************************************************************
/**
/* Format the header */
/* */
/********************************************************************
NMIbuffer->NMIheader.NWMHeaderIdent=NWMHEADERIDENTIFIER;
NMIbuffer->NMIheader.NWMHeaderLength=sizeof(NWMHeader);
NMIbuffer->NMIheader.NWMVersion=NWMVERSION1;
NMIbuffer->NMIheader.NWMType=NWMTCPCONNTYPE;
NMIbuffer->NMIheader.NWMBytesNeeded=0;
NMIbuffer->NMIheader.NWMInputDataDescriptors.
  NWMFiltersDesc.NWMTOffset=sizeof(NWMHeader);
NMIbuffer->NMIheader.NWMInputDataDescriptors.
  NWMFiltersDesc.NWMTLength=sizeof(NWMFilter);
NMIbuffer->NMIheader.NWMInputDataDescriptors.
  NWMFiltersDesc.NWMTNumber=2;
/********************************************************************
/**
/* Format filter 1 */
/* */
/********************************************************************
NMIbuffer->NMIfilter[1].NWMFilterIdent=NWMFILTERIDENTIFIER;
NMIbuffer->NMIfilter[1].NWMFilterFlags=NWMFILTERLCLADDRMASK|
  NWMFILTERLCLPORTMASK;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_family=AF_INET;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_port=5000;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_addr.s_addr=0x09000001;
NMIbuffer->NMIfilter[1].NWMFilterFlags=NWMFILTERLCLADDRMASK|
  NWMFILTERLCLPORTMASK;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_family=AF_INET;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_port=5000;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_addr.s_addr=0x09000001;
NMIbuffer->NMIfilter[1].NWMFilterFlags=NWMFILTERLCLADDRMASK|
  NWMFILTERLCLPORTMASK;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_family=AF_INET;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_port=5000;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_addr.s_addr=0x09000001;
NMIbuffer->NMIfilter[1].NWMFilterFlags=NWMFILTERLCLADDRMASK|
  NWMFILTERLCLPORTMASK;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_family=AF_INET;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_port=5000;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_addr.s_addr=0x09000001;
NMIbuffer->NMIfilter[1].NWMFilterFlags=NWMFILTERLCLADDRMASK|
  NWMFILTERLCLPORTMASK;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_family=AF_INET;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_port=5000;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_addr.s_addr=0x09000001;
NMIbuffer->NMIfilter[1].NWMFilterFlags=NWMFILTERLCLADDRMASK|
  NWMFILTERLCLPORTMASK;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_family=AF_INET;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_port=5000;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_addr.s_addr=0x09000001;
NMIbuffer->NMIfilter[1].NWMFilterFlags=NWMFILTERLCLADDRMASK|
  NWMFILTERLCLPORTMASK;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_family=AF_INET;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_port=5000;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_addr.s_addr=0x09000001;
NMIbuffer->NMIfilter[1].NWMFilterFlags=NWMFILTERLCLADDRMASK|
  NWMFILTERLCLPORTMASK;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_family=AF_INET;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_port=5000;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_addr.s_addr=0x09000001;
NMIbuffer->NMIfilter[1].NWMFilterFlags=NWMFILTERLCLADDRMASK|
  NWMFILTERLCLPORTMASK;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_family=AF_INET;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_port=5000;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_addr.s_addr=0x09000001;
NMIbuffer->NMIfilter[1].NWMFilterFlags=NWMFILTERLCLADDRMASK|
  NWMFILTERLCLPORTMASK;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_family=AF_INET;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_port=5000;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_addr.s_addr=0x09000001;
NMIbuffer->NMIfilter[1].N WMFilterFlags=NWMFILTERLCLADDRMASK|
  NWMFILTERLCLPORTMASK;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_family=AF_INET;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_port=5000;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_addr.s_addr=0x09000001;
NMIbuffer->NMIfilter[1].NWMFilterFlags=NWMFILTERLCLADDRMASK|
  NWMFILTERLCLPORTMASK;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_family=AF_INET;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_port=5000;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_addr.s_addr=0x09000001;
NMIbuffer->NMIfilter[1].NWMFilterFlags=NWMFILTERLCLADDRMASK|
  NWMFILTERLCLPORTMASK;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_family=AF_INET;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_port=5000;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_addr.s_addr=0x09000001;
NMIbuffer->NMIfilter[1].NWMFilterFlags=NWMFILTERLCLADDRMASK|
  NWMFILTERLCLPORTMASK;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_family=AF_INET;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_port=5000;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_addr.s_addr=0x09000001;
NMIbuffer->NMIfilter[1].NWMFilterFlags=NWMFILTERLCLADDRMASK|
  NWMFILTERLCLPORTMASK;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_family=AF_INET;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_port=5000;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_addr.s_addr=0x09000001;
NMIbuffer->NMIfilter[1].NWMFilterFlags=NWMFILTERLCLADDRMASK|
  NWMFILTERLCLPORTMASK;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_family=AF_INET;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_port=5000;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_addr.s_addr=0x09000001;
NMIbuffer->NMIfilter[1].NWMFilterFlags=NWMFILTERLCLADDRMASK|
  NWMFILTERLCLPORTMASK;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_family=AF_INET;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_port=5000;
NMIbuffer->NMIfilter[1].NWMFilterLocal.
  NWMFilterLocalAddr4.sin_addr.s_addr=0x09000001;
NMIbuffer->NMIfilter[1].NWMFilterFlags=NWMFILTERLCLADDRMARK|
Guideline: In z/OS releases prior to V1R7, the current version is 1. In z/OS version V1R7 and later, the current version is 2. Applications coded with NWMVERSION1 (as in example 1) will have the version accepted in z/OS version V1R4 and later. However, applications coded with NWMCURRENTVER and compiled using the version 2 headers work only on z/OS version V1R7 and later releases. Applications using NWMCURRENTVER in z/OS version V1R7 and later releases should recognize that the current version might not be accepted on prior releases of the operating system. When these applications receive an error code indicating an error in the version, they should drop back to the prior (or lowest) version number and verify that that version is acceptable with the current operating system.

The version used does not restrict which functions are available. If an application using version 1 and compiled with a version 2 header is executed on a prior release of the operating system, the application will receive the data corresponding to the release of the operating system on which it executes. Therefore, if the application is executing on a system running version 2 and specifies version 1, it still receives all data including the new version 2 data (STOKEN). If the same application is executed on a release that supports only version 1, it receives everything except the new version 2 data.

Example 2: The following C/C++ code fragment shows how to drop a connection using the following values:

- Resource ID = 003A
- Local IP Address = 9.0.0.1
- Local Port = 5003
- Remote IP Address = 9.0.0.5
- Remote Port = 3000

Guideline: In z/OS releases prior to V1R7, the current version is 1. In z/OS version V1R7 and later, the current version is 2. Applications coded with NWMVERSION1 (as in example 1) will have the version accepted in z/OS version V1R4 and later. However, applications coded with NWMCURRENTVER and compiled using the version 2 headers work only on z/OS version V1R7 and later releases. Applications using NWMCURRENTVER in z/OS version V1R7 and later releases should recognize that the current version might not be accepted on prior releases of the operating system. When these applications receive an error code indicating an error in the version, they should drop back to the prior (or lowest) version number and verify that that version is acceptable with the current operating system.

The version used does not restrict which functions are available. If an application using version 1 and compiled with a version 2 header is executed on a prior release of the operating system, the application will receive the data corresponding to the release of the operating system on which it executes. Therefore, if the application is executing on a system running version 2 and specifies version 1, it still receives all data including the new version 2 data (STOKEN). If the same application is executed on a release that supports only version 1, it receives everything except the new version 2 data.

Example 2: The following C/C++ code fragment shows how to drop a connection using the following values:

- Resource ID = 003A
- Local IP Address = 9.0.0.1
- Local Port = 5003
- Remote IP Address = 9.0.0.5
- Remote Port = 3000

The version used does not restrict which functions are available. If an application using version 1 and compiled with a version 2 header is executed on a prior release of the operating system, the application will receive the data corresponding to the release of the operating system on which it executes. Therefore, if the application is executing on a system running version 2 and specifies version 1, it still receives all data including the new version 2 data (STOKEN). If the same application is executed on a release that supports only version 1, it receives everything except the new version 2 data.

Example 2: The following C/C++ code fragment shows how to drop a connection using the following values:

- Resource ID = 003A
- Local IP Address = 9.0.0.1
- Local Port = 5003
- Remote IP Address = 9.0.0.5
- Remote Port = 3000

The version used does not restrict which functions are available. If an application using version 1 and compiled with a version 2 header is executed on a prior release of the operating system, the application will receive the data corresponding to the release of the operating system on which it executes. Therefore, if the application is executing on a system running version 2 and specifies version 1, it still receives all data including the new version 2 data (STOKEN). If the same application is executed on a release that supports only version 1, it receives everything except the new version 2 data.
Example 3: The following assembler code fragment shows how to format a request to obtain TCP connection information using the filters in the filter definition example (see "Filter example" on page 610):
R0 EQU 0
R1 EQU 1
R2 EQU 2
R3 EQU 3
R4 EQU 4
R5 EQU 5
R6 EQU 6
R7 EQU 7
R8 EQU 8
R9 EQU 9
R10 EQU 10
R11 EQU 11
R12 EQU 12
R13 EQU 13
R14 EQU 14
R15 EQU 15

STORAGE OBTAIN,LENGTH=@DYNSIZE,ADDR=(R13),LOC=ANY
USING NMIdata,R13
ST R13,NMIbuffer

***********************************************************************
** Format the header **
***********************************************************************
LA R2,NMIheader
USING NWMHeader,R2
XC NWMHeader,NWMHeader
MVC NWMHeaderIdent,=A(NWMHeaderIdentifier)
LHI R0,NWMHeaderSize
ST R0,NWMHeaderLength
LHI R0,NWMVersion1
STH R0,NWMTcpConnType
XC NWMBytesNeeded,NWMBytesNeeded
LA R3,NWMInputDataDescriptors
USING NWMTriplet,R3
LHI R0,NWMHeaderSize
ST R0,NWMTOffset
LHI R0,NWMTLength
LHI R0,NMIFilter#
ST R0,NWMType
DROP R3
DROP R2

***********************************************************************
** Format filter 1 **
***********************************************************************
LA R2,NMIfilter1
USING NWMFilter,R2
XC NWMFilter,NWMFilter
MVC NWMFilterIdent,=A(NWMFilterIdentifier)
MVC NWMFilterFlags,=A(NWMFilterLclAddrMask)
OC NWMFilterFlags,=A(NWMFilterLclPortMask)
LA R3,NWMInputDataDescriptors
USING SOCKADDR,R3
LHI R0,AF_INET
STC R0,SOCK_FAMILY
LHI R0,5000
STH R0,SOCK_SIN_PORT
MVC SOCK_SIN_ADDR,=XL4'09000001'
DROP R2,R3

***********************************************************************
** Format filter 2 **
***********************************************************************
LA R2,NMIfilter2
USING NWMFilter,R2
XR NWMFilter,NWMFilter
MVC NWMFilterIdent,=A(NWMFilterIdentifier)
MVC NWMFilterFlags,=A(NWMFilterLclAddrMask)
OC NWMFilterFlags,=A(NWMFilterLclPortMask)
LA R3,NWMInputDataDescriptors
USING SOCKADDR,R3
LHI R0,AF_INET
STC R0,SOCK_FAMILY
LHI R0,5000
STH R0,SOCK_SIN_PORT
MVC SOCK_SIN_ADDR,=XL4'09000001'
DROP R2,R3
SNA network monitoring NMI

z/OS Communications Server VTAM provides a single AF_UNIX socket interface for allowing network management applications to obtain the following types of data:

- Enterprise Extender (EE) connection data
  This data contains information about all EE connections or a desired set of EE connections as specified by the application using the local IP address or host name, the remote IP address or host name, or both.

- Enterprise Extender summary data
  This data contains information comprising a summary of EE activity for this host.

- High Performance Routing (HPR) connection data
  This data contains information about specific HPR connections Rapid Transport Protocol physical units (RTP PUs) as specified by the application using either 1) the RTP PU name, or 2) the RTP partner CP name with an optional APPN COS specification. These RTP PUs are not limited to those using EE connections.

- Common Storage Manager (CSM) statistics
  This data always contains CSM storage pool statistics and CSM summary information and can optionally contain CSM storage owner statistics.
A client network management application polls for information through specific requests using an AF_UNIX stream socket connection that uses VTAM as the server for that socket. The requested data is provided to the application directly using the AF_UNIX stream socket connection.

**SNA network monitoring NMI configuration**

The z/OS system administrator might restrict access to this interface by defining the RACF (or equivalent external security manager product) resource IST.NETMGMT.sysname.SNAMGMT in the SERVAUTH class (sysname represents the MVS system name where the interface is being invoked).

For applications that use the interface, the MVS user ID is permitted to the defined resource. If the resource is not defined, then only superusers (users permitted to BPX.SUPERUSER resource in the FACILITY class) are permitted to it. If you are developing a feature for a product to be used by other parties, include instructions in your documentation indicating that either administrators must define and give appropriate permission to the given security resource to use that feature, or you must run your program as superuser.

**Requirements:**
- The administrator must define an OMVS segment for VTAM if one is not already defined.
- The VTAM OMVS user ID must have write access to the /var directory.

**SNA network monitoring NMI: Enabling and disabling the interface**

You can enable the SNA Network Monitoring data interface by setting the VTAM start option SNAMGMT to YES, and you can disable the interface by setting the VTAM start option SNAMGMT to NO. The default for this start option is NO, and the start option is modifiable after VTAM is started. This start option can be specified in any of the following ways:

- Using the START command for VTAM
  1. IBM default value is NO
  2. Within the default VTAM start option list ATCSTR00 (ATCSTR00 is always used regardless of whether LIST=xx was entered to specify a supplemental VTAM start option list)
  3. Within the supplemental VTAM start list (ATCSTRxx, if LIST=xx entered) as SNAMGMT=YES or SNAMGMT=NO
  4. From the backup start option list (specified by the LISTBKUP start option)
  5. START command options entered by operator as SNAMGMT=YES or SNAMGMT=NO
  6. START command options reentered by the operator

See [Sources of start options](z/OS Communications Server: SNA Network Implementation Guide) for more information.

- Using the MODIFY VTAMOPTS command
  
  MODIFY vtamprocname,VTAMOPTS,SNAMGMT=YES
  MODIFY vtamprocname,VTAMOPTS,SNAMGMT=NO

The current value of the SNAMGMT start option is displayable using any of the following VTAM DISPLAY commands:

DISPLAY NET,VTAMOPTS
SNA network monitoring NMI: Communicating with the server

Applications that need to communicate with the VTAM AF_UNIX server can do so by creating an AF_UNIX stream socket using either the Language Environment C/C++ API or UNIX System Services BPX callable services. The VTAM server provides a well-known AF_UNIX stream socket with a socket path name of /var/sock/SNAMGMT that applications can use in connecting to the server.

Perform the following steps to communicate with the VTAM server:

1. Open an AF_UNIX socket.
2. Connect to the VTAM server using the socket path name /var/sock/SNAMGMT.
3. Read any data on the socket.
4. Build the NMI request packet.
5. Send the packet to the VTAM server.
6. Read the reply.
   - If the reply is a termination record, perform any application cleanup.
   - If the reply is a response to the request, process the response.
7. Repeat the process beginning with step 4 or close the connection.

Tips:

- When an application establishes a successful connection to the VTAM server, the server responds by sending an initialization record to the application. The application must read this record before it can start processing request responses.
- When VTAM needs to close the connection with the application, it attempts to send a termination record to the application before closing the connection.
  VTAM closes the connection when VTAM terminates, when the interface is disabled by an operator, or when there are severe formatting errors in the data requests sent by the application to the VTAM server.
- Both the initialization and termination records conform in structure to the solicited response records sent by VTAM to the application; see “SNA network monitoring NMI request/response format” for details.

SNA network monitoring NMI request/response format

This interface uses a request/response method over the socket. The application builds and sends an NMI request over the socket. The request specifies the type of information to be received and might contain data filters. The application must issue a receive to get the NMI response over the socket. The NMI response provides either 1) data that satisfies the request (matching any input filters specified on the request), or 2) an error response. A severe formatting error in the application’s NMI request results in VTAM sending a termination record and closing the connection.

The SNA Network Management Interface provides the formatted response data directly to the application over the AF_UNIX socket. This is in contrast to the NMIs described in “Real-time TCP/IP network monitoring NMI” on page 540, which return a token to a response buffer that the application must use as input to the EZBTMIC1 callable service in order to obtain the formatted response data.
The NMI request and response mappings are provided for programming to this interface.

**SNA network monitoring NMI request format**

All SNA NMI requests flow on the socket from the client application to the VTAM server. The general format of an SNA NMI request is:

- The request header includes the request type and the request section descriptors (triplets). The following request types can be made:
  - EE Connection Request — obtain information about some or all Enterprise Extender connections.
  - EE Summary Request — obtain summary information about all Enterprise Extender connections.
  - HPR Connection Request — obtain information about one or more HPR connections.
  - CSM Statistics Request — obtain information about global CSM statistics and optionally, about CSM storage owner statistics.

A triplet consists of the offset (in bytes) of the request section relative to the beginning of the request header, the number of elements in the request section, and the length of a request section element.

- The request sections. The only type of request section that can be specified is a filter element.
  - In an EE Connection Request, either zero or one filter elements can be included. The set of all EE connections can be selected either by not including a filter element in the request or by supplying a filter element with no filter parameters specified. A subset of EE connections can be selected by supplying a filter element that includes any combination of the filter parameters in Table 96. z/OS Communications Server does not perform name resolution (to an IP address) on any supplied host name, but simply looks for connections that were established using the given host name.

<table>
<thead>
<tr>
<th>Table 96. EE connection request filter parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Local Hostname</strong></td>
</tr>
<tr>
<td>An EBCDIC name, right-padded with nulls or blanks if less than 64 characters long (applicable to CS for z/OS version V1R5 and later releases only). The Local Hostname parameter is ignored if Local IP Address is specified.</td>
</tr>
<tr>
<td><strong>Local IP Address</strong></td>
</tr>
<tr>
<td>A 32-bit IPv4 address or a 128-bit IPv6 address. (IPv6 address is applicable to z/OS Communications Server V1R5 and later releases only.)</td>
</tr>
<tr>
<td><strong>Remote Hostname</strong></td>
</tr>
<tr>
<td>An EBCDIC name, right-padded with nulls or blanks if less than 64 characters long. The Remote Hostname parameter is ignored if a Remote IP Address value is specified.</td>
</tr>
<tr>
<td><strong>Remote IP Address</strong></td>
</tr>
<tr>
<td>A 32-bit IPv4 address or a 128-bit IPv6 address. (IPv6 address is applicable to z/OS Communications Server V1R5 and later releases only).</td>
</tr>
</tbody>
</table>

- An EE Summary Request cannot contain any filter elements; no filters are applicable to an EE summary request.
- In an HPR Connection Request, you select a subset of HPR connections based on any combination of the following items that includes, at a minimum, either the RTP PU Name or the Partner CP Name (1 - 4 filter elements can be specified per request):
RTP PU Name | An EBCDIC name, right-padded with nulls or blanks if less than 8 characters long.
Partner CP Name | A fully qualified EBCDIC name, right-padded with nulls or blanks if less than 17 characters long. Partner CP Name is ignored if RTP PU Name is specified. If a network identifier is not supplied, the Partner CP Name is qualified with the host’s network ID.
\[
\text{Use a question mark (\texttt{?}) as a wildcard for a single character or an asterisk (\texttt{*}) as a wildcard for zero or more characters.}
\]
\[
\text{For example, the value A?C* matches all names with a first character equal to A and a third character equal to C, but does not match 2-character names or names beginning with characters B through Z.}
\]
\[
\text{To request all known connections, use the string \texttt{*.*}. To request all known connections in the same network as this host, use an asterisk (\texttt{*}).}
\]
COS Name | An EBCDIC name, right-padded with nulls or blanks if less than 8 characters long. COS is ignored if RTP PU Name is specified.

- A CSM Statistics Request can contain 1 – 4 filter elements to request CSM storage ownership statistics. To request statistics about all users that own CSM storage, include a filter element in the request that has an ASID set to the value 0. To request statistics about a subset of users that own CSM storage, supply filter elements that include a nonzero value filter parameter in Table 97 on page 625.

If no filter element is provided on the CSM Statistics Request, no CSM storage ownership statistics are included in the response. The CSM Global Pool Output Section record and the CSM Summary Output Section record are always returned as part of a CSM Statistics Response, regardless of whether filters are included on the request or not.

Restriction: ASID filter parameters are applicable only to z/OS Communications Server V1R11 and later. If the initialization record that was received by the client when the connection was opened specifies that CSM Statistics Request filters are not supported by this VTAM level (any VTAM level prior to z/OS V1R11), then the server rejects any request that contains a filter on the CSM Statistics Request.

| ASID | A 16-bit integer. Specify the ASID value 0 to request all storage owner statistics. |

Table 97 on page 625 shows which filter parameters are required, optional, or not applicable (N/A) for each request type. If you specify inapplicable filters for a particular request type, an EE Connection Request, HPR Connection Request, or CSM Statistics Request, they are ignored. EE Summary Requests that contain filter elements are rejected by VTAM.
Table 97. Required filter parameters

<table>
<thead>
<tr>
<th>Request Type</th>
<th>Local IP Address or Hostname</th>
<th>Remote IP Address or Hostname</th>
<th>RTP PU name or Partner CP name</th>
<th>COS name</th>
<th>ASID</th>
</tr>
</thead>
<tbody>
<tr>
<td>EE Connection Request</td>
<td>Optional; Local Hostname ignored if local IP address is specified</td>
<td>Optional; Remote Hostname is ignored if remote IP address is specified</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>EE Summary Request</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>HPR Connection Request</td>
<td>N/A</td>
<td>N/A</td>
<td>One is required; Partner CP name ignored if RTP PU name is specified</td>
<td>Optional; ignored if RTP PU name is specified</td>
<td>N/A</td>
</tr>
<tr>
<td>CSM Statistics Request</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>Optional; if a filter is provided, an ASID value is required</td>
</tr>
</tbody>
</table>

Every valid request record that is sent to VTAM by the client has the following general request format structure:

<table>
<thead>
<tr>
<th>Common Request/Response Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Triplet information: a single triplet is defined</td>
</tr>
<tr>
<td>- Offset from start of request header to first input section</td>
</tr>
<tr>
<td>- Length of each input section of this type</td>
</tr>
<tr>
<td>- Number of input sections of this type</td>
</tr>
</tbody>
</table>

Start of input information (offset from the start of the request header to this data indicated in the Input Triplet)

SNA network monitoring NMI response format

All SNA NMI responses flow on the socket from the VTAM server to the client application. The general format of an NMI response is as follows:

- The response header, which includes the response type, the return code and reason code, the request section descriptors (triplets), and the response section descriptors (quadruplets). A quadruplet consists of the offset (in bytes) of the response section relative to the beginning of the response header, a reserved field, the number of elements in the response section, and the total number of elements that passed the request filter checks.

  **Tip:** This last field in the quadruplet is applicable only to responses that have a corresponding request. Initialization and termination records do not have corresponding requests. Therefore, this field is reserved and is set to the value 0 on responses that contain initialization and termination records.

- The request sections.
- The response sections.
  - Response sections of the following solicited response types are returned if data is found that matches the corresponding filtered or unfiltered request (if no matches were found, no response data sections are returned):
    - EE connection information
    - EE summary information
    - HPR connection information
- CSM statistics information
- An initialization record always contains a single response section.
- A termination record does not contain a response section (all information is contained within the response header).

The NMI response section consists of one or more records that contain information that passed the request filter checks.

The general format of an NMI response section record is as follows:
- The record header, which contains the overall length of the record and one or more subrecord descriptors (triplets). The record triplet consists of the offset in bytes, relative to the start of the response section record, for the first instance of a given subrecord; the length in bytes of this particular subrecord; and the total number of instances of this subrecord.
- The subrecord sections that are associated with this response section record.

An application that navigates an NMI response must use the overall length value in the response section record to move to the next variable length record. The application should use the response section record triplet data to navigate within the record itself.

The following response section records are returned for the solicited response types:
- EE Summary Response
  1. One EE Summary Global Data Section record
  2. One or more EE Summary IP Address Data Section records
- EE Connection Response
  1. One or more EE Connection Data Section records
- HPR Connection Response
  1. One HPR Connection Global Data Section record
  2. One or more HPR Connection Specific Data Section records
- CSM Statistics Response
  1. One CSM Global Pool Output Section record
  2. One CSM Summary Output Section record
  3. Zero or one CSM Storage Owner Output Section record

Every response record sent by VTAM to the client looks like the format that follows.

<table>
<thead>
<tr>
<th>Common Request/Response Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Triplet information (copied from corresponding request, if any): a single triplet is defined.</td>
</tr>
<tr>
<td>Offset from start of response data to first input record</td>
</tr>
<tr>
<td>Length of each input section of this type</td>
</tr>
<tr>
<td>Number of input sections of this type</td>
</tr>
</tbody>
</table>
Output Quadruplet information: a single quadruplet is defined.
- Offset from start of response data to first output record
  - 0
- Number of output records included in this response. If this value is less than the number of records matching the filters supplied on the corresponding request (if any), then some data was not reported as the result of storage constraints.
- Number of output records matching the filters supplied on corresponding request, if any

Start of input information (copied from corresponding request, if any — offset from start of response data saved in Input Triplet)

Start of output information (offset from start of response data saved in Output Quadruplet)

**SNA network monitoring NMI request and response data structures and records**

The SNA network monitor request and response data structures for C/C++ and assembler programs are located as follows:

<table>
<thead>
<tr>
<th>Header files for C/C++ programs</th>
<th>Macros for assembler programs</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISTEEHNC</td>
<td>ISTEEHNA</td>
<td>The NMI request and response header, initialization record, and termination record structure definitions</td>
</tr>
<tr>
<td>ISTEESUC</td>
<td>ISTEESUA</td>
<td>The EE summary response data structure definitions</td>
</tr>
<tr>
<td>ISTEECOC</td>
<td>ISTEECOA</td>
<td>The EE connection response data structure definitions</td>
</tr>
<tr>
<td>ISTHPRCC</td>
<td>ISTITHPRCA</td>
<td>The HPR connection response data structure definitions</td>
</tr>
<tr>
<td>ISTCSMGC</td>
<td>ISTITCSMGA</td>
<td>The CSM statistics response data structure definitions</td>
</tr>
</tbody>
</table>

These header files and macros are included in SYS1.MACLIB. This data set must be available in the concatenation when compiling or assembling a part that makes use of these definitions. For an example of the mappings of the request and response data structures, see sample SNA network monitoring NMI mappings.

**SNA network monitoring NMI initialization record:** The structure of the initialization record follows.

**Enterprise Extender initialization record format**

<table>
<thead>
<tr>
<th>Common Request/Response Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Triplet information (no corresponding input request): a single triplet is defined.</td>
</tr>
<tr>
<td>- Offset from start of response data to first input section</td>
</tr>
<tr>
<td>- Length of each input section of this type: 0</td>
</tr>
<tr>
<td>- Number of input sections of this type: 0</td>
</tr>
</tbody>
</table>
Enterprise Extender initialization record format

Output Quadruplet information: a single quadruplet is defined.
• Offset from start of response data to first output record
  • 0
• Number of output records included in this response: 1
  • 0

Start of output information (offset from start of response data saved in Output Quadruplet), specifically one:
• Enterprise Extender initialization record

Record Identifier (4 characters): NMII

VTAM Level, from ATCVT (8 bytes)

TOD VTAM Started, from ATCVT (8 bytes)

SNA Network Management Component Name: SNAMGMT

Functions Supported (8 bits)
• IPv6 addresses supported (1 bit)
  – 0 = IPv6 addresses not supported
  – 1 = IPv6 addresses supported
• Local Hostname filter parameter supported (1 bit)
  – 0 = Local Hostname filter parameter not supported
  – 1 = Local Hostname filter parameter supported
• CSM Statistics filters supported (1 bit)
  – 0 = Filters are not accepted on the CSM Statistics request
  – 1 = Filters are accepted on the CSM Statistics request
• Reserved (5 bits): '00000'B

Reserved (15 bytes): 0

SNA network monitoring NMI termination record: The following table describes the structure of the termination record. The termination record contains no output data other than the return code and reason code in the response header.

Enterprise Extender termination record format

Common Request/Response Header

Input Triplet information (no corresponding input request): a single triplet is defined.
• Offset from start of response data to first input section
• Length of each input section of this type: 0
• Number of input sections of this type: 0

Output Quadruplet information: a single quadruplet is defined.
• Offset from start of response data to first output record
  • 0
• Number of output records included in this response: 0
  • 0

SNA network monitoring NMI EE summary response record: The structure of the EE Summary response follows.
Enterprise Extender Summary Response format

<table>
<thead>
<tr>
<th>Common Request/Response Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Triplet information (copied from request): a single triplet is defined.</td>
</tr>
<tr>
<td>• Offset from start of response data to first input section</td>
</tr>
<tr>
<td>• Length of each input section of this type</td>
</tr>
<tr>
<td>• Number of input sections of this type</td>
</tr>
</tbody>
</table>

| Output Quadruplet information: a single quadruplet is defined. |
| • Offset from start of response data to first output record |
| • 0 (since the records that follow are variable length records) |
| • Number of output records included in this response (if this value is less than number of records matching the filters supplied on the corresponding request, then some data was not reported due to storage constraints) |
| • Number of output records matching the filters supplied on the corresponding request |

Start of input information (copied from request, offset from start of response data saved in Input Triplet)

Start of output information (offset from start of response data saved in Output Quadruplet), specifically a collection of:

• Enterprise Extender Summary Global Output Record (one instance)
• One or more Enterprise Extender Summary IP Address Output Records (one instance per IP address being reported)

Enterprise Extender Summary Global Output Record

<table>
<thead>
<tr>
<th>Record Identifier (4 characters) — EESG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length of overall record (4 bytes)</td>
</tr>
<tr>
<td>Reserved field (2 characters)</td>
</tr>
<tr>
<td>Number of triplets for this output record (2 bytes): 1</td>
</tr>
</tbody>
</table>

Output Record Triplet information

• Offset from start of the record to first section of this type within the output record (4 bytes)
• Length of every section of this type within the output record (2 bytes)
• Number of output sections of this type within the output record (2 bytes)

Start of Enterprise Extender Summary static information section (one instance)

Enterprise Extender Summary IP Address Output Record

<table>
<thead>
<tr>
<th>Record Identifier (4 characters): EESI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length of overall record (4 bytes)</td>
</tr>
<tr>
<td>Reserved field (2 characters)</td>
</tr>
<tr>
<td>Number of triplets for this output record (2 bytes): 2</td>
</tr>
</tbody>
</table>

Output Record Triplet information

• Offset from start of the record to first section of this type within the output record (4 bytes)
• Length of every section of this type within the output record (2 bytes)
• Number of output sections of this type within the output record (2 bytes)

Start of Enterprise Extender Summary IP address information section (one instance)

Start of Enterprise Extender Summary Hostname information section (one per host name used to obtain this IP address, zero if no host name resolution was performed)
SNA network monitoring NMI EE connection response record: The structure of the response record is as follows:

<table>
<thead>
<tr>
<th>Common Request/Response Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Triplet information (copied from request): a single triplet is defined.</td>
</tr>
<tr>
<td>• Offset from start of response data to first input section</td>
</tr>
<tr>
<td>• Length of each input section of this type</td>
</tr>
<tr>
<td>• Number of input sections of this type</td>
</tr>
</tbody>
</table>

| Output Quadruplet information: a single quadruplet is defined. |
| • Offset from start of response data to first output record |
| • 0 |
| • Total number of output records |
| • Number of output records included in this response (if this value is not equal to total, then some data was not reported) |

Start of input information (copied from request, offset from start of response data saved in Input Triplet)

Start of output information (offset from start of response data saved in Output Quadruplet), specifically a collection of:

• One or more Enterprise Extender Connection Specific Output Records (one instance per EE connection reported)

Enterprise Extender Connection Specific Output Record

<table>
<thead>
<tr>
<th>Record Identifier (4 characters): EECO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length of overall record (4 bytes)</td>
</tr>
<tr>
<td>Reserved field (2 characters)</td>
</tr>
<tr>
<td>Number of triplets for this output record (2 bytes): 4</td>
</tr>
</tbody>
</table>

Output Record Triplet information

• Offset from start of the record to first section of this type within the output record (4 bytes)
• Length of every section of this type within the output record (2 bytes)
• Number of output sections of this type within the output record (2 bytes)

Start of Enterprise Extender Connection static information section (one instance)

Start of one or more Enterprise Extender Connection Hostname sections (0–2 possible instances, one for local and one for remote host name if applicable)

Start of Enterprise Extender Connection Associated VRN name section (one instance, included only if the EE connection is across a virtual routing node)

Start of one or more Enterprise Extender Connection Associated RTP PU name sections (one instance per RTP PU that is using this EE connection)

SNA network monitoring NMI HPR connection response record:

HPR Connection Response format

<table>
<thead>
<tr>
<th>Common Request/Response Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Triplet information (copied from request): a single triplet is defined.</td>
</tr>
<tr>
<td>• Offset from start of response data to first input section</td>
</tr>
<tr>
<td>• Length of each input section of this type</td>
</tr>
<tr>
<td>• Number of input sections of this type</td>
</tr>
</tbody>
</table>
HPR Connection Response format

Output Quadruplet information: a single quadruplet is defined.

- Offset from start of response data to first output record
- 0
- Number of output records included in this response (if this value is less than number of records matching the filters supplied on the corresponding request, then some data was not reported as the result of storage constraints)
- Number of output records matching the filters supplied on the corresponding request

Start of input information (copied from request, offset from start of response data saved in Input Triplet)

Start of output information (offset from start of response data saved in Output Quadruplet), specifically a collection of:

- HPR Connection Global Output Record (one instance)
- One or more HPR Connection Specific Output Records (one instance per HPR connection reported)

HPR Connection Global Output Record

<table>
<thead>
<tr>
<th>Record Identifier (4 characters): HPRG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length of overall record (4 bytes)</td>
</tr>
<tr>
<td>Reserved field (2 characters)</td>
</tr>
<tr>
<td>Number of triplets for this output record (2 bytes): 1</td>
</tr>
</tbody>
</table>

Output Record Triplet information

- Offset from start of the response data to first section of this type within the output record (4 bytes)
- Length of every section of this type within the output record (2 bytes)
- Number of output sections of this type within the output record (2 bytes)

Start of HPR Connection Global data

HPR Connection Specific Output Record

<table>
<thead>
<tr>
<th>Record Identifier (4 characters): HPRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length of overall record (4 bytes)</td>
</tr>
<tr>
<td>Reserved field (2 characters)</td>
</tr>
<tr>
<td>Number of triplets for this output record (2 bytes): 3</td>
</tr>
</tbody>
</table>

Output Record Triplet information

- Offset from start of the record to first section of this type within the output record (4 bytes)
- Length of every section of this type within the output record (2 bytes)
- Number of output sections of this type within the output record (2 bytes)

Start of HPR Connection static information section (one instance)

Start of HPR Connection Route Selection Control Vector (SNA Control Vector X'2B') section (one instance, potentially none if connection is in the process of performing a pathswitch)

Start of HPR Connection Pathswitch information section (present only if pathswitch had ever occurred on this connection, one instance if present)

SNA network monitoring NMI CSM statistics response record: The structure of the CSM Statistics response is as follows:
**CSM Statistics Response format**

<table>
<thead>
<tr>
<th>Common Request/Response Header</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Triplet information</strong> (copied from request): a single triplet is defined.</td>
</tr>
<tr>
<td>- Offset from start of response data to first input section</td>
</tr>
<tr>
<td>- Length of each input section of this type</td>
</tr>
<tr>
<td>- Number of input sections of this type</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output Quadruplet information: a single quadruplet is defined.</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Offset from start of response data to first output record</td>
</tr>
<tr>
<td>- 0</td>
</tr>
<tr>
<td>- Number of output records included in this response (if this value is less than number of records matching the filters supplied on the corresponding request, then some data was not reported as the result of storage constraints)</td>
</tr>
<tr>
<td>- Number of output records matching the filters supplied on the corresponding request</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Start of input information (copied from request, offset from start of response data saved in Input Triplet)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Start of output information (offset from start of response data saved in Output Quadruplet), specifically a collection of:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- CSM Global Pool Output Section record that contains multiple CSM Global Buffer Pool Data data records (CSMPoolGData), one per pool</td>
</tr>
<tr>
<td>- CSM Summary Output Section record that contains a single CSM Summary Data record (CSMSummGData) that represents CSM system-wide summary information</td>
</tr>
<tr>
<td>- (optionally) CSM Storage Owner Output Section record that contains one or more CSM Storage Owner Data records (CSMStorOData), one per reported owner</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CSM Global Pool Output Section Record</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Record Identifier (4 characters):</strong> CSMP</td>
</tr>
<tr>
<td>Length of overall record (4 bytes)</td>
</tr>
<tr>
<td>Reserved field (2 characters)</td>
</tr>
<tr>
<td>Number of triplets for this output record (2 bytes): 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output Record Triplet information</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Offset from start of the response data to first section of this type within the output record (4 bytes)</td>
</tr>
<tr>
<td>- Length of every section of this type within the output record (2 bytes)</td>
</tr>
<tr>
<td>- Number of output sections of this type within the output record (2 bytes)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Start of CSM Global Buffer Pool Data records (CSMPoolGData), one per CSM pool</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>CSM Summary Output Section Record</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Record Identifier (4 characters):</strong> CSMS</td>
</tr>
<tr>
<td>Length of overall record (4 bytes)</td>
</tr>
<tr>
<td>Reserved field (2 characters)</td>
</tr>
<tr>
<td>Number of triplets for this output record (2 bytes): 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output Record Triplet information</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Offset from start of the response data to first section of this type within the output record (4 bytes)</td>
</tr>
<tr>
<td>- Length of every section of this type within the output record (2 bytes)</td>
</tr>
<tr>
<td>- Number of output sections of this type within the output record (2 bytes)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Start of CSM Summary Data record (CSMSummGData), one single system wide record</th>
</tr>
</thead>
</table>
CSM Storage Owner Output Section Record

Record Identifier (4 characters): CSMO
Length of overall record (4 bytes)
Reserved field (2 characters)
Number of triplets for this output record (2 bytes): 1
Output Record Triplet information
- Offset from start of the response data to first section of this type within the output record (4 bytes)
- Length of every section of this type within the output record (2 bytes)
- Number of output sections of this type within the output record (2 bytes)

Start of CSM Storage Owner Data records (CSMStorOData), one per reported owner

NMI request errors

The following table describes the errors in an NMI request for which VTAM sends a termination record with the given return code and reason code, and then closes the connection.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EINVAL (121)</td>
<td>X'00007110'</td>
<td>Request header too short.</td>
</tr>
<tr>
<td>EINVAL (121)</td>
<td>X'00007111'</td>
<td>Unsupported version number in request header.</td>
</tr>
<tr>
<td>EINVAL (121)</td>
<td>X'00007112'</td>
<td>Triplet format is not valid; first request section is not contiguous to request header.</td>
</tr>
<tr>
<td>EINVAL (121)</td>
<td>X'00007112'</td>
<td>Triplet format is not valid; length of the filter element is insufficient for the request.</td>
</tr>
<tr>
<td>EINVAL (121)</td>
<td>X'00007113'</td>
<td>Length of request header plus length of request sections does not equal total length of request.</td>
</tr>
<tr>
<td>EINVAL (121)</td>
<td>X'00007114'</td>
<td>Eyecatcher in request header is not valid.</td>
</tr>
</tbody>
</table>

The following table describes the error in an NMI request for which VTAM returns a negative response of the same type as the request. VTAM leaves the connection active after returning the negative response for these errors.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EINVAL (121)</td>
<td>X'00007115'</td>
<td>Unrecognized request type.</td>
</tr>
<tr>
<td>EINVAL (121)</td>
<td>X'00007116'</td>
<td>Too many filter elements (request sections) included for request type.</td>
</tr>
<tr>
<td>EINVAL (121)</td>
<td>X'00007117'</td>
<td>Too few filter elements (request sections) included for request type.</td>
</tr>
<tr>
<td>EINVAL (121)</td>
<td>X'00007118'</td>
<td>Undefined filter parameter indicator set in filter element.</td>
</tr>
<tr>
<td>EINVAL (121)</td>
<td>X'00007119'</td>
<td>Required filter parameter missing from filter element.</td>
</tr>
<tr>
<td>EINVAL (121)</td>
<td>X'0000711A'</td>
<td>Unsupported filter parameter indicator set in filter element.</td>
</tr>
<tr>
<td>EINVAL (121)</td>
<td>X'0000711B'</td>
<td>Request not valid for HPR or EE information in a pure subarea VTAM node.</td>
</tr>
</tbody>
</table>

Chapter 14. Network management interfaces 633
Network management diagnosis

The interfaces that are described in this topic are designed to return error information as either a return_value, return_code, or reason_code, where applicable. The information in this section should be used to further diagnose the problem that is being reported.

When the return_value is -1, the return_code and reason_code indicate the problem that was incurred by the interface. See the section that describes the interface that is being used for return_value, return_code, and reason_code descriptions.

If you are not able to diagnose the problem using the returned error information, gather the following information that documents the error and contact IBM Customer Support.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local IPSec NMI</td>
<td>Collect a dump of the IKED address space.</td>
</tr>
<tr>
<td>Network security services (NSS) NMI</td>
<td>Collect a dump of the NSSD address space.</td>
</tr>
<tr>
<td>Real-time TCP/IP network monitoring NMI</td>
<td>Set the SYSTCPIP MISC trace as active. Collect a dump of the TCP/IP address space and data space.</td>
</tr>
<tr>
<td>Packet and data trace formatting NMI</td>
<td>Collect a dump of the TCP/IP address space and data space.</td>
</tr>
<tr>
<td>TCP/IP network management NMI (EZBNMIFR)</td>
<td>Collect a dump of the TCP/IP address space and data space.</td>
</tr>
<tr>
<td>SNA network monitoring NMI</td>
<td>Collect a dump of the VTAM address space.</td>
</tr>
</tbody>
</table>

File storage locations

The following table shows parts that are needed in order to compile Network management interface (NMI) applications and their locations. Your compiler should be configured to have access to these libraries.

<table>
<thead>
<tr>
<th>Function</th>
<th>File name</th>
<th>Type</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow applications to capture data packets</td>
<td>EZBYTMIA (1)</td>
<td>MACRO</td>
<td>SEZANMAC</td>
</tr>
<tr>
<td></td>
<td>EZBYTMIH (1)</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>Packet and data trace formatting NMI</td>
<td>EZBCTAPI</td>
<td>MACRO</td>
<td>SEZANMAC</td>
</tr>
<tr>
<td></td>
<td>EZBYPTO</td>
<td>MACRO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EZBYPTHA</td>
<td>MACRO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EZBCTHDR</td>
<td>MACRO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EZBYCTHH</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EZBYPPTHH</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>File name</td>
<td>Type</td>
<td>Library</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------</td>
<td>-----------------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>Allow applications to obtain TCP connection information</td>
<td>EZBYTMIA (1)</td>
<td>MACRO</td>
<td>SEZANMAC</td>
</tr>
<tr>
<td></td>
<td>EZBYTMIH (1)</td>
<td>H</td>
<td>SEZANMAC</td>
</tr>
<tr>
<td></td>
<td>EZASMF77</td>
<td>MACRO</td>
<td>SYS1.MACLIB</td>
</tr>
<tr>
<td></td>
<td>EZASMF (2)</td>
<td>H</td>
<td>SEZANMAC</td>
</tr>
<tr>
<td>Allow applications to obtain IPSec information</td>
<td>EZASMF77</td>
<td>MACRO</td>
<td>SYS1.MACLIB</td>
</tr>
<tr>
<td></td>
<td>EZASMF (2)</td>
<td>H</td>
<td>SEZANMAC</td>
</tr>
<tr>
<td>TCP/IP callable NMI to retrieve local TCP and UDP endpoint data</td>
<td>EZBNMRHA</td>
<td>MACRO</td>
<td>SEZANMAC</td>
</tr>
<tr>
<td></td>
<td>EZBNMRHC (2)</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>TCP/IP callable NMI to retrieve new TCP/IP storage statistics details</td>
<td>EZBNMRHA</td>
<td>MACRO</td>
<td>SEZANMAC</td>
</tr>
<tr>
<td></td>
<td>EZBNMRHC (2)</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>Enterprise Extender Network Management</td>
<td>ISTEZHNC</td>
<td>H</td>
<td>SYS1.MACLIB</td>
</tr>
<tr>
<td></td>
<td>ISTEESUC</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ISTEECOC</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ISTHPRCC</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ISTCSMGCC</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ISTEZHNA</td>
<td>MACRO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ISTEESUA</td>
<td>MACRO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ISTEECOAH</td>
<td>MACRO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ISTHPRCA</td>
<td>MACRO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ISTCSMGCA</td>
<td>MACRO</td>
<td></td>
</tr>
<tr>
<td>Real-time TCP/IP network monitoring NMI</td>
<td>EZBYTMIA (1)</td>
<td>MACRO</td>
<td>SEZANMAC</td>
</tr>
<tr>
<td></td>
<td>EZBYTMIH (1)</td>
<td>H</td>
<td>SEZANMAC</td>
</tr>
<tr>
<td></td>
<td>EZANMFTA</td>
<td>MACRO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EZANMFTC</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>Local IPSec NMI</td>
<td>EZBNMSEA</td>
<td>MACRO</td>
<td>SEZANMAC</td>
</tr>
<tr>
<td>Local IPSec NMI</td>
<td>EZBNMSEC</td>
<td>H</td>
<td>/usr/include</td>
</tr>
<tr>
<td>Network security services (NSS) NMI</td>
<td>EZBNMSEA</td>
<td>MACRO</td>
<td>SEZANMAC</td>
</tr>
<tr>
<td>Network security services (NSS) NMI</td>
<td>EZBNMSEC</td>
<td>H</td>
<td>/usr/include</td>
</tr>
</tbody>
</table>

(1) Part used for multiple functions.

(2) These parts require the XL C/C++ Run-Time functions, macros, and header files.
Chapter 15. Application Transparent Transport Layer Security (AT-TLS)

Application Transparent Transport Layer Security (AT-TLS) creates a secure session on behalf of an application. Instead of implementing TLS in every application that requires a secure connection, AT-TLS provides encryption and decryption of data based on policy statements that are coded in the Policy Agent. The application sends and receives cleartext (unencrypted data) as usual while AT-TLS encrypts and decrypts data at the TCP transport layer. For more information about AT-TLS and AT-TLS policy setup, see the Application Transparent Transport Layer Security (AT-TLS) information in the z/OS Communications Server: IP Configuration Guide and the Policy Agent information in the z/OS Communications Server: IP Configuration Reference.

Most applications do not need any awareness of the security negotiations and encryption that is done by TCP/IP on its behalf. However, you might want some applications to be aware of AT-TLS or have control over the security functions that are being performed by TCP/IP. For example, if the application is a server requesting client authentication, you might want the application to get the partner certificate or the user ID associated with the partner certificate. Or the application might negotiate in cleartext with its partner to decide whether a secure session is necessary. If both agree to a secure session, then the application needs to tell AT-TLS to set up a secure session. The SIOCTTLSCTL ioctl provides the interface for the application to query or control AT-TLS.

Applications that are taking advantage of AT-TLS can be separated into three different types (basic, aware and controlling) as described in Table 99. An application’s type is based on whether an awareness of the service is needed and, if so, the amount of control that the application is given over the security functions. Basic applications are unchanged. Aware applications are changed to invoke the SIOCTTLSCTL ioctl to query a socket about AT-TLS status using a TTLSi_Req_Type value of TTLS_QUERY_ONLY or TTLS_RETURN_CERTIFICATE. Controlling applications are changed to invoke the SIOCTTLSCTL ioctl to control the secure session on a socket using a TTLSi_Req_Type value of TTLS_INIT_CONNECTION, TTLS_RESET_SESSION or TTLS_RESET_CIPHER.

<table>
<thead>
<tr>
<th>Application type</th>
<th>SIOCTTLSCTL ioctl calls issued</th>
<th>ApplicationControlled setting in AT-TLS policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>application does not issue any AT-TLS ioctl calls</td>
<td>Off</td>
</tr>
<tr>
<td>Aware</td>
<td>query requests</td>
<td>Off</td>
</tr>
<tr>
<td>Controlling</td>
<td>query and control requests</td>
<td>On</td>
</tr>
</tbody>
</table>

- A basic application is unaware that AT-TLS is performing encryption or decryption of data. Most applications can match this model.
- An aware application is aware of AT-TLS and can query information such as AT-TLS status, partner certificate, and derived RACF user ID without any advanced setting in AT-TLS policy. A server that requires a RACF user ID derived from a partner certificate matches this model.
A controlling application is aware of AT-TLS and needs to control the secure session. It must have the ApplicationControlled parameter in AT-TLS policy set to ON. Any application that must control when the initial handshake is done or when sessions or ciphers must be reset matches this model.

The SIOCTTLSCTL ioctl blocks during the initial handshake if the socket is in blocking mode. If the socket is non-blocking, SIOCTTLSCTL returns EWouldBlock during the initial handshake.

Applications that use non-blocking sockets can use the select function to wait for the socket to become writable. When the socket becomes writable, the initial handshake is complete.

The following APIs are supported by AT-TLS:

- Macro API (EZASMI)
- CALL instruction API (EZASOKET) supporting COBOL, PL/I, and System/370™ assembler languages
- REXX socket API
- Language Environment C socket call [ioctl()]
- UNIX System Services Assembler Callable Service (BPX1IOC or BPX4IOC)
- CICS® C socket calls
- CICS CALL instruction API (EZASOKET - by including EZACICAL or EZACICSO)
- IMS™ CALL instruction API (EZASOKET)

Restrictions: The following APIs are not supported by AT-TLS:

- TCP C Socket API
- X/Open Transport Interface (XTI)
- Pascal API

CICS transaction considerations

CICS transaction security environments are not visible to AT-TLS support. The CICS job and all of its transactions appear to the stack as a single server application with a single z/OS UNIX callable services process ID running in the security environment of the CICS job. Connections established, whether active or passive, can perform TLS handshake processing as either CLIENT or SERVER. All of the connections that are established by a single CICS job are able to share the Session ID cache in the SSL environment. The CICS job should use a private keyring with a server certificate. The keyring used must contain the chain of root certificates needed to validate the server certificate it presents to the client. If the server requires the CLIENT AUTHENTICATION call, it must also have any other root certificates necessary to validate presented client certificates on its keyring.

TCP/IP CICS Socket Support provides a Listener transaction that has a configuration option to get the client’s certificate-associated user ID. When this option is configured, the Listener waits for the TLS handshake to complete on the accepted connection (select for write) and then uses the SIOCTTLSCTL ioctl to see whether an associated user ID is present. A user ID is present when the HandshakeRole parameter is defined in AT-TLS policy as ServerWithClientAuth, the client passed in a certificate, and the certificate was registered with RACF with an associated user ID. This user ID is passed into the Listener security exit, if one
is configured. The security exit can remove or change the user ID. The Listener then starts the transaction to process the connection under this user ID.

A CICS transaction that participates in a TLS handshake as CLIENT when the server requests CLIENT AUTHENTICATION presents a certificate identifying the CICS job, not the transaction user.

See the Application Transparent Transport Layer Security (AT-TLS) information in the z/OS Communications Server: IP CICS Sockets Guide for more information on configuring TCP/IP CICS socket support.

### Using the SIOCTTLSCTL ioctl

An application uses the SIOCTTLSCTL ioctl to query AT-TLS information for a connection and to control the use of AT-TLS on a connection.

#### Starting AT-TLS on a connection

Use the SIOCTTLSCTL ioctl with option TTLS_INIT_CONNECTION to start AT-TLS on a connection. This starts the SSL handshake. If using non-blocking sockets, the server can wait for the handshake to complete by waiting for the socket to become writable. If using blocking sockets, the ioctl blocks until the handshake is complete. If the handshake times out or fails for any reason, the connection is reset.

Some server applications need to support some clients using cleartext security negotiation and other clients using implicit security. This means that the SSL handshake starts as soon as the connection is established with the server. For server applications that support both types of clients, the TTLS_ALLOW_HSTIMEOUT option is helpful. This option enables the server to request an SSL handshake and keep the TCP connection active if the SSL handshake times out. This option is most effective if the server normally sends data to the client first. The server application must request both the TTLS_INIT_CONNECTION and the TTLS_ALLOW_HSTIMEOUT option on the SIOCTTLSCTL start handshake request to keep the connection active after an SSL handshake timeout.

The server application waits for the SSL handshake to complete, either by blocking the socket or by waiting for the socket to become writable. After the handshake completes, the server application can check the SIOCTTLSCTL status to determine the state of the connection, the protocol and cipher used, and other information. If a non-blocking socket is used, the final status is queried by issuing another SIOCTTLSCTL ioctl with option TTLS_QUERY_ONLY. If a blocking socket is used, the final status is contained in the returned SIOCTTLSCTL. Ensure that your server application checks the SIOCTTLSCTL status and takes appropriate action based on the returned status.

**Restriction:** The TTLS_ALLOW_HSTIMEOUT option is supported only when the HandshakeRole value is Server or ServerWithClientAuth and the HandshakeTimer value is nonzero.

#### Stopping AT-TLS on a connection

Use the SIOCTTLSCTL ioctl with option TTLS_STOP_CONNECTION to stop secure traffic on the TCP connection. The SSL session ends on the connection and the TCP connection returns to cleartext communication. The connection retains the
policy mapping, but the connection is in the same state as before a SIOCTTLSCTL ioctl with the TTLS_INIT_CONNECTION option was issued.

Applications that negotiate security can use this option to stop the secure connection. For example, an application negotiates, using cleartext, that a secure session needs to be established. Later, the application performs a separate negotiation to stop the secure connection. After both sides agree to stop security, the application issues the SIOCTTLSCTL ioctl with option TTLS_STOP_CONNECTION. The application must clear all application data from the connection before issuing the TTLS_STOP_CONNECTION request. The connection is reset if any unread application data exists when the application issues the TTLS_STOP_CONNECTION request. If nonblocking sockets are used, the application can wait for the request to complete by waiting for the socket to become writable. If blocking sockets are used, the ioctl blocks until the request is complete. After the request completes, the connection state is NONSECURE.

Restriction: The TTLS_STOP_CONNECTION option cannot be used on SSLv2 connections.

Tip: Do not use the TTLS_STOP_CONNECTION option if the application is not going to send or receive any clear text data after the request completes. AT-TLS closes the SSL session when the application closes the TCP socket.

Requesting AT-TLS queries and additional functions

Use the TTLSHeader structure pointed to by the TTLSi_BufferPtr pointer to query additional information for the secure connection. The TTLSHeader structure can be used to obtain the TTLSRule, TTLSGroupAction, TTLESvironmentAction, TTLSConnectionAction names, and the partner certificate. The application can also provide a host name that is validated against the host name in the partner's certificate.

Steps for implementing an aware server application

To implement an aware server application, create or update the server application as follows:

1. If the server is using non-blocking sockets, the server should issue select on the new socket to wait for the socket to become writable, which indicates that the initial handshake is complete. If using blocking sockets, the select is not needed.

2. When the new socket is writeable the server can issue the SIOCTTLSCTL ioctl with TTLSi_Req_Type set to TTLS_RETURN_CERTIFICATE to retrieve the certificate presented by the client (if provided). The ioctl should return with a policy status of TTLS_POL_ENABLED and a connection status of TTLS_CONN_SECURE. The server program can examine the negotiated session attributes and the certificate that is supplied by the client (if provided). If this certificate is registered with the security product and associated with a user ID, then the user ID fields are also returned in the ioctl data.

Steps for implementing a controlling server application

To implement a simple aware and controlling application as a server, create or update the server application as follows:
1. When a new connection is accepted, the server should issue an SIOCTTLSCTL ioctl with TTLSi_Req_Type value set to TTLS_QUERY_ONLY to verify that policy is correctly set up for this connection. The ioctl should return a policy status of TTLS_POL_APPLCNTRL and a connection status of TTLS_CONN_NOTSECURE. This means that the security of the connection is application-controlled and that the connection is not yet secure. If any other status is returned, the application cannot initiate a secure session for the connection. See "Coding the SIOCTTLSCTL ioctl" on page 643 for an explanation of all status values. If you are sure the connection will be set for application control, this step can be omitted.

2. The server and client send and receive cleartext data to negotiate the use of TLS. The negotiation protocol is the responsibility of the applications and is not performed by the stack. Ensure that your negotiation protocol causes all cleartext data to be read on both ends before continuing.

3. If both sides agree to use a secure connection, the server should issue an SIOCTTLSCTL ioctl with TTLSi_Req_Type value set to TTLS_INIT_CONNECTION to start the handshake. The client must also initiate a secure connection at this time.

4. If using non-blocking sockets, the server can wait for the handshake to complete by waiting for the socket to become writable. If using blocking sockets, the ioctl will block until the handshake is complete.

5. If the server wants to verify that the session is now secure, it can issue an SIOCTTLSCTL ioctl with TTLSi_Req_Type value set to TTLS_QUERY_ONLY to retrieve the negotiated session attributes. The ioctl should return a connection status of TTLS_CONN_SECURE along with additional information. To retrieve the certificate that is presented by the client (if one is provided), use the TTLS buffer with a TTLSK_Certificate Get request. If this certificate is registered with the security product and associated with a user ID, then the user ID fields are also returned in the ioctl data.

**Steps for starting an aware or controlling server application**

To start an aware or controlling server application, perform the following steps:

1. Code or modify existing AT-TLS policy to cover the TCP server port. Ensure that the HandshakeRole parameter in the rule is set to Server or ServerWithClientAuth. The action should specify one of the following:
   - ApplicationControlled Off for an aware server application (this is also the default).
   - ApplicationControlled On for a controlling server application.

2. Install the policy to the appropriate TCP stack or stacks using Policy Agent. If any policy errors are reported, you must correct them and reinstall the policy.

3. Issue the `pasearch` command to verify the policy rule and actions for the server.
4. Start the aware or controlling server application.

The client and server exchange data, which to them is cleartext, but which is automatically encrypted by AT-TLS.

If at some point a controlling server application should need to reset the secure session or the current cipher, it can issue an SIOCTTLSCTL ioctl with the TTLSi_Req_Type value set to TTLS_RESET_SESSION or TTLS_RESET_CIPHER.

In addition to connection status and policy status, the SIOCTTLSCTL ioctl also provides the following secure session attributes after the handshake completes: security type, SSL protocol in use, negotiated cipher in use, certificate associated user ID, and, if requested, partner certificate information. The certificate associated user ID is available when the HandshakeRole parameter is specified in AT-TLS policy as ServerWithClientAuth and the partner-supplied certificate has an associated user ID.

Additional SIOCTTLSCTL request type options are available to reset an SSL session ID so that it is not reused by this or another connection and to reset the cipher used by the SSL session. AT-TLS policy also provides ways for these resets to occur automatically based on elapsed time. Resets requested by issuing the SIOCTTLSCTL ioctl reset the beginning of the elapsed time intervals specified in the policy.

\[\text{Figure 32: SIOCTTLSCTL with TTLS_Query_Only}\] and \[\text{Figure 33 on page 643}\] and \[\text{Figure 34 on page 643}\] show when the SIOCTTLSCTL ioctl can be issued and which AT-TLS functions are performed based on the ioctl call. The SIOCTTLSCTL ioctl initiates AT-TLS obtaining the user ID and job name of the requesting application and performing an AT-TLS rule search if these steps have not already been performed. An AT-TLS policy lookup assigns a rule and actions to the connection if a match is found. An SIOCTTLSCTL ioctl with a TTLS_Init_Connection request type initiates a System SSL environment search or creates a new environment and initiates the SSL connection (which consists of the SSL handshake). Policy and connection status fields are returned on all SIOCTTLSCTL ioctls.
Coding the SIOCTTLSCTL ioctl

General coding guidelines for the sockets ioctl calls can be found in the following publications:

- **z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference**
  - Macro API (EZASMI) for assembler programs
  - CALL instruction API (EZASOKET) supporting COBOL, PL/I, and System/370 assembler languages
  - REXX socket API
- **z/OS Communications Server: IP CICS Sockets Guide**
  - CICS C socket calls (EZACIC07 or EZACIC17, which calls EZASOKET with entry in EZACICAL)
  - CICS CALL instruction API (EZASOKET with entry in EZACICAL) supporting COBOL, PL/I, and System/370 assembler languages
- **z/OS Communications Server: IP IMS Sockets Guide**
  - IMS CALL instruction API (EZASOKET) supporting COBOL, PL/I, and System/370 assembler languages
- **z/OS XL C/C++ Run-Time Library Reference**
  - z/OS IBM C/C++ sockets API within the z/OS Language Environment
- **z/OS UNIX System Services Programming: Assembler Callable Services Reference**
  - Assembler Callable Services (BPX1IOC or BPX4IOC)
Each programming language has its own control block structure mapping. All mappings and header files are stored in SEZANMAC and the C language headers are also installed in file system directory /usr/include. The following programming languages are supported:

**Assembler**
Include EZBZTLSP mapping. See the [z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference](https://www.ibm.com/support/docview.zhtml?docid=507009) for coding an ioctl call in assembler or coding an ioctl call for a callable API.

See the [z/OS UNIX System Services Programming: Assembler Callable Services Reference](https://www.ibm.com/support/docview.zhtml?docid=507009) for coding BPX1IOC or BPX4IOC.

**PL/I**

**COBOL**

**REXX**
No mapping or header file used. See the [z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference](https://www.ibm.com/support/docview.zhtml?docid=507009) for coding an ioctl call in REXX.

**C**
Include EZBZTLSLC header file, which is installed in SEZANMAC and in the file system directory /usr/include. See the [z/OS XL C/C++ Run-Time Library Reference](https://www.ibm.com/support/docview.zhtml?docid=507009) or [z/OS Communications Server: IP CICS Sockets Guide](https://www.ibm.com/support/docview.zhtml?docid=507009) for coding an ioctl call in C.

See the control block structures in SEZANMAC and in the /usr/include directory for variable names and locations and their enumerated values.

- All ioctl calls must set the Version field. See the control block structures in SEZANMAC for constants declared for these variables.

  **Guideline:** The TTLS_CURRENT_VERSION constant in EZBZTLSC.h is being deprecated and will remain defined as 1. Use a specific TTLS version level constant, such TTLS_VERSION2, to set the TTLSi_Ver level when coding SIOCTTLSCTL ioctl requests.

- Any field not used must be set to 0.

- If the additional buffer is required for the SIOCTTLSCTL ioctl, the buffer pointer is specified in TTLSi_BufferPtr and must point to the beginning of the buffer. The length of the buffer area is specified in TTLSi_BufferLen. Obtain enough buffer storage to hold the returned data. The buffer area can be part of the storage obtained for the base ioctl or it can be stand-alone storage. If the buffer is not large enough, the variable errno indicates the value ENOBUFFS with the required buffer size specified in TTLSi_Cert_Len.

**SIOCTTLSCTL (X’C038D90B’)**
The SIOCTTLSCTL ioctl provides an interface for the application to query and control AT-TLS for the connection. The following data items are returned by the SIOCTTLSCTL ioctl:

**TTLSi_Stat_Policy**
Indicates the level of AT-TLS enablement for the connection. Possible values include the following:
TTLS_POL_OFF (1)
AT-TLS was not enabled on the stack when AT-TLS policy mapping was performed for the connection.

TTLS_POL_NO_POLICY (2)
No matching policy rule was found when AT-TLS policy mapping was performed for the connection. The application should issue a message, if appropriate, directing the system administrator to create a policy rule that matches this connection.

TTLS_POL_NOT_ENABLED (3)
The policy rule that matches this connection indicates that AT-TLS should not be used. The application should issue a message, if appropriate, directing the system administrator to change the policy rule for this connection.

TTLS_POL_ENABLED (4)
AT-TLS is enabled for this connection, but application control has not been granted. The application should issue a message, if appropriate, directing the system administrator to change the policy rule to enable application control for this connection.

TTLS_POL_APPLCNTRL (5)
AT-TLS is enabled and is application controllable.

TTLSi_Stat_Conn
Indicates the current level of secure session on the connection.

TTLS_CONN_NOTSECURE (1)
The connection does not have a secure session established.

TTLS_CONN_HS_INPROGRESS (2)
Connection initial handshake in progress.

TTLS_CONN_SECURE (3)
The connection has a secure session.

In addition to connection status and policy status, the SIOCTTLSCTL ioctl also provides the following connection information, when available:

TTLSi_Sec_Type
Indicates the security type for the connection if AT-TLS policy is defined for the connection. Valid values are:

TTLS_SEC_UNKNOWN (0)
The connection does not have a secure session established.

TTLS_SEC_CLIENT (1)
The security type is Client.

TTLS_SEC_SERVER (2)
The security type is Server.

TTLS_SEC_SRV_CA_PASS (3)
The security type is Server with Client Authentication. Client Authentication Type is PassThru.

TTLS_SEC_SRV_CA_FULL (4)
The security type is Server with Client Authentication. Client Authentication Type is Full.
**TTLS_SEC_SRV_CA_REQD (5)**
The security type is Server with Client Authentication. Client Authentication Type is Required.

**TTLS_SEC_SRV_CA_SAFCHK (6)**
The security type is Server with Client Authentication. Client Authentication Type is SAFCheck.

**TTLSi_SSL_Prot**
Indicates the SSL protocol that is in use for the connection if the connection is secure. Valid values are:

- **TTLS_PROT_UNKNOWN (0x0000)**
  The connection does not have a secure session established.

- **TTLS_PROT_SSLV2 (0x0200)**
  SSL version 2 is in use.

- **TTLS_PROT_SSLV3 (0x0300)**
  SSL version 3 is in use.

- **TTLS_PROT_TLSV1 (0x0301)**
  TLS version 1.0 is in use.

- **TTLS_PROT_TLSV1_1 (0x0302)**
  TLS version 1.1 is in use.

**TTLSi_FIPS140**
Indicates whether Federal Information Processing Standard (FIPS) 140 is in effect. Valid values are:

- **TTLS_FIPS140_OFF (0x00)**
  FIPS 140 support is not in effect.

- **TTLS_FIPS140_ON (0x01)**
  FIPS 140 support is in effect.

**TTLSi_Neg_Cipher**
Indicates the cipher in use for the connection if the connection is secure. For the list of cipher suites that are supported, see the [gsk_environment_open()](https://www.ibm.com) API information in the z/OS Cryptographic Services System SSL Programming.

**TTLSi_UserID/TTLSi_UserID_Len**
**TTLSi_UserID** is a null terminated character string.
**TTLSi_UserID_Len** indicates the number of characters returned prior to the first null.

These fields are returned when the HandshakeRole parameter is specified as ServerWithClientAuth, the client provides a valid certificate, and the certificate is associated with a user ID in the DIGTCERT General Resource Class. See the RACF and Digital Certificates topic in the z/OS Security Server RACF Security Administrator’s Guide for more information on associating user IDs with certificates.

**TTLSi_Cert_Len**
Indicates the size of the partner’s certificate if the connection is secure and a certificate was supplied during negotiation.

If the TTLS_RETURN_CERTIFICATE request type is specified on the SIOCCTTLSCTL ioctl and the partner certificate is known, the certificate is returned in the additional buffer provided (using fields **TTLSi_BufferPtr** and }
TTLSi_BufferLen) with the ioctl call. The length of the returned certificate is returned in TTLSi_Cert_Len. If the buffer provided is not large enough to hold the certificate, then the variable errno indicates the value ENOBUFS and the required buffer size is returned in TTLSi_Cert_Len.

The following optional behaviors can be requested on the SIOCTTLSCTL ioctl using the TTLSi_Req_Type field:

**TTLS_QUERY_ONLY (0x0000)**
Query the connection status. If more advanced query information is required, use the optional buffer that includes the TTLSHeader control block.

**Restriction:** The TTLS_QUERY_ONLY option must be specified alone without any other request option.

**TTLS_RETURN_CERTIFICATE (0x0001)**
Return the partner certificate used for authentication if it is available.

**Restriction:** This request is not valid with TTLS Version 2. For TTLS version 2, use the TTLSHeader structure to request the partner certificate.

**TTLS_INIT_CONNECTION (0x0002)**
Initialize the secure SSL connection using the role defined by the HandshakeRole parameter.

**Restriction:** The connection must be application controlled to use this request.

**TTLS_RESET_SESSION (0x0004)**
Reset a session ID to avoid its reuse by another connection.

**Restriction:** The connection must be application controlled to use this request.

**TTLS_RESET_CIPHER (0x0008)**
Reset and renegotiate the cipher used for the secure session. If the session ID has timed out or has been reset, a full handshake is performed. Otherwise, a short handshake is performed.

**Restriction:** The connection must be application controlled to use this request.

**TTLS_STOP_CONNECTION (0x0010)**
Close the SSL connection. Data will no longer be encrypted or decrypted on the connection. The state of the TCP connection is unchanged.

**Restrictions:**
- The application must read all secure application data before the TTLS_STOP_CONNECTION request is issued. The connection is reset if any secure application data exists when the TTLS_STOP_CONNECTION request is issued.
- The connection must be application controlled to use this request.

**TTLS_ALLOW_HSTIMEOUT (0x0020)**
Allow the TCP connection to remain active if the SSL handshake fails because no data was received from the client. The timeout value is determined by the HandshakeTimeout value from the policy. The HandshakeTimeout must be a nonzero value. This option is valid only if the HandshakeRole value is Server or ServerWithClientAuth. Any non-SSL data received ends the handshake request and leaves the connection state nonsecure.
Restriction: The TTLS_INIT_CONNECTION option must be specified when the TTLS_ALLOW_HSTIMEOUT option is requested.

Tip: Use this option for servers that send the first application data to the client and must support SSL and non-SSL clients on the same port.

Using the TTLSHeader control block
The TTLSHeader control block extends the SIOCTTLSCTL ioctl. You can use Get requests to obtain additional information about the AT-TLS connection. You can use Set requests to change the AT-TLS behavior for the connection. TTLS Verion 2 is required to use the TTLSHeader control block. The TTLSHeader control block includes a fixed section and a variable length self-defining section that contains Set and Get requests followed by the set and get data area. The set data area is not overlaid with the Get request data when the SIOCTTLSCTL returns.

Rule: Set request quadruplets must immediately follow the TTLSHeader control block in the buffer, followed by Get request quadruplets. The data for the Set requests must follow the set and get quadruplets. The data area for the Get requests must follow the set data area.

Table 100. TTLSHeader control block structure

<table>
<thead>
<tr>
<th>TTLSHeader fixed section</th>
<th>TTLSHeader optional Set request quadruplets</th>
<th>TTLSHeader optional Get request quadruplets</th>
<th>Beginning of set data area</th>
<th>Beginning of get data area</th>
</tr>
</thead>
</table>

TTLSHeader fixed section: The TTLSHeader fixed section contains control information that describes the Set and Get requests that follow. The fixed section is defined by the TTLSHeader structure as shown in Table 101.

Table 101. TTLSHeader fixed section

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTLSHeaderIdent</td>
<td>0</td>
<td>8 bytes</td>
<td>EBCDIC</td>
<td>TTLSHeader identifier; set to TTLSHdr_Ident(EBCDIC 'TTLSHDR' )</td>
</tr>
<tr>
<td>TTLSHdr_Rsvd1</td>
<td>8</td>
<td>8 bytes</td>
<td>Binary</td>
<td>Reserved; set to 0</td>
</tr>
<tr>
<td>TTLSHdrgetBytesNeeded</td>
<td>16</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Length of the buffer needed to contain TTLSHeader, TTLSQuadruplet structures, data for Set requests, and data for Get requests. This value is set on return if all data does not fit in the buffer provided</td>
</tr>
<tr>
<td>TTLSHdr_SetCount</td>
<td>20</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Number of Set requests in the buffer. Each Set request is represented by a TTLSQuadruplet</td>
</tr>
<tr>
<td>TTLSHdr_GetCount</td>
<td>24</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Number of Get requests in the buffer. Each Get request is represented by a TTLSQuadruplet</td>
</tr>
<tr>
<td>TTLSHdr_Rsvd2</td>
<td>28</td>
<td>16 bytes</td>
<td>Binary</td>
<td>Reserved; set to 0</td>
</tr>
</tbody>
</table>

TTLSHeader variable length structure: The variable length section is used to request options, such as the following:

- Validating a host name and get options
- Returning the partner certificate
Each Set or Get request is defined by a TTLSQuadruplet structure as shown in Table 102.

**Table 102. TTLSQuadruplet structure**

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTLSQ_Key</td>
<td>0</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Constant identifying the request</td>
</tr>
<tr>
<td>TTLSQ_Offset</td>
<td>4</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Offset to the first value for the request, measured in bytes from the start of the TTLSHeader structure. For Set requests, this must be a nonzero value. For Get requests, this value must be 0. On return, this value is nonzero if data was returned for this request.</td>
</tr>
<tr>
<td>TTLSQ_Length</td>
<td>8</td>
<td>4 bytes</td>
<td>Binary</td>
<td>On input, this is the length of the value for a Set request or has the value 0 for a Get request. On return for a Get request, this value is the length of the data that is returned.</td>
</tr>
<tr>
<td>TTLSQ_Rcode</td>
<td>12</td>
<td>4 bytes</td>
<td>Binary</td>
<td>Return code of the request operation. See each request operation for the possible return codes.</td>
</tr>
</tbody>
</table>

**Get request**: Use the Get request to obtain additional information about the AT-TLS connection.

**Table 103. Get request structure**

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Constant</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTLSK_Host_Status</td>
<td>4000</td>
<td>1 byte</td>
<td>Binary</td>
<td>Validates the host name that is provided with the partner certificate. The host name must end with a null character. The TTLSQ_Offset field must be set to the value of the start of the host name that is in the buffer. The following values can be returned in the TTLSQ_Rcode field:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0 The host name has successfully validated against the partner certificate.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1 Partner certificate is not available.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2 Host name did not match the name in the partner certificate.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3 Host name validation failed with an unexpected gsk_validate_hostname value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4 Host name validation failed with an unexpected gsk_decode_certificate value.</td>
</tr>
</tbody>
</table>

Chapter 15. Application Transparent Transport Layer Security (AT-TLS)  649
Table 103. Get request structure (continued)

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Constant</th>
<th>Length</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTLSK_Certificate</td>
<td>4001</td>
<td>Unknown</td>
<td>Binary</td>
<td>Returns the partner certificate. The certificate length is not known until the secure connection is established. The value of the TTLSi_Cert_Len field in the SIOCTTLSCTL structure can be used to determine the certificate length when the secure connection is complete. The following values can be returned in TTLSQ_Rcode: 0 The request completed successfully.</td>
</tr>
<tr>
<td>TTLSK_TTLSRule_Name</td>
<td>4002</td>
<td>48 bytes, ending with a null character</td>
<td>EBCDIC</td>
<td>Returns the name of the TTLSRule field that is mapped to the connection. The following value can be returned in TTLSQ_Rcode: 0 The request completed successfully.</td>
</tr>
<tr>
<td>TTLSK_GroupAction_Name</td>
<td>4003</td>
<td>48 bytes, ending with a null character</td>
<td>EBCDIC</td>
<td>Returns the name of the TTLSGroupAction field that is mapped to the connection. The following value can be returned in TTLSQ_Rcode: 0 The request completed successfully.</td>
</tr>
<tr>
<td>TTLSK_EnvironmentAction_Name</td>
<td>4004</td>
<td>48 bytes, ending with a null character</td>
<td>EBCDIC</td>
<td>Returns the name of the TTLSEnvironmentAction field that is mapped to the connection. The following value can be returned in TTLSQ_Rcode: 0 The request completed successfully.</td>
</tr>
<tr>
<td>TTLSK_ConnectionAction_Name</td>
<td>4005</td>
<td>48 bytes, ending with a null character</td>
<td>EBCDIC</td>
<td>Returns the name of the TTLSConnectionAction field that is mapped to the connection. The following value can be returned in TTLSQ_Rcode: 0 The request completed successfully.</td>
</tr>
</tbody>
</table>

For example, assume that an application made a secure connection to a server. However, the application needs to verify that the certificate is from the server the application is connected to. The application has two known server host names, mvs.telnet.raleigh.ibm.com and mvs.prod.rtp.ibm.com. The application would use the following TTLSHeader structure, pointed to by the TTLSI_BufPtr pointer on the SIOCTTLSCTL ioctl request to validate the server’s certificate against these host names:

Table 104. Example TTLSHeader structure

<table>
<thead>
<tr>
<th>TTLSHeader</th>
<th>Offset</th>
<th>Format</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TTLSHeaderIdent</td>
<td>0</td>
<td>EBCDIC</td>
<td>TTLSHDR</td>
</tr>
</tbody>
</table>
### Table 104. Example TTLSHeader structure (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Format</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTLSHdrRsvd1</td>
<td>8</td>
<td>Binary</td>
<td>00000000 00000000</td>
</tr>
<tr>
<td>TTLSHdrBytesNeeded</td>
<td>16</td>
<td>Binary</td>
<td>00000000</td>
</tr>
<tr>
<td>TTLSHdrSetCount</td>
<td>20</td>
<td>Binary</td>
<td>00000000</td>
</tr>
<tr>
<td>TTLSHdrGetCount</td>
<td>24</td>
<td>Binary</td>
<td>00000002</td>
</tr>
<tr>
<td>TTLSHdrRsvd2</td>
<td>28</td>
<td>Binary</td>
<td>00000000 00000000 00000000 00000000</td>
</tr>
</tbody>
</table>

**TTLSQuadruplet**

- Get Request buffer

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Format</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTLSQKey</td>
<td>48</td>
<td>Binary</td>
<td>00000FA0</td>
</tr>
<tr>
<td>TTLSQOffset</td>
<td>52</td>
<td>Binary</td>
<td>00000050</td>
</tr>
<tr>
<td>TTLSQLength</td>
<td>56</td>
<td>Binary</td>
<td>00000000</td>
</tr>
<tr>
<td>TTLSQRcode</td>
<td>60</td>
<td>Binary</td>
<td>00000000</td>
</tr>
</tbody>
</table>

**TTLSQuadruplet**

- Get Request buffer

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Format</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTLSQKey</td>
<td>64</td>
<td>Binary</td>
<td>00000FA0</td>
</tr>
<tr>
<td>TTLSQOffset</td>
<td>68</td>
<td>Binary</td>
<td>0000006B</td>
</tr>
<tr>
<td>TTLSQLength</td>
<td>72</td>
<td>Binary</td>
<td>00000000</td>
</tr>
<tr>
<td>TTLSQRcode</td>
<td>76</td>
<td>Binary</td>
<td>00000000</td>
</tr>
</tbody>
</table>

**Buffer Data**

- Hostname | 80 | EBCDIC | mvs.telnet.raleigh.ibm.com
- Null character | 106 | Binary | 00
- Hostname | 107 | EBCDIC | mvs.prod.rtp.ibm.com
- Null Character | 127 | Binary | 00

Assuming that the certificate listed mvs.prod.rtp.ibm.com as the hostname value, the following TTLSHeader structure would be returned to the application:

### Table 105. Example returned TTLSHeader structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Format</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTLSHeaderIdent</td>
<td>0</td>
<td>EBCDIC</td>
<td>TTLSHDR</td>
</tr>
<tr>
<td>TTLSHdrRsvd1</td>
<td>8</td>
<td>Binary</td>
<td>00000000 00000000</td>
</tr>
<tr>
<td>TTLSHdrBytesNeeded</td>
<td>16</td>
<td>Binary</td>
<td>00000080</td>
</tr>
<tr>
<td>TTLSHdrSetCount</td>
<td>20</td>
<td>Binary</td>
<td>00000000</td>
</tr>
<tr>
<td>TTLSHdrGetCount</td>
<td>24</td>
<td>Binary</td>
<td>00000002</td>
</tr>
<tr>
<td>TTLSHdrRsvd2</td>
<td>28</td>
<td>Binary</td>
<td>00000000 00000000 00000000 00000000</td>
</tr>
</tbody>
</table>

**TTLSQuadruplet**

- Get Request buffer

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Format</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTLSQKey</td>
<td>48</td>
<td>Binary</td>
<td>00000FA0</td>
</tr>
<tr>
<td>TTLSQOffset</td>
<td>52</td>
<td>Binary</td>
<td>00000050</td>
</tr>
<tr>
<td>TTLSQLength</td>
<td>56</td>
<td>Binary</td>
<td>00000000</td>
</tr>
<tr>
<td>TTLSQRcode</td>
<td>60</td>
<td>Binary</td>
<td>00000000</td>
</tr>
</tbody>
</table>

**TTLSQuadruplet**

- Get Request buffer

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
<th>Format</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTLSQKey</td>
<td>64</td>
<td>Binary</td>
<td>00000FA0</td>
</tr>
<tr>
<td>TTLSQOffset</td>
<td>68</td>
<td>Binary</td>
<td>0000006B</td>
</tr>
<tr>
<td>TTLSQLength</td>
<td>72</td>
<td>Binary</td>
<td>00000001</td>
</tr>
<tr>
<td>Field</td>
<td>Offset</td>
<td>Format</td>
<td>Value</td>
</tr>
<tr>
<td>-------------</td>
<td>--------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>TTLSQ_Rcode</td>
<td>76</td>
<td>Binary</td>
<td>00000000</td>
</tr>
<tr>
<td>Hostname</td>
<td>80</td>
<td>EBCDIC</td>
<td>mvs.telnet.raleigh.ibm.com</td>
</tr>
<tr>
<td>Null character</td>
<td>106</td>
<td>Binary</td>
<td>00</td>
</tr>
<tr>
<td>Hostname</td>
<td>107</td>
<td>EBCDIC</td>
<td>mvs.prod.rtp.ibm.com</td>
</tr>
<tr>
<td>Null character</td>
<td>127</td>
<td>Binary</td>
<td>00</td>
</tr>
</tbody>
</table>

**SIOCTTLSCTL ioctl return values**

The following are possible return values:

0       Successful completion.

-1      An error occurred. Check the return code and reason code. The following are possible values:

**EProtoType**

Socket is not TCP.

**EINVAL**

The error depends on the reason code. The following are possible reason codes:

**JrInvalidVersion**

Version not valid in TTLS_IOCTL data structure.

**JrSocketCallParmError**

Denotes one of the following conditions:

- TTLS_RETURN_CERTIFICATE request type is specified along with a zero value in either TTLSi_BufferPtr or TTLSi_BufferLen
- TTLS_RETURN_CERTIFICATE request type is specified and TTLS_Version is not set to 1
- TTLS_RETURN_CERTIFICATE request type is not specified along with a nonzero value in either TTLSi_BufferPtr or TTLSi_BufferLen and TTLS_Version is set to 1
  - Request type is not valid.
  - Length of input data is not length of ioctl structure.

**EPerm**

Denotes one of the following error conditions:

- The TTLS_INIT_CONNECTION option was requested, along with one of the following:
  - TTLS_RESET_SESSION
  - TTLS_RESET_CIPHER
  - TTLS_STOP_CONNECTION
- The TTLS_STOP_CONNECTION option was requested along with the TTLS_RESET_SESSION or TTLS_RESET_CIPHER option
- The TTLS_ALLOW_HSTIMEOUT option was requested without the TTLS_INIT_CONNECTION option
ENotConn
The connection has not reached the established state or has been closed.

EPipe
TTLS_INIT_CONNECTION, TTLS_STOP_CONNECTION, or TTLS_RESET_CIPHER option was requested and the connection is no longer in established state.

EMVSERR
Internal failure while mapping AT-TLS policy.

EOpNotSupp
The TTLS_INIT_CONNECTION, TTLS_STOP_CONNECTION, TTLS_RESET_SESSION, or TTLS_RESET_CIPHER option was requested and one of the following is true:
• TCPCONFIG NOTTLS is configured or is the default.
• The connection has no policy.
• The AT-TLS policy for the connection specifies TTLSEnabled=No.

EAcces
The TTLS_INIT_CONNECTION, TTLS_STOP_CONNECTION, TTLS_RESET_SESSION, or TTLS_RESET_CIPHER option was requested and the AT-TLS policy for the connection specifies ApplicationControlled=No.

EAlready
TTLS_INIT_CONNECTION was requested and the connection is already secure or TTLS_STOP_CONNECTION was requested and the connection is not secure.

EProto
Denotes one of the following reason codes:
JrGetConnErr
The TTLS_RESET_SESSION or TTLS_RESET_CIPHER option was requested and the connection is not secure.

JrInvalidVersion
The TTLS_RESET_CIPHER or TTLS_STOP_CONNECTION option was requested; the connection is secure but is SSLv2.

JrConnDeniedPolicy
The TTLS_ALLOW_HSTIMEOUT option was requested but the HandshakeRole value is client or the HandshakeTimeout value is 0.

EInProgress
The TTLS_INIT_CONNECTION or TTLS_STOP_CONNECTION option was requested and handshake is in progress.

EWouldBlock
The socket is a non-blocking socket and an SSL handshake is in progress.

ENoBufs
Denotes one of the following reason codes:
JrBuffTooSmall
• For TTLS_Version1, the TTLS_RETURN_CERTIFICATE option was requested and the buffer provided using
TTLSi_BufferPtr field is too small. See the TTLSi_Cert_Len value for the number of bytes required to hold the certificate.

- For TLS Version 2, the buffer supplied was too small. See the TTLSHdr_BytesNeeded field value for the number of bytes required.

**SIOCTTLSCTL ioctl coding examples**

The following examples show sample code for building and issuing the SIOCTTLSCTL ioctl.

**SIOCTTLSCTL ioctl assembler example**

The following sample assembler code builds an SIOCTTLSCTL ioctl and issues the ioctl using the Macro API (EZASMI). The ioctl requests initialization of the secure connection and the return of the partner's certificate in the provided buffer.

```assembly
  XC TTLS_IOCTL(TTLS_IOCTL_V1Len),TTLS_IOCTL
  MVI TTLS_IOCTL(0),TTLS_IOCTL
  L R8,=A(TTLS_INIT_CONNECTION)
  O R8,=A(TTLS_RETURN_CERTIFICATE)
  STH R8,TTLS_IOCTL
  LA R8,BUFFERA
  ST R8,TTLS_IOCTL
  MVC TTLS_IOCTL,BUFFERA

  EZASMI TYPE=IOCTL, ISSUE IOCTL MACRO
  S=SOCDESCA,
  COMMAND='SIOCTTLSCTL',
  REQARG=TTLS_IOCTL,
  RETARG=TTLS_IOCTL,
  REQAREA=REQAREA, IN CASE WE ARE DOING EXITS OR ECBS
  ERRNO=ERRNO, RETURN ERRNO HERE
  RETCODE=RETCODE, RETURN RETCODE HERE
  ERROR=ERROR

  EZBTLSLP DSECT=NO  TTLS ioctl structure
```

**SIOCTTLSCTL ioctl PL/I example**

The following sample PL/I code builds and issues an SIOCTTLSCTL ioctl that requests secure connection initialization and the return of the partner's certificate in the provided buffer.

```pli
/* get the SIOCTTLSCTL ioctl mapping and constants */
% include EZBTLS1;

/* area to return the certificate data if available */
DCL CERTIF CHAR(1000) INIT('B');
```
/* allocate storage for the SIOCTTLSCTL ioctl */
Allocate TTLS_IOCTL;
TTLSI_VER = TTLS_VERSION1;
TTLSI_REQ_TYPE = TTLS_INIT_CONNECTION | TTLS_RETURN_CERTIFICATE;

/* if you DO NOT want to get the certificate then you must */
/* clear the following two fields */
/* TTLSi_BufferPtr = SYSNULL; */
/* TTLSi_BufferLen = 0; */
/* if you DO want to get the certificate then you must */
/* set the following two fields */
TTLSi_BufferPtr = ADDR(CERTIF);
TTLSi_BufferLen = LENGTH(CERTIF);

call ezasoket(IOCTL,
SOCKET,
SIOCTTLSCTL, /* TTLS ioctl */
TTLS_IOCTL, /* input buffer */
TTLS_IOCTL, /* output buffer */
ERRNO,
RETCODE);
if RETCODE < 0 then do;
/* do failure logic. If the socket is in non- */
/* blocking mode then you may also receive RETCODE */
/* of -1 with an ERRNO of EINPROGRESS. This does */
/* not indicate an error. Wait for the completion */
/* of the handshake with SELECT for WRITEABLE. */
...
end;

SIOCTTLSCTL ioctl COBOL example
The following sample COBOL code builds and issues an SIOCTTLSCTL ioctl that requests the initialization of the secure connection.

***
Data Division.
***
* Variables used by the SIOCTTLSCTL IOCTL call *
*---------------------------------------------------------------*
01 ttls_ioctl-data.
   COPY EZBZTLSB.
***
Procedure Division.
***
TTLS-Init.
   move low-values to ttls_ioctl-data.
   set TTLSI-BUFFERPTR to NULL.
   move TTLS-VERSION1 to TTLSI-VER.
   move TTLS-INIT-CONNECTION to TTLSI-REQ-TYPE.
   Call 'EZASOKET' using soket_ioctl socket-descriptor-new
         SIOCTTLSCTL
         TTLS-IOCTL TTLS-IOCTL
         errno retcode.
   * if error other than EINPROGRESS then *
   * IF ((RETCODE < 0) AND (ERRNO NOT = EINPROGRESS)) THEN *
   * handle error here *
   ELSE *
   * normal case *
   * TTLS-INIT-Exit.
   Exit.
SIOCTTLSCTL ioctl C example

The following sample C code builds and issues an SIOCTTLSCTL ioctl that requests initialization of the secure connection and the return of the partner's certificate in the provided buffer.

```c
#include "ezbztlsc.h" /* SIOCTTLSCTL ioctl */

struct TTLS_IOCTL ioc; /* ioctl data structure */
char buff[1000]; /* buffer for certificate */

/* issue the SIOCTTLSCTL ioctl */
memset(&ioc, 0, sizeof(ioc)); /* set all unused fields to zero */
ioc.TTLSi_Ver = TTLS_VERSION1;
ioc.TTLSi_Req_Type = TTLS_INIT_CONNECTION | TTLS_RETURN_CERTIFICATE;
ioc.TTLSi_BufferPtr = &buff
ioc.TTLSi_BufferLen = sizeof(buff);

rc = ioctl(s, SIOCTTLSCTL, (char *)&ioc);
if (rc < 0)
{
    /* do failure logic. If the socket is in non- */
    /* blocking mode then you may also receive rc */
    /* of -1 with an errno of EINPROGRESS. This does */
    /* not indicate an error. Wait for the completion */
    /* of the handshake with select() for WRITEABLE. */
}
```
Chapter 16. Interfacing with the Digital Certificate Access Server (DCAS)

The DCAS can be used by providers of logon and single sign-on services where access to z/OS-based applications is needed. The DCAS is a TCP/IP server that enables clients to connect over the network and obtain a passticket and z/OS user ID from a SAF-compliant product, such as RACF. This topic refers to RACF as the SAF product.

A passticket is like a temporary password, because it is valid for only a short period of time. Applications on z/OS can be configured to support logon access with passtickets. For information about passtickets, see the z/OS Security Server RACF Security Administrator’s Guide.

IBM does not provide header files or samples for programming DCAS clients, but the specifications for developing a client are defined in the following topics.

Understanding how clients interface to DCAS

Clients connect to DCAS using the TCP protocol.

- By default, the DCAS listens on port 8990, but it can listen on any configured port.
- Clients that connect to DCAS must use the SSL protocol (DCAS supports SSL Version 3). Client authentication is performed.
- DCAS provides a request and response interface that enables clients to obtain two types of information. After the TCP connection and SSL handshake processing completes, the DCAS client sends a request and in turn receives a response. The request and its response determine which of the following types of information DCAS provides:
  - Clients can request a user ID and passticket for an application. The client sends a Format 1 type request that includes an application ID1 and an x.509 certificate. DCAS returns a user ID and passticket in the Format 1 response. In this case, DCAS converts the x.509 certificate to a valid user ID, which is returned.
    
    **Requirement:** The x.509 certificate must have been mapped to a valid user ID in RACF.
  - Clients can request a passticket for an application. The client sends a Format 2 type request that includes an application ID1 and a user ID. DCAS returns a passticket in the Format 1 response.

See the Configuring RACF services for Express Logon information in the z/OS Communications Server: IP Configuration Guide for more details about passtickets.

The request and response formats are described in the following topic.

Where text is required in the formatted request, DCAS requires that they are encoded in EBCDIC (IBM-1047 codepage). Responses that contain text are also encoded in EBCDIC (IBM-1047 codepage).
If the request from the client was not processed successfully, DCAS returns error code information in the response. The client must be designed to examine this information.

**Interfacing with the DCAS: Defining the format for request and response specifications**

Table 106 contains format 1 request information.

<table>
<thead>
<tr>
<th>Field byte offset</th>
<th>Field name</th>
<th>Field description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>opcode</td>
<td>01 = request</td>
</tr>
<tr>
<td>1</td>
<td>Format</td>
<td>01 = request user ID and passticket</td>
</tr>
<tr>
<td>2-5</td>
<td>Correlator</td>
<td>User-defined value</td>
</tr>
<tr>
<td>6-25</td>
<td>Appl ID</td>
<td>Application for which the passticket is generated. This must have the same name as the passticket data profile that is defined for the application using the RACF PTKTDATA class.¹ (EBCDIC).</td>
</tr>
<tr>
<td>26-27</td>
<td>reserved</td>
<td>not used</td>
</tr>
<tr>
<td>28-31</td>
<td>Certificate Length</td>
<td>Input certificate length. Maximum length is 32 767 bytes. This field is a binary integer.</td>
</tr>
<tr>
<td>32-n</td>
<td>Certificate</td>
<td>Base-64 encoded certificate</td>
</tr>
</tbody>
</table>

¹ The application ID required in the DCAS Format 1 and Format 2 requests must match the name of a valid passticket data profile defined in RACF using the PTKTDATA class. See the z/OS Security Server RACF Security Administrator's Guide for information about defining PTKTDATA for applications.

Table 107 contains format 1 response information.

<table>
<thead>
<tr>
<th>Field byte offset</th>
<th>Field name</th>
<th>Field description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>opcode</td>
<td>02 = response</td>
</tr>
<tr>
<td>1</td>
<td>Format</td>
<td>01 = request user ID and passticket</td>
</tr>
<tr>
<td>2-5</td>
<td>Correlator</td>
<td>User-defined value that matches the value of the request.</td>
</tr>
<tr>
<td>6-7</td>
<td>Return Code 1</td>
<td>If nonzero, examine the extended return codes: Return Code 2, Return Code 3, Return Code 4</td>
</tr>
<tr>
<td>8-11</td>
<td>Return Code 2</td>
<td>Extended (see Table 109 on page 659)</td>
</tr>
<tr>
<td>12-15</td>
<td>Return Code 3</td>
<td>Extended (see Table 109 on page 659)</td>
</tr>
<tr>
<td>16-19</td>
<td>Return Code 4</td>
<td>Extended (see Table 109 on page 659)</td>
</tr>
<tr>
<td>20-28</td>
<td>User ID</td>
<td>If Return Code 1 is 0, a user ID is returned (EBCDIC)</td>
</tr>
<tr>
<td>29</td>
<td>reserved</td>
<td>null</td>
</tr>
<tr>
<td>30-37</td>
<td>Passticket</td>
<td>If Return Code 1 is 0, a passticket is returned.</td>
</tr>
</tbody>
</table>
Table 108 contains format 2 request information.

### Table 108. Format 2 request

<table>
<thead>
<tr>
<th>Field byte offset</th>
<th>Field name</th>
<th>Field description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>opcode</td>
<td>02 = request</td>
</tr>
<tr>
<td>1</td>
<td>Format</td>
<td>02 = request passticket</td>
</tr>
<tr>
<td>2-5</td>
<td>Correlator</td>
<td>User-defined value</td>
</tr>
<tr>
<td>6-25</td>
<td>Appl ID</td>
<td>Application for which the passticket is generated. Must have the same name as the passticket data profile that is defined for the application using the RACF PTKTDATA class.¹ (EBCDIC)</td>
</tr>
<tr>
<td>26-27</td>
<td>reserved</td>
<td>Not used</td>
</tr>
<tr>
<td>28-31</td>
<td>User ID Length</td>
<td>Length of the input user ID (binary integer)</td>
</tr>
<tr>
<td>32-n</td>
<td>User ID</td>
<td>Input user ID (EBCDIC)</td>
</tr>
</tbody>
</table>

¹ The response to a Format 2 request is a Format 1 Response. The application ID required in the DCAS Format 1 and Format 2 requests must match the name of a valid passticket data profile defined in RACF using the PTKTDATA class. See the z/OS Security Server RACF Security Administrator’s Guide for information about defining PTKTDATA for applications.

### Table 109. Understanding return codes in the response

<table>
<thead>
<tr>
<th>Return Code 1</th>
<th>Return Code 2</th>
<th>Return Code 3</th>
<th>Return Code 4</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not Set</td>
<td>Not Set</td>
<td>Not Set</td>
<td>The response indicates that the request completed successfully.</td>
</tr>
<tr>
<td>250</td>
<td>Not Set</td>
<td>Not Set</td>
<td>Not Set</td>
<td>An internal error occurred on the DCAS server. Request that the system operator obtain a DCAS trace. See the z/OS Communications Server: IP Diagnosis Guide for instructions.</td>
</tr>
<tr>
<td>251</td>
<td>Not Set</td>
<td>Not Set</td>
<td>Not Set</td>
<td>Passticket generation failed. The most likely cause is that the application ID in the DCAS Format 1 or 2 request does not match a valid passticket data profile name defined in the RACF PTKTDATA class.¹</td>
</tr>
<tr>
<td>252</td>
<td>8</td>
<td>8</td>
<td></td>
<td>For a Format 1 type request, RACF has determined that the input certificate is in error or has not been mapped to a valid RACF user ID. For return codes other than the ones described, see the z/OS Communications Server: IP Diagnosis Guide.</td>
</tr>
</tbody>
</table>

³ The response to a Format 2 request is a Format 1 Response. The application ID required in the DCAS Format 1 and Format 2 requests must match the name of a valid passticket data profile defined in RACF using the PTKTDATA class. See the z/OS Security Server RACF Security Administrator’s Guide for information about defining PTKTDATA for applications.
<table>
<thead>
<tr>
<th>Return Code 1</th>
<th>Return Code 2</th>
<th>Return Code 3</th>
<th>Return Code 4</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>253</td>
<td>10</td>
<td>Not Set</td>
<td>Not Set</td>
<td>The input format 1 or 2 request is incorrect. Examine Return Code 2 for details.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Verify that the input request to DCAS matches the defined format specifications.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Verify that DCAS is configured with a SERVERTYPE in the DCAS profile that is consistent with the input request format.</td>
</tr>
<tr>
<td>254</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>DCAS failed to authenticate the client.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The DCAS server has been configured with AUTHTYPE LOCAL2. This requires that the certificate of the DCAS client (as a result of the SSL handshake) be mapped to a defined and valid user ID in RACF. The user ID must be permitted to the following SERVAUTH class profile: EZA.DCAS.&lt;cvtsysname&gt;. If the DCAS client receives this error, then the certificate does not map to a valid user ID.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>For return codes other than the ones described, see the Diagnosing problems with Express Logon information in the z/OS Communications Server: IP Diagnosis Guide for diagnosing DCAS.</td>
</tr>
<tr>
<td>255</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>DCAS failed to authenticate the client.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The DCAS server has been configured with AUTHTYPE LOCAL2. This requires that the certificate of the DCAS client (as a result of the SSL handshake) be mapped to a defined and valid user ID in RACF. If the DCAS client receives this error, then the certificate does not map to a valid user ID.</td>
</tr>
</tbody>
</table>

1 The application ID required in the DCAS Format 1 and Format 2 requests must match the name of a valid passticket data profile defined in RACF using the PTKTDATA class. See the z/OS Security Server RACF Security Administrator's Guide for information about defining PTKTDATA for applications.
## Configuring the DCAS server to work with your solution

When interfacing to DCAS as a provider of logon services, work with the system administrator to verify that DCAS is configured to work with your solution. For more details about configuring DCAS, see the EXPRESS LOGON using DCAS (Digital Certificate Access Server) information in the z/OS Communications Server: IP Configuration Reference. The DCAS configuration statements described in Table 110 require coordination between the DCAS client and server.

Table 110. DCAS client and server coordination

<table>
<thead>
<tr>
<th>DCAS client interface</th>
<th>DCAS server configuration statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Request Format</td>
<td>SERVERTYPE options</td>
<td><strong>ALLTYPES</strong> – Allows Format 1 and Format 2 input requests to be accepted by the DCAS server.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>CERTTYPE</strong> – Allows only Format 1 requests.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>USERIDTYPE</strong> – Allows only Format 2 requests.</td>
</tr>
<tr>
<td>SSL connection parameters</td>
<td>V3CIPHER</td>
<td>The DCAS client must communicate with the DCAS server using SSL. The V3CIPHER allows for specification of the cipher.</td>
</tr>
<tr>
<td>TCP port used for connection</td>
<td>PORT</td>
<td>The default DCAS listening port is 8990 but DCAS can be configured to use any port.</td>
</tr>
</tbody>
</table>
Chapter 17. Miscellaneous programming interfaces

This topic contains descriptions of the following:

- "SIOCSAPPLDATA IOCTL"
- “SIOCSMOCTL IOCTL” on page 665
- “TCP_KeepAlive socket option” on page 668

SIOCSAPPLDATA IOCTL

The SIOCSAPPLDATA ioctl enables applications to associate 40 bytes of application-specific information with TCP sockets they own. This information can assist problem determination, capacity planning, and accounting applications. This ioctl supports both 31-bit and 64-bit addressing modes.

This application-specific information, which is referred to as ApplData, is available from the following sources:

- In the Netstat ALL/-A, ALLConn/-a, and CONn/-c reports where it can be searched using the APPLD/-G filter. See the z/OS Communications Server: IP System Administrator’s Commands for additional information on using ApplData information with Netstat.
- In the SMF 119 TCP connection termination record. See the z/OS Communications Server: IP Configuration Reference for additional information.
- Through the callable TCP/IP network management interface. See “TCP/IP callable NMI (EZBNMIFR)” on page 598 for more information.

The SIOCSAPPLDATA IOCTL constant and data structures for assembler applications are defined in the EZBYAPPL macro in the SEZANMAC data set, and for C/C++ applications they are defined in the EZBYAPLC header file in the SEZANMAC data set and the /usr/include file system directory.

Consider the following guidelines when using this ioctl:

- The application is responsible for documenting the content, format, and meaning of the ApplData strings that it might associate 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 TCP/IP’s EZA or EZB, are reserved for IBM use. IBM product identifiers begin with a letter in the range A – I.
- You should use printable EBCDIC characters for the entire string to enable searching with Netstat filters.

SIOCSAPPLDATA input

Input is provided using a pointer to a SetApplData structure, which in turn defines the version, size, and location of the SetADcontainer structure that contains the application data to be associated with the stream socket.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetAD_eye1</td>
<td>8</td>
<td>constant SETADEYE1</td>
</tr>
<tr>
<td>SetAD_ver</td>
<td>4</td>
<td>constant SETADVER</td>
</tr>
</tbody>
</table>
Table 111. SetApplData (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetAD_len</td>
<td>4</td>
<td>sizeof(SetADcontainer)</td>
</tr>
<tr>
<td>SetAD_ptr</td>
<td>8</td>
<td>A bimodal pointer to a SetADcontainer structure. In 31-bit addressing mode the first 4 bytes are reserved and should be 0; the second 4 bytes contain the 31-bit address.</td>
</tr>
</tbody>
</table>

Table 112. SetADcontainer

<table>
<thead>
<tr>
<th>Field name</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetAD_eye2</td>
<td>8</td>
<td>constant SETADEYE2</td>
</tr>
<tr>
<td>SetAD_buffer</td>
<td>40</td>
<td>A character buffer that contains the data to associate with this end of the connection. This buffer should be padded on the right with space characters.</td>
</tr>
</tbody>
</table>

**SIOCSAPPLDATA output**

The SIOCSAPPLDATA IOCTL sets the following return codes and reason codes:

Table 113. SIOCSAPPLDATA IOCTL return and reason codes

<table>
<thead>
<tr>
<th>ReturnValue</th>
<th>ReturnCode</th>
<th>ReasonCode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>The request was successful.</td>
</tr>
<tr>
<td>-1</td>
<td>EProtoType</td>
<td>JrSocketTypeNotSupported</td>
<td>The request was not successful. The socket is not a stream (TCP) socket.</td>
</tr>
<tr>
<td>-1</td>
<td>EINVAL</td>
<td>JrSocketCallParmError</td>
<td>The input parameter is not a correctly formatted SetApplData structure. Either the SetAD_eye1 or the SetAD_ver field is incorrect or the storage pointed to by the SetAD_ptr field did not contain a correctly formatted SetADcontainer structure. The SetAD_eye2 field is incorrect.</td>
</tr>
<tr>
<td>-1</td>
<td>EINVAL</td>
<td>JrBuffLenInvalid</td>
<td>SetAD_len contains an incorrect length for the SetAD_ver value of the SetADcontainer structure.</td>
</tr>
<tr>
<td>-1</td>
<td>EFault</td>
<td>JrBadInputBufAddr</td>
<td>An abend occurred while attempting to copy the SetADcontainer structure from the address provided in SetAD_ptr field.</td>
</tr>
<tr>
<td>-1</td>
<td>ENOBUFFS</td>
<td>JrSmNoStorage</td>
<td>There was no storage available to store the associated data.</td>
</tr>
</tbody>
</table>

The SIOCSAPPLDATA call can be issued on stream sockets only. No application authorization is required. Each time the ioctl call is issued, the application data is replaced. If the call is issued on a socket prior to issuing a listen() call, the application data is inherited by all connections accepted over that socket. If the call is issued on a socket after issuing a listen() call, the application data is inherited by all connections accepted over that socket that arrive after the ioctl call is processed.

**SIOCSAPPLDATA C language example**

```c
#include <ezbyaplc.h>
char myappldata[SETADBUFLEN+1]; /* extra byte for null string terminator */
SetApplData myioctlParm;
SetADcontainer myBuffer;

sprintf(myappldata, "%HRSERVR\%8.8s\%8.8s\%8.8s\%8.8s", a, b, c, d); /* prefix and 4 char[8] fields */
memcpy(myioctlParm.SetAD_eye1, SETADEYE1, sizeof(myioctlParm.SetAD_eye1));
myioctlParm.SetAD_ver = SETADVER;
```
Applications that use the UNIX System Services optimized Asynchronous Socket I/O option (designated by the AioCommBuff bit in the AIOCB control block) can exploit 64-bit shared memory objects. The application allocates a shared memory object and issues a new IOCTL SIOCSMOCTL that enables TCP/IP to establish access to the memory object or to remove access to the memory object.

Table 114 lists SIOCSMOCTL requirements.

<table>
<thead>
<tr>
<th>Minimum authorization:</th>
<th>Executing in supervisor state, in system key, or APF authorized</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatchable unit mode:</td>
<td>Task or SRB</td>
</tr>
<tr>
<td>Cross memory mode:</td>
<td>PASN=HASN=SASN</td>
</tr>
<tr>
<td>Addressing mode:</td>
<td>AMODE31 or AMODE64</td>
</tr>
<tr>
<td>ASC mode:</td>
<td>Primary</td>
</tr>
<tr>
<td>Interrupt status:</td>
<td>Enabled for I/O and external interrupts</td>
</tr>
<tr>
<td>Locks:</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Control parameters:</td>
<td>Must reside in an addressable area in the primary address space and must be accessible using caller’s execution key</td>
</tr>
</tbody>
</table>

A SIOCSMOCTL IOCTL can be issued on any type of socket (stream, datagram, or raw), and requires that the application be authorized. After access to shared memory objects is established, the application can use buffers in the memory objects for asynchronous I/O by setting the AioCommBuff bit in the AIOCB control block on any stream socket that it has created. TCP/IP internally associates access to shared memory objects with the socket that was used to issue the SIOCSMOCTL IOCTL; if any shared memory object associations remain when that socket is closed, these memory associations are automatically broken and TCP/IP access to those objects is removed. The application must ensure that the socket that is used to issue the SIOCSMOCTL IOCTL is closed only after all other stream sockets that use buffers in those shared memory objects are closed.

For more information about the use of the BPX1AIO and BPX4AIO services and about the use of the AioCommBuff bit, see z/OS UNIX System Services Programming: Assembler Callable Services Reference.

For more information about the use of shared memory objects, see z/OS MVS Programming: Extended Addressability Guide.

### SIOCSMOCTL input

SIOCSMOCTL input consists of a pointer to a SMOCTL_IOCTL structure that contains the following:

```c
myIoctlParm.SetAD_len = sizeof(SetADcontainer);
myIoctlParm.SetAD_ptr = &myBuffer
memcpy(myBuffer.SetAD_eye2, SETADEYE2, sizeof(myIoctlParm.SetAD_eye2));
memcpy(myBuffer.SetADbuffer, myappldata, SETADBUFLEN);
rc = ioctl(soc, SIOCSAPPLDATA, (char *)&myIoctlParm);
```
Table 115. SIOCSMOCTL input structure

<table>
<thead>
<tr>
<th>Data item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMOCTL_Version</td>
<td>Required input that contains the signed 31-bit version is Version 1 of the SIOCSMOCTL IOCTL.</td>
</tr>
</tbody>
</table>
| SMOCTL_Request     | Required input that contains the signed 31-bit request type, which can be one of the following:  
   - Attach request (establish access from TCP/IP to the shared memory object)  
   - Detach request (remove access from TCP/IP to the shared memory object) |
| SMOCTL_ObjectAddr  | Required input that contains the 64-bit starting address of a shared memory object to be attached or detached. |
| SMOCTL_IARV64_Retcode | Output field that contains the IARV64 return code. |
| SMOCTL_IARV64_Rsncode | Output field that contains the IARV64 reason code. |

The SIOCSMOCTL IOCTL parameter list for assembler applications is defined in the EZBITSIA macro in the SEZANMAC data set. For C/C++ applications, the parameter list is defined in the header file, ezbitsic.h. This header file is installed in the SEZANMAC data set and in the file system directory, /usr/include.

**SIOCSMOCTL output**

The SMOCTL_IOCTL structure is updated with status information that pertains to the attach or delete request. The SIOCSMOCTL ioctl sets the return codes and reason codes that are described in Table 116.

Table 116. SIOCSMOCTL return and reason codes

<table>
<thead>
<tr>
<th>ReturnValue</th>
<th>ReturnCode</th>
<th>ReasonCode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>The request was successful.</td>
</tr>
<tr>
<td>-1</td>
<td>EACCESS</td>
<td>JRIOCTLAccessAuthorization</td>
<td>The request was not successful. The issuer of the IOCTL is not authorized.</td>
</tr>
<tr>
<td>-1</td>
<td>EINVAL</td>
<td>JRSocketCallParmError</td>
<td>The request was not successful. The input parameter length is incorrect, the version is not valid, or the request type is not valid.</td>
</tr>
<tr>
<td>-1</td>
<td>ENOMEM</td>
<td>JRSmNoStorage</td>
<td>The request was not successful. The attach request failed due to a storage shortage.</td>
</tr>
<tr>
<td>-1</td>
<td>EINVAL</td>
<td>JRDuplicateSmoAttach</td>
<td>The request was not successful. For an attach request, the specified shared memory object has already been attached.</td>
</tr>
</tbody>
</table>
Table 116. SIOCSMOCTL return and reason codes (continued)

<table>
<thead>
<tr>
<th>ReturnValue</th>
<th>ReturnCode</th>
<th>ReasonCode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>EINVAL</td>
<td>JRSmoNotAttached</td>
<td>The request was not successful. For a detach request, the specified shared memory object is not attached.</td>
</tr>
<tr>
<td>-1</td>
<td>EMVSPARM</td>
<td>JRIARV64Error</td>
<td>The request was not successful. For an attach or detach request, the IARV64 services encountered an error. Fields SMOI_IARV64_Retcode and SMOI_IARV64_Rsncode describe the error.</td>
</tr>
</tbody>
</table>

Tips:
- Create shared memory objects in system key (0-7) fetch-protected storage to maintain integrity of the data.
- Create an ancillary socket that is used only to issue SIOCSMOCTL IOCTL requests.

Steps for creating an ancillary socket

Before you begin: Ensure that the ancillary socket is not closed until all stream sockets that might reference those shared memory objects are closed. This can be accomplished by taking advantage of the fact that the UNIX System Services process cleanup service closes sockets sequentially from lowest-numbered socket descriptor to highest-numbered socket descriptor.

Perform the following steps:

1. Issue a getrlimits() request to discover the largest socket descriptor available for the process.

2. Issue a dup2() request to copy the original ancillary socket descriptor to the largest socket descriptor obtained from the getrlimits() request.

3. Close the original ancillary socket descriptor.

Applications in a common INET environment

When a generic application in a common INET environment creates a socket, UNIX System Services creates socket sub-instances to each active TCP/IP instance. When the application then issues an IOCTL on its socket, UNIX System Services propagates the IOCTL to all active TCP/IP instances until the sockets all indicate that the IOCTL was successful. If a TCP/IP instance indicates a failure, IOCTL processing stops at that point and a failure return value, return code, and reason code are returned to the application. If the generic application issues a SIOCSMOCTL attach request and encounters a failure, some TCP/IP instances might have access to the shared memory object, and some might not. For problem determination purposes, all TCP/IP instances should gain access to the shared
memory object, or no TCP/IP instances should gain access to it. When a SIOCSMOCTL attach request fails in a common INET environment, the application should immediately issue a SIOCSMOCTL detach request to ensure that no TCP/IP instance has access to the shared memory object and should thereafter not set a value in the AioCommBuff bit in the AIOCB.

A generic application in a common INET environment can use the SOCK#SO_EIOIFNEWTP socket option on its listening socket so that the application is notified when a TCP/IP instance is stopped and restarted. When a TCP/IP instance is recycled, the application’s response is to close the listening socket and create a new listening socket, which cause new listening socket sub-instances to each active TCP/IP instance to be created. The application should do the following to ensure that the recycled TCP/IP instance gains access to the shared memory object and that the other TCP/IP instances retain their access to the share memory object:

1. After the new listening socket is created, create a new ancillary socket (which is propagated to all active TCP/IP instances).
2. Issue a SIOCSMOCTL attach request for the shared memory object on the new ancillary socket.
3. Close the original ancillary socket descriptor.
4. Perform the “Steps for creating an ancillary socket” on page 667 to ensure that the new ancillary socket is not closed until all stream sockets that might reference the shared memory object are closed.

**TCP_KeepAlive socket option**

Some TCP/IP users require a keep alive function with better timing granularity (in seconds) than that provided by the existing SO_KeepAlive socket option, which uses a stack-wide time value provided by configuration data.

The Posix.1g standard defines an alternative keep alive function, TCP_KeepAlive, which provides a value in seconds that is specific to a particular socket.

The value of TCP_KeepAlive, which is used for the current connection in place of the configuration default keep alive time (when keep alive timing is made active by the SO_KeepAlive socket option), can be in the range 1 – 2,147,460 seconds. If a value greater than 2,147,460 is specified, 2,147,460 is used. If the TCP_KeepAlive value 0 is specified for a specific socket, keep alive timing for that socket is disabled.

**SetSockOpt for TCP_KeepAlive**

Specifies a socket-specific timer value that remains in effect until it is respecified by the SetSockOpt option or until the socket is closed. Timeout values in the range 1 – 2,147,460 seconds (or 0) are valid for TCP_KeepAlive. If a value larger than the allowed range is specified, the value 2,147,460 seconds is used.

**GetSockOpt for TCP_KeepAlive**

Returns the specific timer value (in seconds) that is in effect for the given socket, or the value 0 if keep alive timing is not active.

Unlike the current algorithm used for issuing probes during an SO_KeepAlive cycle, the TCP_KeepAlive function time values the number of probes issued before terminating the connection. Probe retry intervals are scaled in proportion to the interval specified, as shown in Table 117 on page 669.
### Table 117. TCP KeepAlive time values

<table>
<thead>
<tr>
<th>TCP KeepAlive time (T) specified in seconds</th>
<th>Seconds to first probe</th>
<th>Number of probes</th>
<th>Probe interval</th>
<th>Maximum interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>T = 0 (KeepAlive Disabled)</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>0 &lt; T &lt;= 5</td>
<td>T</td>
<td>1</td>
<td>1</td>
<td>T + 1</td>
</tr>
<tr>
<td>5 &lt; T &lt;= 10</td>
<td>T</td>
<td>1</td>
<td>2</td>
<td>T + 2</td>
</tr>
<tr>
<td>10 &lt; T &lt;= 30</td>
<td>T</td>
<td>1</td>
<td>5</td>
<td>T + 5</td>
</tr>
<tr>
<td>30 &lt; T &lt;= 60</td>
<td>T</td>
<td>1</td>
<td>10</td>
<td>T + 10</td>
</tr>
<tr>
<td>60 &lt; T &lt;= 120</td>
<td>T</td>
<td>1</td>
<td>20</td>
<td>T + 20</td>
</tr>
<tr>
<td>120 &lt; T &lt;= 300</td>
<td>T</td>
<td>2</td>
<td>20</td>
<td>T + 40</td>
</tr>
<tr>
<td>300 &lt; T &lt;= 600</td>
<td>T</td>
<td>2</td>
<td>30</td>
<td>T + 160</td>
</tr>
<tr>
<td>600 &lt; T &lt;= 1800</td>
<td>T</td>
<td>5</td>
<td>30</td>
<td>T + 150</td>
</tr>
<tr>
<td>1800 &lt; T &lt;= 3600</td>
<td>T</td>
<td>5</td>
<td>60</td>
<td>T + 300</td>
</tr>
<tr>
<td>3600 &lt; T &lt;= 7200</td>
<td>T</td>
<td>9</td>
<td>60</td>
<td>T + 540</td>
</tr>
<tr>
<td>7200 &lt; T &lt;= 2 147 460 (35 791 x 60 = 2 147 460)</td>
<td>T</td>
<td>9</td>
<td>75</td>
<td>T + 675</td>
</tr>
<tr>
<td>T &gt; 2 147 460</td>
<td>2 147 460</td>
<td>9</td>
<td>75</td>
<td>2 147 460 + 675</td>
</tr>
</tbody>
</table>

The TCP KeepAlive option value can range from 1 – 2 147 460 seconds. For values greater than 2 hours (7200 seconds), the probe interval and number of probes are adjusted as the specified interval increases until they coincide with the default algorithm. If no response is received from the remote partner after the listed number of probes, the connection is terminated.

**Tips:**

1. The SO KeepAlive function must be activated before any keep alive processing is done. The KEEPALIVEOPTIONS configuration value is used for timing unless a specific value has been provided through the TCP KeepAlive option.
2. The TCP KeepAlive option can be set before or after the SO KeepAlive function is activated, but timing does not take effect until the SO KeepAlive status is set to active.
Appendix A. TCP/IP in the sysplex

This topic introduces the SO_CLUSTERCONNTYPE socket option. This socket option provides sysplex-specific connection routing information to a sockets application, which might enable the application to offer better function, performance, and scalability. Specific information about how to use SO_CLUSTERCONNTYPE can be found in the z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference.

Restriction: This enhancement does not apply to UDP or raw socket connections.

Sockets applications are typically designed to communicate with an appropriate partner on any platform, but they might be able to perform better if both partners know they are on zSeries, within the same sysplex, or on the same MVS image.

TCP sockets applications can benefit from information about the partner. Table 118 lists examples of these benefits.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Potential benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same Cluster (sysplex)</td>
<td>Avoid parameter conversions, because both sides of the connection use the same machine architectures and data representation (z/OS).</td>
</tr>
<tr>
<td>Same MVS image</td>
<td>Share memory information that is costly to generate (for example, security contexts).</td>
</tr>
<tr>
<td>Internal</td>
<td>Communications are not exposed outside the cluster. This means that application security might cost less. For example, the application might not encrypt application data.</td>
</tr>
</tbody>
</table>

The internal indication is returned only when the partner is part of the same cluster. This means that the data flows to the partner over a link or interface that is never exposed outside the cluster (sysplex), and the link or interface type is one of the following:

- CTC
- HiperSockets™ (iQDIO)
- MPCPTP (including XCF and IUTSAMEH)
- Loopback
- Both connection partners are owned by the same multi-homed stack

These devices/interfaces are assumed to participate in the same physical security as the cluster itself, so that the links/interfaces carrying IP traffic have the same physical security as links/interfaces to the attached DASD. When the internal indication is returned, the application can choose not to encrypt data exchanged with a partner application in the same cluster. This saves CPU cycles and improves throughput. The application itself determines whether or not to exploit the internal indication.
Tip: If the destination IP address for a connection (for example, the partner’s IP address) is a dynamic VIPA or distributed dynamic VIPA residing in the cluster, the internal indication will not be on since traffic for these connections can be forwarded to the target TCP/IP stacks over links or interfaces that are external to the cluster.

For example, exploiting the internal indication might be used by an application to avoid the cost of encrypting data. If an application has just established a connection for which SSL would be the appropriate protection if the partner were not in the sysplex, and the application has assumed or has been configured to know that data within a sysplex is protected by physical security (controlled physical access), then the application might choose to implement the following:

- Immediately after connection setup, but before initiating SSL handshaking, issue the GETSOCKOPT call to obtain SO_CLUSTERCONNTYPE information. If the internal indication is not returned, proceed to initiate the SSL handshaking with appropriate levels of encryption specified (negotiated) between the two connection endpoints.

- If the internal indication is returned, initiate SSL handshaking as usual to gain the benefits of authenticating the partner, but specify only null encryption as an encryption choice. Because support for null encryption is a required feature of SSL, the SSL handshake is not destined to fail for architectural (IETF RFC) reasons. It is then up to the partner to determine whether a negotiated null encryption is acceptable to the partner or the connection should be closed.

While the expensive SSL handshaking cannot be avoided in any case, encryption of the data exchanged between the partners can be turned off as appropriate. If the applications were doing bulk data transfer, and normal encryption would be triple-DES, the savings in CPU cycles might be considerable.

Additional benefits include:

- Avoiding costly application operations (such as parameter marshalling) at the discretion of the application
- Sharing of information that provides the following:
  - Reduced CPU utilization
  - Reduced application workload
  - Better application performance

In general, sockets applications are designed so that any partners (client and server) using the same protocol can be used to connect with each other to do useful work. Typically, applications had to determine (for each partner) its platform and then exchange (through application protocol) this information with its partner. In some cases, this application level exchange cannot be performed:

- If both sides of the connection are not owned by the same company
- If the application protocol is governed by industry standards that do not include platform-related information

The SO_CLUSTERCONNTYPE socket option reports the same image (same MVS image or Virtual Server), same cluster (same sysplex), or cluster internal to a sockets application when a connection is established. The information is determined and reported only when specifically requested, so that the application

---

1. When all of the TCP/IP stacks in the sysplex have been initialized and are in a steady state, they will have exchanged information within the sysplex, such that each stack recognizes all of the IP addresses supported by the other stacks in the sysplex, and which particular stacks support which IP addresses. The name of the MVS image for each stack is also made known.
that does not need to use the function does not incur the expense. This option performs similarly whether the sockets application was the listening (server) application or the initiating (client) application.

If the TCP/IP stacks within a sysplex have been partitioned using a sysplex subplex, then results from the SO_CLUSTERCONNTYPE socket option request are impacted in the following ways depending on what is reported:

- **Same cluster (sysplex) indicator**
  
  If two stacks on the same sysplex belong to different TCP/IP subplexes, this indicator is no longer on. Communication between these two stacks must go across a network interface, as opposed to using a dynamic XCF connection.

- **Same MVS image indicator**
  
  If two stacks on the same MVS system belong to different TCP/IP subplexes, this indicator is no longer on.

- **Internal indicator**
  
  If two TCP/IP stacks are in separate subplexes, connections between them no longer have this indicator on if CTC, MPCPTP (including XCF and IUTSAMEH), or HiperSockets is used. The reason for this is that each stack is not aware of the IP addresses for the partner stack because they belong to different subplexes. Connections using Loopback or one of the local interfaces on a stack will continue to have this indicator on.

The reason the indicators can no longer be set is because when stacks are in different subplexes, they do not appear to each other to be in the same image or cluster.
Appendix B. Well-known port assignments

This topic lists the well-known port assignments for transport protocols TCP and UDP, and includes port number, keyword, and a description of the reserved port assignment. You can also find a list of these well-known port numbers in the hlq.ETC.SERVICES data set. The official assignment of port numbers is managed by the Internet Assigned Numbers Authority. The current list can be viewed at http://www.iana.org/assignments/port-numbers.

Table 119 lists the well-known port assignments for TCP.

<table>
<thead>
<tr>
<th>Port number</th>
<th>Keyword</th>
<th>Assigned to</th>
<th>Services description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>rje</td>
<td>remote job entry</td>
<td>remote job entry</td>
</tr>
<tr>
<td>7</td>
<td>echo</td>
<td>echo</td>
<td>echo</td>
</tr>
<tr>
<td>9</td>
<td>discard</td>
<td>discard</td>
<td>sink null</td>
</tr>
<tr>
<td>11</td>
<td>systat</td>
<td>active users</td>
<td>active users</td>
</tr>
<tr>
<td>13</td>
<td>daytime</td>
<td>daytime</td>
<td>daytime</td>
</tr>
<tr>
<td>15</td>
<td>netstat</td>
<td>netstat</td>
<td>who is up or netstat</td>
</tr>
<tr>
<td>19</td>
<td>chargen</td>
<td>ttytst source</td>
<td>character generator</td>
</tr>
<tr>
<td>21</td>
<td>ftp</td>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>23</td>
<td>telnet</td>
<td>telnet</td>
<td>telnet</td>
</tr>
<tr>
<td>25</td>
<td>smtp</td>
<td>mail</td>
<td>Simple Mail Transfer Protocol</td>
</tr>
<tr>
<td>37</td>
<td>time</td>
<td>timeserver</td>
<td>timeserver</td>
</tr>
<tr>
<td>39</td>
<td>rlp</td>
<td>resource</td>
<td>Resource Location Protocol</td>
</tr>
<tr>
<td>42</td>
<td>nameserver</td>
<td>name</td>
<td>host name server</td>
</tr>
<tr>
<td>43</td>
<td>nicname</td>
<td>who is</td>
<td>who is</td>
</tr>
<tr>
<td>53</td>
<td>domain</td>
<td>name server</td>
<td>domain name server</td>
</tr>
<tr>
<td>57</td>
<td>mtp</td>
<td>private terminal access</td>
<td>private terminal access</td>
</tr>
<tr>
<td>69</td>
<td>tftp</td>
<td>TFTP</td>
<td>Trivial File Transfer protocol</td>
</tr>
<tr>
<td>77</td>
<td>rje</td>
<td>netrjs</td>
<td>any private RJE service</td>
</tr>
<tr>
<td>79</td>
<td>finger</td>
<td>finger</td>
<td>finger</td>
</tr>
<tr>
<td>80</td>
<td>http</td>
<td>http</td>
<td>Web Server</td>
</tr>
<tr>
<td>87</td>
<td>link</td>
<td>ttylink</td>
<td>any private terminal link</td>
</tr>
<tr>
<td>95</td>
<td>supdup</td>
<td>supdup</td>
<td>SUPDUP protocol</td>
</tr>
<tr>
<td>101</td>
<td>hostname</td>
<td>hostname</td>
<td>nic hostname server, usually from SRI-NIC</td>
</tr>
<tr>
<td>109</td>
<td>pop</td>
<td>postoffice</td>
<td>Post Office Protocol</td>
</tr>
<tr>
<td>111</td>
<td>sunrpc</td>
<td>sunrpc</td>
<td>Sun remote procedure call</td>
</tr>
<tr>
<td>113</td>
<td>auth</td>
<td>authentication</td>
<td>authentication service</td>
</tr>
<tr>
<td>115</td>
<td>sftp</td>
<td>sftp</td>
<td>Simple File Transfer Protocol</td>
</tr>
<tr>
<td>117</td>
<td>uucp-path</td>
<td>UUCP path service</td>
<td>UUCP path service</td>
</tr>
<tr>
<td>119</td>
<td>untp</td>
<td>readnews untp</td>
<td>USENET News Transfer Protocol</td>
</tr>
</tbody>
</table>
### Table 119. TCP well-known port assignments (continued)

<table>
<thead>
<tr>
<th>Port number</th>
<th>Keyword</th>
<th>Assigned to</th>
<th>Services description</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>ntp</td>
<td>NTP</td>
<td>Network Time Protocol</td>
</tr>
<tr>
<td>160–223</td>
<td></td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>712</td>
<td>vexec</td>
<td>vice-exec</td>
<td>Andrew File System authenticated service</td>
</tr>
<tr>
<td>713</td>
<td>vlogin</td>
<td>vice-login</td>
<td>Andrew File System authenticated service</td>
</tr>
<tr>
<td>714</td>
<td>vshell</td>
<td>vice-shell</td>
<td>Andrew File System authenticated service</td>
</tr>
<tr>
<td>2001</td>
<td>datasetsrv</td>
<td></td>
<td>Andrew File System service</td>
</tr>
<tr>
<td>2106</td>
<td>venus.itc</td>
<td></td>
<td>Andrew File System service, for the Venus process</td>
</tr>
</tbody>
</table>

### Well-known UDP port assignments

Table 120 lists the well-known port assignments for UDP.

### Table 120. Well-known UDP port assignments

<table>
<thead>
<tr>
<th>Port number</th>
<th>Keyword</th>
<th>Assigned to</th>
<th>Services description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>rje</td>
<td>remote job entry</td>
<td>remote job entry</td>
</tr>
<tr>
<td>7</td>
<td>echo</td>
<td>echo</td>
<td>echo</td>
</tr>
<tr>
<td>9</td>
<td>discard</td>
<td>discard</td>
<td>sink null</td>
</tr>
<tr>
<td>11</td>
<td>users</td>
<td>active users</td>
<td>active users</td>
</tr>
<tr>
<td>13</td>
<td>daytime</td>
<td>daytime</td>
<td>daytime</td>
</tr>
<tr>
<td>15</td>
<td>netstat</td>
<td>Netstat</td>
<td>Netstat</td>
</tr>
<tr>
<td>19</td>
<td>chargen</td>
<td>tty tst source</td>
<td>character generator</td>
</tr>
<tr>
<td>37</td>
<td>time</td>
<td>timeserver</td>
<td>timeserver</td>
</tr>
<tr>
<td>39</td>
<td>rlp</td>
<td>resource</td>
<td>Resource Location Protocol</td>
</tr>
<tr>
<td>42</td>
<td>nameserver</td>
<td>name</td>
<td>host name server</td>
</tr>
<tr>
<td>43</td>
<td>nicname</td>
<td>who is</td>
<td>who is</td>
</tr>
<tr>
<td>53</td>
<td>domain</td>
<td>nameserver</td>
<td>domain name server</td>
</tr>
<tr>
<td>69</td>
<td>tftp</td>
<td>TFTP</td>
<td>Trivial File Transfer Protocol</td>
</tr>
<tr>
<td>75</td>
<td></td>
<td>any private dial out service</td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>rje</td>
<td>netrjs</td>
<td>any private RJE service</td>
</tr>
<tr>
<td>79</td>
<td>finger</td>
<td>finger</td>
<td>finger</td>
</tr>
<tr>
<td>111</td>
<td>sunrpc</td>
<td>sunrpc</td>
<td>Sun remote procedure call</td>
</tr>
<tr>
<td>123</td>
<td>ntp</td>
<td>NTP</td>
<td>Network Time Protocol</td>
</tr>
<tr>
<td>135</td>
<td>llbd</td>
<td>NCS LLBD</td>
<td>NCS local location broker daemon</td>
</tr>
<tr>
<td>161</td>
<td>snmp</td>
<td>SNMP</td>
<td>SNMP server</td>
</tr>
<tr>
<td>162</td>
<td>snmptrap</td>
<td>SNMPTRAP</td>
<td>SNMP trap</td>
</tr>
<tr>
<td>531</td>
<td>rvd-control</td>
<td></td>
<td>rvd control port</td>
</tr>
<tr>
<td>2001</td>
<td>rauth2</td>
<td></td>
<td>Andrew File System service, for the Venus process</td>
</tr>
<tr>
<td>2002</td>
<td>rfilebulk</td>
<td></td>
<td>Andrew File System service, for the Venus process</td>
</tr>
<tr>
<td>Port number</td>
<td>Keyword</td>
<td>Assigned to</td>
<td>Services description</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
<td>-------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>2003</td>
<td>rfilesrv</td>
<td></td>
<td>Andrew File System service, for the Venus process</td>
</tr>
<tr>
<td>2018</td>
<td>console</td>
<td></td>
<td>Andrew File System service</td>
</tr>
<tr>
<td>2115</td>
<td>ropcons</td>
<td></td>
<td>Andrew File System service, for the Venus process</td>
</tr>
<tr>
<td>2131</td>
<td>rupdsrv</td>
<td></td>
<td>assigned in pairs; bulk must be srv +1</td>
</tr>
<tr>
<td>2132</td>
<td>rupdbulk</td>
<td></td>
<td>assigned in pairs; bulk must be srv +1</td>
</tr>
<tr>
<td>2133</td>
<td>rupdsrv1</td>
<td></td>
<td>assigned in pairs; bulk must be srv +1</td>
</tr>
<tr>
<td>2134</td>
<td>rupdbulk1</td>
<td></td>
<td>assigned in pairs; bulk must be srv +1</td>
</tr>
<tr>
<td>12000</td>
<td>entextxid</td>
<td></td>
<td>IBM Enterprise Extender SNA XID Exchange</td>
</tr>
<tr>
<td>12001</td>
<td>entextnetwk</td>
<td></td>
<td>IBM Enterprise Extender SNA COS Network Priority</td>
</tr>
<tr>
<td>12002</td>
<td>entexthigh</td>
<td></td>
<td>IBM Enterprise Extender SNA COS High Priority</td>
</tr>
<tr>
<td>12003</td>
<td>entextmed</td>
<td></td>
<td>IBM Enterprise Extender SNA COS Medium Priority</td>
</tr>
<tr>
<td>12004</td>
<td>entextlow</td>
<td></td>
<td>IBM Enterprise Extender SNA COS Low Priority</td>
</tr>
</tbody>
</table>
Appendix C. Programming interfaces for providing classification data to be used in differentiated services policies

Applications and users of TCP/IP networks might have different requirements for the service they receive from those networks. A network that treats all traffic as best effort might not meet the needs of such users. Service differentiation is a mechanism to provide different service levels to different traffic types based on their requirements and importance in an enterprise network. For example, it might be critical to provide Enterprise Resource Planning (ERP) traffic better service during peak hours than that of FTP or web traffic. The overall service provided to applications or users, in terms of elements such as throughput and delay, is termed Quality of Service (QoS).

One aspect of QoS is Differentiated Services (DS), which provides QoS to broad classes of traffic or users, for example all outbound web traffic accessed by a particular subnet. z/OS provides support for DS by allowing network administrators to define policies that describe how different z/OS TCP/IP workload traffic should be treated. Administrators can define service policy rules that identify desired workloads and map them to service policy actions that dictate the DS attributes assigned to these workloads. For more information on QoS and DS, see the z/OS Communications Server: IP Configuration Guide.

Service policy rules can specify generic attributes to identify a given workload, such as the server’s well-known port or jobname. However, there are cases where a more granular level of classification for a server’s outgoing TCP/IP traffic is desired. For example, a server application might provide services for several different types of requests using a single well-known port. A network administrator might want to be able to specify unique DS attributes for each service type the application supports. One way of accomplishing this is by allowing applications to provide additional information that can be used by an administrator to define more granular service policy rules and actions. The programming interfaces described in this topic provide this capability.

Application defined policy classification data can be specified using extensions to the sendmsg() socket API. The sendmsg() API is similar to other socket APIs, such as send() and write() that allow an application to send data, but also provides the capability of specifying ancillary data. Ancillary data allows applications to pass additional option data to the TCP/IP protocol stack along with the normal data that is sent to the TCP/IP network. This ancillary data can be used by the application to define the attributes of the outgoing traffic for a particular TCP connection or for the specific data being sent in that sendmsg() invocation. These extensions to the sendmsg() API are only available to applications using the TCP protocol and the following socket API libraries:

- z/OS IBM C/C++ sockets with the z/OS Language Environment. For more information about these APIs, see the z/OS XL C/C++ Run-Time Library Reference.
- z/OS UNIX System Services Assembler Callable services socket APIs. For more information about these APIs, see the Assembler Callable Services Reference.

The policy classification data is defined by the application and contains one (or both) of the following two formats:

© Copyright IBM Corp. 1989, 2009
- **Application defined token**: This token is a free format character string that can represent any application defined resource (for example, as transaction identifier, user ID, URL, and so on). When an application passes this token in sendmsg(), TCP/IP will invoke the policy classification function passing it the application-defined token in addition to any of the existing classification attributes (local/remote IP address and port, job name, and so on). The application defined token maps to the ApplicationData attribute of a DS policy rule.

- **Application priority levels**: An application specified priority that maps to one of five predefined QoS service levels: Expedited, High, Medium, Low and Best Effort. Applications using this format of application classification data need to map their outgoing data types to one of these priority levels. For example, the application might already have a concept of transaction priority that it can use to map to one of these priority levels. It is important to note that the priority specified by the application does not automatically translate to a QoS service level. The actual service level assigned is derived by the contents of the service policy. Application priority rules are mapped to the ApplicationPriority attribute of a DS policy rule.

Applications might decide to pass classification data of either format or for both formats. The latter option allows applications to specify the same application defined token yet associate it with different priorities depending on the type of request being processed. For example, an application can pass an application token of ORDER and a HIGH priority for one user and a token of ORDER with a LOW priority for another user. The policy administrator would then be able to distinguish the service level assigned to these two different classes of users. When passing classification data on the sendmsg() API, applications also need to determine the scope of the classification:

- **Connection-Level**: The DS policy action assigned will be used for all traffic on this TCP connection until another sendmsg() with different classification data is specified.

- **Message-Level**: The DS policy action assigned will be used only for the outgoing data passed on this sendmsg() invocation. Any future data sent on this connection without the specification of any classification data will use the original DS policy action that was assigned to this TCP connection.

---

**Passing application classification data on SENDMSG**

A key difference in the sendmsg() API versus the more common send() API is that most parameters are passed in a message header input parameter. The mapping for the message header is defined in *socket.h* for C/C++ and in the *BPXYMSGH* macro for users of the UNIX System Services Assembler Callable services. For simplicity, only the C/C++ version of the data structures are shown in this topic:

```c
struct msghdr {
    void *msg_name; /* optional address */
    size_t msg_namelen; /* size of address */
    struct iovec *msg_iov; /* scatter/gather array */
    int msg_iovlen; /* # elements in msg_iov */
    void *msg_control; /* ancillary data */
    size_t msg_controllen; /* ancillary data length */
    int msg_flags; /* flags on received msg */
};
```

The following are some key points regarding the usage of sendmsg() for the purpose of passing application defined classification data:
Since application policy classification data is only supported for TCP sockets, the `msg_name` and `msg_name_len` parameters are not applicable.

Data to be sent using `sendmsg()` needs to be described in the `msg_iov` structure.

The address of the ancillary data is passed in the `msg_control` field.

`msg_controllen` contains the length of the ancillary data passed.

**Note:** If multiple ancillary data sections are passed, this length should reflect the total length of ancillary data sections.

`msg_flags` is not applicable for `sendmsg()`

The ancillary data (in this case the application classification data) is pointed to by the `msg_control` parameter. This `msg_control` pointer points to the following structure (C/C++ example shown below) that describes the ancillary data (also defined in `socket.h` and `BPXYSMSGH` respectively):

```c
struct cmsghdr {
    size_t cmsg_len; /* data byte count includes hdr */
    int cmsg_level; /* originating protocol */
    int cmsg_type; /* protocol-specific type */
    /* followed by u_char cmsg_data[]; */
};
```

- The `cmsg_len` should be set to the length of the `cmsghdr` plus the length of all application classification data that follows immediately after the `cmsghdr`. This is represented by the commented out `cmsg_data` field.
- The `cmsg_level` must be set to the constant `IPPROTO_IP` for AF_INET sockets and `IPPROTO_IPV6` for AF_INET6 sockets. `IPPROTO_IP` and `IPPROTO_IPV6` are defined in `in.h` and `BPXYSOCK`.
- The `cmsg_type` must be set to the constant `IP_QOS_CLASSIFICATION_DATA` (defined in header file `ezaqosdc.h` for C/C++ users and in macro `EZAQOSDA` for assembler users). The header file and macro are both included in the SEZANMAC data set. This data set must be available in the concatenation when compiling or assembling a part that makes use of these definitions.

The data that follows the `cmsghdr` structure is described by the following structure:

```c
struct ip_qos_classification_data {
    int ip_qos_version; /* Version of structure */
    int ip_qos_classification_scope; /* Classification Scope */
    int ip_qos_classification_type; /* Type of QoS classification */
    u_char ip_qos_reserved[12]; /* Reserved for IBM use */
    int ip_qos_appl_token_len; /* Length of application data */
    /* u_char ip_qos_appl_token[128]; */ /* Application Classification Token */
};
```

The `ip_qos_classification_data` structure should be filled in as follows:

- `ip_qos_version`: This field indicates version of the structure. This must be filled in using the constant `IP_QOS_CURRENT_VERSION`.
- `ip_qos_classification_scope`: Specify a connection level scope (use constant `IP_QOS_CONNECTION_LEVEL`) or a message level scope (constant `IP_QOS_MESSAGE_LEVEL`).

Connection level scope indicates that the DS policy action assigned by the way of classification of this message will remain in effect for all subsequent messages sent until a `sendmsg()` with new classification data is issued. Message level scope indicates that the DS policy action assigned will only be used for the message data included in this `sendmsg()` invocation. Future data sent without classification data will inherit the previous connection level DS policy action.
assignment (from last Connection Level classification by the way of sendmsg() or from the original TCP connection classification during connection establishment).

- **ip_gos_classification_type**: This specification indicates the type of classification data being passed. An application can choose to pass an application defined token, an application specified priority, or both a token and a priority. If the latter option is selected the two selected classification types should be logically ORed together. The following types can be specified:
  - Application defined token classification. A single type should be specified. If more than one type is specified the results are unpredictable.
    - `IP_SET_QOSLEVEL_W_APPL_TOKEN_ASCII`: This indicates that the classification data is a character string in ASCII format. When this option is specified the application token needs to be passed in the `ip_gos_appl_token` field.
  - `IP_SET_QOSLEVEL_W_APPL_TOKEN_EBCDIC`: Same as above except that the string is in EBCDIC format.

  **Note:** The `IP_SET_QOSLEVEL_W_APPL_TOKEN_ASCII` does perform slightly better than this option as the application data specified in the policy is saved in ASCII format inside of the TCP/IP stack, thereby eliminating the need to translate the application defined token on every sendmsg() request.
  - Application defined priority classification. A single type should be specified. If multiple priority types are specified the results are unpredictable.
    - `IP_SET_QOSLEVEL_EXPEDITED`: Indicates that Expedited priority is requested.
    - `IP_SET_QOSLEVEL_HIGH`: Indicates that High priority is requested.
    - `IP_SET_QOSLEVEL_MEDIUM`: Indicates that Medium priority is requested.
    - `IP_SET_QOSLEVEL_LOW`: Indicates that Low priority is requested.
    - `IP_SET_QOSLEVEL_BEST_EFFORT`: Indicates that Best Effort priority is requested.
  - `ip_gos_appl_token_len`: The length of the `ip_gos_appl_token` specified. This length should not include any null terminating characters.
  - `ip_gos_appl_token`: This virtual field immediately follows the `ip_gos_classification_len` field and contains the application classification token string in either ASCII or EBCDIC format depending on which flavor of `IP_SET_QOSLEVEL_W_APPL_TOKEN_xxxx` was specified for the classification type. This field is only referenced when an application defined token type is specified. Note that this string should not exceed 128 bytes. If a larger size is specified, only the first 128 bytes will be used.
Additional SENDMSG considerations

The sendmsg() enhancements to allow for QoS classification data will only be available through the Language Environment C/C++ sendmsg() API and the UNIX System Services BPX2SMS service. The sendmsg() API supported across the TCP/IP provided socket API libraries (C, Macro, Callable, CICS, and so on) do not currently support the passing of ancillary data. Some additional considerations for these sendmsg() enhancements follow:

- UNIX System Services Assembler Callable Services Environment
  - Applications should ensure that the BPX2SMS (sendmsg) service is invoked. An older version of sendmsg(), named BPX1SMS, also exists but does not support the application classification enhancements described in this topic.
  - Include the EZAQOSDA macro from the SEZANMAC library for the definitions needed for the application classification ancillary data.
  - Include the BPXYSOCK and BPXYMSGH macros from SYS1.MACLIB.

- IBM C/C++ applications using the z/OS Language Environment:
  - Applications need to include the following header files:
    - socket.h, in.h
    - ezaqosdc.h (from SEZANMAC)

- AF_INET6 considerations
  The sendmsg() enhancements for QoS classification data are supported for AF_INET6 sockets. However, they are supported only for AF_INET6 sockets when the connection’s traffic flows over an IPv4 network (such as, the remote partner’s IP address is an IPv4-mapped IPv6 address). This feature is not supported for AF_INET6 sockets when the connection’s traffic flows over an IPv6 network (such as, the remote partner’s IP address is an IPv6 address); the sendmsg() enhancements will be ignored if used on an IPv6 connection.

  In order to exploit these enhancements for an AF_INET6 socket, the application should be coded as indicated in this topic, but should substitute IPPROTO_IPV6 for IPPROTO_IP in the cmsghdr’s cmsg_level field.

Note: The Language Environment C/C++ library supports 2 versions of the sendmsg() API. The key difference is in the definition of the msghdr structure. In order to use the correct version of sendmsg() the application needs to ensure that the macro symbolic _OE_SOCKETS is not specified. _OE_SOCKETS causes the older version of msghdr and sendmsg() to be used. The older version does not support passing of application classification data.

Applications providing classification data should document the content and format of this data so that network administrators can use this information when defining DS policies.
Appendix D. X Window System interface V11R4 and Motif version 1.1

Support for X Window System Version 11 Release 4 and Motif Version 1.1 is available as feature HIP614X and is documented here.

The current support, provided as part of the base IP support in z/OS Communications Server, is documented in Chapter 7, “X Window System interface in the z/OS Communications Server environment,” on page 195.

The X Window System support provided with TCP/IP includes the following APIs from the X Window System Version 11 Release 4:
• SEZAXI1L (Xlib, Xmu, Xext, and Xau routines)
• SEZAOLDX (X Release 10 compatibility routines)
• SEZAXTLB (Xt Intrinsics)
• SEZAXAWL (Athena widget set)
• Header files needed for compiling X clients
• Standard MIT X clients
• Sample X clients (XSAMP1, XSAMP2, and XSAMP3)
• SEZARNT1 (a combination of the X Window System libraries listed previously and SEZACMTX)

Note: SEZARNT1 contains the reentrant versions of the libraries.
• SEZARNT2 (Athena widget set for reentrant modules)
• SEZARNT3 (Motif widget set for reentrant modules). The SEZARNT1, SEZARNT2, and SEZARNT3 library members are:
  – Fixed block 80, in object deck format.
  – Compiled with the C/370 RENT compile-time option.
  – Used as input for X Window System and socket programmers who make their programs reentrant.
  – Passed to the C/370 prelinker. Use the prelink utility to combine all input text decks into a single text deck.

The X Window System support provided with TCP/IP also includes the following APIs based on Release 1.1 of the Motif-based widget set:
• SEZAXMLB Motif-based widget set
• Header files needed for compiling clients using the Motif-based widget set.

Three-dimensional graphics are available as an extension of the X Window System. For information about using three-dimensional graphics, see PEXlib Specification and C Language Binding, SR28-5166.

In addition, the X Window System support provided with TCP/IP includes support for z/OS UNIX System Services. For information about the z/OS UNIX System Services support provided, see "X Window System routines: z/OS UNIX System Services support" on page 738.
Software requirements for X Window System interface V11R4 and Motif version 1.1

Application programs using the X Window System API are written in C and should be compiled, linked, and run using the z/OS Language Environment z/OS C/C++ compiler and run-time environment.

To run sample X clients (XSAMP1, XSAMP2, and XSAMP3), you require IBM C for System/370, Library Licensed Program (5688-188).

How the X Window System interface works in the MVS environment

The X Window System is a network transparent protocol that supports windowing and graphics. The protocol is communicated between a client or application and an X server over a reliable bidirectional byte stream. This byte stream is provided by the TCP/IP communication protocol. In the MVS environment, X Window System support consists of a set of application calls that create the X protocol, as requested by the application. This application programming interface allows an application to be created, which uses the X Window System protocol to be displayed on an X server.

In an X Window System environment, the X server distributes user input to and accepts requests from various client programs located either on the same system or elsewhere on a network. The X client code uses sockets to communicate with the X server.

Figure 35 on page 687 shows a high-level abstraction of how the X Window System works in a MVS environment. As an application writer, you need to be concerned only with the client API in writing your application.
The communication path from the MVS X Window System application to the server involves the client code and TCP/IP. The application program that you create is the client part of a client-server relationship. The X server provides access to the resources that are shared among many X applications, such as the screen, keyboard, mouse, fonts, and graphics contexts. A single X server can control more than one physical screen.

Each client can interact with multiple servers, and each server can interact with multiple clients.

If your application is written to the Xlib interface, it calls XOpenDisplay() to start communication with an X server on a workstation. The Xlib code opens a communication path called a socket to the X server, and sends the appropriate X protocol to initiate client-server communication.

The X protocol generated by the Window System client code uses an ISO Latin-1 encoding for character strings, while the MVS encoding for character strings is EBCDIC. The X Window System client code in the MVS environment automatically transforms character strings from EBCDIC to ISO Latin-1 or from ISO Latin-1 to EBCDIC, as needed using internal translate tables.

In the MVS environment, external names must be eight characters or less. Many of the X Window System application programming interface names exceed this limit. To support the X API in MVS, all X names longer than eight characters are remapped to unique names using the C compiler preprocessor. This name remapping is found in a file called X11GLUE.H, which is automatically included in.
your program when you include the standard X header file called XLIB.H. When debugging your application, you can see the XI1GLUE.H file to find the remapped names of the X API routines.

**X Window System interface in the MVS environment: Identifying the target display**

The `user_id.XWINDOWS.DISPLAY` data set is used by the X Window System to identify the host name of the target display.

The following is the format of the environment variable in the `user_id.XWINDOWS.DISPLAY` data set:

```
host_name:target_server.target_screen
```

The environment variable in the `user_id.XWINDOWS.DISPLAY` data set contains the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host_name</td>
<td>Specifies the host name or IP address of the host machine on which the X Window System server is running.</td>
</tr>
<tr>
<td>target_server</td>
<td>Specifies the number of the display server on the host machine.</td>
</tr>
<tr>
<td>target_screen</td>
<td>Specifies the screen to be used on the same target server.</td>
</tr>
</tbody>
</table>

**Notes:**
1. You should be aware that the `userid.XWINDOWS.DISPLAY` data set cannot contain sequence numbers.
2. For information about identifying the target display in z/OS UNIX System Services see, “Identifying the target display in z/OS UNIX System Services” on page 739.

**X Window System interface in the MVS environment: Application resource file**

The X Window System allows you to modify certain characteristics of an application at run time by means of application resources. Typically, application resources are set to tailor the appearance and possibly the behavior of an application. The application resources can specify information about an application’s window sizes, placement, coloring, font usage, and other functional details.

On a UNIX system, this information can be found in the user’s home directory in a file called `.Xdefaults`. In the MVS environment, this data set is called `user_id.X.DEFAULTS`. Each line of this data set represents resource information for an application.

**Note:** For information about the application resource file in z/OS UNIX System Services, see “X Window System routines: z/OS UNIX System Services support” on page 738.
Figure 36 on page 689 shows an example of a set of resources specified for a typical X Window System application.

<table>
<thead>
<tr>
<th>XClock+geometry:</th>
<th>500x60+5-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>XClock+font:</td>
<td>-bitstream-<em><strong>-bold-r-</strong></em>-33-240-***</td>
</tr>
<tr>
<td>XClock+foreground:</td>
<td>orange</td>
</tr>
<tr>
<td>XClock+background:</td>
<td>skyblue</td>
</tr>
<tr>
<td>XClock+borderWidth:</td>
<td>4</td>
</tr>
<tr>
<td>XClock+borderColor:</td>
<td>blue</td>
</tr>
<tr>
<td>XClock+analog:</td>
<td>false</td>
</tr>
</tbody>
</table>

Figure 36. Resources specified for a typical X Window System application

In this example, the xclock application automatically creates a window in the lower left corner of the screen with a digital display in orange letters on a skyblue background.

These resources can also be set on the RESOURCE_MANAGER property of the X server, which allows a single, central place where resources are found, that control all applications, displayed on an X server. You can use the xrdb program to control the X server resource database in the resource property.

xrdb is an X client that you can use either to get or to set the contents of the RESOURCE_MANAGER property on the root window of screen 0. This property is then used by all applications at startup to control the application resource.

---

**X Window System interface in the MVS environment: Creating an application**

To create an application that uses the X Window System protocol, you should study the X Window System application programming interface. In addition, sample programs called XSAMP1, XSAMP2, and XSAMP3 (see “Using sample X Window System programs” on page 695) illustrate simple examples of programs that use the X Window System API. These programs are distributed with TCP/IP.

You should ensure that the first X header file your program includes is the XLIB.H header file. This file defines a number of preprocessor symbols, which enable your program to compile correctly. If your program uses the Xt Intrinsic, you should ensure that the INTRINSIC.H header file is the first X header file included in your program. This file contains a number of preprocessor symbols that allow your program to compile correctly. In addition, these header files include the MVS header files that remap the external names of the X Window System routines to the shorter names used by the X Window System that is supported by TCP/IP.

**X Window System header files**

This topic describes the X Window System, X Intrinsic, Athena widget set, and Motif-based widget set headers used by X Window System applications.
X Window System and Xt Intrinsics header files
The following is a list of X Window System and Xt Intrinsics headers:

- ap@keysy.h
- Atoms.h
- Callback.h
- CharSet.h
- CloseHoo.h
- ComposI.h
- ComposP.h
- Composit.h
- Constral.h
- Constrap.h
- Converte.h
- ConvertI.h
- copyrigh.h
- Core.h
- CoreP.h
- cursorfo.h
- CurrUtil.h
- CvtCache.h
- DECkeysy.h
- DisplayQ.h
- Drawing.h
- Error.h
- EventI.h
- Extutil.h
- fd.h
- InitialI.h
- Initier.h

- IntriniI.h
- IntriniP.h
- Intrinsi.h
- keysym.h
- keysymde.h
- ks@names.h
- Misc.h
- MITEMisc.h
- mitmiscs.h
- multibst.h
- multibuf.h
- Object.h
- ObjectP.h
- PassivGr.h
- poly.h
- Quarks.h
- RectObj.h
- RectObjP.h
- region.h
- Resource.h
- Selectio.h
- shape.h
- shapestr.h
- Shell.h
- ShellP.h
- StdCmap.h
- StdSel.h

- StringDe.h
- SysUtil.h
- Translat.h
- VarargsI.h
- Vendor.h
- VendorP.h
- WinUtil.h
- X.h
- Xatom.h
- Xatomtyp.h
- Xct.h
- Xext.h
- Xkeymap.h
- Xlib.h
- Xlibint.h
- Xlibos.h
- X11glue.h
- Xmu.h
- Xos.h
- Xproto.h
- Xproost.h
- Xresourc.h
- Xtoremap.h

- XawInit.h

Athena widget set header files
The following is a list of the Athena widget set headers:

- ACommand.h
- ACommanP.h
- AForm.h
- AFormP.h
- ALabel.h
- ALabelP.h
- ALIST.h
- ALISTP.h
- ASCROLLb.h
- ASCROLLP.h
- AText.h
- ATextP.h
- ATextSrP.h
- AsciISin.h
- AsciISnkP.h
- AsciISrc.h
- AsciISrcP.h
- AsciITex.h
- AsciITexP.h
- Box.h

- BoxP.h
- Cardinal.h
- Clock.h
- ClockP.h
- CommandI.h
- CommandP.h
- Dialog.h
- DialogP.h
- Grip.h
- GripP.h
- Logo.h
- LogoP.h
- Mailbox.h
- MailboxP.h
- MenuButP.h
- MenuButt.h
- Paned.h
- PanedP.h
- Scroll.h
- Simple.h
- SimpleMP.h

- SimpleMe.h
- SimpleP.h
- Sme.h
- SmeBSBP.h
- SmeBSBP.h
- SmeLine.h
- SmeLineP.h
- SmeP.h
- StripChP.h
- StripCha.h
- Template.h
- TemplateP.h
- TextSink.h
- TextSinP.h
- TextSrc.h
- Toggle.h
- ToggleP.h
- VPaned.h
- Viewport.h
- ViewporP.h
- XawInit.h

690  z/OS V1R11.0 Comm Svr: IP Programmer's Guide and Reference
Motif header files

The following is a list of headers for the Motif-based widget set:

- ArrowB.h
- ArrowBG.h
- ArrowBGP.h
- ArrowBP.h
- bitmaps.h
- BulletBP.h
- Bulletin.h
- CascaBGP.h
- CascadBG.h
- CascaBGP.h
- CascadBP.h
- CascadeB.h
- Command.h
- CommandP.h
- CutPaste.h
- CutPasteP.h
- DialogSP.h
- DrawingA.h
- DrawinAP.h
- DrawnB.h
- DrawnBP.h
- FileSB.h
- FileSBP.h
- Form.h
- FormP.h
- SashP.h
- Frame.h
- FrameP.h
- Scale.h
- ScaleP.h
- Label.h
- LabelGP.h
- ScrollBP.h
- Scrollin.h
- List.h
- SelectBP.h
- ListP.h
- SelectIB.h
- MainW.h
- MainWP.h
- SeparatG.h
- CascadeBP.h
- MenuShel.h
- Separato.h
- Command.h
- MenuShep.h
- SeparatP.h
- CommandP.h
- MessageBP.h
- StringSr.h
- CutPaste.h
- MessageB.h
- Text.h
- CutPasteP.h
- PanedW.h
- TextInP.h
- DialogSP.h
- PanedWP.h
- TextOutP.h
- DrawingA.h
- PushB.h
- TextP.h
- DrawinAP.h
- PushBG.h
- TextSrcP.h
- DrawnB.h
- PushBP.h
- ToggleBGP.h
- DrawnBP.h
- RowColum.h
- ToggleBP.h
- FileSB.h
- RowColuP.h
- Xm.h
- FileSBP.h
- XmP.h

X Window System interface in the MVS environment: Compiling and linking

You can use several methods to compile, link-edit, and execute your program in MVS. This topic contains information about the data sets that you must include to run your C source program under MVS batch using cataloged procedures supplied by IBM.

The following list contains partitioned data set names, which are used as examples in the JCL statements below:

<table>
<thead>
<tr>
<th>Data Set Name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_id.MYPROG.C</td>
<td>Contains user C source programs.</td>
</tr>
<tr>
<td>user_id.MYPROG.C(PROGRAM1)</td>
<td>Member PROGRAM1 in user_id.MYPROG.C partitioned data set.</td>
</tr>
<tr>
<td>user_id.MYPROG.H</td>
<td>Contains user #include files.</td>
</tr>
<tr>
<td>user_id.MYPROG.OBJ</td>
<td>Contains object code for the compiled versions of user C programs in user_id.MYPROG.C.</td>
</tr>
<tr>
<td>user_id.MYPROG.LOAD</td>
<td>Contains link-edited versions of user programs in user_id.MYPROG.OBJ.</td>
</tr>
</tbody>
</table>
X Window System interface in the MVS environment:  
Nonreentrant modules

The following lines describe the additions that you must make to the compile step of your cataloged procedure to compile a nonreentrant module. Catalogued procedures are included in the samples supplied by IBM for your MVS system.

**Note:** Compile all C source using the `def(IBMCPP)` preprocessor symbol.

- Add the following statement as the first `//SYSLIB DD` statement:
  ```
  //SYSLIB DD DSN=SEZACMAC,DISP=SHR
  ```
- Add the following `//USERLIB DD` statement:
  ```
  //USERLIB DD DSN=user_id.MYPROG.H,DISP=SHR
  ```

The following lines describe the additions that you must make to the link-edit step of your cataloged procedure to link-edit a nonreentrant module:

- To link-edit programs that use only X11 library functions, add the following statements as the first `//SYSLIB DD` statements:
  ```
  //    DD DSN=SEZAX11L,DISP=SHR
  //    DD DSN=SEZACMTX,DISP=SHR
  ```
- You must include the following statements when you link-edit your application code, because not all entry points are defined as external references in `SEZAX11L`:
  ```
  INCLUDE SYSLIB(XMACROS)
  INCLUDE SYSLIB(XLIBINT)
  INCLUDE SYSLIB(XRM)
  ```
- To link-edit programs that use the Athena Toolkit functions, including Athena widget sets, add the following after the `//SYSLIB DD` statement:
  ```
  //    DD DSN=SEZAXAWL,DISP=SHR
  //    DD DSN=SEZAXTLB,DISP=SHR
  //    DD DSN=SEZAX11L,DISP=SHR
  //    DD DSN=SEZACMTX,DISP=SHR
  ```
- You must include the following when you link-edit your application code, because not all entry points are defined as external references in `SEZAX11L`, `SEZAXTLB`, and `SEZAXAWL`:
  ```
  INCLUDE SYSLIB(XMACROS)
  INCLUDE SYSLIB(XLIBINT)
  INCLUDE SYSLIB(XRM)
  INCLUDE SYSLIB(CALLBACK)
  INCLUDE SYSLIB(CONVERT)
  INCLUDE SYSLIB(CONVERTE)
  INCLUDE SYSLIB(INTRINSI)
  INCLUDE SYSLIB(DISPLAY)
  INCLUDE SYSLIB(ERROR)
  INCLUDE SYSLIB(EVENT)
  INCLUDE SYSLIB(NEXTEVEN)
  INCLUDE SYSLIB(TMSTATE)
  INCLUDE SYSLIB(ASCTEXT)
  INCLUDE SYSLIB(ATOMS)
  INCLUDE SYSLIB(ATEXT)
  ```
- To link-edit programs that use the Motif Toolkit functions, add the following after the `//SYSLIB DD` statement:
  ```
  //    DD DSN=SEZAXMLB,DISP=SHR
  //    DD DSN=SEZAXTLB,DISP=SHR
  //    DD DSN=SEZAX11L,DISP=SHR
  //    DD DSN=SEZACMTX,DISP=SHR
  ```
You must include the following when you link-edit your application code, because not all entry points are defined as external references in SEZAX1L, SEZAXTLB, and SEZAXMLB.

```
INCLUDE SYSLIB(XMACROS)
INCLUDE SYSLIB(XLIBLENT)
INCLUDE SYSLIB(XRM)
INCLUDE SYSLIB(CALLBACK)
INCLUDE SYSLIB(CONVERT)
INCLUDE SYSLIB(CONVERTE)
INCLUDE SYSLIB(INTRINSI)
INCLUDE SYSLIB(DISPLAY)
INCLUDE SYSLIB(ERROR)
INCLUDE SYSLIB(EVENT)
INCLUDE SYSLIB(NEXTEVEN)
INCLUDE SYSLIB(TMSTATE)
INCLUDE SYSLIB(ATOMS)
INCLUDE SYSLIB(CUTPASTE)
INCLUDE SYSLIB(FILESB)
INCLUDE SYSLIB(GEOOUTILS)
INCLUDE SYSLIB(LIST)
INCLUDE SYSLIB(MANAGER)
INCLUDE SYSLIB(PRIMITIV)
INCLUDE SYSLIB(RESIND)
INCLUDE SYSLIB(ROWCOLUM)
INCLUDE SYSLIB(MSELECTI)
INCLUDE SYSLIB(TEXT)
INCLUDE SYSLIB(TEXTF)
INCLUDE SYSLIB(TRAVERSA)
INCLUDE SYSLIB(VISUAL)
INCLUDE SYSLIB(XMSTRING)
```

**Note:** If you are using X Release 10 compatibility routines, add the following in the //SYSLIB DD statement:

```
// DD DSN=SEZAOIXL,DISP=SHR
```

The following steps describe how to execute your program:

1. Specify the IP address of the X server on which you want to display the application output by creating or modifying the `user_id.XWINDOWS.DISPLAY` data set. The following is an example of a line in this data set.

   `CHARM.RALEIGH.IBM.COM:0.0` or `9.67.43.79:0.0`

2. Allow the host application access to the X server.

3. On the workstation where you want to display the application output, you must grant permission for the MVS host to access the X server. To do this, enter the `xhost` command:

   `xhost ralmvs1`

4. To execute your program under TSO, enter the following:

   `CALL 'user_id.MYPKGLOAD(PROGRAM1)'`

---

**X Window System interface in the MVS environment:**

**Reentrant modules**

The following lines describe the additions that you must make to the compile step of your cataloged procedure to compile a reentrant module. Cataloged procedures are included in the samples supplied by IBM for your MVS system.
Note: Compile all C source using the def(IBMCPP) preprocessor symbol. See "X Window System interface in the MVS environment: Compiling and linking" on page 691 for information about compiling and linking your program in MVS.

- Add the following statement as the first //SYSLIB DD statement:
  
  
- Add the following //USERLIB DD statement:

The following lines describe the additions that you must make to the prelink-edit and link-edit steps of your cataloged procedure to create a reentrant module.

- To create reentrant modules that use only the X11 library functions, do the following:
  - Add the following statement as the first //SYSLIB DD statement in the prelink-edit step:
  
  - Add the following statement as the first //SYSLIB DD statement in the link-edit step:

- To create reentrant modules that use only the Athena Toolkit functions, including Athena widget sets, do the following:
  - Add the following statements as the first //SYSLIB DD statements in the prelink-edit step:
  
  - Add the following statement as the first //SYSLIB DD statement in the link-edit step:

- To create reentrant modules that use only the Motif Toolkit functions, do the following:
  - Add the following statements as the first //SYSLIB DD statements in the prelink-edit step:
  
  - Add the following statement as the first //SYSLIB DD statement in the link-edit step:

Following is a sample cataloged procedure for an X11 library function.

```c
/*-------------------------------------------------------------
/* PRELINK-EDIT STEP:
/*-------------------------------------------------------------
/*PRELNK EXEC PGM=EDCPRLK,REGION=4096K,COND=(4,LT),
   //       PARM='MAP,NONCAL'
   //STEPLIB DD DSN=C370.LL.V2R1M0.SEDCLINK,DISP=SHR
   //   DD DSN=C370.LL.V2R1M0.COMMON.SIBMLINK,DISP=SHR
   //   DD DSN=C370.LL.V2R1M0.SEDCCOMP,DISP=SHR
   //SYSLIB DD DSN=B37.SEZARNT1,DISP=SHR
   //OBJLIB DD DSN=&OBJLIB;,DISP=SHR;
   //SYSMOD DD UNIT=VIO,SPACE=(TRK,(50,10)),DISP=(MOD,PASS),
   //       DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
   //SYMSGS DD DSN=C370.V2R1M0.SEDCMSGS(EDCMSGE),DISP=SHR
   //SYSPRINT DD SYSOUT=&SYSOUT;
   //SYSOUT DD SYSOUT=&SYSOUT;
/*----------------------------------*/
```
Note: For more information about installing a reentrant module in the LPA area, see the
z/OS XL C/C++ User's Guide.

The following steps describe how to execute your program:

1. Specify the IP address of the X server on which you want to display the
application output by creating or modifying the user_id.XWINDOWS.DISPLAY
data set. The following is an example of a line in this data set:

   CHARM.raleigh.ibm.com:2:0.0 or 9.67.43.79:0.0

2. Allow the host application access to the X server.

   On the workstation where you want to display the application output, you
must grant permission for the MVS host to access the X server. To do this, enter
the xhost command:

   xhost ralmvs

3. If you have installed your program in the LPA as a reentrant module and you
want to run it under TSO, enter the following:

   PROGRAM1

Note: For more information about compiling and linking, see the z/OS XL C/C++
User's Guide.

Using sample X Window System programs

This topic contains information about the sample X programs that are provided.
The C source code can be found in the SEZAINST data set.

The following are sample C source programs:

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSAMP1</td>
<td>Xlib sample program</td>
</tr>
<tr>
<td>XSAMP2</td>
<td>Athena widget sample program</td>
</tr>
<tr>
<td>XSAMP3</td>
<td>Motif-based widget sample program</td>
</tr>
</tbody>
</table>

For information about running a sample program, see "X Window System
interface in the MVS environment: Compiling and linking" on page 691 and
"Compiling and linking with z/OS UNIX System Services" on page 739.
### Table 121: Environment variables for X Window System Interface V11r4

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOME</td>
<td>The system initializes this variable at login to the path name of the user’s home directory.</td>
</tr>
<tr>
<td>LANG</td>
<td>Determines the locale category for the native language, local customs, and coded character set in the absence of the LC_ALL and other LC_* (LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, LC_NUMERIC, LC_TIME) environment variables. See Note.</td>
</tr>
<tr>
<td>LOGNAME</td>
<td>The system initializes this variable at login to the user’s login name.</td>
</tr>
<tr>
<td>RESOURCE_NAME</td>
<td>Used by XtOpenDisplay as an alternative specification of an application name. There is no default value.</td>
</tr>
<tr>
<td>SESSION_MANAGER</td>
<td>If defined, this environment variable causes a Session Shell widget to connect to a session manager. There is no default value.</td>
</tr>
<tr>
<td>USER</td>
<td>The name of the user account; this is determined by the name that was entered at login.</td>
</tr>
<tr>
<td>XAPPLRESDIR</td>
<td>Specifies the directory to search for files that contain application defaults.</td>
</tr>
<tr>
<td>XAUTHORITY</td>
<td>Specifies the name of the authority file on the local host.</td>
</tr>
<tr>
<td>XBLANGPATH</td>
<td>Used to locate desktop icons, if XMICONBMSEARCHPATH or XMICONSEARCHPATH are not set.</td>
</tr>
<tr>
<td>XENVIRONMENT</td>
<td>Contains the full path name of the file that contains resource defaults. There is no default value.</td>
</tr>
<tr>
<td>XFILESEARCHPATH</td>
<td>Specifies where the X resources file for the current locale is located.</td>
</tr>
<tr>
<td>XLOCALEDIR</td>
<td>Specifies the directory to search for locale files. The default value is /usr/lib/X11/locale.</td>
</tr>
<tr>
<td>XMODIFIERS</td>
<td>Used by the XSetLocaleModifiers function to specify additional modifiers. There is no default value.</td>
</tr>
<tr>
<td>XPROPFORMATS</td>
<td>Specifies the name of a file from which to obtain additional formats.</td>
</tr>
<tr>
<td>XUSERFILEPATH</td>
<td>Specifies the search paths for files containing application defaults. There is no default value.</td>
</tr>
</tbody>
</table>
Table 121. Environment variables for X Window System Interface V11r4 (continued)

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| XWTRACE              | Controls the generation of traces of the socket level communications between Xlib and the X Window System server. These traces are as follows:
  - XWTRACE undefined or 0: No trace generated
  - XWTRACE=1: Error messages
  - XWTRACE>=2: API function tracing for TRANS functions
  - There is no default value. The output is sent to stderr. |

Note: This can be used by applications to determine the language to use for error messages, instructions, collating sequences, date formats, and so on.

Standard X client applications

The following standard MIT X clients are also provided with TCP/IP as examples of how to use the X Window System API:

<table>
<thead>
<tr>
<th>Application</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>appres</td>
<td>Lists application resource database</td>
</tr>
<tr>
<td>atobm</td>
<td>Bit map conversion utilities</td>
</tr>
<tr>
<td>bitmap</td>
<td>Bit map editor</td>
</tr>
<tr>
<td>bmtoa</td>
<td>Bit map conversion utilities</td>
</tr>
<tr>
<td>listres</td>
<td>Lists resources in widgets</td>
</tr>
<tr>
<td>oclock</td>
<td>Displays time of day</td>
</tr>
<tr>
<td>xauth</td>
<td>X authority data set utility</td>
</tr>
<tr>
<td>xcalc</td>
<td>Scientific calculator for X</td>
</tr>
<tr>
<td>xclock</td>
<td>Analog/digital clock for X</td>
</tr>
<tr>
<td>xdpynfo</td>
<td>Displays information utility for X</td>
</tr>
<tr>
<td>xfd</td>
<td>Font displayer for X</td>
</tr>
<tr>
<td>xfontsel</td>
<td>Point and click interface for selecting X11 font names</td>
</tr>
<tr>
<td>xkill</td>
<td>Stops a client by its X resource</td>
</tr>
<tr>
<td>xlogo</td>
<td>X Window System logo</td>
</tr>
<tr>
<td>xlsatoms</td>
<td>Lists interned atoms defined on server</td>
</tr>
<tr>
<td>xlsclients</td>
<td>Lists client applications running on a display</td>
</tr>
<tr>
<td>xlsfonts</td>
<td>Displays server font list displayer for X</td>
</tr>
<tr>
<td>xlswins</td>
<td>Displays server window list displayer for X</td>
</tr>
<tr>
<td>xmag</td>
<td>Magnify parts of the screen</td>
</tr>
<tr>
<td>xprop</td>
<td>Property displayer for X</td>
</tr>
<tr>
<td>xrdb</td>
<td>X server resource database utility</td>
</tr>
<tr>
<td>xrefresh</td>
<td>Refreshes all or part of an X screen</td>
</tr>
</tbody>
</table>
xset User preference utility for X
xsetroot Root window parameter setting utility for X
xwd Dumps an image of an X window
xwininfo Window information utility for X
xwud Displays image displayer for X

These standard X Window client application programs also contain information about X Window System programming techniques.

Consult the following members of the SEZAINST data set for documentation about the MIT X clients:

<table>
<thead>
<tr>
<th>Member Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HLPAPPRE</td>
<td>Help for APPRES module</td>
</tr>
<tr>
<td>HLPBITMA</td>
<td>Help for BITMAP module</td>
</tr>
<tr>
<td>HLPLISTR</td>
<td>Help for LISTRES module</td>
</tr>
<tr>
<td>HLPLOCLOC</td>
<td>Help for OCLOCK module</td>
</tr>
<tr>
<td>HLPXAUTH</td>
<td>Help for XAUTH module</td>
</tr>
<tr>
<td>HLPXCALC</td>
<td>Help for XCALC module</td>
</tr>
<tr>
<td>HLPXCLOC</td>
<td>Help for XCLOCK module</td>
</tr>
<tr>
<td>HLPXDPYI</td>
<td>Help for XDPYINFO module</td>
</tr>
<tr>
<td>HLPXFDM</td>
<td>Help for XFD module</td>
</tr>
<tr>
<td>HLPXFONT</td>
<td>Help for XFONTSEL module</td>
</tr>
<tr>
<td>HLPXKILL</td>
<td>Help for XKILL module</td>
</tr>
<tr>
<td>HLPXLOGO</td>
<td>Help for XLOGO module</td>
</tr>
<tr>
<td>HLPXLSAT</td>
<td>Help for XLSATOMS module</td>
</tr>
<tr>
<td>HLPXLSCL</td>
<td>Help for XLSCLIEN module</td>
</tr>
<tr>
<td>HLPXLSFO</td>
<td>Help for XLSFONTS module</td>
</tr>
<tr>
<td>HLPXLSWI</td>
<td>Help for XLSWINS module</td>
</tr>
<tr>
<td>HLPXMAG</td>
<td>Help for XMAG module</td>
</tr>
<tr>
<td>HLPXPROP</td>
<td>Help for XPROP module</td>
</tr>
<tr>
<td>HLPXRDB</td>
<td>Help for XRDB module</td>
</tr>
<tr>
<td>HLPXREFR</td>
<td>Help for XREFRESH module</td>
</tr>
<tr>
<td>HLPXSET</td>
<td>Help for XSET module</td>
</tr>
<tr>
<td>HLPXSETR</td>
<td>Help for XSETROOT module</td>
</tr>
<tr>
<td>HLPXWD</td>
<td>Help for XWD module</td>
</tr>
<tr>
<td>HLPXWINI</td>
<td>Help for XWININFO module</td>
</tr>
<tr>
<td>HLPXWUD</td>
<td>Help for XWUD module</td>
</tr>
</tbody>
</table>

The SEZAINST data set also contains default application resource data sets for XCALC, XCLOCK, XFD, and XFONTSEL. Copy these data sets from:

- SEZAINST(XXCALC)
to the following data sets for TSO users:

- user_id.XAPDF.XCALC
- user_id.XAPDF.XCLOCK
- user_id.XAPDF.XFD
- user_id.XAPDF.XFONTSEL

Notes:

1. The EZAGETIN job includes JCL to copy the sample members from SEZAINST to user_id.XAPDF.classname, where classname is the application specified class name. The high-level qualifier should be tailored to be the user ID using these data sets.

2. For information on default application resource data sets for z/OS UNIX System Services users, see “X Window System routines: z/OS UNIX System Services support” on page 738.

Building X client modules

The support for X Window System Version 11 Release 4 provides standard MIT X clients. The C source and header files are found in SEZAINST and SEZACMAC data sets respectively.

You can build the following X client modules based on X11 functions:

Table 122. Building X client modules based on X11 functions.

<table>
<thead>
<tr>
<th>To build module</th>
<th>Do the following</th>
</tr>
</thead>
</table>
| ATOBM           | 1. Compile the ATOBM C source program.  
|                 | 2. Link-edit the ATOBM object module. |
| BITMAP          | 1. Compile the BITMAP C source program.  
|                 | 2. Compile the BMDIALOG C source program.  
|                 | 3. Link-edit the BITMAP and BMDIALOG object modules. |
| BMTOA           | 1. Compile the BMTOA C source program.  
|                 | 2. Link-edit the BMTOA object module. |
| XAUTH           | 1. Compile the XAUTH C source program.  
|                 | 2. Compile the GHOSTXA C source program.  
|                 | 3. Compile the PROCESS source program.  
|                 | 4. Compile the PARSEDPY C source program.  
|                 | 5. Link-edit the XAUTH, GHOSTXA, PROCESS, and PARSEDPY object modules. |
| XDPYINFO C      | 1. Compile the XDPYINFO C source program.  
|                 | 2. Link-edit the XDPYINFO object module. |
| XKILL           | 1. Compile the XKILL C source program.  
|                 | 2. Link-edit the XKILL object module. |
| XLSATOMS        | 1. Compile the XLSATOMS C source program.  
|                 | 2. Link-edit the XLSATOMS object module. |
**Table 122. Building X client modules based on X11 functions. (continued)**

<table>
<thead>
<tr>
<th>To build module</th>
<th>Do the following</th>
</tr>
</thead>
</table>
| XLSCLIEN        | 1. Compile the XLSCLIEN C source program.  
                  2. Link-edit the XLSCLIEN object module. |
| XLSFONTS        | 1. Compile the XLSFONTS C source program.  
                  2. Compile the DSIMPLE C source program.  
                  3. Link-edit the XLSFONTS and DSIMPLE object modules. |
| XLSWINS         | 1. Compile the XLSWINS C source program.  
                  2. Link-edit the XLSWINS object module. |
| XMAG            | 1. Compile the XMAG C source program.  
                  2. Link-edit the XMAG object module. |
| XPROP           | 1. Compile the XPROP C source program.  
                  2. Compile the DSIMPLE C source program.  
                  3. Link-edit the XPROP and DSIMPLE object modules. |
| XRDB            | 1. Compile the XRDB C source program.  
                  2. Link-edit the XRDB object module. |
| XREFRESH        | 1. Compile the XREFRESH C source program.  
                  2. Link-edit the XREFRESH object module. |
| XSET            | 1. Compile the XSET C source program.  
                  2. Link-edit the XSET object module. |
| XSETROOT        | 1. Compile the XSETROOT C source program.  
                  2. Link-edit the XSETROOT object module. |
| XWD             | 1. Compile the XWD C source program.  
                  2. Compile the DSIMPLE C source program.  
                  3. Link-edit the XWD and DSIMPLE object modules. |
| XWININFO        | 1. Compile the XWININFO C source program.  
                  2. Compile the DSIMPLE C source program.  
                  3. Link-edit the XWININFO and DSIMPLE object modules. |
| XWUD            | 1. Compile the XWUD C source program.  
                  2. Link-edit the XWUD object module. |

You can build the following X client modules based on Xt Intrinsics and Athena Toolkit functions:

**Table 123. Building X client modules based on Xt Intrinsics and Athena Toolkit functions.**

<table>
<thead>
<tr>
<th>To build module</th>
<th>Do the following</th>
</tr>
</thead>
</table>
| APPRES          | 1. Compile the APPRES C source program.  
                  2. Link-edit the APPRES object module. |
| OCLOCK          | 1. Compile the OCLOCK C source program.  
                  2. Compile the NCLOCK C source program.  
                  3. Compile the TRANSFOR C source program.  
                  4. Link-edit the OCLOCK, NCLOCK, and TRANSFOR object modules. |
Table 123. Building X client modules based on Xt Intrinsics and Athena Toolkit functions. (continued)

<table>
<thead>
<tr>
<th>To build module</th>
<th>Do the following</th>
</tr>
</thead>
<tbody>
<tr>
<td>LISTRES</td>
<td>1. Compile the LISTRES C source program.</td>
</tr>
<tr>
<td></td>
<td>2. Compile the UTIL C source program.</td>
</tr>
<tr>
<td></td>
<td>3. Compile the WIDGETS C source program.</td>
</tr>
<tr>
<td></td>
<td>4. Link-edit the LISTRES, UTIL, and WIDGETS object modules.</td>
</tr>
<tr>
<td>XCALC</td>
<td>1. Compile the XCALC C source program.</td>
</tr>
<tr>
<td></td>
<td>2. Compile the ACTIONS C source program.</td>
</tr>
<tr>
<td></td>
<td>3. Compile the MATH C source program.</td>
</tr>
<tr>
<td></td>
<td>4. Link-edit the XCALC, ACTIONS, and MATH object modules.</td>
</tr>
<tr>
<td>XCLOCK</td>
<td>1. Compile the XCLOCK C source program.</td>
</tr>
<tr>
<td></td>
<td>2. Link-edit the XCLOCK object module.</td>
</tr>
<tr>
<td>XFD</td>
<td>1. Compile the XFD C source program.</td>
</tr>
<tr>
<td></td>
<td>2. Compile the FONTGRID C source program.</td>
</tr>
<tr>
<td></td>
<td>3. Link-edit the XFD and FONTGRID object modules.</td>
</tr>
<tr>
<td>XFONTSEL</td>
<td>1. Compile the XFONTSEL C source program.</td>
</tr>
<tr>
<td></td>
<td>2. Link-edit the XFONTSEL object module.</td>
</tr>
<tr>
<td>XLOGO</td>
<td>1. Compile the XLOGO C source program.</td>
</tr>
<tr>
<td></td>
<td>2. Link-edit the XLOGO object module.</td>
</tr>
</tbody>
</table>

X Window System routines

The following tables list the routines supported by TCP/IP. The routines are grouped according to the type of function provided.

X Window System routines: Opening and closing a display

Table 124 provides the routines for opening and closing a display.

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCloseDisplay()</td>
<td>Closes a display.</td>
</tr>
<tr>
<td>XFree()</td>
<td>Frees in-memory data created by Xlib function.</td>
</tr>
<tr>
<td>XNoOp()</td>
<td>Executes a NoOperation protocol request.</td>
</tr>
<tr>
<td>XOpenDisplay()</td>
<td>Opens a display.</td>
</tr>
</tbody>
</table>

X Window System routines: Creating and destroying windows

Table 125 provides the routines for creating and destroying windows.

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XConfigureWindow()</td>
<td>Configures the specified window.</td>
</tr>
<tr>
<td>XCreateSimpleWindow()</td>
<td>Creates unmapped InputOutput subwindow.</td>
</tr>
</tbody>
</table>
### Table 125. Creating and destroying windows (continued)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCreateWindow()</td>
<td>Creates unmapped subwindow.</td>
</tr>
<tr>
<td>XDestroySubwindows()</td>
<td>Destroys all subwindows of specified window.</td>
</tr>
<tr>
<td>XDestroyWindow()</td>
<td>Unmaps and destroys window and all subwindows.</td>
</tr>
</tbody>
</table>

### X Window System routines: Manipulating windows

Table 126 provides the routines for manipulating windows.

### Table 126. Manipulating windows

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCirculateSubwindows()</td>
<td>Circulates a subwindow up or down.</td>
</tr>
<tr>
<td>XCirculateSubwindowsUp()</td>
<td>Raises the lowest mapped child of window.</td>
</tr>
<tr>
<td>XCirculateSubwindowsDown()</td>
<td>Lowers the highest mapped child of window.</td>
</tr>
<tr>
<td>XIconifyWindow()</td>
<td>Sends a WM_CHANGE_STATE ClientMessage to the root window of the specified screen.</td>
</tr>
<tr>
<td>XLowerWindow()</td>
<td>Lowers the specified window.</td>
</tr>
<tr>
<td>XMapRaised()</td>
<td>Maps and raises the specified window.</td>
</tr>
<tr>
<td>XMapSubwindows()</td>
<td>Maps all subwindows of the specified window.</td>
</tr>
<tr>
<td>XMapWindow()</td>
<td>Maps the specified window.</td>
</tr>
<tr>
<td>XMoveResizeWindow()</td>
<td>Changes the specified window size and location.</td>
</tr>
<tr>
<td>XMoveWindow()</td>
<td>Moves the specified window.</td>
</tr>
<tr>
<td>XRaiseWindow()</td>
<td>Raises the specified window.</td>
</tr>
<tr>
<td>XReconfigureWMWindow()</td>
<td>Issues a ConfigureWindow request on the specified top-level window.</td>
</tr>
<tr>
<td>XResizeWindow()</td>
<td>Changes the specified window’s size.</td>
</tr>
<tr>
<td>XRestackWindows()</td>
<td>Restacks a set of windows from top to bottom.</td>
</tr>
<tr>
<td>XSetWindowBorderWidth()</td>
<td>Changes the border width of the window.</td>
</tr>
<tr>
<td>XUnmapSubwindows()</td>
<td>Unmaps all subwindows of the specified window.</td>
</tr>
<tr>
<td>XUnmapWindow()</td>
<td>Unmaps the specified window.</td>
</tr>
<tr>
<td>XWithdrawWindow()</td>
<td>Unmaps the specified window and sends a synthetic UnmapNotify event to the root window of the specified screen.</td>
</tr>
</tbody>
</table>

### X Window System routines: Changing window attributes

Table 127 provides the routines for changing window attributes.

### Table 127. Changing window attributes

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XChangeWindowAttributes()</td>
<td>Changes one or more window attributes.</td>
</tr>
<tr>
<td>XSetWindowBackground()</td>
<td>Sets the window background to a specified pixel.</td>
</tr>
<tr>
<td>XSetWindowBackgroundPixmap()</td>
<td>Sets the window background to a specified pixmap.</td>
</tr>
<tr>
<td>XSetWindowBorder()</td>
<td>Changes the window border to a specified pixel.</td>
</tr>
<tr>
<td>XSetWindowBorderPixmap()</td>
<td>Changes the window border tile.</td>
</tr>
</tbody>
</table>
### Table 127. Changing window attributes (continued)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XTranslateCoordinates()</td>
<td>Transforms coordinates between windows.</td>
</tr>
</tbody>
</table>

### X Window System routines: Obtaining window information

Table 128 provides the routines for obtaining window information.

### Table 128. Obtaining window information

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XGetGeometry()</td>
<td>Gets the current geometry of the specified drawable.</td>
</tr>
<tr>
<td>XGetWindowAttributes()</td>
<td>Gets the current attributes for the specified window.</td>
</tr>
<tr>
<td>XQueryPointer()</td>
<td>Gets the pointer coordinates and the root window.</td>
</tr>
<tr>
<td>XQueryTree()</td>
<td>Obtains the IDs of the children and parent windows.</td>
</tr>
</tbody>
</table>

### X Window System routines: Obtaining properties and atoms

Table 129 provides the routines for obtaining properties and atoms.

### Table 129. Properties and atoms

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XGetAtomName()</td>
<td>Gets a name for the specified atom ID.</td>
</tr>
<tr>
<td>XInternAtom()</td>
<td>Gets an atom for the specified name.</td>
</tr>
</tbody>
</table>

### X Window System routines: Manipulating window properties

Table 130 provides the routines for manipulating the properties of windows.

### Table 130. Manipulating window properties

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XChangeProperty()</td>
<td>Changes the property for the specified window.</td>
</tr>
<tr>
<td>XDeleteProperty()</td>
<td>Deletes a property for the specified window.</td>
</tr>
<tr>
<td>XGetWindowProperty()</td>
<td>Gets the atom type and property format for the window.</td>
</tr>
<tr>
<td>XListProperties()</td>
<td>Gets the specified window property list.</td>
</tr>
<tr>
<td>XRotateWindowProperties()</td>
<td>Rotates the properties in a property array.</td>
</tr>
</tbody>
</table>

### X Window System routines: Setting window selections

Table 131 provides the routines for setting window selections.

### Table 131. Setting window selections

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XConvertSelection()</td>
<td>Converts a selection.</td>
</tr>
<tr>
<td>XGetSelectionOwner()</td>
<td>Gets the selection owner.</td>
</tr>
<tr>
<td>XSetSelectionOwner()</td>
<td>Sets the selection owner.</td>
</tr>
</tbody>
</table>

### X Window System routines: Manipulating colormaps

Table 132 on page 704 provides the routines for manipulating color maps.
### Table 132. Manipulating colormaps

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XAllocStandardColormap()</td>
<td>Allocates an XStandardColormap structure.</td>
</tr>
<tr>
<td>XCopyColormapAndFree()</td>
<td>Creates a new colormap from a specified colormap.</td>
</tr>
<tr>
<td>XCreateColormap()</td>
<td>Creates a colormap.</td>
</tr>
<tr>
<td>XFreeColormap()</td>
<td>Frees the specified colormap.</td>
</tr>
<tr>
<td>XQueryColor()</td>
<td>Queries the RGB value for a specified pixel.</td>
</tr>
<tr>
<td>XQueryColors()</td>
<td>Queries the RGB values for an array of pixels.</td>
</tr>
<tr>
<td>XSetWindowColormap()</td>
<td>Sets the colormap of the specified window.</td>
</tr>
</tbody>
</table>

### X Window System routines: Manipulating color cells

Table 133 provides the routines for manipulating color cells.

### Table 133. Manipulating color cells

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XAllocColor()</td>
<td>Allocates a read-only color cell.</td>
</tr>
<tr>
<td>XAllocColorCells()</td>
<td>Allocates read/write color cells.</td>
</tr>
<tr>
<td>XAllocColorPlanes()</td>
<td>Allocates read/write color resources.</td>
</tr>
<tr>
<td>XAllocNamedColor()</td>
<td>Allocates a read-only color cell by name.</td>
</tr>
<tr>
<td>XFreeColors()</td>
<td>Frees colormap cells.</td>
</tr>
<tr>
<td>XLookupColor()</td>
<td>Looks up a colorname.</td>
</tr>
<tr>
<td>XStoreColor()</td>
<td>Stores an RGB value into a single colormap cell.</td>
</tr>
<tr>
<td>XStoreColors()</td>
<td>Stores RGB values into colormap cells.</td>
</tr>
<tr>
<td>XStoreNamedColor()</td>
<td>Sets a pixel color to the named color.</td>
</tr>
</tbody>
</table>

### X Window System routines: Creating and freeing pixmaps

Table 134 provides the routines for creating and freeing pixmaps.

### Table 134. Creating and freeing pixmaps

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCreatePixmap()</td>
<td>Creates a pixmap of a specified size.</td>
</tr>
<tr>
<td>XFreePixmap()</td>
<td>Frees all storage associated with specified pixmap.</td>
</tr>
</tbody>
</table>

### X Window System routines: Manipulating graphics contexts

Table 135 provides the routines for manipulating graphics contexts.

### Table 135. Manipulating graphics contexts

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XChangeGC()</td>
<td>Changes the components in the specified Graphics Context (GC).</td>
</tr>
<tr>
<td>XCopyGC()</td>
<td>Copies the components from a source GC to a destination GC.</td>
</tr>
<tr>
<td>XCreateGC()</td>
<td>Creates a new GC.</td>
</tr>
<tr>
<td>XFreeGC()</td>
<td>Frees the specified GC.</td>
</tr>
</tbody>
</table>
Table 135. Manipulating graphics contexts (continued)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XGetGCValues()</td>
<td>Returns the GC values in the specified structure.</td>
</tr>
<tr>
<td>XGContextFromGC()</td>
<td>Obtains the GContext resource ID for GC.</td>
</tr>
<tr>
<td>XQueryBestTile()</td>
<td>Gets the best fill tile shape.</td>
</tr>
<tr>
<td>XQueryBestSize()</td>
<td>Gets the best size tile, stipple, or cursor.</td>
</tr>
<tr>
<td>XQueryBestStipple()</td>
<td>Gets the best stipple shape.</td>
</tr>
<tr>
<td>XSetArcMode()</td>
<td>Sets the arc mode of the specified GC.</td>
</tr>
<tr>
<td>XSetBackground()</td>
<td>Sets the background of the specified GC.</td>
</tr>
<tr>
<td>XSetClipmask()</td>
<td>Sets the clip_mask of the specified GC to a specified pixmap.</td>
</tr>
<tr>
<td>XSetClipOrigin()</td>
<td>Sets the clip origin of the specified GC.</td>
</tr>
<tr>
<td>XSetClipRectangles()</td>
<td>Sets the clip_mask of GC to a list of rectangles.</td>
</tr>
<tr>
<td>XSetDashes()</td>
<td>Sets the dashed line style components of a specified GC.</td>
</tr>
<tr>
<td>XSetFillRule()</td>
<td>Sets the fill rule of the specified GC.</td>
</tr>
<tr>
<td>XSetFillStyle()</td>
<td>Sets the fill style of the specified GC.</td>
</tr>
<tr>
<td>XSetFont()</td>
<td>Sets the current font of the specified GC.</td>
</tr>
<tr>
<td>XSetForeground()</td>
<td>Sets the foreground of the specified GC.</td>
</tr>
<tr>
<td>XSetFunction()</td>
<td>Sets display function in the specified GC.</td>
</tr>
<tr>
<td>XSetGraphicsExposures()</td>
<td>Sets the graphics exposure flag of the specified GC.</td>
</tr>
<tr>
<td>XSetLineAttributes()</td>
<td>Sets the line drawing components of the GC.</td>
</tr>
<tr>
<td>XSetPlaneMask()</td>
<td>Sets the plane mask of the specified GC.</td>
</tr>
<tr>
<td>XSetState()</td>
<td>Sets the foreground, background, plane mask, and function in GC.</td>
</tr>
<tr>
<td>XSetStipple()</td>
<td>Sets the stipple of the specified GC.</td>
</tr>
<tr>
<td>XSetSubwindowMode()</td>
<td>Sets the subwindow mode of the specified GC.</td>
</tr>
<tr>
<td>XSetTile()</td>
<td>Sets the fill tile of the specified GC.</td>
</tr>
<tr>
<td>XSetTSOrigin()</td>
<td>Sets the tile or stipple origin of the specified GC.</td>
</tr>
</tbody>
</table>

X Window System routines: Clearing and copying areas

Table 136 provides the routines for clearing and copying areas.

Table 136. Clearing and copying areas

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XClearArea()</td>
<td>Clears a rectangular area of the window.</td>
</tr>
<tr>
<td>XClearWindow()</td>
<td>Clears the entire window.</td>
</tr>
<tr>
<td>XCopyArea()</td>
<td>Copies the drawable area between drawables of the same root and the same depth.</td>
</tr>
<tr>
<td>XCopyPlane()</td>
<td>Copies single bit plane of the drawable.</td>
</tr>
</tbody>
</table>

X Window System routines: Drawing lines

Table 137 on page 706 provides the routines for drawing lines.
Table 137. Drawing lines

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XDraw()</td>
<td>Draws an arbitrary polygon or curve that is defined by the specified list of Vertexes as specified in vlist.</td>
</tr>
<tr>
<td>XDrawArc()</td>
<td>Draws a single arc in the drawable.</td>
</tr>
<tr>
<td>XDrawArcs()</td>
<td>Draws multiple arcs in a specified drawable.</td>
</tr>
<tr>
<td>XDrawFilled()</td>
<td>Draws arbitrary polygons or curves and then fills them.</td>
</tr>
<tr>
<td>XDrawLine()</td>
<td>Draws a single line between two points in a drawable.</td>
</tr>
<tr>
<td>XDrawLines()</td>
<td>Draws multiple lines in the specified drawable.</td>
</tr>
<tr>
<td>XDrawPoint()</td>
<td>Draws a single point in the specified drawable.</td>
</tr>
<tr>
<td>XDrawPoints()</td>
<td>Draws multiple points in the specified drawable.</td>
</tr>
<tr>
<td>XDrawRectangle()</td>
<td>Draws an outline of a single rectangle in the drawable.</td>
</tr>
<tr>
<td>XDrawRectangles()</td>
<td>Draws an outline of multiple rectangles in the drawable.</td>
</tr>
<tr>
<td>XDrawSegments()</td>
<td>Draws multiple line segments in the specified drawable.</td>
</tr>
</tbody>
</table>

X Window System routines: Filling areas

Table 138 provides the routines for filling areas.

Table 138. Filling areas

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XFillArc()</td>
<td>Fills single arc in drawable.</td>
</tr>
<tr>
<td>XFillArcs()</td>
<td>Fills multiple arcs in drawable.</td>
</tr>
<tr>
<td>XFillPolygon()</td>
<td>Fills a polygon area in the drawable.</td>
</tr>
<tr>
<td>XFillRectangle()</td>
<td>Fills single rectangular area in the drawable.</td>
</tr>
<tr>
<td>XFillRectangles()</td>
<td>Fills multiple rectangular areas in the drawable.</td>
</tr>
</tbody>
</table>

X Window System routines: Loading and freeing fonts

Table 139 provides the routines for loading and freeing fonts.

Table 139. Loading and freeing fonts

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XFreeFont()</td>
<td>Unloads the font and frees the storage used by the font.</td>
</tr>
<tr>
<td>XFreeFontInfo()</td>
<td>Frees the font information array.</td>
</tr>
<tr>
<td>XFreeFontNames()</td>
<td>Frees a font name array.</td>
</tr>
<tr>
<td>XFreeFontPath()</td>
<td>Frees data returned by XGetFontPath.</td>
</tr>
<tr>
<td>XGetFontPath()</td>
<td>Gets the current font search path.</td>
</tr>
<tr>
<td>XGetFontProperty()</td>
<td>Gets the specified font property.</td>
</tr>
<tr>
<td>XListFontsWithInfo()</td>
<td>Gets names and information about loaded fonts.</td>
</tr>
<tr>
<td>XLoadFont()</td>
<td>Loads a font.</td>
</tr>
<tr>
<td>XLoadQueryFont()</td>
<td>Loads and queries font in one operation.</td>
</tr>
<tr>
<td>XListFonts()</td>
<td>Gets a list of available font names.</td>
</tr>
<tr>
<td>XQueryFont()</td>
<td>Gets information about a loaded font.</td>
</tr>
<tr>
<td>XSetFontPath()</td>
<td>Sets the font search path.</td>
</tr>
</tbody>
</table>
Table 139. Loading and freeing fonts (continued)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XUnloadFont()</td>
<td>Unloads the specified font.</td>
</tr>
</tbody>
</table>

**X Window System routines: Querying character string sizes**

Table 140 provides the routines for querying the character size of a string.

Table 140. Querying character string sizes

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XFreeStringList()</td>
<td>Frees the in-memory data associated with the specified string list.</td>
</tr>
<tr>
<td>XQueryTextExtents()</td>
<td>Gets a 1-byte character string bounding box from the server.</td>
</tr>
<tr>
<td>XQueryTextExtents16()</td>
<td>Gets a 2-byte character string bounding box from the server.</td>
</tr>
<tr>
<td>XStringListToTextProperty()</td>
<td>Converts lists of pointers to character strings and text properties.</td>
</tr>
<tr>
<td>XTextExtents()</td>
<td>Gets a bounding box of a 1-byte character string.</td>
</tr>
<tr>
<td>XTextExtents16()</td>
<td>Gets a bounding box of a 2-byte character string.</td>
</tr>
<tr>
<td>XTextPropertyToStringList()</td>
<td>Returns a list of strings representing the elements of the specified XTextProperty structure.</td>
</tr>
<tr>
<td>XTextWidth()</td>
<td>Gets the width of an 8-bit character string.</td>
</tr>
<tr>
<td>XTextWidth16()</td>
<td>Gets the width of a 2-byte character string.</td>
</tr>
</tbody>
</table>

**X Window System routines: Drawing text**

Table 141 provides the routines for drawing text.

Table 141. Drawing text

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XDrawImageString()</td>
<td>Draws 8-bit image text in the specified drawable.</td>
</tr>
<tr>
<td>XDrawImageString16()</td>
<td>Draws 2-byte image text in the specified drawable.</td>
</tr>
<tr>
<td>XDrawString()</td>
<td>Draws 8-bit text in the specified drawable.</td>
</tr>
<tr>
<td>XDrawString16()</td>
<td>Draws 2-byte text in the specified drawable.</td>
</tr>
<tr>
<td>XDrawText()</td>
<td>Draws 8-bit complex text in the specified drawable.</td>
</tr>
<tr>
<td>XDrawText16()</td>
<td>Draws 2-byte complex text in the specified drawable.</td>
</tr>
</tbody>
</table>

**X Window System routines: Transferring images**

Table 142 provides the routines for transferring images.

Table 142. Transferring images

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XGetImage()</td>
<td>Gets the image from the rectangle in the drawable.</td>
</tr>
<tr>
<td>XGetSubImage()</td>
<td>Copies the rectangle on the display to the image.</td>
</tr>
<tr>
<td>XPutImage()</td>
<td>Puts the image from memory into the rectangle in the drawable.</td>
</tr>
</tbody>
</table>
### X Window System routines: Manipulating cursors

Table 143 provides the routines for manipulating cursors.

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCreateFontCursor()</td>
<td>Creates a cursor from a standard font.</td>
</tr>
<tr>
<td>XCreateGlyphCursor()</td>
<td>Creates a cursor from font glyphs.</td>
</tr>
<tr>
<td>XDefineCursor()</td>
<td>Defines a cursor for a window.</td>
</tr>
<tr>
<td>XFreeCursor()</td>
<td>Frees a cursor.</td>
</tr>
<tr>
<td>XQueryBestCursor()</td>
<td>Gets useful cursor sizes.</td>
</tr>
<tr>
<td>XRecolorCursor()</td>
<td>Changes the color of a cursor.</td>
</tr>
<tr>
<td>XUndefineCursor()</td>
<td>Undefines a cursor for a window.</td>
</tr>
</tbody>
</table>

### X Window System routines: Handling window manager functions

Table 144 provides the routines for handling the window manager functions.

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XAddToSaveSet()</td>
<td>Adds a window to the client saveset.</td>
</tr>
<tr>
<td>XAllowEvents()</td>
<td>Allows events to be processed after a device is frozen.</td>
</tr>
<tr>
<td>XChangeActivePointerGrab()</td>
<td>Changes the active pointer grab.</td>
</tr>
<tr>
<td>XChangePointerControl()</td>
<td>Changes the interactive feel of the pointer device.</td>
</tr>
<tr>
<td>XChangeSaveSet()</td>
<td>Adds or removes a window from the client’s saveset.</td>
</tr>
<tr>
<td>XGetInputFocus()</td>
<td>Gets the current input focus.</td>
</tr>
<tr>
<td>XGetPointerControl()</td>
<td>Gets the current pointer parameters.</td>
</tr>
<tr>
<td>XGrabButton()</td>
<td>Grabs a mouse button.</td>
</tr>
<tr>
<td>XGrabKey()</td>
<td>Grabs a single key of the keyboard.</td>
</tr>
<tr>
<td>XGrabKeyboard()</td>
<td>Grabs the keyboard.</td>
</tr>
<tr>
<td>XGrabPointer()</td>
<td>Grabs the pointer.</td>
</tr>
<tr>
<td>XGrabServer()</td>
<td>Grabs the server.</td>
</tr>
<tr>
<td>XInstallColormap()</td>
<td>Installs a colormap.</td>
</tr>
<tr>
<td>XKillClient()</td>
<td>Removes a client.</td>
</tr>
<tr>
<td>XListInstalledColormaps()</td>
<td>Gets a list of currently installed colormaps.</td>
</tr>
<tr>
<td>XRemoveFromSaveSet()</td>
<td>Removes a window from the client’s saveset.</td>
</tr>
<tr>
<td>XReparentWindow()</td>
<td>Changes the parent of a window.</td>
</tr>
<tr>
<td>XSetCloseDownMode()</td>
<td>Changes the close down mode.</td>
</tr>
<tr>
<td>XSetInputFocus()</td>
<td>Sets the input focus.</td>
</tr>
<tr>
<td>XUngrabButton()</td>
<td>Ungrabs a mouse button.</td>
</tr>
<tr>
<td>XUngrabKey()</td>
<td>Ungrabs a key.</td>
</tr>
<tr>
<td>XUngrabKeyboard()</td>
<td>Ungrabs the keyboard.</td>
</tr>
<tr>
<td>XUngrabPointer()</td>
<td>Ungrabs the pointer.</td>
</tr>
<tr>
<td>XUngrabServer()</td>
<td>Ungrabs the server.</td>
</tr>
</tbody>
</table>
Table 144. Handling window manager functions (continued)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XUninstallColormap()</td>
<td>Uninstalls a colormap.</td>
</tr>
<tr>
<td>XWarpPointer()</td>
<td>Moves the pointer to an arbitrary point on the screen.</td>
</tr>
</tbody>
</table>

X Window System routines: Manipulating keyboard settings

Table 145 provides the routines for manipulating keyboard settings.

Table 145. Manipulating keyboard settings

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XAutoRepeatOff()</td>
<td>Turns off the keyboard auto-repeat.</td>
</tr>
<tr>
<td>XAutoRepeatOn()</td>
<td>Turns on the keyboard auto-repeat.</td>
</tr>
<tr>
<td>XBell()</td>
<td>Sets the volume of the bell.</td>
</tr>
<tr>
<td>XChangeKeyboardControl()</td>
<td>Changes the keyboard settings.</td>
</tr>
<tr>
<td>XChangeKeyboardMapping()</td>
<td>Changes the mapping of symbols to keycodes.</td>
</tr>
<tr>
<td>XDeleteModifiermapEntry()</td>
<td>Deletes an entry from the XModifierKeymap structure.</td>
</tr>
<tr>
<td>XFreeModifiermap()</td>
<td>Frees XModifierKeymap structure.</td>
</tr>
<tr>
<td>getXKeyboardControl()</td>
<td>Gets the current keyboard settings.</td>
</tr>
<tr>
<td>getXKeyboardMapping()</td>
<td>Gets the mapping of symbols to keycodes.</td>
</tr>
<tr>
<td>getXModifierMapping()</td>
<td>Gets keycodes to be modifiers.</td>
</tr>
<tr>
<td>getXPointerMapping()</td>
<td>Gets the mapping of buttons on the pointer.</td>
</tr>
<tr>
<td>xInsertModifiermapEntry()</td>
<td>Adds an entry to the XModifierKeymap structure.</td>
</tr>
<tr>
<td>XNewModifiermap()</td>
<td>Creates the XModifierKeymap structure.</td>
</tr>
<tr>
<td>XQueryKeymap()</td>
<td>Gets the state of the keyboard keys.</td>
</tr>
<tr>
<td>XSetPointerMapping()</td>
<td>Sets the mapping of buttons on the pointer.</td>
</tr>
<tr>
<td>XSetModifierMapping()</td>
<td>Sets keycodes to be modifiers.</td>
</tr>
</tbody>
</table>

X Window System routines: Controlling the screen saver

Table 146 provides the routines for controlling the screen saver.

Table 146. Controlling the screen saver

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XActivateScreenSaver()</td>
<td>Activates the screen saver.</td>
</tr>
<tr>
<td>XForceScreenSaver()</td>
<td>Turns the screen saver on or off.</td>
</tr>
<tr>
<td>XGetScreenSaver()</td>
<td>Gets the current screen saver settings.</td>
</tr>
<tr>
<td>XResetScreenSaver()</td>
<td>Resets the screen saver.</td>
</tr>
<tr>
<td>XSetScreenSaver()</td>
<td>Sets the screen saver.</td>
</tr>
</tbody>
</table>

X Window System routines: Manipulating hosts and access control

Table 147 on page 710 provides the routines for manipulating hosts and toggling the access control.
Table 147. Manipulating hosts and access control

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XDisableAccessControl()</td>
<td>Disables access control.</td>
</tr>
<tr>
<td>XEnableAccessControl()</td>
<td>Enables access control.</td>
</tr>
<tr>
<td>XListHosts()</td>
<td>Gets the list of hosts.</td>
</tr>
<tr>
<td>XSetAccessControl()</td>
<td>Changes access control.</td>
</tr>
</tbody>
</table>

**X Window System routines: Handling events**

Table 148 provides the routines for handling events.

Table 148. Handling events

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCheckIfEvent()</td>
<td>Checks event queue for the specified event without blocking.</td>
</tr>
<tr>
<td>XCheckMaskEvent()</td>
<td>Removes the next event that matches a specified mask without blocking.</td>
</tr>
<tr>
<td>XCheckTypedEvent()</td>
<td>Gets the next event that matches event type.</td>
</tr>
<tr>
<td>XCheckTypedWindowEvent()</td>
<td>Gets the next event for the specified window.</td>
</tr>
<tr>
<td>XCheckWindowEvent()</td>
<td>Removes the next event that matches the specified window and mask without blocking.</td>
</tr>
<tr>
<td>XEventsQueued()</td>
<td>Checks the number of events in the event queue.</td>
</tr>
<tr>
<td>XFlush()</td>
<td>Flushes the output buffer.</td>
</tr>
<tr>
<td>XGetMotionEvents()</td>
<td>Gets the motion history for the specified window.</td>
</tr>
<tr>
<td>XIfEvent()</td>
<td>Checks the event queue for the specified event and removes it.</td>
</tr>
<tr>
<td>XMaskEvent()</td>
<td>Removes the next event that matches a specified mask.</td>
</tr>
<tr>
<td>XNextEvent()</td>
<td>Gets the next event and removes it from the queue.</td>
</tr>
<tr>
<td>XPeekEvent()</td>
<td>Peeks at the event queue.</td>
</tr>
<tr>
<td>XPeekIfEvent()</td>
<td>Checks the event queue for the specified event.</td>
</tr>
<tr>
<td>XPending()</td>
<td>Returns the number of events that are pending.</td>
</tr>
<tr>
<td>XPutBackEvent()</td>
<td>Pushes the event back to the top of the event queue.</td>
</tr>
<tr>
<td>XSelectInput()</td>
<td>Selects events to be reported to the client.</td>
</tr>
<tr>
<td>XSendEvent()</td>
<td>Sends an event to a specified window.</td>
</tr>
<tr>
<td>XSync()</td>
<td>Flushes the output buffer and waits until all requests are completed.</td>
</tr>
<tr>
<td>XWindowEvent()</td>
<td>Removes the next event that matches the specified window and mask.</td>
</tr>
</tbody>
</table>

**X Window System routines: Enabling and disabling synchronization**

Table 149 provides the routines for toggling synchronization.

Table 149. Enabling and disabling synchronization

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSetAfterFunction()</td>
<td>Sets the previous after function.</td>
</tr>
</tbody>
</table>
### Table 149. Enabling and disabling synchronization (continued)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSynchronize()</td>
<td>Enables or disables synchronization.</td>
</tr>
</tbody>
</table>

**X Window System routines: Using default error handling**

Table 150 provides the routines for using the default error handling.

### Table 150. Using default error handling

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XDisplayName()</td>
<td>Gets the name of the display currently being used.</td>
</tr>
<tr>
<td>XGetErrorText()</td>
<td>Gets the error text for the specified error code.</td>
</tr>
<tr>
<td>XGetErrorDatabaseText()</td>
<td>Gets the error text from the error database.</td>
</tr>
<tr>
<td>XSetErrorHandler()</td>
<td>Sets the error handler.</td>
</tr>
<tr>
<td>XSetIOErrorHandler()</td>
<td>Sets the error handler for unrecoverable I/O errors.</td>
</tr>
</tbody>
</table>

**X Window System routines: Communicating with window managers**

Table 151 provides the routines for communicating with window managers.

### Table 151. Communicating with window managers

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XAllocClassHints()</td>
<td>Allocates storage for an XClassHint structure.</td>
</tr>
<tr>
<td>XAllocIconSize()</td>
<td>Allocates storage for an XIconSize structure.</td>
</tr>
<tr>
<td>XAllocSizeHints()</td>
<td>Allocates storage for an XSizeHints structure.</td>
</tr>
<tr>
<td>XAllocWMHints()</td>
<td>Allocates storage for an XWMHints structure.</td>
</tr>
<tr>
<td>XGetClassHint()</td>
<td>Gets the class of a window.</td>
</tr>
<tr>
<td>XFetchName()</td>
<td>Gets the name of a window.</td>
</tr>
<tr>
<td>XGetCommand()</td>
<td>Gets a window WM_COMMAND property.</td>
</tr>
<tr>
<td>XGetIconName()</td>
<td>Gets the name of an icon window.</td>
</tr>
<tr>
<td>XGetIconSizes()</td>
<td>Gets the values of icon size atom.</td>
</tr>
<tr>
<td>XGetNormalHints()</td>
<td>Gets size hints for window in normal state.</td>
</tr>
<tr>
<td>XGetRGBColormaps()</td>
<td>Gets colormap associated with specified atom.</td>
</tr>
<tr>
<td>XGetSizeHints()</td>
<td>Gets the values of type WM_SIZE_HINTS properties.</td>
</tr>
<tr>
<td>XGetStandardColormap()</td>
<td>Gets colormap associated with specified atom.</td>
</tr>
<tr>
<td>XGetTextProperty()</td>
<td>Gets window property of type TEXT.</td>
</tr>
<tr>
<td>XGetTransientForHint()</td>
<td>Gets WM_TRANSIENT_FOR property for window.</td>
</tr>
<tr>
<td>XGetWM_CLIENT_MACHINE</td>
<td>Gets the value of a window WM_CLIENT_MACHINE property.</td>
</tr>
<tr>
<td>XGetWMColormapWindows()</td>
<td>Gets the value of a window WM_COLOMAP_WINDOWS property.</td>
</tr>
<tr>
<td>XGetWMHints()</td>
<td>Gets the value of the window manager hints atom.</td>
</tr>
<tr>
<td>XGetWMName()</td>
<td>Gets the value of the WM_NAME property.</td>
</tr>
<tr>
<td>XGetWMIcoName()</td>
<td>Gets the value of the WM_ICON_NAME property.</td>
</tr>
</tbody>
</table>
Table 151. Communicating with window managers  (continued)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XGetWMNormalHints()</td>
<td>Gets the value of the window manager hints atom.</td>
</tr>
<tr>
<td>XGetWMProtocols()</td>
<td>Gets the value of a window WM_PROTOCOLS property.</td>
</tr>
<tr>
<td>XGetWMSizeHints()</td>
<td>Gets the values of type WM_SIZE_HINTS properties.</td>
</tr>
<tr>
<td>XGetZoomHints()</td>
<td>Gets values of the zoom hints atom.</td>
</tr>
<tr>
<td>XSetCommand()</td>
<td>Sets the value of the command atom.</td>
</tr>
<tr>
<td>XSetClassHint()</td>
<td>Sets the class of a window.</td>
</tr>
<tr>
<td>XSetIconName()</td>
<td>Assigns a name to an icon window.</td>
</tr>
<tr>
<td>XSetIconSizes()</td>
<td>Sets the values of icon size atom.</td>
</tr>
<tr>
<td>XSetNormalHints()</td>
<td>Sets size hints for a window in normal state.</td>
</tr>
<tr>
<td>XSetRGBColormaps()</td>
<td>Sets the colormap associated with the specified atom.</td>
</tr>
<tr>
<td>XSetSizeHints()</td>
<td>Sets the values of the type WM_SIZE_HINTS properties.</td>
</tr>
<tr>
<td>XSetStandardColormap()</td>
<td>Sets the colormap associated with the specified atom.</td>
</tr>
<tr>
<td>XSetStandardProperties()</td>
<td>Specifies a minimum set of properties.</td>
</tr>
<tr>
<td>XSetTextProperty()</td>
<td>Sets window properties of type TEXT.</td>
</tr>
<tr>
<td>XSetTransientForHint()</td>
<td>Sets WM_TRANSIENT_FOR property for window.</td>
</tr>
<tr>
<td>XSetWMClientMachine()</td>
<td>Sets window WM_CLIENT_MACHINE property.</td>
</tr>
<tr>
<td>XSetWMColormapWindows()</td>
<td>Sets a window WM_COLORMAP_WINDOWS property.</td>
</tr>
<tr>
<td>XSetWMHints()</td>
<td>Sets the value of the window manager hints atom.</td>
</tr>
<tr>
<td>XSetWMIconName()</td>
<td>Sets the value of the WM_ICON_NAME property.</td>
</tr>
<tr>
<td>XSetWMName()</td>
<td>Sets the value of the WM_NAME property.</td>
</tr>
<tr>
<td>XSetWMNormalHints()</td>
<td>Sets the value of the window manager hints atom.</td>
</tr>
<tr>
<td>XSetWMProperties()</td>
<td>Sets the values of properties for a window manager.</td>
</tr>
<tr>
<td>XSetWMProtocols()</td>
<td>Sets the value of the WM_PROTOCOLS property.</td>
</tr>
<tr>
<td>XSetWMSizeHints()</td>
<td>Sets the values of type WM_SIZE_HINTS properties.</td>
</tr>
<tr>
<td>XSetZoomHints()</td>
<td>Sets the values of the zoom hints atom.</td>
</tr>
<tr>
<td>XStoreName()</td>
<td>Assigns a name to a window.</td>
</tr>
</tbody>
</table>

**X Window System routines: Manipulating keyboard event functions**

Table 152 provides the routines for manipulating keyboard event functions.

Table 152. Manipulating keyboard event functions

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XKeycodeToKeysym()</td>
<td>Converts keycode to a keysym value.</td>
</tr>
<tr>
<td>XKeysymToKeyCode()</td>
<td>Converts keysym value to keycode.</td>
</tr>
<tr>
<td>XKeysymToString()</td>
<td>Converts keysym value to keysym name.</td>
</tr>
<tr>
<td>XLookupKeysym()</td>
<td>Translates a keyboard event into a keysym value.</td>
</tr>
<tr>
<td>XLookupMapping()</td>
<td>Gets the mapping of a keyboard event from a keymap file.</td>
</tr>
<tr>
<td>XLookupString()</td>
<td>Translates the keyboard event into a character string.</td>
</tr>
</tbody>
</table>
Table 152. Manipulating keyboard event functions (continued)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XRebindCode()</td>
<td>Changes the keyboard mapping in the keymap file.</td>
</tr>
<tr>
<td>XRebindKeysym()</td>
<td>Maps the character string to a specified keysym and modifiers.</td>
</tr>
<tr>
<td>XRefreshKeyboardMapping()</td>
<td>Refreshes the stored modifier and keymap information.</td>
</tr>
<tr>
<td>XStringToKeysym()</td>
<td>Converts the keysym name to the keysym value.</td>
</tr>
<tr>
<td>XUseKeymap()</td>
<td>Changes the keymap files.</td>
</tr>
<tr>
<td>XGeometry()</td>
<td>Parses window geometry given padding and font values.</td>
</tr>
<tr>
<td>XGetDefault()</td>
<td>Gets the default window options.</td>
</tr>
<tr>
<td>XParseColor()</td>
<td>Obtains RGB values from color name.</td>
</tr>
<tr>
<td>XParseGeometry()</td>
<td>Parses standard window geometry options.</td>
</tr>
<tr>
<td>XWMGeometry()</td>
<td>Obtains a window’s geometry information.</td>
</tr>
</tbody>
</table>

**X Window System routines: Manipulating regions**

Table 153 provides the routines for manipulating regions.

Table 153. Manipulating regions

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XClipBox()</td>
<td>Generates the smallest enclosing rectangle in the region.</td>
</tr>
<tr>
<td>XCreateRegion()</td>
<td>Creates a new empty region.</td>
</tr>
<tr>
<td>XEmptyRegion()</td>
<td>Determines whether a specified region is empty.</td>
</tr>
<tr>
<td>XEqualRegion()</td>
<td>Determines whether two regions are the same.</td>
</tr>
<tr>
<td>XIntersectRegion()</td>
<td>Computes the intersection of two regions.</td>
</tr>
<tr>
<td>XDestroyRegion()</td>
<td>Frees storage associated with the specified region.</td>
</tr>
<tr>
<td>XOffsetRegion()</td>
<td>Moves the specified region by the specified amount.</td>
</tr>
<tr>
<td>XPointInRegion()</td>
<td>Determines if a point lies in the specified region.</td>
</tr>
<tr>
<td>XPolygonRegion()</td>
<td>Generates a region from points.</td>
</tr>
<tr>
<td>XRectInRegion()</td>
<td>Determines if a rectangle lies in the specified region.</td>
</tr>
<tr>
<td>XSetRegion()</td>
<td>Sets the GC to the specified region.</td>
</tr>
<tr>
<td>XShrinkRegion()</td>
<td>Reduces the specified region by a specified amount.</td>
</tr>
<tr>
<td>XSubtractRegion()</td>
<td>Subtracts two regions.</td>
</tr>
<tr>
<td>XUnionRegion()</td>
<td>Computes the union of two regions.</td>
</tr>
<tr>
<td>XUnionRectWithRegion()</td>
<td>Creates a union of source region and rectangle.</td>
</tr>
<tr>
<td>XXorRegion()</td>
<td>Gets the difference between the union and intersection of regions.</td>
</tr>
</tbody>
</table>

**X Window System routines: Using cut and paste buffers**

Table 154 provides the routines for using cut and paste buffers.

Table 154. Using cut and paste buffers

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XFetchBuffer()</td>
<td>Gets data from a specified cut buffer.</td>
</tr>
</tbody>
</table>
Table 154. Using cut and paste buffers (continued)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XFetchBytes()</td>
<td>Gets data from the first cut buffer.</td>
</tr>
<tr>
<td>XRotateBuffers()</td>
<td>Rotates the cut buffers.</td>
</tr>
<tr>
<td>XStoreBuffer()</td>
<td>Stores data in a specified cut buffer.</td>
</tr>
<tr>
<td>XStoreBytes()</td>
<td>Stores data in first cut buffer.</td>
</tr>
</tbody>
</table>

**X Window System routines: Querying visual types**

Table 155 provides the routines for querying visual types.

Table 155. Querying visual types

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XGetVisualInfo()</td>
<td>Gets a list of visual information structures.</td>
</tr>
<tr>
<td>XListDepths()</td>
<td>Determines the number of depths that are available on a given screen.</td>
</tr>
<tr>
<td>XListPixmapFormats()</td>
<td>Gets the pixmap format information for a given display.</td>
</tr>
<tr>
<td>XMatchVisualInfo()</td>
<td>Gets visual information matching screen depth and class.</td>
</tr>
<tr>
<td>XPixmapFormatValues()</td>
<td>Gets the pixmap format information for a given display.</td>
</tr>
</tbody>
</table>

**X Window System routines: Manipulating images**

Table 156 provides the routines for manipulating images.

Table 156. Manipulating images

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XAddPixel()</td>
<td>Increases each pixel in pixmap by a constant value.</td>
</tr>
<tr>
<td>XCreateImage()</td>
<td>Allocates memory for the XImage structure.</td>
</tr>
<tr>
<td>XDestroyImage()</td>
<td>Frees memory for the XImage structure.</td>
</tr>
<tr>
<td>XGetPixel()</td>
<td>Gets a pixel value in an image.</td>
</tr>
<tr>
<td>XPutPixel()</td>
<td>Sets a pixel value in an image.</td>
</tr>
<tr>
<td>XSubImage()</td>
<td>Creates an image that is a subsection of a specified image.</td>
</tr>
</tbody>
</table>

**X Window System routines: Manipulating bit maps**

Table 157 provides the routines for manipulating bit maps.

Table 157. Manipulating bit maps

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCreateBitmapFromData()</td>
<td>Includes a bit map in the C program.</td>
</tr>
<tr>
<td>XCreatePixmapFromBitmapData()</td>
<td>Creates a pixmap using bit map data.</td>
</tr>
<tr>
<td>XDeleteContext()</td>
<td>Deletes data associated with the window and context type.</td>
</tr>
<tr>
<td>XFindContext()</td>
<td>Gets data associated with the window and context type.</td>
</tr>
<tr>
<td>XReadBitmapFile()</td>
<td>Reads in a bit map from a file.</td>
</tr>
<tr>
<td>XSaveContext()</td>
<td>Stores data associated with the window and context type.</td>
</tr>
<tr>
<td>XUniqueContext()</td>
<td>Allocates a new context.</td>
</tr>
</tbody>
</table>
### X Window System routines: Using the resource manager

**Table 158. Using the resource manager**

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xpermalloc()</td>
<td>Allocates memory that is never freed.</td>
</tr>
<tr>
<td>XrmDestroyDatabase()</td>
<td>Destroys a resource database and frees its allocated memory.</td>
</tr>
<tr>
<td>XrmGetFileDatabase()</td>
<td>Creates a database from a specified file.</td>
</tr>
<tr>
<td>XrmGetResource()</td>
<td>Retrieves a resource from a database.</td>
</tr>
<tr>
<td>XrmGetStringDatabase()</td>
<td>Creates a database from a specified string.</td>
</tr>
<tr>
<td>XrmInitialize()</td>
<td>Initializes the resource manager.</td>
</tr>
<tr>
<td>XrmMergeDatabases()</td>
<td>Merges two databases.</td>
</tr>
<tr>
<td>XrmParseCommand()</td>
<td>Stores command options in a database.</td>
</tr>
<tr>
<td>XrmPutFileDatabase()</td>
<td>Copies the database into a specified file.</td>
</tr>
<tr>
<td>XrmPutLineResource()</td>
<td>Stores a single resource entry in a database.</td>
</tr>
<tr>
<td>XrmPutResource()</td>
<td>Stores a resource in a database.</td>
</tr>
<tr>
<td>XrmPutStringResource()</td>
<td>Stores string resource in a database.</td>
</tr>
<tr>
<td>XrmQGetResource()</td>
<td>Retrieves a quark from a database.</td>
</tr>
<tr>
<td>XrmQGetSearchList()</td>
<td>Gets a resource search list of database levels.</td>
</tr>
<tr>
<td>XrmQGetSearchResource()</td>
<td>Gets a quark search list of database levels.</td>
</tr>
<tr>
<td>XrmQPutResource()</td>
<td>Stores binding and quarks in a database.</td>
</tr>
<tr>
<td>XrmQPutStringResource()</td>
<td>Stores string binding and quarks in a database.</td>
</tr>
<tr>
<td>XrmQuarkToString()</td>
<td>Converts a quark to a character string.</td>
</tr>
<tr>
<td>XrmStringToQuark()</td>
<td>Converts a character string to a quark.</td>
</tr>
<tr>
<td>XrmStringToQuarkList()</td>
<td>Converts character strings to a quark list.</td>
</tr>
<tr>
<td>XrmStringToBindingQuarkList()</td>
<td>Converts strings to bindings and quarks.</td>
</tr>
<tr>
<td>XrmUniqueQuark()</td>
<td>Allocates a new quark.</td>
</tr>
</tbody>
</table>

### X Window System routines: Manipulating display functions

**Table 159. Manipulating display functions**

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllPlanes()</td>
<td>XAllPlanes() Returns all bits suitable for use in plane argument.</td>
</tr>
<tr>
<td>BitMapBitOrder()</td>
<td>XBitMapOrder() Returns either the most or least significant bit in each bit map unit.</td>
</tr>
<tr>
<td>BitMapPad()</td>
<td>XBitMapPad() Returns the multiple of bits padding each scanline.</td>
</tr>
<tr>
<td>BitMapUnit()</td>
<td>XBitMapUnit() Returns the size of a bit map unit in bits.</td>
</tr>
</tbody>
</table>
### Table 159. Manipulating display functions (continued)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlackPixel() XBlackPixel()</td>
<td>Returns the black pixel value of the screen specified.</td>
</tr>
<tr>
<td>BlackPixelOfScreen() XBlackPixelOfScreen()</td>
<td>Returns the black pixel value of the screen specified.</td>
</tr>
<tr>
<td>CellsOfScreen() XCellsOfScreen()</td>
<td>Returns the number of colormap cells.</td>
</tr>
<tr>
<td>ConnectionNumber() XConnectionNumber()</td>
<td>Returns the file descriptor of the connection.</td>
</tr>
<tr>
<td>CreatePixmapCursor() XCreatePixmapCursor()</td>
<td>Creates a pixmap of a specified size.</td>
</tr>
<tr>
<td>CreateWindow() XCreateWindow()</td>
<td>Creates an unmapped subwindow for a specified parent window.</td>
</tr>
<tr>
<td>DefaultColormap() XDefaultColormap()</td>
<td>Returns a default colormap ID for allocation on the screen specified.</td>
</tr>
<tr>
<td>DefaultColormapOfScreen()</td>
<td>Returns the default colormap ID of the screen specified.</td>
</tr>
<tr>
<td>DefaultDepth() XDefaultDepth()</td>
<td>Returns the depth of the default root window.</td>
</tr>
<tr>
<td>DefaultDepthOfScreen() XDefaultDepthOfScreen()</td>
<td>Returns the default depth of the screen specified.</td>
</tr>
<tr>
<td>DefaultGC() XDefaultGC()</td>
<td>Returns the default GC of the default root window.</td>
</tr>
<tr>
<td>DefaultGCOfScreen() XDefaultGCOfScreen()</td>
<td>Returns the default GC of the screen specified.</td>
</tr>
<tr>
<td>DefaultScreen() XDefaultScreen()</td>
<td>Obtains the default screen referred to in the XOpenDisplay routine.</td>
</tr>
<tr>
<td>DefaultScreenOfDisplay() XDefaultScreenOfDisplay()</td>
<td>Returns the default screen of the display specified.</td>
</tr>
<tr>
<td>DefaultRootWindow() XDefaultRootWindow()</td>
<td>Obtains the root window for the default screen specified.</td>
</tr>
<tr>
<td>DefaultVisual() XDefaultVisual()</td>
<td>Returns the default visual type of the screen specified.</td>
</tr>
<tr>
<td>DefaultVisualOfScreen() XDefaultVisualOfScreen()</td>
<td>Returns the default visual type of the screen specified.</td>
</tr>
<tr>
<td>DisplayCells() XDisplayCells()</td>
<td>Displays the number of entries in the default colormap.</td>
</tr>
<tr>
<td>DisplayHeight() XDisplayHeight()</td>
<td>Displays the height of the screen in pixels.</td>
</tr>
<tr>
<td>DisplayHeightMM() XDisplayHeightMM()</td>
<td>Displays the height of the screen in millimeters.</td>
</tr>
<tr>
<td>DisplayOfScreen() XDisplayOfScreen()</td>
<td>Displays the type of screen specified.</td>
</tr>
<tr>
<td>DisplayPlanes() XDisplayPlanes()</td>
<td>Displays the depth (number of planes) of the root window of the screen specified.</td>
</tr>
<tr>
<td>DisplayString() XDisplayString()</td>
<td>Displays the string passed to XOpenDisplay when the current display was opened.</td>
</tr>
<tr>
<td>DisplayWidth() XDisplayWidth()</td>
<td>Displays the width of the specified screen in pixels.</td>
</tr>
<tr>
<td>DisplayWidthMM() XDisplayWidthMM()</td>
<td>Displays the width of the specified screen in millimeters.</td>
</tr>
<tr>
<td>DoesBackingStore() XDoesBackingStore()</td>
<td>Indicates whether the specified screen supports backing stores.</td>
</tr>
<tr>
<td>DoesSaveUnders() XDoesSaveUnders()</td>
<td>Indicates whether the specified screen supports save unders.</td>
</tr>
<tr>
<td>EventMaskOfScreen() XEventMaskOfScreen()</td>
<td>Returns the initial root event mask for a specified screen.</td>
</tr>
<tr>
<td>HeightMMOfScreen() XHeightMMOfScreen()</td>
<td>Returns the height of a specified screen in millimeters.</td>
</tr>
<tr>
<td>HeightOfScreen() XHeightOfScreen()</td>
<td>Returns the height of a specified screen in pixels.</td>
</tr>
<tr>
<td>ImageByteOrder() XImageByteOrder()</td>
<td>Specifies the required byte order for each scanline unit of an image.</td>
</tr>
<tr>
<td>IsCursorKey()</td>
<td>Returns TRUE if keysym is on cursor key.</td>
</tr>
<tr>
<td>IsFunctionKey()</td>
<td>Returns TRUE if keysym is on function keys.</td>
</tr>
<tr>
<td>IsKeypadKey()</td>
<td>Returns TRUE if keysym is on keypad.</td>
</tr>
</tbody>
</table>
Table 159. Manipulating display functions (continued)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsMiscFunctionKey()</td>
<td>Returns TRUE if keysym is on miscellaneous function keys.</td>
</tr>
<tr>
<td>IsModifierKey()</td>
<td>Returns TRUE if keysym is on modifier keys.</td>
</tr>
<tr>
<td>IsPFKey()</td>
<td>Returns TRUE if keysym is on PF keys.</td>
</tr>
<tr>
<td>LastKnownRequestProcessed()</td>
<td>Extracts the full serial number of the last known request processed by the X server.</td>
</tr>
<tr>
<td>XLastKnownRequestProcessed()</td>
<td></td>
</tr>
<tr>
<td>MaxCmapsOfScreen() XMaxCmapsOfScreen()</td>
<td>Returns the maximum number of colormaps supported by the specified screen.</td>
</tr>
<tr>
<td>MinCmapsOfScreen() XMinCmapsOfScreen()</td>
<td>Returns the minimum number of colormaps supported by the specified screen.</td>
</tr>
<tr>
<td>NextRequest() XNextRequest()</td>
<td>Extracts the full serial number to be used for the next request to be processed by the X Server.</td>
</tr>
<tr>
<td>PlanesOfScreen() XPlanesOfScreen()</td>
<td>Returns the depth (number of planes) in a specified screen.</td>
</tr>
<tr>
<td>ProtocolRevision() XProtocolRevision()</td>
<td>Returns the minor protocol revision number (0) of the X server associated with the display.</td>
</tr>
<tr>
<td>ProtocolVersion() XProtocolVersion()</td>
<td>Returns the major version number (11) of the protocol associated with the display.</td>
</tr>
<tr>
<td>QLength() XQLength()</td>
<td>Returns the length of the event queue for the display.</td>
</tr>
<tr>
<td>RootWindow() XRootWindow()</td>
<td>Returns the root window of the current screen.</td>
</tr>
<tr>
<td>RootWindowOfScreen() XRootWindowOfScreen()</td>
<td>Returns the root window of the specified screen.</td>
</tr>
<tr>
<td>ScreenCount() XScreenCount()</td>
<td>Returns the number of screens available.</td>
</tr>
<tr>
<td>XScreenNumberOfScreen()</td>
<td>Returns the screen index number of the specified screen.</td>
</tr>
<tr>
<td>ScreenOfDisplay() XScreenOfDisplay()</td>
<td>Returns the pointer to the screen of the display specified.</td>
</tr>
<tr>
<td>ServerVendor() XServerVendor()</td>
<td>Returns the pointer to a null-determined string that identifies the owner of the X server implementation.</td>
</tr>
<tr>
<td>VendorRelease() XVendorRelease()</td>
<td>Returns the number related to the vendor’s release of the X server.</td>
</tr>
<tr>
<td>WhitePixel() XWhitePixel()</td>
<td>Returns the white pixel value for the current screen.</td>
</tr>
<tr>
<td>WhitePixelOfScreen() XWhitePixelOfScreen()</td>
<td>Returns the white pixel value of the specified screen.</td>
</tr>
<tr>
<td>WidthMMOfScreen() XWidthMMOfScreen()</td>
<td>Returns the width of the specified screen in millimeters.</td>
</tr>
<tr>
<td>WidthOfScreen() XWidthOfScreen()</td>
<td>Returns the width of the specified screen in pixels.</td>
</tr>
</tbody>
</table>

**X Window System routines: Extension routines**

X Window System Extension Routines allow you to create extensions to the core Xlib functions with the same performance characteristics. The following are the protocol requests for X Window System extensions:

- XQueryExtension
- XListExtensions
- XFreeExtensionList

Table 160 on page 718 lists the X Window System Extension Routines and provides a short description of each routine.
Table 160. Extension routines

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XAllocID()</td>
<td>Returns a resource ID that can be used when creating new resources.</td>
</tr>
<tr>
<td>XSetCloseDisplay()</td>
<td>Defines a procedure to call when XCloseDisplay is called.</td>
</tr>
<tr>
<td>XSetCopyGC()</td>
<td>Defines a procedure to call when a GC is copied.</td>
</tr>
<tr>
<td>XSetCreateFont()</td>
<td>Defines a procedure to call when XLoadQueryFont is called.</td>
</tr>
<tr>
<td>XSetCreateGC()</td>
<td>Defines a procedure to call when a new GC is created.</td>
</tr>
<tr>
<td>XSetError()</td>
<td>Suppresses the call to an external error handling routine and defines an alternative routine for error handling.</td>
</tr>
<tr>
<td>XSetErrorString()</td>
<td>Defines a procedure to call when an I/O error is detected.</td>
</tr>
<tr>
<td>XSetEventToWire()</td>
<td>Defines a procedure to call when an event must be converted from the host to wire format.</td>
</tr>
<tr>
<td>XSetFreeFont()</td>
<td>Defines a procedure to call when XFreeFont is called.</td>
</tr>
<tr>
<td>XSetFreeGC()</td>
<td>Defines a procedure to call when a GC is freed.</td>
</tr>
<tr>
<td>XSetWireToEvent()</td>
<td>Defines a procedure to call when an event is converted from the wire to the host format.</td>
</tr>
<tr>
<td>XFreeExtensionList()</td>
<td>Frees memory allocated by XListExtensions.</td>
</tr>
<tr>
<td>XListExtensions()</td>
<td>Returns a list of all extensions supported by the server.</td>
</tr>
<tr>
<td>XQueryExtension()</td>
<td>Indicates whether a named extension is present.</td>
</tr>
</tbody>
</table>

Table 161. MIT extensions to X

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XShapeQueryExtension</td>
<td>Queries to see if server supports the SHAPE extension.</td>
</tr>
<tr>
<td>XShapeQueryVersion</td>
<td>Checks the version number of the server SHAPE extension.</td>
</tr>
<tr>
<td>XShapeCombineRegion</td>
<td>Converts the specified region into a list of rectangles and calls XShapeRectangles.</td>
</tr>
<tr>
<td>XShapeCombineRectangles</td>
<td>Performs a CombineRectangles operation.</td>
</tr>
<tr>
<td>XShapeCombineMask</td>
<td>Performs a CombineMask operation.</td>
</tr>
<tr>
<td>XShapeCombineShape</td>
<td>Performs a CombineShape operation.</td>
</tr>
<tr>
<td>XShapeOffsetShape</td>
<td>Performs an OffsetShape operation.</td>
</tr>
<tr>
<td>XShapeQueryExtents</td>
<td>Sets the extents of the bounding and clip shapes.</td>
</tr>
</tbody>
</table>

**X Window System routines: MIT extensions to X**

The AIX extensions described in the *IBM AIX X-Window Programmer’s Reference* are not supported by the X Window System API provided by the TCP/IP library routines.

The following MIT extensions are supported by the TCP/IP X client code:
- SHAPE
- MITMISC
- MULTIBUF

Table 161 lists the routines that allow an application to use these extensions.
### Table 161. MIT extensions to X (continued)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XShapeSelectInput</td>
<td>Selects Input Events.</td>
</tr>
<tr>
<td>XShapeInputSelected</td>
<td>Returns the current input mask for extension events on the specified window.</td>
</tr>
<tr>
<td>XShapeGetRectangles</td>
<td>Gets a list of rectangles describing the region specified.</td>
</tr>
<tr>
<td>XMITMiscQueryExtension</td>
<td>Queries to see if server supports the MITMISC extension.</td>
</tr>
<tr>
<td>XMITMiscSetBugMode</td>
<td>Sets the compatibility mode switch.</td>
</tr>
<tr>
<td>XMITMiscGetBugMode</td>
<td>Queries the compatibility mode switch.</td>
</tr>
<tr>
<td>XmbufQueryExtension</td>
<td>Queries to see if server supports the MULTIBUF extension.</td>
</tr>
<tr>
<td>XmbufGetVersion</td>
<td>Gets the version number of the extension.</td>
</tr>
<tr>
<td>XmbufCreateBuffers</td>
<td>Requests that multiple buffers be created.</td>
</tr>
<tr>
<td>XmbufDestroyBuffers</td>
<td>Requests that the buffers be destroyed.</td>
</tr>
<tr>
<td>XmbufDisplayBuffers</td>
<td>Displays the indicated buffers.</td>
</tr>
<tr>
<td>XmbufGetWindowAttributes</td>
<td>Gets the multibuffering attributes.</td>
</tr>
<tr>
<td>XmbufChangeWindowAttributes</td>
<td>Sets the multibuffering attributes.</td>
</tr>
<tr>
<td>XmbufGetBufferAttributes</td>
<td>Gets the attributes for the indicated buffer.</td>
</tr>
<tr>
<td>XmbufChangeBufferAttributes</td>
<td>Sets the attributes for the indicated buffer.</td>
</tr>
<tr>
<td>XmbufGetScreenInfo</td>
<td>Gets the parameters controlling how mono and stereo windows can be created on the indicated screen.</td>
</tr>
<tr>
<td>XmbufCreateStereoWindow</td>
<td>Creates a stereo window.</td>
</tr>
</tbody>
</table>

### X Window System routines: Associate table functions

When you need to associate arbitrary information with resource IDs, the XAssocTable allows you to associate your own data structures with X resources, such as bit maps, pixmaps, fonts, and windows.

An XAssocTable can be used to type X resources. For example, to create three or four types of windows with different properties, each window ID is associated with a pointer to a user-defined window property data structure. (A generic type, called XID, is defined in XLIB.H.)

Follow these guidelines when using an XAssocTable.

- Ensure the correct display is active before initiating an XAssocTable function, because all XIDs are relative to a specified display.
- Restrict the size of the table (number of buckets in the hashing system) to a power of two, and assign no more than eight XIDs for each bucket to maximize the efficiency of the table.

There is no restriction on the number of XIDs for each table or display, or the number of displays for each table.

Table 162 lists the Associate table functions and provides a short description of each function.
Table 162. Associate table functions

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCreateAssocTable()</td>
<td>Returns a pointer to the newly created associate table.</td>
</tr>
<tr>
<td>XDeleteAssoc()</td>
<td>Deletes an entry from the specified associate table.</td>
</tr>
<tr>
<td>XDestroyAssocTable()</td>
<td>Frees memory allocated to the specified associate table.</td>
</tr>
<tr>
<td>XLookUpAssoc()</td>
<td>Obtains data from the specified associate table.</td>
</tr>
<tr>
<td>XMakeAssoc()</td>
<td>Creates an entry in the specified associate table.</td>
</tr>
</tbody>
</table>

X Window System routines: Miscellaneous utility routines

The MIT X Miscellaneous Utility routines are included in SEZAX11L. These are a set of common utility functions that have been useful to application writers.

Table 163 lists the Miscellaneous utility routines and provides a short description of each routine.

Table 163. Miscellaneous utility routines

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XctCreate()</td>
<td>Creates an XctData structure for parsing a Compound Text string.</td>
</tr>
<tr>
<td>XctFree()</td>
<td>Frees all data associated with the XctData structure.</td>
</tr>
<tr>
<td>XcNextItem()</td>
<td>Parses the next item from the Compound Text string.</td>
</tr>
<tr>
<td>XcReset()</td>
<td>Resets the XctData structure to reparse the Compound Text string.</td>
</tr>
<tr>
<td>XmuAddCloseDisplayHook()</td>
<td>Adds a callback for the given display.</td>
</tr>
<tr>
<td>XmuAddInitializer()</td>
<td>Registers a procedure to be invoked the first time XmuCallInitializers is called on a given application context.</td>
</tr>
<tr>
<td>XmuAllStandardColormaps()</td>
<td>Creates all of the appropriate standard colormaps.</td>
</tr>
<tr>
<td>XmuCallInitializers()</td>
<td>Calls each of the procedures that have been registered with XmuAddInitializer.</td>
</tr>
<tr>
<td>XmuClientWindow()</td>
<td>Finds a window at or below the specified window.</td>
</tr>
<tr>
<td>XmuCompareISOLatin1()</td>
<td>Compares two strings, ignoring case differences.</td>
</tr>
<tr>
<td>XmuConvertStandardSelection()</td>
<td>Converts many standard selections.</td>
</tr>
<tr>
<td>XmuCopyISOLatin1Lowered()</td>
<td>Copies a string, changing all Latin-1 uppercase letters to lowercase.</td>
</tr>
<tr>
<td>XmuCopyISOLatin1Uppered()</td>
<td>Copies a string, changing all Latin-1 lower case letters to uppercase.</td>
</tr>
<tr>
<td>XmuCreateColormap()</td>
<td>Creates a colormap.</td>
</tr>
<tr>
<td>XmuCreatePixmapFromBitmap()</td>
<td>Creates a pixmap of the specified width, height, and depth.</td>
</tr>
<tr>
<td>XmuCreateStippledPixmap()</td>
<td>Creates a two-pixel by one-pixel stippled pixmap of specified depth on the specified screen.</td>
</tr>
<tr>
<td>XmuCursorNameToIndex()</td>
<td>Returns the index in the standard cursor font for the name of a standard cursor.</td>
</tr>
<tr>
<td>XmuCvtFunctionToCallback()</td>
<td>Converts a callback procedure to a callback list containing that procedure.</td>
</tr>
<tr>
<td>Routine</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>XmuCvtStringToBackingStore()</td>
<td>Converts a string to a backing-store integer.</td>
</tr>
<tr>
<td>XmuCvtStringToBitmap()</td>
<td>Creates a bit map suitable for window manager icons.</td>
</tr>
<tr>
<td>XmuCvtStringToCursor()</td>
<td>Converts a string to a Cursor.</td>
</tr>
<tr>
<td>XmuCvtStringToJustify()</td>
<td>Converts a string to an XtJustify enumeration value.</td>
</tr>
<tr>
<td>XmuCvtStringToLong()</td>
<td>Converts a string to an integer of type long.</td>
</tr>
<tr>
<td>XmuCvtStringToOrientation()</td>
<td>Converts a string to an XtOrientation enumeration value.</td>
</tr>
<tr>
<td>XmuCvtStringToShapeStyle()</td>
<td>Converts a string to an integer shape style.</td>
</tr>
<tr>
<td>XmuCvtStringToWidget()</td>
<td>Converts a string to an immediate child widget of the parent widget passed as an argument.</td>
</tr>
<tr>
<td>XmuDeleteStandardColormap()</td>
<td>Removes the specified property from the specified screen.</td>
</tr>
<tr>
<td>XmuDQAddDisplay()</td>
<td>Adds the specified display to the queue.</td>
</tr>
<tr>
<td>XmuDQCreate()</td>
<td>Creates and returns an empty XmuDisplayQueue.</td>
</tr>
<tr>
<td>XmuDQDestroy()</td>
<td>Releases all memory associated with the specified queue.</td>
</tr>
<tr>
<td>XmuDQLookupDisplay()</td>
<td>Returns the queue entry for the specified display.</td>
</tr>
<tr>
<td>XmuDQNDDisplays()</td>
<td>Returns the number of displays in the specified queue.</td>
</tr>
<tr>
<td>XmuDQRemoveDisplay()</td>
<td>Removes the specified display from the specified queue.</td>
</tr>
<tr>
<td>XmuDrawLogo()</td>
<td>Draws the official X Window System logo.</td>
</tr>
<tr>
<td>XmuDrawRoundedRectangle()</td>
<td>Draws a rounded rectangle.</td>
</tr>
<tr>
<td>XmuFillRoundedRectangle()</td>
<td>Draws a filled rounded rectangle.</td>
</tr>
<tr>
<td>XmuGetAtomName()</td>
<td>Returns the name of an Atom.</td>
</tr>
<tr>
<td>XmuGetColormapAllocation()</td>
<td>Determines the best allocation of reds, greens, and blues in a standard colormap.</td>
</tr>
<tr>
<td>XmuGetHostname()</td>
<td>Returns the host name.</td>
</tr>
<tr>
<td>XmuInternAtom()</td>
<td>Caches the Atom value for one or more displays.</td>
</tr>
<tr>
<td>XmuInternStrings()</td>
<td>Converts a list of atom names into Atom values.</td>
</tr>
<tr>
<td>XmuLocateBitmapFile()</td>
<td>Reads a file in standard bit map file format.</td>
</tr>
<tr>
<td>XmuLookupAPL()</td>
<td>This function is similar to XLookupString, except that it maps a key event to an APL string.</td>
</tr>
<tr>
<td>XmuLookupArabic()</td>
<td>This function is similar to XLookupString, except that it maps a key event to a Latin and Arabic (ISO 8859-6) string.</td>
</tr>
<tr>
<td>XmuLookupCloseDisplayHook()</td>
<td>Determines if a callback is installed.</td>
</tr>
<tr>
<td>XmuLookupCyrillic()</td>
<td>This function is similar to XLookupString, except that it maps a key event to a Latin and Cyrillic (ISO 8859-5) string.</td>
</tr>
<tr>
<td>XmuLookupGreek()</td>
<td>This function is similar to XLookupString, except that it maps a key event to a Latin and Greek (ISO 8859-7) string.</td>
</tr>
<tr>
<td>XmuLookupHebrew()</td>
<td>This function is similar to XLookupString, except that it maps a key event to a Latin and Hebrew (ISO 8859-8) string.</td>
</tr>
<tr>
<td>XmuLookupJISX0201()</td>
<td>This function is similar to XLookupString, except that it maps a key event to a string in the JIS X0201-1976 encoding.</td>
</tr>
</tbody>
</table>
Table 163. Miscellaneous utility routines (continued)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XmuLookupKana()</td>
<td>This function is similar to XLookupString, except that it maps a key event to a string in the JIS X0201-1976 encoding.</td>
</tr>
<tr>
<td>XmuLookupLatin1()</td>
<td>This function is identical to XLookupString.</td>
</tr>
<tr>
<td>XmuLookupLatin2()</td>
<td>This function is similar to XLookupString, except that it maps a key event to a Latin-2 (ISO 8859-2) string.</td>
</tr>
<tr>
<td>XmuLookupLatin3()</td>
<td>This function is similar to XLookupString, except that it maps a key event to a Latin-3 (ISO 8859-3) string.</td>
</tr>
<tr>
<td>XmuLookupLatin4()</td>
<td>This function is similar to XLookupString, except that it maps a key event to a Latin-4 (ISO 8859-4) string.</td>
</tr>
<tr>
<td>XmuLookupStandardColormap()</td>
<td>Creates or replaces a standard colormap if one does not currently exist.</td>
</tr>
<tr>
<td>XmuLookupString()</td>
<td>Maps a key event into a specific key symbol set.</td>
</tr>
<tr>
<td>XmuMakeAtom()</td>
<td>Creates and initializes an opaque object.</td>
</tr>
<tr>
<td>XmuNameOfAtom()</td>
<td>Returns the name of an AtomPtr.</td>
</tr>
<tr>
<td>XmuPrintDefaultErrorMessage()</td>
<td>Prints an error message, equivalent to Xlib’s default error message.</td>
</tr>
<tr>
<td>XmuReadBitmapData()</td>
<td>Reads a standard bit map file description.</td>
</tr>
<tr>
<td>XmuReadBitmapDataFromFile()</td>
<td>Reads a standard bit map file description from the specified file.</td>
</tr>
<tr>
<td>XmuReleaseStippledPixmap()</td>
<td>Frees a pixmap created with XmuCreateStippledPixmap.</td>
</tr>
<tr>
<td>XmuRemoveCloseDisplayHook()</td>
<td>Deletes a callback that has been added with XmuAddCloseDisplayHook.</td>
</tr>
<tr>
<td>XmuReshapeWidget()</td>
<td>Reshapes the specified widget, using the Shape extension.</td>
</tr>
<tr>
<td>XmuScreenOfWindow()</td>
<td>Returns the screen on which the specified window was created.</td>
</tr>
<tr>
<td>XmuSimpleErrorHandler()</td>
<td>A simple error handler for Xlib error conditions.</td>
</tr>
<tr>
<td>XmuStandardColormap()</td>
<td>Creates a standard colormap for the given screen.</td>
</tr>
<tr>
<td>XmuUpdateMapHints()</td>
<td>Clears the PPosition and PSize flags and sets the USPosition and USSize flags.</td>
</tr>
<tr>
<td>XmuVisualStandardColormaps()</td>
<td>Creates all of the appropriate standard colormaps for a given visual.</td>
</tr>
</tbody>
</table>

**X Window System routines: X authorization routines**

The MIT X Authorization routines are included in SEZAX11L. These routines are used to deal with X authorization data in X clients. Table 164 lists the X authorization routines and provides a short description of each routine.

Table 164. Authorization routines

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XauFileName()</td>
<td>Generates the default authorization file name.</td>
</tr>
<tr>
<td>XauReadAuth()</td>
<td>Reads the next entry from the authfile.</td>
</tr>
<tr>
<td>XuWriteAuth()</td>
<td>Writes an authorization entry to the authfile.</td>
</tr>
</tbody>
</table>
Table 164. Authorization routines (continued)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XauGetAuthByAddr()</td>
<td>Searches for an authorization entry.</td>
</tr>
<tr>
<td>XauLockAuth()</td>
<td>Does the work necessary to synchronously update an authorization file.</td>
</tr>
<tr>
<td>XauUnlockAuth()</td>
<td>Undoes the work of XauLockAuth.</td>
</tr>
<tr>
<td>XauDisposeAuth()</td>
<td>Frees storage allocated to hold an authorization entry.</td>
</tr>
</tbody>
</table>

**X Window System toolkit**

An X Window System Toolkit is a set of library functions layered on top of the X Window System Xlib functions that allows you to simplify the design of applications by providing an underlying set of common user interface functions. Included are mechanisms for defining and expanding interclient and intracomponent interaction independently, masking implementation details from both the application and component implementor.

An X Window System Toolkit consists of the following:
- A set of programming mechanisms, called Intrinsics, that are used to build widgets.
- An architectural model to help programmers design new widgets, with enough flexibility to accommodate different application interface layers.
- A consistent interface, in the form of a coordinated set of widgets and composition policies, some of which are application domain-specific, while others are common across several application domains.

The fundamental data type of the X Window System Toolkit is the widget. A widget is allocated dynamically and contains state information. Every widget belongs to one widget class that is allocated statically and initialized. The widget class contains the operations allowed on widgets of that class.

An X Window System Toolkit manages the following functions:
- Toolkit initialization
- Widgets and widget geometry
- Memory
- Window, data set, and timer events
- Input focus
- Selections
- Resources and resource conversion
- Translation of events
- Graphics contexts
- Pixmaps
- Errors and warnings

You must remap many of the X Widget and X Intrinsics routine names. This remapping is done in a header file called XT@REMAP.H. This file is automatically included by the INTRINSIC.H header file. In debugging your application, see the XT@REMAP.H file to find the remapped names of the X Toolkit routines.
Some of the X Window System header data sets have been renamed from their original distribution names, because of the data set naming conventions in the MVS environment. Such name changes are generally restricted to those header files used internally by the actual widget code, rather than the application header files, to minimize the number of changes required for an application to be ported to the MVS environment.

In porting applications to the MVS environment, you might have to make changes to header file names in Table 165.

<table>
<thead>
<tr>
<th>MIT distribution name</th>
<th>TCP/IP name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CompositeI.h</td>
<td>ComposiI.h</td>
</tr>
<tr>
<td>CompositeP.h</td>
<td>ComposiP.h</td>
</tr>
<tr>
<td>ConstrainP.h</td>
<td>ConstraP.h</td>
</tr>
<tr>
<td>IntrinsicI.h</td>
<td>IntriniI.h</td>
</tr>
<tr>
<td>IntrinsicP.h</td>
<td>IntriniP.h</td>
</tr>
<tr>
<td>PassivGraI.h</td>
<td>PassivGr.h</td>
</tr>
<tr>
<td>ProtocolsP.h</td>
<td>ProtocoP.h</td>
</tr>
<tr>
<td>SelectionI.h</td>
<td>SelectI.h</td>
</tr>
<tr>
<td>WindowObjP.h</td>
<td>WindowOP.h</td>
</tr>
</tbody>
</table>

**Table 165. X Intrinsic header file names**

*Xt Intrinsics routines*

Table 166 provides the Xt Intrinsics routines and a short description of each routine.

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CompositeClassPartInitialize</td>
<td>Initializes the CompositeClassPart of a composite widget.</td>
</tr>
<tr>
<td>CompositeDeleteChild</td>
<td>Deletes a child widget from a composite widget.</td>
</tr>
<tr>
<td>CompositeDestroy</td>
<td>Destroys a composite widget.</td>
</tr>
<tr>
<td>CompositeInitialize</td>
<td>Initializes a composite widget structure.</td>
</tr>
<tr>
<td>CompositInsertChild</td>
<td>Inserts a child widget in a composite widget.</td>
</tr>
<tr>
<td>RemoveCallback</td>
<td>Removes a callback procedure from a callback list.</td>
</tr>
<tr>
<td>XrmCompileResourceList</td>
<td>Compiles an XtResourceList into an XrmResourceList.</td>
</tr>
<tr>
<td>XtAddActions</td>
<td>Declares an action table and registers it with the translation manager.</td>
</tr>
<tr>
<td>XtAddCallback</td>
<td>Adds a callback procedure to the callback list of the specified widget.</td>
</tr>
<tr>
<td>XtAddCallbacks</td>
<td>Adds a list of callback procedures to the callback list of specified widget.</td>
</tr>
<tr>
<td>XtAddConverter</td>
<td>Adds a new converter.</td>
</tr>
<tr>
<td>XtAddEventHandler</td>
<td>Registers an event handler procedure with the dispatch mechanism when an event matching the mask occurs on the specified widget.</td>
</tr>
<tr>
<td>XtAddExposureToRegion</td>
<td>Computes the union of the rectangle defined by the specified exposure event and region.</td>
</tr>
<tr>
<td>Routine</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>XtAddGrab</td>
<td>Redirects user input to a model widget.</td>
</tr>
<tr>
<td>XtAddInput</td>
<td>Registers a new source of events.</td>
</tr>
<tr>
<td>XtAddRawEventHandler</td>
<td>Registers an event handler procedure with the dispatch mechanism without causing the server to select for that event.</td>
</tr>
<tr>
<td>XtAddTimeOut</td>
<td>Creates a timeout value in the default application context and returns an identifier for it.</td>
</tr>
<tr>
<td>XtAddWorkProc</td>
<td>Registers a work procedure in the default application context.</td>
</tr>
<tr>
<td>XtAppAddActionHook</td>
<td>Adds an actionhook procedure to an application context.</td>
</tr>
<tr>
<td>XtAppAddActions</td>
<td>Declares an action table and registers it with the translation manager.</td>
</tr>
<tr>
<td>XtAppAddConverter</td>
<td>Registers a new converter.</td>
</tr>
<tr>
<td>XtAppAddInput</td>
<td>Registers a new file as an input source for a specified application.</td>
</tr>
<tr>
<td>XtAppAddTimeOut</td>
<td>Creates a timeout value and returns an identifier for it.</td>
</tr>
<tr>
<td>XtAppAddWorkProc</td>
<td>Registers a work procedure for a specified procedure.</td>
</tr>
<tr>
<td>XtAppCreateShell</td>
<td>Creates a top-level widget that is the root of a widget tree.</td>
</tr>
<tr>
<td>XtAppError</td>
<td>Calls the installed unrecoverable error procedure.</td>
</tr>
<tr>
<td>XtAppErrorMsg</td>
<td>Calls the high-level error handler.</td>
</tr>
<tr>
<td>XtAppGetErrorDatabase</td>
<td>Obtains the error database and merges it with an application or database specified by a widget.</td>
</tr>
<tr>
<td>XtAppGetErrorDatabaseText</td>
<td>Obtains the error database text for an error or warning for an error message handler.</td>
</tr>
<tr>
<td>XtAppGetSelectionTimeout</td>
<td>Gets and returns the current selection timeout (ms) value.</td>
</tr>
<tr>
<td>XtAppInitialize</td>
<td>A convenience routine for initializing the toolkit.</td>
</tr>
<tr>
<td>XtAppMainLoop</td>
<td>Process input by calling XtAppNextEvent and XtDispatchEvent.</td>
</tr>
<tr>
<td>XtAppNextEvent</td>
<td>Returns the value from the top of a specified application input queue.</td>
</tr>
<tr>
<td>XtAppPeekEvent</td>
<td>Returns the value from the top of a specified application input queue without removing input from queue.</td>
</tr>
<tr>
<td>XtAppPending</td>
<td>Determines if the input queue has any events for a specified application.</td>
</tr>
<tr>
<td>XtAppProcessEvent</td>
<td>Processes applications that require direct control of the processing for different types of input.</td>
</tr>
<tr>
<td>XtAppReleaseCacheRefs</td>
<td>Decrements the reference count for the conversion entries identified by the refs argument.</td>
</tr>
<tr>
<td>XtAppSetErrorHandler</td>
<td>Registers a procedure to call on unrecoverable error conditions. The default error handler prints the message to standard error.</td>
</tr>
<tr>
<td>XtAppSetErrorMsgHandler</td>
<td>Registers a procedure to call on unrecoverable error conditions. The default error handler constructs a string from the error resource database.</td>
</tr>
<tr>
<td>Routine</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>XtAppSetFallbackResources</td>
<td>Sets the fallback resource list that will be loaded at display initialization time.</td>
</tr>
<tr>
<td>XtAppSetSelectionTimeout</td>
<td>Sets the Intrinsics selection timeout value.</td>
</tr>
<tr>
<td>XtAppSetTypeConverter</td>
<td>Registers the specified type converter and destructor in all application contexts created by the calling process.</td>
</tr>
<tr>
<td>XtAppSetWarningHandler</td>
<td>Registers a procedure to call on nonfatal error conditions. The default warning handler prints the message to standard error.</td>
</tr>
<tr>
<td>XtAppSetWarningMsgHandler</td>
<td>Registers a procedure to call on nonfatal error conditions. The default warning handler constructs a string from error resource database.</td>
</tr>
<tr>
<td>XtAppWarning</td>
<td>Calls the installed nonfatal error procedure.</td>
</tr>
<tr>
<td>XtAppWarningMsg</td>
<td>Calls the installed high-level warning handler.</td>
</tr>
<tr>
<td>XtAugmentTranslations</td>
<td>Merges new translations into an existing widget translation table.</td>
</tr>
<tr>
<td>XtBuildEventMask</td>
<td>Retrieves the event mask for a specified widget.</td>
</tr>
<tr>
<td>XtCallAcceptFocus</td>
<td>Calls the accept_focus procedure for the specified widget.</td>
</tr>
<tr>
<td>XtCallActionProc</td>
<td>Searches for the named action routine and, if found, calls it.</td>
</tr>
<tr>
<td>XtCallbackExclusive</td>
<td>Calls customized code for callbacks to create pop-up shell.</td>
</tr>
<tr>
<td>XtCallbackNone</td>
<td>Calls customized code for callbacks to create pop-up shell.</td>
</tr>
<tr>
<td>XtCallbackNonexclusive</td>
<td>Calls customized code for callbacks to create pop-up shell.</td>
</tr>
<tr>
<td>XtCallbackPopdown</td>
<td>Pops down a shell that was mapped by callback functions.</td>
</tr>
<tr>
<td>XtCallbackReleaseCacheRef</td>
<td>A callback that can be added to a callback list to release a previously returned XtCacheRef value.</td>
</tr>
<tr>
<td>XtCallbackReleaseCacheRefList</td>
<td>A callback that can be added to a callback list to release a list of previously returned XtCacheRef value.</td>
</tr>
<tr>
<td>XtCallbackList</td>
<td>Calls all callbacks on a callback list.</td>
</tr>
<tr>
<td>XtCallback</td>
<td>Lists the callback procedures in a widget callback list.</td>
</tr>
<tr>
<td>XtCallbackConverter</td>
<td>Looks up the specified type converter in the application context and invokes the conversion routine.</td>
</tr>
<tr>
<td>XtCalloc</td>
<td>Allocates and initializes an array.</td>
</tr>
<tr>
<td>XtClass</td>
<td>Obtains the class of a widget and returns a pointer to the widget class structure.</td>
</tr>
<tr>
<td>XtCloseDisplay</td>
<td>Closes a display and removes it from an application context.</td>
</tr>
<tr>
<td>XtConfigureWidget</td>
<td>Moves and resizes the sibling widget of the child making the geometry request.</td>
</tr>
<tr>
<td>XtConvert</td>
<td>Invokes resource conversions.</td>
</tr>
<tr>
<td>XtConvertAndStore</td>
<td>Looks up the type converter registered to convert from_type to to_type and then calls XtCallConverter.</td>
</tr>
<tr>
<td>Routine</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>XtConvertCase</td>
<td>Determines upper and lowercase equivalents for a KeySym.</td>
</tr>
<tr>
<td>XtCopyAncestorSensitive</td>
<td>Copies the sensitive value from a widget record.</td>
</tr>
<tr>
<td>XtCopyDefaultColormap</td>
<td>Copies the default colormap from a widget record.</td>
</tr>
<tr>
<td>XtCopyDefaultDepth</td>
<td>Copies the default depth from a widget record.</td>
</tr>
<tr>
<td>XtCopyFromParent</td>
<td>Copies the parent from a widget record.</td>
</tr>
<tr>
<td>XtCopyScreen</td>
<td>Copies the screen from a widget record.</td>
</tr>
<tr>
<td>XtCreateApplicationContext</td>
<td>Creates an opaque type application context.</td>
</tr>
<tr>
<td>XtCreateApplicationShell</td>
<td>Creates an application shell widget by calling XtAppCreateShell.</td>
</tr>
<tr>
<td>XtCreateManagedWidget</td>
<td>Creates and manages a child widget in a single procedure.</td>
</tr>
<tr>
<td>XtCreatePopupShell</td>
<td>Creates a pop-up shell.</td>
</tr>
<tr>
<td>XtCreateWidget</td>
<td>Creates an instance of a widget.</td>
</tr>
<tr>
<td>XtCreateWindow</td>
<td>Calls XcreateWindow with the widget structure and parameter.</td>
</tr>
<tr>
<td>XtDatabase</td>
<td>Obtains the resource database for a particular display.</td>
</tr>
<tr>
<td>XtDestroyApplicationContext</td>
<td>Destroys an application context.</td>
</tr>
<tr>
<td>XtDestroyGC</td>
<td>Deallocates graphics context when it is no longer needed.</td>
</tr>
<tr>
<td>XtDestroyWidget</td>
<td>Destroys a widget instance.</td>
</tr>
<tr>
<td>XtDirectConvert</td>
<td>Invokes resource conversion.</td>
</tr>
<tr>
<td>XtDisownSelection</td>
<td>Informs the Intrinsics selection mechanism that the specified widget is to lose ownership of the selection.</td>
</tr>
<tr>
<td>XtDispatchEvent</td>
<td>Receives X events and calls appropriate event handlers.</td>
</tr>
<tr>
<td>XtDisplay</td>
<td>Returns the display pointer for the specified widget.</td>
</tr>
<tr>
<td>XtDisplayInitialize</td>
<td>Initializes a display and adds it to an application context.</td>
</tr>
<tr>
<td>XtDisplayOfObject</td>
<td>Returns the display pointer for the specified widget.</td>
</tr>
<tr>
<td>XtDisplayStringConversionWarning</td>
<td>Issues a warning message for conversion routines.</td>
</tr>
<tr>
<td>XtDisplayToApplicationContext</td>
<td>Retrieves the application context associated with a Display.</td>
</tr>
<tr>
<td>XtError</td>
<td>Calls the installed unrecoverable error procedure.</td>
</tr>
<tr>
<td>XtErrorMsg</td>
<td>A low-level error and warning handler procedure type.</td>
</tr>
<tr>
<td>XtFindFile</td>
<td>Searches for a file using substitutions in a path list.</td>
</tr>
<tr>
<td>XtFree</td>
<td>Frees an allocated block of storage.</td>
</tr>
<tr>
<td>XtGetActionKeysym</td>
<td>Retrieves the KeySym and modifiers that matched the final event specification in a translation table entry.</td>
</tr>
<tr>
<td>XtGetApplicationNameAndClass</td>
<td>Returns the application name and class as passed to XtDisplayInitialize</td>
</tr>
<tr>
<td>XtGetApplicationResources</td>
<td>Retrieves resources that are not specific to a widget, but apply to the overall application.</td>
</tr>
<tr>
<td>XtGetConstraintResourceList</td>
<td>Returns the constraint resource list for a particular widget.</td>
</tr>
</tbody>
</table>
### Table 166. Xt Intrinsics routines (continued)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtGetErrorDatabase</td>
<td>Obtains the error database and returns the address of the error database.</td>
</tr>
<tr>
<td>XtGetErrorDatabaseText</td>
<td>Obtains the error database text for an error or warning.</td>
</tr>
<tr>
<td>XtGetGC</td>
<td>Returns a read-only sharable GC.</td>
</tr>
<tr>
<td>XtGetKeysymTable</td>
<td>Returns a pointer to the KeySym to KeyCode mapping table for a particular display.</td>
</tr>
<tr>
<td>XtGetMultiClickTime</td>
<td>Returns the multiclick time setting.</td>
</tr>
<tr>
<td>XtGetResourceList</td>
<td>Obtains the resource list structure for a particular class.</td>
</tr>
<tr>
<td>XtGetSelectionRequest</td>
<td>Retrieves the SelectionRequest event that triggered the convert_selection procedure.</td>
</tr>
<tr>
<td>XtGetSelectionTimeout</td>
<td>Obtains the current selection timeout.</td>
</tr>
<tr>
<td>XtGetSelectionValue</td>
<td>Obtains the selection value in a single, logical unit.</td>
</tr>
<tr>
<td>XtGetSelectionValueIncremental</td>
<td>Obtains the selection value using incremental transfers.</td>
</tr>
<tr>
<td>XtGetSelectionValues</td>
<td>Takes a list of target types and client data and obtains the current value of the selection converted to each of the targets.</td>
</tr>
<tr>
<td>XtGetSelectionValuesIncremental</td>
<td>A function similar to XtGetSelectionValueIncremental except that it takes a list of targets and client_data.</td>
</tr>
<tr>
<td>XtGetSubresources</td>
<td>Obtains resources other than widgets.</td>
</tr>
<tr>
<td>XtGetSubvalues</td>
<td>Retrieves the current value of a nonwidget resource data associated with a widget instance.</td>
</tr>
<tr>
<td>XtGetValues</td>
<td>Retrieves the current value of a resource associated with a widget instance.</td>
</tr>
<tr>
<td>XtGrabButton</td>
<td>Passively grabs a single pointer button.</td>
</tr>
<tr>
<td>XtGrabKey</td>
<td>Passively grabs a single key of the keyboard.</td>
</tr>
<tr>
<td>XtGrabKeyboard</td>
<td>Actively grabs the keyboard.</td>
</tr>
<tr>
<td>XtGrabPointer</td>
<td>Actively grabs the pointer.</td>
</tr>
<tr>
<td>XtHasCallbacks</td>
<td>Finds the status of a specified widget callback list.</td>
</tr>
<tr>
<td>XtInitialize</td>
<td>Initializes the toolkit, application, and shell.</td>
</tr>
<tr>
<td>XtInitializeWidgetClass</td>
<td>Initializes a widget class without creating any widgets.</td>
</tr>
<tr>
<td>XtInsertEventHandler</td>
<td>Registers an event handler procedure that receives events before or after all previously registered event handlers.</td>
</tr>
<tr>
<td>XtInsertRawEventHandler</td>
<td>Registers an event handler procedure that receives events before or after all previously registered event handlers without selecting for the events.</td>
</tr>
<tr>
<td>XtInstallAccelerators</td>
<td>Installs accelerators from a source widget to destination widget.</td>
</tr>
<tr>
<td>XtInstallAllAccelerators</td>
<td>Installs all the accelerators from a widget and all the descendants of the widget onto one destination widget.</td>
</tr>
<tr>
<td>XtIsApplicationShell</td>
<td>Determines whether a specified widget is a subclass of an application shell widget.</td>
</tr>
<tr>
<td>XtIsComposite</td>
<td>Determines whether a specified widget is a subclass of a composite widget.</td>
</tr>
<tr>
<td>XtIsConstraint</td>
<td>Determines whether a specified widget is a subclass of a constraint widget.</td>
</tr>
<tr>
<td>Routine</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>XtIsManaged</td>
<td>Determines the managed state of a specified child widget.</td>
</tr>
<tr>
<td>XtIsObject</td>
<td>Determines whether a specified widget is a subclass of an object widget.</td>
</tr>
<tr>
<td>XtIsOverrideShell</td>
<td>Determines whether a specified widget is a subclass of an override shell widget.</td>
</tr>
<tr>
<td>XtIsRealized</td>
<td>Determines if a widget has been realized.</td>
</tr>
<tr>
<td>XtIsRectObj</td>
<td>Determines whether a specified widget is a subclass of a RectObj widget.</td>
</tr>
<tr>
<td>XtIsSensitive</td>
<td>Determines the current sensitivity state of a widget.</td>
</tr>
<tr>
<td>XtIsShell</td>
<td>Determines whether a specified widget is a subclass of a shell widget.</td>
</tr>
<tr>
<td>XtIsSubclass</td>
<td>Determines whether a specified widget is in a specific subclass.</td>
</tr>
<tr>
<td>XtIsTopLevelShell</td>
<td>Determines whether a specified widget is a subclass of a TopLevelShell widget.</td>
</tr>
<tr>
<td>XtIsTransientShell</td>
<td>Determines whether a specified widget is a subclass of a TransientShell widget.</td>
</tr>
<tr>
<td>XtIsVendorShell</td>
<td>Determines whether a specified widget is a subclass of a VendorShell widget.</td>
</tr>
<tr>
<td>XtIsWidget</td>
<td>Determines whether a specified widget is a subclass of a Widget widget.</td>
</tr>
<tr>
<td>XtIsWMShell</td>
<td>Determines whether a specified widget is a subclass of a WMShell widget.</td>
</tr>
<tr>
<td>XtKeysymToKeycodeList</td>
<td>Returns the list of KeyCodes that map to a particular KeySym.</td>
</tr>
<tr>
<td>XtLastTimestampProcessed</td>
<td>Retrieves the timestamp from the most recent call to XtDispatchEvent.</td>
</tr>
<tr>
<td>XtMainLoop</td>
<td>An infinite loop that processes input.</td>
</tr>
<tr>
<td>XtMakeGeometryRequest</td>
<td>A request from the child widget to a parent widget for a geometry change.</td>
</tr>
<tr>
<td>XtMakeResizeRequest</td>
<td>Makes a resize request from a widget.</td>
</tr>
<tr>
<td>XtMalloc</td>
<td>Allocates storage.</td>
</tr>
<tr>
<td>XtManageChild</td>
<td>Adds a single child to a parent widget list of managed children.</td>
</tr>
<tr>
<td>XtManageChildren</td>
<td>Adds a list of widgets to the geometry-managed, displayable, subset of its composite parent widget.</td>
</tr>
<tr>
<td>XtMapWidget</td>
<td>Maps a widget explicitly.</td>
</tr>
<tr>
<td>XtMenuPopupAction</td>
<td>Pops up a menu when a pointer button is pressed or when the pointer is moved into the widget.</td>
</tr>
<tr>
<td>XtMergeArgLists</td>
<td>Merges two ArgList structures.</td>
</tr>
<tr>
<td>XtMoveWidget</td>
<td>Moves a sibling widget of the child making the geometry request.</td>
</tr>
<tr>
<td>XtName</td>
<td>Returns a pointer to the instance name of the specified object.</td>
</tr>
<tr>
<td>XtNameToWidget</td>
<td>Translates a widget name to a widget instance.</td>
</tr>
</tbody>
</table>
Table 166. Xt Intrinsic routines (continued)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtNewString</td>
<td>Copies an instance of a string.</td>
</tr>
<tr>
<td>XtNextEvent</td>
<td>Returns the value from the header of the input queue.</td>
</tr>
<tr>
<td>XtOpenDisplay</td>
<td>Opens, initializes, and adds a display to an application context.</td>
</tr>
<tr>
<td>XtOverrideTranslations</td>
<td>Overwrites existing translations with new translations.</td>
</tr>
<tr>
<td>XtOwnSelection</td>
<td>Sets the selection owner when using atomic transfer.</td>
</tr>
<tr>
<td>XtOwnSelectionIncremental</td>
<td>Sets the selection owner when using incremental transfers.</td>
</tr>
<tr>
<td>XtParent</td>
<td>Returns the parent widget for the specified widget.</td>
</tr>
<tr>
<td>XtParseAcceleratorTable</td>
<td>Parses an accelerator table into the opaque internal representation.</td>
</tr>
<tr>
<td>XtParseTranslationTable</td>
<td>Compiles a translation table into the opaque internal representation.</td>
</tr>
<tr>
<td>XtPeekEvent</td>
<td>Returns the value from the front of the input queue without removing it from the queue.</td>
</tr>
<tr>
<td>XtPending</td>
<td>Determines if the input queue has events pending.</td>
</tr>
<tr>
<td>XtPopdown</td>
<td>Unmaps a pop-up from within an application.</td>
</tr>
<tr>
<td>XtPopup</td>
<td>Maps a pop-up from within an application.</td>
</tr>
<tr>
<td>XtPopupSpringLoaded</td>
<td>Maps a spring-loaded pop-up from within an application.</td>
</tr>
<tr>
<td>XtProcessEvent</td>
<td>Processes one input event, timeout, or alternate input source.</td>
</tr>
<tr>
<td>XtQueryGeometry</td>
<td>Queries the preferred geometry of a child widget.</td>
</tr>
<tr>
<td>XtRealizeWidget</td>
<td>Realizes a widget instance.</td>
</tr>
<tr>
<td>XtRealloc</td>
<td>Changes the size of an allocated block of storage, sometimes moving it.</td>
</tr>
<tr>
<td>XtRegisterCaseConverter</td>
<td>Registers a specified case converter.</td>
</tr>
<tr>
<td>XtRegisterGrabAction</td>
<td>Registers button and key grabs for a widget window according to the event bindings in the widget translation table.</td>
</tr>
<tr>
<td>XtReleaseGC</td>
<td>Deallocates a shared GC when it is no longer needed.</td>
</tr>
<tr>
<td>XtRemoveActionHook</td>
<td>Removes an action hook procedure without destroying the application context.</td>
</tr>
<tr>
<td>XtRemoveAllCallbacks</td>
<td>Deletes all callback procedures from a specified widget callback list.</td>
</tr>
<tr>
<td>XtRemoveCallback</td>
<td>Deletes a callback procedure from a specified widget callback list only if both the procedure and the client data match.</td>
</tr>
<tr>
<td>XtRemoveCallbacks</td>
<td>Deletes a list of callback procedures from a specified widget callback list.</td>
</tr>
<tr>
<td>XtRemoveEventHandler</td>
<td>Removes a previously registered event handler.</td>
</tr>
<tr>
<td>XtRemoveGrab</td>
<td>Removes the redirection of user input to a modal widget.</td>
</tr>
<tr>
<td>XtRemoveInput</td>
<td>Discontinues a source of input by causing the Intrinsic read routine to stop watching for input from the input source.</td>
</tr>
<tr>
<td>XtRemoveRawEventHandler</td>
<td>Removes previously registered raw event handler.</td>
</tr>
</tbody>
</table>
### Table 166. Xt Intrinsics routines (continued)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtRemoveTimeOut</td>
<td>Clears a timeout value by removing the timeout.</td>
</tr>
<tr>
<td>XtRemoveWorkProc</td>
<td>Removes the specified background work procedure.</td>
</tr>
<tr>
<td>XtResizeWidget</td>
<td>Resizes a sibling widget of the child making the geometry request.</td>
</tr>
<tr>
<td>XtResizeWindow</td>
<td>Resizes a child widget that already has the values for its width, height, and border width.</td>
</tr>
<tr>
<td>XtResolvePathname</td>
<td>Searches for a file using standard substitutions in a path list.</td>
</tr>
<tr>
<td>XtScreen</td>
<td>Returns the screen pointer for the specified widget.</td>
</tr>
<tr>
<td>XtScreenOfObject</td>
<td>Returns the screen pointer for the nearest ancestor of object that is of class Widget.</td>
</tr>
<tr>
<td>XtSetErrorHandler</td>
<td>Registers a procedure to call under unrecoverable error conditions.</td>
</tr>
<tr>
<td>XtSetErrorMsgHandler</td>
<td>Registers a procedure to call under unrecoverable error conditions.</td>
</tr>
<tr>
<td>XtSetKeyboardFocus</td>
<td>Redirects keyboard input to a child of a composite widget without calling XSetInputFocus.</td>
</tr>
<tr>
<td>XtSetKeyTranslator</td>
<td>Registers a key translator.</td>
</tr>
<tr>
<td>XtSetMappedWhenManaged</td>
<td>Changes the widget map_when_managed field.</td>
</tr>
<tr>
<td>XtSetMultiClickTime</td>
<td>Sets the multi-click time for an application.</td>
</tr>
<tr>
<td>XtSetSelectionTimeout</td>
<td>Sets the Intrinsics selection timeout.</td>
</tr>
<tr>
<td>XtSetSensitive</td>
<td>Sets the sensitivity state of a widget.</td>
</tr>
<tr>
<td>XtSetSubvalues</td>
<td>Sets the current value of a nonwidget resource associated with an instance.</td>
</tr>
<tr>
<td>XtSetTypeConverter</td>
<td>Registers a type converter for all application contexts in a process.</td>
</tr>
<tr>
<td>XtSetValues</td>
<td>Modifies the current value of a resource associated with widget instance.</td>
</tr>
<tr>
<td>XtSetWarningHandler</td>
<td>Registers a procedure to be called on non-fatal error conditions.</td>
</tr>
<tr>
<td>XtSetWarningMsgHandler</td>
<td>Registers a procedure to be called on nonfatal error conditions.</td>
</tr>
<tr>
<td>XtSetWMColormapWindows</td>
<td>Sets the value of the WM_COLORMAP_WINDOWS property on a widget's window.</td>
</tr>
<tr>
<td>XtStringConversionWarning</td>
<td>A convenience routine for old format resource converters that convert from strings.</td>
</tr>
<tr>
<td>XtSuperclass</td>
<td>Obtains the superclass of a widget by returning a pointer to the superclass structure of the widget.</td>
</tr>
<tr>
<td>XtToolkitInitialize</td>
<td>Initializes the X Toolkit internals.</td>
</tr>
<tr>
<td>XtTranslateCoords</td>
<td>Translates an [x,y] coordinate pair from widget coordinates to root coordinates.</td>
</tr>
<tr>
<td>XtTranslateKey</td>
<td>The default key translator routine.</td>
</tr>
<tr>
<td>XtTranslateKeyCode</td>
<td>Registers a key translator.</td>
</tr>
<tr>
<td>XtUngrabButton</td>
<td>Cancels a passive button grab.</td>
</tr>
<tr>
<td>XtUngrabKey</td>
<td>Cancels a passive key grab.</td>
</tr>
</tbody>
</table>
Table 166. Xt Intrinsics routines (continued)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XtUngrabKeyboard</td>
<td>Cancels an active keyboard grab.</td>
</tr>
<tr>
<td>XtUngrabPointer</td>
<td>Cancels an active pointer grab.</td>
</tr>
<tr>
<td>XtUninstallTranslations</td>
<td>Causes the entire translation table for widget to be removed.</td>
</tr>
<tr>
<td>XtUnmanageChild</td>
<td>Removes a single child from the managed set of its parent.</td>
</tr>
<tr>
<td>XtUnmanageChildren</td>
<td>Removes a list of children from the managed list of the parent, but does not destroy the children widgets.</td>
</tr>
<tr>
<td>XtUnmapWidget</td>
<td>Unmaps a widget explicitly.</td>
</tr>
<tr>
<td>XtUnrealizeWidget</td>
<td>Destroys the associated widget and its descendants.</td>
</tr>
<tr>
<td>XtVaAppCreateShell</td>
<td>Creates a top-level widget that is the root of a widget tree using varargs lists.</td>
</tr>
<tr>
<td>XtVaAppInitialize</td>
<td>Initializes the Xtk internals, creates an application context, opens and initializes a display, and creates the initial application shell instance using varargs lists.</td>
</tr>
<tr>
<td>XtVaCreateArgsList</td>
<td>Dynamically allocates a varargs list for use with XtVaNestedList in multiple calls.</td>
</tr>
<tr>
<td>XtVaCreateManagedWidget</td>
<td>Creates and manages a child widget in a single procedure using varargs lists.</td>
</tr>
<tr>
<td>XtVaCreatePopupShell</td>
<td>Creates a pop-up shell using varargs lists.</td>
</tr>
<tr>
<td>XtVaCreateWidget</td>
<td>Creates an instance of a widget using varargs lists.</td>
</tr>
<tr>
<td>XtVaGetApplicationResources</td>
<td>Retrieves resources for the overall application using varargs list.</td>
</tr>
<tr>
<td>XtVaGetSubresources</td>
<td>Fetches resources for widget subparts using varargs list.</td>
</tr>
<tr>
<td>XtVaGetSubvalues</td>
<td>Retrieves the current values of nonwidget resources associated with a widget instance using varargs lists.</td>
</tr>
<tr>
<td>XtVaGetValues</td>
<td>Retrieves the current values of resources associated with a widget instance using varargs lists.</td>
</tr>
<tr>
<td>XtVaSetSubvalues</td>
<td>Sets the current values of nonwidget resources associated with a widget instance using varargs lists.</td>
</tr>
<tr>
<td>XtVaSetValues</td>
<td>Modifies the current values of resources associated with a widget instance using varargs lists.</td>
</tr>
<tr>
<td>XtWarning</td>
<td>Calls the installed non-fatal error procedure.</td>
</tr>
<tr>
<td>XtWarningMsg</td>
<td>Calls the installed high-level warning handler.</td>
</tr>
<tr>
<td>XtWidgetToApplicationContext</td>
<td>Gets the application context for given widget.</td>
</tr>
<tr>
<td>XtWindow</td>
<td>Returns the window of the specified widget.</td>
</tr>
<tr>
<td>XtWindowOfObject</td>
<td>Returns the window for the nearest ancestor of object that is of class Widget.</td>
</tr>
<tr>
<td>XtWindowToWidget</td>
<td>Translates a window and display pointer into a widget instance.</td>
</tr>
</tbody>
</table>
X Window System toolkit: Application resources

X applications can be modified at run time by a set of resources. Applications that make use of an X Window System toolkit can be modified by additional sets of application resources. These resources are searched until a resource specification is found. The X Intrinsics determine the actual search order used for determining a resource value.

The search order used in the TSO environment, in descending order of preference, is:

1. Command Line
   Standard arguments include:
   a. Command switches (-display, -fg, -foreground, +rv)
   b. Resource manager directives (-name, -xrm)
   c. Natural language directive (-xnllanguage)
2. User Environment File
   Use the source found from the user_id.XDEFAULT.host data set. In this case, host is the string returned by the gethostname() call.
3. Server and User Preference Resources
   Use the first source found from:
   a. RESOURCE_MANAGER property on the root window [screen0]
   b. user_id.X.DEFAULTS data set
4. Application Class Resources
   Use the first source found from:
   a. The default application resource data set named user_id.XAPDF.classname, where classname is the application specified class name.
      The MVS data set name XAPDF is modified, if a natural language directive is specified as xnllanguageXAPDF, where xnllanguage is the string specified by the natural language directive.
   b. Fallback resources defined by XtAppSetFallbackResources within the application.

X Window System routines: Athena widget support

The X Window System support with TCP/IP includes the widget set developed at Massachusetts Institute of Technology (MIT), which is generally known as the Athena widget set.
The Athena widget set supports the following widgets:

- AsciiSink
- AsciiSrc
- AsciiText
- Box
- Clock
- Command
- Dialog
- Form
- Grip
- Label
- List
- Logo
- Mailbox
- MenuButton
- Paned
- Scrollbar
- Simple
- SimpleMenu
- Sme (Simple Menu Entry)
- SmeBSB (BSB Menu Entry)
- SmeLine
- StripChart
- Text
- TextSink
- TextSrc
- Toggle
- VPaned
- Viewport

Table 167 provides the Athena widget routines with a short description of each routine.

### Table 167. Athena widget routines

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XawAsciiSave</td>
<td>Saves the changes made in the current text source into a file.</td>
</tr>
<tr>
<td>XawAsciiSaveAsFile</td>
<td>Saves the contents of the current text buffer into a named file.</td>
</tr>
<tr>
<td>XawAsciiSourceChanged</td>
<td>Determines if the text buffer in an AsciiSrc object has changed.</td>
</tr>
<tr>
<td>XawAsciiSourceFreeString</td>
<td>Frees the storage associated with the string from an AsciiSrc widget requested with a call to XtGetValues.</td>
</tr>
<tr>
<td>XawDialogAddButton</td>
<td>Adds a new button to a Dialog widget.</td>
</tr>
<tr>
<td>XawDialogGetValueString</td>
<td>Returns the character string in the text field of a Dialog Widget.</td>
</tr>
<tr>
<td>XawDiskSourceCreate</td>
<td>Creates a disk source.</td>
</tr>
<tr>
<td>XawFormDoLayout</td>
<td>Forces or defers a relayout of the Form.</td>
</tr>
<tr>
<td>XawInitializeWidgetSet</td>
<td>Forces a reference to vendor shell so that the one in this widget is installed.</td>
</tr>
<tr>
<td>XawListChange</td>
<td>Changes the list that is displayed.</td>
</tr>
<tr>
<td>XawListHighlight</td>
<td>Highlights an item in the list.</td>
</tr>
<tr>
<td>XawListShowCurrent</td>
<td>Retrieves the list element that is currently set.</td>
</tr>
<tr>
<td>XawListUnhighlight</td>
<td>Unhighlights an item in the list.</td>
</tr>
<tr>
<td>XawPanedAllowResize</td>
<td>Enables or disables a child’s request for pane resizing.</td>
</tr>
<tr>
<td>XawPanedGetMinMax</td>
<td>Retrieves the minimum and maximum height settings for a pane.</td>
</tr>
<tr>
<td>XawPanedGetNumSub</td>
<td>Retrieves the number of panes in a paned widget.</td>
</tr>
<tr>
<td>XawPanedSetMinMax</td>
<td>Sets the minimum and maximum height settings for a pane.</td>
</tr>
<tr>
<td>XawPanedSetRefigureMode</td>
<td>Enables or disables automatic recalculation of pane sizes and positions.</td>
</tr>
<tr>
<td>XawScrollbarSetThumb</td>
<td>Sets the position and length of a Scrollbar thumb.</td>
</tr>
<tr>
<td>Routine</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>XawSimpleMenuAddGlobalActions</td>
<td>Registers an XawPositionSimpleMenu global action routine.</td>
</tr>
<tr>
<td>XawSimpleMenuClearActiveEntry</td>
<td>Clears the SimpleMenu widget internal information about the currently</td>
</tr>
<tr>
<td></td>
<td>highlighted menu entry.</td>
</tr>
<tr>
<td>XawSimpleMenuGetActiveEntry</td>
<td>Gets the currently highlighted menu entry.</td>
</tr>
<tr>
<td>XawStringSourceCreate</td>
<td>Creates a string source.</td>
</tr>
<tr>
<td>XawTextDisableRedisplay</td>
<td>Disables redisplay while making several changes to a Text Widget.</td>
</tr>
<tr>
<td>XawTextDisplay</td>
<td>Displays batched updates.</td>
</tr>
<tr>
<td>XawTextDisplayCaret</td>
<td>Enables and disables the insert point.</td>
</tr>
<tr>
<td>XawTextEnableRedisplay</td>
<td>Enables redisplay.</td>
</tr>
<tr>
<td>XawTextGetInsertionPoint</td>
<td>Returns the current position of the insert point.</td>
</tr>
<tr>
<td>XawTextGetSelectionPos</td>
<td>Retrieves the text that has been selected by this text widget.</td>
</tr>
<tr>
<td>XawTextGetSource</td>
<td>Retrieves the current text source for the specified widget.</td>
</tr>
<tr>
<td>XawTextInvalidate</td>
<td>Redisplays a range of characters.</td>
</tr>
<tr>
<td>XawTextReplace</td>
<td>Modifies the text in an editable Text widget.</td>
</tr>
<tr>
<td>XawTextSearch</td>
<td>Searches for a string in a Text widget.</td>
</tr>
<tr>
<td>XawTextSetInsertionPoint</td>
<td>Moves the insert point to the specified source position.</td>
</tr>
<tr>
<td>XawTextSetLastPos</td>
<td>Sets the last position data in an AsciiSource Object.</td>
</tr>
<tr>
<td>XawTextSetSelection</td>
<td>Selects a piece of text.</td>
</tr>
<tr>
<td>XawTextSetSelectionArray</td>
<td>Assigns a new selection array to a text widget.</td>
</tr>
<tr>
<td>XawTextSetSource</td>
<td>Replaces the text source in the specified widget.</td>
</tr>
<tr>
<td>XawTextSinkClearToBackground</td>
<td>Clears a region of the sink to the background color.</td>
</tr>
<tr>
<td>XawTextSinkDisplayText</td>
<td>Stub function that in subclasses will display text.</td>
</tr>
<tr>
<td>XawTextSinkFindDistance</td>
<td>Finds the Pixel Distance between two text positions.</td>
</tr>
<tr>
<td>XawTextSinkFindPosition</td>
<td>Finds a position in the text.</td>
</tr>
<tr>
<td>XawTextSinkGetCursorBounds</td>
<td>Finds the bounding box for the insert cursor.</td>
</tr>
<tr>
<td>XawTextSinkInsertCursor</td>
<td>Places the InsertCursor.</td>
</tr>
<tr>
<td>XawTextSinkMaxHeight</td>
<td>Finds the minimum height that contains a given number of lines.</td>
</tr>
<tr>
<td>XawTextSinkMaxLines</td>
<td>Finds the maximum number of lines that fit in a given height.</td>
</tr>
<tr>
<td>XawTextSinkResolve</td>
<td>Resolves a location to a position.</td>
</tr>
<tr>
<td>XawTextSinkSetTabs</td>
<td>Sets the Tab stops.</td>
</tr>
<tr>
<td>XawTextSourceConvertSelection</td>
<td>Dummy selection converter.</td>
</tr>
<tr>
<td>XawTextSourceRead</td>
<td>Reads the source into a buffer.</td>
</tr>
<tr>
<td>XawTextSourceReplace</td>
<td>Replaces a block of text with new text.</td>
</tr>
<tr>
<td>XawTextSourceScan</td>
<td>Scans the text source for the number and type of item specified.</td>
</tr>
<tr>
<td>XawTextSourceSearch</td>
<td>Searches the text source for the text block passed.</td>
</tr>
<tr>
<td>XawTextSourceSetSelection</td>
<td>Allows special setting of the selection.</td>
</tr>
</tbody>
</table>
Some of the header files have been renamed from their original distribution names because of the data set naming conventions in the MVS environment. In addition, some of the header file names were changed to eliminate duplicate data set names with the Motif-based widget support. If your application uses these header files, you must use the new header file names in Table 168. These data set members can be found in the SEZACMAC partitioned data set. They carry an H extension in this text to distinguish them as header files.

Table 168. Athena header file names

<table>
<thead>
<tr>
<th>MIT distribution name</th>
<th>TCP/IP name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsciiSinkP.h</td>
<td>AscSinkP.h</td>
</tr>
<tr>
<td>AsciiSrcP.h</td>
<td>AscSrcP.h</td>
</tr>
<tr>
<td>AsciiTextP.h</td>
<td>AscTextP.h</td>
</tr>
<tr>
<td>Command.h</td>
<td>ACommand.h</td>
</tr>
<tr>
<td>CommandP.h</td>
<td>ACommandP.h</td>
</tr>
<tr>
<td>Form.h</td>
<td>AForm.h</td>
</tr>
<tr>
<td>FormP.h</td>
<td>AFormP.h</td>
</tr>
<tr>
<td>Label.h</td>
<td>ALabel.h</td>
</tr>
<tr>
<td>LabelP.h</td>
<td>ALabelP.h</td>
</tr>
<tr>
<td>List.h</td>
<td>AList.h</td>
</tr>
<tr>
<td>ListP.h</td>
<td>AListP.h</td>
</tr>
<tr>
<td>MenuButtoP.h</td>
<td>MenuButP.h</td>
</tr>
<tr>
<td>Scrollbar.h</td>
<td>AScrollbar.h</td>
</tr>
<tr>
<td>ScrollbarP.h</td>
<td>AScrollP.h</td>
</tr>
<tr>
<td>SimpleMenP.h</td>
<td>SimpleMP.h</td>
</tr>
<tr>
<td>StripCharP.h</td>
<td>StripChP.h</td>
</tr>
<tr>
<td>TemplateP.h</td>
<td>TemplatP.h</td>
</tr>
<tr>
<td>Text.h</td>
<td>AText.h</td>
</tr>
<tr>
<td>TextP.h</td>
<td>ATextP.h</td>
</tr>
<tr>
<td>TextSinkP.h</td>
<td>TextSinkP.h</td>
</tr>
<tr>
<td>TextSrcP.h</td>
<td>ATextSrP.h</td>
</tr>
<tr>
<td>ViewportP.h</td>
<td>ViewporP.h</td>
</tr>
</tbody>
</table>
X Window System routines: Motif-based widget support

The X Window System support with TCP/IP includes the Motif-based widget set (Release 1.1).

The Motif-based widget set supports the following gadgets and widgets:

- ArrowButton, ArrowGadget, and ArrowButtonGadget
- BulletinBoard
- CascadeButton and CascadeButtonGadget
- Command
- DialogShell
- DrawingArea
- DrawnButton
- Form
- Frame
- Label and LabelGadget
- List
- MainWindow
- MenuShell
- MessageBox
- PanedWindow
- PushButton and PushButtonGadget
- RowColumn
- Sash
- Scale
- ScrollBar
- ScrolledWindow
- SelectionBox and SelectionDialog
- Separator and SeparatorGadget
- Text
- ToggleButton and ToggleButtonGadget

FileSelectionBox and FileSelectionDialog widgets are not supported in TCP/IP Version 3 Release 2 for MVS.

To run a Motif-based application, you must copy the module SEZAINST(KEYSYMDB) to hlq.XKEYSYM.DB or user_id.XKEYSYM.DB to make it available to your application at run-time.

Note: The EZAGETIN job copies SEZAINST(KEYSYMDB) to hlq.XKEYSYM.DB.

Some of the header files have been renamed from their original distribution names because of the data set naming conventions in the MVS environment. Such name changes are generally restricted to those header files used internally by the actual widget code, rather than the application header data sets, to minimize the number of changes required for an application to be ported to the MVS environment.

When porting applications to the MVS environment, you might have to make changes to the header file names in Table 169. These data set members can be found in the SEZACMAC partitioned data set. They carry an H extension in this text to distinguish them as header files.

<table>
<thead>
<tr>
<th>Motif distribution name</th>
<th>TCP/IP name</th>
</tr>
</thead>
<tbody>
<tr>
<td>BulletinBP.h</td>
<td>BulletBP.h</td>
</tr>
<tr>
<td>CascadeBG.h</td>
<td>CascadBG.h</td>
</tr>
<tr>
<td>CascadeBGP.h</td>
<td>CascaBGP.h</td>
</tr>
<tr>
<td>CascadeBP.h</td>
<td>CascadBP.h</td>
</tr>
<tr>
<td>CutPasteP.h</td>
<td>CutPastP.h</td>
</tr>
<tr>
<td>DrawingAP.h</td>
<td>DrawinAP.h</td>
</tr>
<tr>
<td>ExtObjectP.h</td>
<td>ExtObjeP.h</td>
</tr>
<tr>
<td>MenuShellP.h</td>
<td>MenuSheP.h</td>
</tr>
<tr>
<td>MessageBP.h</td>
<td>MessagBP.h</td>
</tr>
</tbody>
</table>
Table 169. Motif header file names (continued)

<table>
<thead>
<tr>
<th>Motif distribution name</th>
<th>TCP/IP name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProtocolsP.h</td>
<td>ProtocoP.h</td>
</tr>
<tr>
<td>RowColumnP.h</td>
<td>RowColuP.h</td>
</tr>
<tr>
<td>ScrollBarP.h</td>
<td>ScrollBP.h</td>
</tr>
<tr>
<td>ScrolledWP.h</td>
<td>ScrollWP.h</td>
</tr>
<tr>
<td>SelectioB.h</td>
<td>SelectB.h</td>
</tr>
<tr>
<td>SelectioBP.h</td>
<td>SelectBP.h</td>
</tr>
<tr>
<td>SeparatoG.h</td>
<td>SeperatG.h</td>
</tr>
<tr>
<td>SeparatoGP.h</td>
<td>SeparatGP.h</td>
</tr>
<tr>
<td>SeparatorP.h</td>
<td>SeparatP.h</td>
</tr>
<tr>
<td>ToggleBGP.h</td>
<td>TogglBGP.h</td>
</tr>
<tr>
<td>TraversalI.h</td>
<td>TraversI.h</td>
</tr>
<tr>
<td>VirtKeysP.h</td>
<td>VirtKeyP.h</td>
</tr>
</tbody>
</table>

X Window System routines: z/OS UNIX System Services support

The following topics provide information about using z/OS UNIX System Services for the X Window System.

For information about using z/OS UNIX System Services sockets, see the z/OS XL C/C++ Run-Time Library Reference.

X Window System routines: What is provided with z/OS UNIX System Services

The z/OS UNIX System Services X Window System support provided with TCP/IP includes the following APIs:

- SEZAROE1 and SEZACMTX compiled to run under z/OS UNIX System Services. SEZAROE1 is a combination of the reentrant versions of the X Window System libraries (see the z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference for information about data sets).
- .SEZAROE2 (z/OS UNIX System Services Athena Widget set for reentrant modules).
- SEZAROE3 (z/OS UNIX System Services Motif Widget set for reentrant modules).

The SEZAROE1, SEZAROE2, and SEZAROE3 library members are:

- Fixed block 80, in object deck format.
- Compiled with the C/370 RENT compile-time option.
- Used as input for reentrant z/OS UNIX System Services X Window System and socket programs.
- Passed to the C/370 prelinker. Use the prelink utility to combine all input text decks into a single text deck.

X Window System routines: z/OS UNIX System Services software requirements

Application programs using the X Window System API in z/OS UNIX System Services are written in C and should be compiled, linked, and executed using the
You must have the AD/Cycle C/370 Library V1R2M0 and the AD/Cycle LE/370 Library V1R3M0 available when you compile and link your program.

You must include MANIFEST.H as the first #include statement in the source of every z/OS UNIX System Services MVS X Window System application program to remap the socket functions to the correct run-time library names.

In z/OS UNIX System Services, the DISPLAY environment variable is used by the X Window System to identify the host name of the target display.

**X Window System routines: z/OS UNIX System Services**

**application resource file**

The X Window System allows you to modify certain characteristics of an application at run time by means of application resources. Typically, application resources are set to tailor the appearance and possibly the behavior of an application. The application resources can specify information about application window sizes, placement, coloring, font usage, and other functional details.

In the z/OS UNIX System Services environment, this information can be found in the file:

```
/u/user_id/.Xdefaults
```

where:

```
/u/user_id
```

is found from the environment variable *home*.

**Identifying the target display in z/OS UNIX System Services**

The DISPLAY environment variable is used by the X Window System to identify the host name of the target display.

The following is the format of the DISPLAY environment variable:

```
host_name:target_server.target_screen
```

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host_name</td>
<td>Specifies the host name or IP address of the host machine on which the X Window System server is running.</td>
</tr>
<tr>
<td>target_server</td>
<td>Specifies the number of the display server on the host machine.</td>
</tr>
<tr>
<td>target_screen</td>
<td>Specifies the screen to be used on the target server.</td>
</tr>
</tbody>
</table>

For more information about resolving a host name to an IP address, see the [z/OS XL C/C++ Run-Time Library Reference](https://www.ibm.com/support/knowledgecenter/en/SSLVMBTM_11.1.0/com.ibm.zos.v11r4.doc/c/ke_dclstdx.html).

**Compiling and linking with z/OS UNIX System Services**

The following steps describe how to compile, link-edit, and run your z/OS UNIX System Services X Window System application under MVS batch, using the EDCCLG cataloged procedure supplied by IBM.
You must make the following changes to the EDCCLG cataloged procedure, which is supplied with AD/Cycle C/370 Version 1 Release 2 Compiler Licensed Program (5688-216).

In the compile step, make the following changes:

- Change the CPARM parameters to specify one of the following:
  - `CPARM='DEF(IBMCPP),RENT,LO'`
  - `CPARM='DEF(IBMCPP,_POSIX1_SOURCE=1),RENT,LO'`
  - `CPARM='DEF(IBMCPP,_OPEN_SYS),RENT,LO'`
  - `CPARM='DEF(IBMCPP,_OPEN_SOCKETS,_POSIX1_SOURCE=1),RENT,LO'`
  - `CPARM='DEF(IBMCPP,_OPEN_SOCKETS,_OPEN_SYS),RENT,LO'`

**Note:** The recommended CPARMS are:

`CPARM='DEF(IBMCPP,_OPEN_SOCKETS,_POSIX1_SOURCE=1),RENT,LO'`

RENT is the reentrant option and LO is the long-name option. You must specify these options to use z/OS UNIX System Services MVS functions. You must also specify the feature text macro, IBMCPP.

If you choose to just access the z/OS UNIX System Services MVS functions defined by the POSIX standards 1003.1, 1003.1a, 1003.2, and 1003.4a, then specify the feature test macro POSIX1_SOURCE=1 to expose the appropriate definitions for the read(), write(), fcntl(), and close() functions.

If you choose to access all of the z/OS UNIX System Services MVS functions supported by C/370, including those defined by the POSIX standards 1003.1, 1003.1a, 1003.2, and 1003.4a, then specify the feature test macro _OPEN_SYS.

If you choose to access the z/OS UNIX System Services MVS socket functions or errno values, then specify the feature test macro _OPEN_SOCKETS to expose the socket-related definitions in all of the include files.

**Note:** Compile all C source using the `def(IBMCPP)` preprocessor symbol. See "X Window System interface in the MVS environment: Compiling and linking" on page 691 for information about compiling and linking your program in MVS.

For a complete discussion of all of the AD/Cycle C/370 parameters, see the AD/Cycle C/370 Programming Guide.

- Add the following lines after the //SYSLIB DD statement for the IBM C/370 library edc.v1r2m0.SEDCDHDR:
  ```
  // DD DSN=sys1.SFOMHDRS,DISP=SHR
  // DD DSN=SEZACMAC,DISP=SHR
  ```

- Add the following //USERLIB DD statement:
  ```
  //USERLIB DD DSN=USER.MYPROG.H,DISP=SHR
  ```

In the prelink edit step, make the following changes:

- Add the following prelink parameter:
  ```
  PPARAM='OMVS'
  ```

- To link-edit programs that use only X11 library functions, add the following line after the prelink //SYSLIB DD statement for the IBM AD/Cycle C/370 library cee.v1r3m0.SCEEEOBJ:
  ```
  // DD DSN=SEZAROE1,DISP=SHR
  ```

- To link-edit programs that use the Athena Toolkit functions, including Athena widget sets, add the following lines after the prelink //SYSLIB DD statement for the IBM AD/Cycle C/370 library cee.v1r3m0.SCEEEOBJ:
To link-edit programs that use the Motif Toolkit functions, add the following lines after the prelink //SYSLIB DD statement for the IBM AD/Cycle C/370 library cee.v1r3m0.SCEEOBJ:

```
// DD DSN=SEZAROE3,DISP=SHR
// DD DSN=SEZAROE1,DISP=SHR
```

For a complete discussion about compiling and link-editing the X Window System in MVS z/OS UNIX System Services, see the [z/OS XL C/C++ Run-Time Library Reference](#).

To execute your program in the z/OS UNIX System Services shell, make the following changes:

- Set the DISPLAY environment variable to the name or IP address of the X server on which you want to display the application output. The following is an example:

  ```
  DISPLAY=CHARM.RALEIGH.IBM.COM:0.0
  export DISPLAY
  ```

- Allow the host application access to the X server.

  On the workstation where you want to display the application output, you must grant permission for the MVS host to access the X server. To do this, enter the xhost command:

  ```
  xhost ralmvs1
  ```

## Compiling and linking with z/OS UNIX System Services using c89

The following c89 utility options can be specified:

- IBMCC must always be specified.
- The c89 utility assumes _OPEN_SYS and includes all of the z/OS UNIX System Services MVS functions supported by C/370. However, _OPEN_SOCKETS must be specified if z/OS UNIX System Services MVS sockets are being used by the application program.

```
-D IBMCC
-D _OPEN_SOCKETS
```

### Notes:

1. When you compile and link-edit your application program using the c89 utility with z/OS UNIX System Services sockets and TCP/IP Version 3 Release 1 for X Window System, you must include the z/OS UNIX System Services socket library before the X Window System include files:

   ```
   -1"/'sys1.SFOMHDRS'
   -1"/'SEZACMAC'
   -1"/'SEZAROE1'
   ```

2. The flag for the prelinker libraries, -l, is a dash followed by the lowercase letter L.

- If the Athena Toolkit functions are required, then also specify:
  ```
  -1"/'SEZAROE2'
  ```
- If the Motif Toolkit functions are required, then also specify:
  ```
  -1"/'SEZAROE3'
  ```

To execute your program under TSO, enter the following:
CALL 'USER.MYP prog.LOAD(PROGRAM1)' 'POSIX(ON)'

This loads the run-time library from cee.v1r3m0.SCEERUN. To use the z/OS UNIX System Services MVS C/370 functions, you must either specify the run-time option:

POSIX(ON)

or include the following statement in your C source program:

#pragma runopts(POSIX(ON))

Standard X client applications for z/OS UNIX System Services

For information about standard X Client applications for X Window System on z/OS UNIX System Services, see "Standard X client applications" on page 697.

Application resources for z/OS UNIX System Services

X applications can be modified at run time by a set of resources. Applications that make use of an X Window System toolkit can be modified by additional sets of application resources. These resources are searched until a resource specification is found. The X Intrinsics determine the actual search order used for determining a resource value.

The search order used in the z/OS UNIX System Services environment, in descending order of preference, is:

1. Command Line
   Standard arguments include:
   a. Command switches (-display, -fg, -foreground, +rv)
   b. Resource manager directives (-name, -xrm)
   c. Natural language directives (-xnllanguage)

2. User Environment File
   Use the source found from the file /u/user_id/.Xdefault-host.
   /u/user_id/.Xdefault-host is found from the environment variable home, and host is the string returned by the gethostname() call.

3. Server and User Preference Resources
   Use the first source found from:
   a. RESOURCE_MANAGER property on the root window [screen0]
   b. /u/user_id/.Xdefaults
      /u/user_id is found from the environment variable home.

4. Application Class Resources
   Use the first source found from:
   a. The default application resource file
      /u/user_id/classname
      where classname is the application specified class name, and /u/user_id is found from the environment variable home.
   b. Fallback resources defined in the file /usr/lib/X11/app-defaults/
      classname where classname is the application-specified class name.
Appendix E. 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.
Attention: Network Information Center
14200 Park Meadow Drive
Suite 200
Chantilly, VA 22021

Hard copies of all RFCs are available from the NIC, either individually or by subscription. Online copies are available at the following Web address:

http://www.rfc-editor.org/rfc.html

See "Internet drafts" on page 758 for draft RFCs implemented in this and previous Communications Server releases.

Many features of TCP/IP Services are based on the following RFCs:

<table>
<thead>
<tr>
<th>RFC</th>
<th>Title and Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFC 652</td>
<td>Telnet output carriage-return disposition option D. Crocker</td>
</tr>
<tr>
<td>RFC 653</td>
<td>Telnet output horizontal tabstops option D. Crocker</td>
</tr>
<tr>
<td>RFC 654</td>
<td>Telnet output horizontal tab disposition option D. Crocker</td>
</tr>
<tr>
<td>RFC 655</td>
<td>Telnet output formfeed disposition option D. Crocker</td>
</tr>
<tr>
<td>RFC 657</td>
<td>Telnet output vertical tab disposition option D. Crocker</td>
</tr>
<tr>
<td>RFC 658</td>
<td>Telnet output linefeed disposition D. Crocker</td>
</tr>
<tr>
<td>RFC 698</td>
<td>Telnet extended ASCII option T. Mock</td>
</tr>
<tr>
<td>RFC 726</td>
<td>Remote Controlled Transmission and Echoing Telnet option J. Postel, D. Crocker</td>
</tr>
<tr>
<td>RFC 727</td>
<td>Telnet logout option M.R. Crispin</td>
</tr>
<tr>
<td>RFC 732</td>
<td>Telnet Data Entry Terminal option J.D. Day</td>
</tr>
<tr>
<td>RFC 733</td>
<td>Standard for the format of ARPA network text messages D. Crocker, J. Vittal, K.T. Pogran, D.A. Henderson</td>
</tr>
</tbody>
</table>
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 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
Appendix E. Related protocol specifications
RFC 1044  Internet Protocol on Network System’s HYPERchannel: Protocol specification  K. Hardwick, J. Lekashman
RFC 1053  Telnet X.3 PAD option S. Levy, T. Jacobson
RFC 1055  Nonstandard for transmission of IP datagrams over serial lines: SLIP J. Romkey
RFC 1058  Routing Information Protocol C. Hedrick
RFC 1060  Assigned numbers J. Reynolds, J. Postel
RFC 1071  Computing the Internet checksum R.T. Braden, D.A. Borman, C. Partridge
RFC 1072  TCP extensions for long-delay paths V. Jacobson, R.T. Braden
RFC 1073  Telnet window size option D. Waitzman
RFC 1079  Telnet terminal speed option C. Hedrick
RFC 1085  ISO presentation services on top of TCP/IP based internets M.T. Rose
RFC 1091  Telnet terminal-type option J. VanBokkelen
RFC 1094  NFS: Network File System Protocol specification Sun Microsystems
RFC 1096  Telnet X display location option G. Marcy
RFC 1101  DNS encoding of network names and other types P. Mockapetris
RFC 1112  Host extensions for IP multicasting S.E. Deering
RFC 1113  Privacy enhancement for Internet electronic mail: Part I — message encipherment and authentication procedures J. Linn
RFC 1118  Hitchhiker’s Guide to the Internet E. Krol
RFC 1122  Requirements for Internet Hosts—Communication Layers R. Braden, Ed.
RFC 1123  Requirements for Internet Hosts—Application and Support R. Braden, Ed.
RFC 1146  TCP alternate checksum options J. Zweig, C. Partridge
RFC 1155  Structure and identification of management information for TCP/IP-based internets M. Rose, K. McCloghrie
RFC 1156  Management Information Base for network management of TCP/IP-based internets K. McCloghrie, M. Rose
RFC 1158  Management Information Base for network management of TCP/IP-based internets: MIB-II M. Rose
RFC 1166  Internet numbers S. Kirkpatrick, M.K. Stahl, M. Recker
RFC 1179  Line printer daemon protocol L. McLaughlin
RFC 1180  TCP/IP tutorial T. Socolofsky, C. Kale
<table>
<thead>
<tr>
<th>RFC</th>
<th>Title</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1183</td>
<td>New DNS RR Definitions</td>
<td>C.F. Everhart, L.A. Mamakos, R. Ullmann, P.V. Mockapetris</td>
</tr>
<tr>
<td>1184</td>
<td>Telnet Linemode Option</td>
<td>D. Borman</td>
</tr>
<tr>
<td>1186</td>
<td>MD4 Message Digest Algorithm</td>
<td>R.L. Rivest</td>
</tr>
<tr>
<td>1187</td>
<td>Bulk Table Retrieval with the SNMP</td>
<td>M. Rose, K. McCloghrie, J. Davin</td>
</tr>
<tr>
<td>1188</td>
<td>Proposed Standard for the Transmission of IP Datagrams over FDDI Networks</td>
<td>D. Katz</td>
</tr>
<tr>
<td>1190</td>
<td>Experimental Internet Stream Protocol: Version 2 (ST-II)</td>
<td>C. Topolcic</td>
</tr>
<tr>
<td>1191</td>
<td>Path MTU discovery</td>
<td>J. Mogul, S. Deering</td>
</tr>
<tr>
<td>1198</td>
<td>FYI on the X window system</td>
<td>R. Scheifler</td>
</tr>
<tr>
<td>1207</td>
<td>FYI on Questions and Answers: Answers to commonly asked “experienced Internet user” questions</td>
<td>G. Malkin, A. Marine, J. Reynolds</td>
</tr>
<tr>
<td>1208</td>
<td>Glossary of networking terms</td>
<td>O. Jacobsen, D. Lynch</td>
</tr>
<tr>
<td>1213</td>
<td>Management Information Base for Network Management of TCP/IP-based internets: MIB-II</td>
<td>K. McCloghrie, M.T. Rose</td>
</tr>
<tr>
<td>1215</td>
<td>Convention for defining traps for use with the SNMP</td>
<td>M. Rose</td>
</tr>
<tr>
<td>1227</td>
<td>SNMP MUX protocol and MIB</td>
<td>M.T. Rose</td>
</tr>
<tr>
<td>1228</td>
<td>SNMP-DPI: Simple Network Management Protocol Distributed Program Interface</td>
<td>G. Carpenter, B. Wijnen</td>
</tr>
<tr>
<td>1229</td>
<td>Extensions to the generic-interface MIB</td>
<td>K. McCloghrie</td>
</tr>
<tr>
<td>1230</td>
<td>IEEE 802.4 Token Bus MIB</td>
<td>K. McCloghrie, R. Fox</td>
</tr>
<tr>
<td>1231</td>
<td>IEEE 802.5 Token Ring MIB</td>
<td>K. McCloghrie, R. Fox, E. Decker</td>
</tr>
<tr>
<td>1236</td>
<td>IP to X.121 address mapping for DDN</td>
<td>L. Morales, P. Hasse</td>
</tr>
<tr>
<td>1256</td>
<td>ICMP Router Discovery Messages</td>
<td>S. Deering, Ed.</td>
</tr>
<tr>
<td>1267</td>
<td>Border Gateway Protocol 3 (BGP-3)</td>
<td>K. Lougheed, Y. Rekhter</td>
</tr>
<tr>
<td>1268</td>
<td>Application of the Border Gateway Protocol in the Internet</td>
<td>Y. Rekhter, P. Gross</td>
</tr>
<tr>
<td>1269</td>
<td>Definitions of Managed Objects for the Border Gateway Protocol: Version 3</td>
<td>S. Willis, J. Burruss</td>
</tr>
<tr>
<td>1270</td>
<td>SNMP Communications Services</td>
<td>F. Kastenholz, ed.</td>
</tr>
<tr>
<td>1285</td>
<td>FDDI Management Information Base</td>
<td>J. Case</td>
</tr>
<tr>
<td>1315</td>
<td>Management Information Base for Frame Relay DTEs</td>
<td>C. Brown, F. Baker, C. Carvalho</td>
</tr>
<tr>
<td>1321</td>
<td>The MD5 Message-Digest Algorithm</td>
<td>R. Rivest</td>
</tr>
<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 &quot;New Internet User&quot; 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>
</tbody>
</table>

Appendix E. Related protocol specifications 747
RFC 1340  Assigned Numbers J. Reynolds, J. Postel
RFC 1344  Implications of MIME for Internet Mail Gateways N. Bornstein
RFC 1349  Type of Service in the Internet Protocol Suite P. Almquist
RFC 1350  The TFTP Protocol (Revision 2) K.R. Sollins
RFC 1351  SNMP Administrative Model J. Davin, J. Galvin, K. McCloghrie
RFC 1352  SNMP Security Protocols J. Galvin, K. McCloghrie, J. Davin
RFC 1353  Definitions of Managed Objects for Administration of SNMP Parties K. McCloghrie, J. Davin, J. Galvin
RFC 1354  IP Forwarding Table MIB F. Baker
RFC 1356  Multiprotocol Interconnect on X.25 and ISDN in the Packet Mode A. Malis, D. Robinson, R. Ullmann
RFC 1358  Charter of the Internet Architecture Board (IAB) L. Chapin
RFC 1363  A Proposed Flow Specification C. Partridge
RFC 1368  Definition of Managed Objects for IEEE 802.3 Repeater Devices D. McMaster, K. McCloghrie
RFC 1372  Telnet Remote Flow Control Option C. L. Hedrick, D. Borman
RFC 1374  IP and ARP on HIPPI J. Renwick, A. Nicholson
RFC 1381  SNMP MIB Extension for X.25 LAPB D. Throop, F. Baker
RFC 1382  SNMP MIB Extension for the X.25 Packet Layer D. Throop
RFC 1387  RIP Version 2 Protocol Analysis G. Malkin
RFC 1388  RIP Version 2 Carrying Additional Information G. Malkin
RFC 1389  RIP Version 2 MIB Extensions G. Malkin, F. Baker
RFC 1390  Transmission of IP and ARP over FDDI Networks D. Katz
RFC 1393  Traceroute Using an IP Option G. Malkin
RFC 1398  Definitions of Managed Objects for the Ethernet-Like Interface Types F. Kastenholz
RFC 1408  Telnet Environment Option D. Borman, Ed.
RFC 1413  Identification Protocol M. St. Johns
RFC 1416  Telnet Authentication Option D. Borman, ed.
RFC 1420  SNMP over IPX S. Bostock
RFC 1428  Transition of Internet Mail from Just-Send-8 to 8bit-SMTP/MIME G. Vaudreuil
RFC 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 1464  Using the Domain Name System to Store Arbitrary String Attributes R. Rosenbaum

RFC 1469  IP Multicast over Token-Ring Local Area Networks T. Pusateri

RFC 1483  Multiprotocol Encapsulation over ATM Adaptation Layer 5 Juha Heinanen

RFC 1514  Host Resources MIB P. Grillo, S. Waldbusser

RFC 1516  Definitions of Managed Objects for IEEE 802.3 Repeater Devices D. McMaster, K. McCloghrie

RFC 1521  MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies N. Borenstein, N. Freed

RFC 1535  A Security Problem and Proposed Correction With Widely Deployed DNS Software E. Gavron

RFC 1536  Common DNS Implementation Errors and Suggested Fixes A. Kumar, J. Postel, C. Neuman, P. Danzig, S. Miller

RFC 1537  Common DNS Data File Configuration Errors P. Beertema

RFC 1540  Internet Official Protocol Standards J. Postel

RFC 1571  Telnet Environment Option Interoperability Issues D. Borman

RFC 1572  Telnet Environment Option S. Alexander

RFC 1573  Evolution of the Interfaces Group of MIB-II K. McCloghrie, F. Kastenholz

RFC 1577  Classical IP and ARP over ATM M. Laubach

RFC 1583  OSPF Version 2 J. Moy

RFC 1591  Domain Name System Structure and Delegation J. Postel


RFC 1594  FYI on Questions and Answers—Answers to Commonly Asked "New Internet User" Questions A. Marine, J. Reynolds, G. Malkin

RFC 1644  T/TCP — TCP Extensions for Transactions Functional Specification R. Braden

RFC 1646  TN3270 Extensions for LUname and Printer Selection C. Graves, T. Butts, M. Angel

RFC 1647  TN3270 Enhancements B. Kelly

RFC 1652  SMTP Service Extension for 8bit-MIMEtransport J. Klensin, N. Freed, M. Rose, E. Stefferrud, 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

Appendix E. Related protocol specifications  749
<table>
<thead>
<tr>
<th>RFC 1701</th>
<th>Generic Routing Encapsulation (GRE) S. Hanks, T. Li, D. Farinacci, P. Traina</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFC 1702</td>
<td>Generic Routing Encapsulation over IPv4 networks S. Hanks, T. Li, D. Farinacci, P. Traina</td>
</tr>
<tr>
<td>RFC 1706</td>
<td>DNS NSAP Resource Records B. Manning, R. Colella</td>
</tr>
<tr>
<td>RFC 1712</td>
<td>DNS Encoding of Geographical Location C. Farrell, M. Schulze, S. Pleitner D. Baldoni</td>
</tr>
<tr>
<td>RFC 1713</td>
<td>Tools for DNS debugging A. Romao</td>
</tr>
<tr>
<td>RFC 1723</td>
<td>RIP Version 2—Carrying Additional Information G. Malkin</td>
</tr>
<tr>
<td>RFC 1752</td>
<td>The Recommendation for the IP Next Generation Protocol S. Bradner, A. Mankin</td>
</tr>
<tr>
<td>RFC 1766</td>
<td>Tags for the Identification of Languages H. Alvestrand</td>
</tr>
<tr>
<td>RFC 1771</td>
<td>A Border Gateway Protocol 4 (BGP-4) Y. Rekhter, T. Li</td>
</tr>
<tr>
<td>RFC 1794</td>
<td>DNS Support for Load Balancing T. Brisco</td>
</tr>
<tr>
<td>RFC 1826</td>
<td>IP Authentication Header R. Atkinson</td>
</tr>
<tr>
<td>RFC 1828</td>
<td>IP Authentication using Keyed MD5 P. Metzger, W. Simpson</td>
</tr>
<tr>
<td>RFC 1829</td>
<td>The ESP DES-CBC Transform P. Karn, P. Metzger, W. Simpson</td>
</tr>
<tr>
<td>RFC 1830</td>
<td>SMTP Service Extensions for Transmission of Large and Binary MIME Messages G. Vaudreuil</td>
</tr>
<tr>
<td>RFC 1832</td>
<td>XDR: External Data Representation Standard R. Srinivasan</td>
</tr>
<tr>
<td>RFC 1833</td>
<td>Binding Protocols for ONC RPC Version 2 R. Srinivasan</td>
</tr>
<tr>
<td>RFC 1850</td>
<td>OSPF Version 2 Management Information Base F. Baker, R. Coltun</td>
</tr>
<tr>
<td>RFC 1854</td>
<td>SMTP Service Extension for Command Pipelining N. Freed</td>
</tr>
<tr>
<td>RFC 1869</td>
<td>SMTP Service Extensions J. Klensin, N. Freed, M. Rose, E. Stefferud, D. Crocker</td>
</tr>
<tr>
<td>RFC 1870</td>
<td>SMTP Service Extension for Message Size Declaration J. Klensin, N. Freed, K. Moore</td>
</tr>
<tr>
<td>RFC 1876</td>
<td>A Means for Expressing Location Information in the Domain Name System C. Davis, P. Vixie, T. Goodwin, I. Dickinson</td>
</tr>
<tr>
<td>RFC 1883</td>
<td>Internet Protocol, Version 6 (IPv6) Specification S. Deering, R. Hinden</td>
</tr>
<tr>
<td>RFC 1886</td>
<td>DNS Extensions to support IP version 6 S. Thomson, C. Huitema</td>
</tr>
<tr>
<td>RFC 1888</td>
<td>OSI NSAPs and IPv6 J. Bound, B. Carpenter, D. Harrington, J. Houldsworth, A. Lloyd</td>
</tr>
<tr>
<td>RFC 1891</td>
<td>SMTP Service Extension for Delivery Status Notifications K. Moore</td>
</tr>
<tr>
<td>RFC 1892</td>
<td>The Multipart/Report Content Type for the Reporting of Mail System Administrative Messages G. Vaudreuil</td>
</tr>
<tr>
<td>RFC</td>
<td>Title</td>
</tr>
<tr>
<td>-------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>1894</td>
<td>An Extensible Message Format for Delivery Status Notifications</td>
</tr>
<tr>
<td>1901</td>
<td>Introduction to Community-based SNMPv2</td>
</tr>
<tr>
<td></td>
<td>Network Management Protocol (SNMPv2)</td>
</tr>
<tr>
<td></td>
<td>Protocol (SNMPv2)</td>
</tr>
<tr>
<td></td>
<td>Protocol (SNMPv2)</td>
</tr>
<tr>
<td></td>
<td>Protocol (SNMPv2)</td>
</tr>
<tr>
<td></td>
<td>Protocol (SNMPv2)</td>
</tr>
<tr>
<td></td>
<td>Management Protocol (SNMPv2)</td>
</tr>
<tr>
<td>1908</td>
<td>Coexistence between Version 1 and Version 2 of the Internet-standard</td>
</tr>
<tr>
<td></td>
<td>Network Management Framework</td>
</tr>
<tr>
<td>1912</td>
<td>Common DNS Operational and Configuration Errors</td>
</tr>
<tr>
<td>1918</td>
<td>Address Allocation for Private Internets</td>
</tr>
<tr>
<td>1930</td>
<td>Guidelines for creation, selection, and registration of an Autonomous System (AS)</td>
</tr>
<tr>
<td>1939</td>
<td>Post Office Protocol-Version 3</td>
</tr>
<tr>
<td>1941</td>
<td>Path MTU Discovery for IP version 6</td>
</tr>
<tr>
<td>1981</td>
<td>Serial Number Arithmetic</td>
</tr>
<tr>
<td>1985</td>
<td>SMTP Service Extension for Remote Message Queue Starting</td>
</tr>
<tr>
<td>1995</td>
<td>Incremental Zone Transfer in DNS</td>
</tr>
<tr>
<td>1996</td>
<td>A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)</td>
</tr>
<tr>
<td>2010</td>
<td>Operational Criteria for Root Name Servers</td>
</tr>
<tr>
<td></td>
<td>SMIv2</td>
</tr>
<tr>
<td>2012</td>
<td>SNMPv2 Management Information Base for the Transmission Control</td>
</tr>
<tr>
<td></td>
<td>Protocol using SMIv2</td>
</tr>
<tr>
<td></td>
<td>using SMIv2</td>
</tr>
</tbody>
</table>
\appendix
\section{Related protocol specifications}

\begin{itemize}
  \item RFC 2219 \textit{Use of DNS Aliases for Network Services} M. Hamilton, R. Wright
  \item RFC 2228 \textit{FTP Security Extensions} M. Horowitz, S. Lunt
  \item RFC 2230 \textit{Key Exchange Delegation Record for the DNS} R. Atkinson
  \item RFC 2233 \textit{The Interfaces Group MIB using SMIm2} K. McCloghrie, F. Kastenholz
  \item RFC 2240 \textit{A Legal Basis for Domain Name Allocation} O. Vaughn
  \item RFC 2246 \textit{The TLS Protocol Version 1.0} T. Dierks, C. Allen
  \item RFC 2251 \textit{Lightweight Directory Access Protocol (v3)} M. Wahl, T. Howes, S. Kille
  \item RFC 2254 \textit{The String Representation of LDAP Search Filters} T. Howes
  \item RFC 2251 \textit{Lightweight Directory Access Protocol (v3)} M. Wahl, T. Howes, S. Kille
  \item RFC 2254 \textit{The String Representation of LDAP Search Filters} T. Howes
  \item RFC 2261 \textit{An Architecture for Describing SNMP Management Frameworks} D. Harrington, R. Presuhn, B. Wijnen
  \item RFC 2262 \textit{Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)} J. Case, D. Harrington, R. Presuhn, B. Wijnen
  \item RFC 2271 \textit{An Architecture for Describing SNMP Management Frameworks} D. Harrington, R. Presuhn, B. Wijnen
  \item RFC 2273 \textit{SNMPv3 Applications} D. Levi, P. Meyer, B. Stewart
  \item RFC 2274 \textit{User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)} U. Blumenthal, B. Wijnen
  \item RFC 2275 \textit{View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)} B. Wijnen, R. Presuhn, K. McCloghrie
  \item RFC 2279 \textit{UTF-8, a transformation format of ISO 10646} F. Yergeau
  \item RFC 2292 \textit{Advanced Sockets API for IPv6} W. Stevens, M. Thomas
  \item RFC 2308 \textit{Negative Caching of DNS Queries (DNS NCACHE)} M. Andrews
  \item RFC 2317 \textit{Classless IN-ADDR.ARPA delegation} H. Eidnes, G. de Groot, P. Vixie
  \item RFC 2320 \textit{Definitions of Managed Objects for Classical IP and ARP Over ATM Using SMIm2 (IPOA-MIB)} M. Greene, J. Luciani, K. White, T. Kuo
  \item RFC 2328 \textit{OSPF Version 2} J. Moy
  \item RFC 2345 \textit{Domain Names and Company Name Retrieval} J. Klensin, T. Wolf, G. Oglesby
  \item RFC 2352 \textit{A Convention for Using Legal Names as Domain Names} O. Vaughn
  \item RFC 2355 \textit{TN3270 Enhancements} B. Kelly
  \item RFC 2358 \textit{Definitions of Managed Objects for the Ethernet-like Interface Types} J. Flick, J. Johnson
  \item RFC 2373 \textit{IP Version 6 Addressing Architecture} R. Hinden, S. Deering
  \item RFC 2374 \textit{An IPv6 Aggregatable Global Unicast Address Format} R. Hinden, M. O'Dell, S. Deering
  \item RFC 2375 \textit{IPv6 Multicast Address Assignments} R. Hinden, S. Deering
  \item RFC 2385 \textit{Protection of BGP Sessions via the TCP MD5 Signature Option} A. Hefferman
\end{itemize}
| RFC 2389 | Feature negotiation mechanism for the File Transfer Protocol P. Hethmon, R. Elz |
| RFC 2401 | Security Architecture for Internet Protocol S. Kent, R. Atkinson |
| RFC 2402 | IP Authentication Header S. Kent, R. Atkinson |
| RFC 2403 | The Use of HMAC-MD5-96 within ESP and AH C. Madson, R. Glenn |
| RFC 2404 | The Use of HMAC-SHA–1–96 within ESP and AH C. Madson, R. Glenn |
| RFC 2405 | The ESP DES-CBC Cipher Algorithm With Explicit IV C. Madson, N. Doraswamy |
| RFC 2406 | IP Encapsulating Security Payload (ESP) S. Kent, R. Atkinson |
| RFC 2407 | The Internet IP Security Domain of Interpretation for ISAKMPD. Piper |
| RFC 2409 | The Internet Key Exchange (IKE) D. Harkins, D. Carrel |
| RFC 2410 | The NULL Encryption Algorithm and Its Use With IPSec R. Glenn, S. Kent, |
| RFC 2428 | FTP Extensions for IPv6 and NATs M. Allman, S. Ostermann, C. Metz |
| RFC 2445 | Internet Calendaring and Scheduling Core Object Specification (iCalendar) F. Dawson, D. Stenerson |
| RFC 2459 | Internet X.509 Public Key Infrastructure Certificate and CRL Profile R. Housley, W. Ford, W. Polk, D. Solo |
| RFC 2460 | Internet Protocol, Version 6 (IPv6) Specification S. Deering, R. Hinden |
| RFC 2462 | IPv6 Stateless Address Autoconfiguration S. Thomson, T. Narten |
| RFC 2463 | Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification A. Conta, S. Deering |
| RFC 2464 | Transmission of IPv6 Packets over Ethernet Networks M. Crawford |
| RFC 2476 | Message Submission R. Gellens, J. Klensin |
| RFC 2487 | SMTP Service Extension for Secure SMTP over TLS P. Hoffman |
| RFC 2505 | Anti-Spam Recommendations for SMTP MTAs G. Lindberg |
| RFC 2523 | Photuris: Extended Schemes and Attributes P. Karn, W. Simpson |
| RFC 2535 | Domain Name System Security Extensions D. Eastlake 3rd |
| RFC 2538 | Storing Certificates in the Domain Name System (DNS) D. Eastlake 3rd, O. Guimandsson |
| 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 2573 | SNMP Applications | D. Levi, P. Meyer, B. Stewart |
| RFC 2575 | View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP) | B. Wijnen, R. Presuhn, K. McCloghrie |
| 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 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 |

Appendix E. Related protocol specifications 755
| RFC 2782 | A DNS RR for specifying the location of services (DNS SRV) | A. Gubrandsen, P. Vixie, L. Esibov |
| RFC 2821 | Simple Mail Transfer Protocol | J. Klensin, Ed. |
| RFC 2822 | Internet Message Format | P. Resnick, Ed. |
| RFC 2840 | TELNET KERMIT OPTION | J. Altman, F. da Cruz |
| RFC 2845 | Secret Key Transaction Authentication for DNS (TSIG) | P. Vixie, O. Gudmundsson, D. Eastlake 3rd, B. Wellington |
| RFC 2851 | Textual Conventions for Internet Network Addresses | M. Daniele, B. Haberman, S. Routhier, J. Schoenwaelder |
| RFC 2852 | Deliver By SMTP Service Extension | D. Newman |
| RFC 2874 | DNS Extensions to Support IPv6 Address Aggregation and Renumbering | M. Crawford, C. Huitema |
| RFC 2915 | The Naming Authority Pointer (NAPTR) DNS Resource Record | M. Mealling, R. Daniel |
| RFC 2920 | SMTP Service Extension for Command Pipelining | N. Freed |
| RFC 2930 | Secret Key Establishment for DNS (TKEY RR) | D. Eastlake, 3rd |
| RFC 2941 | Telnet Authentication Option | T. Ts’o, ed., J. Altman |
| RFC 2942 | Telnet Authentication: Kerberos Version 5 | T. Ts’o |
| RFC 2946 | Telnet Data Encryption Option | T. Ts’o |
| RFC 2952 | Telnet Encryption: DES 64 bit Cipher Feedback | T. Ts’o |
| RFC 2953 | Telnet Encryption: DES 64 bit Output Feedback | T. Ts’o |
| RFC 2992 | Analysis of an Equal-Cost Multi-Path Algorithm | C. Hopps |
| RFC 3019 | IP Version 6 Management Information Base for The Multicast Listener Discovery Protocol | B. Haberman, R. Wozzella |
| 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 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 3415 | View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP) B. Wijnen, R. Presuhn, K. McCloghrie |
| RFC 3419 | Textual Conventions for Transport Addresses M. Daniele, J. Schoenwaelder |
| RFC 3484 | Default Address Selection for Internet Protocol version 6 (IPv6) R. Draves |
| RFC 3513 | Internet Protocol Version 6 (IPv6) Addressing Architecture R. Hinden, S. Deering |
| RFC 3526 | More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE) T. Kivinen, M. Kojo |
| RFC 3542 | Advanced Sockets Application Programming Interface (API) for IPv6 W. Richard Stevens, M. Thomas, E. Nordmark, T. Jinmei |
| RFC 3569 | An Overview of Source-Specific Multicast (SSM) S. Bhattacharyya, Ed. |
| RFC 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 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. Schoenwaelder

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 4113  Management Information Base for the User Datagram Protocol (UDP) B. Fenner, J. Flick

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 4308  Cryptographic Suites for IPsec P. Hoffman

RFC 4552  Authentication/Confidentiality for OSPFv3 M. Gupta, N. Melam

RFC 4678  Server/Application State Protocol v1 A. Bivens

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 4941  Privacy Extensions for Stateless Address Autoconfiguration in IPv6 T. Narten, R. Draves, S. Krishnan

RFC 5095  Deprecation of Type 0 Routing Headers in IPv6 J. Abley, P. Savola, G. Neville-Neil

Internet drafts

Internet drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Other groups may also distribute working documents as Internet drafts. You can see Internet drafts at http://www.ietf.org/ID.html.

Several areas of IPv6 implementation include elements of the following Internet drafts and are subject to change during the RFC review process.
Appendix F. Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to z/OS TSO/E Primer, z/OS TSO/E User's Guide, and z/OS ISPF User's Guide Vol I for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

z/OS information

z/OS information is accessible using screen readers with the BookServer/Library Server versions of z/OS books in the Internet library at [www.ibm.com/systems/z/os/zos/bkserv/](http://www.ibm.com/systems/z/os/zos/bkserv/)
IBM may not offer all of the products, services, or features discussed in this document. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user’s responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.
Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Site Counsel  
IBM Corporation  
P.O. Box 12195  
3039 Cornwallis Road  
Research Triangle Park, North Carolina 27709-2195  
U.S.A

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM’s future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM’s suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application
programs conforming to the application programming interface for the operating
platform for which the sample programs are written. These examples have not
been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or
imply reliability, serviceability, or function of these programs. You may copy,
modify, and distribute these sample programs in any form without payment to
IBM for the purposes of developing, using, marketing, or distributing application
programs conforming to IBM’s application programming interfaces.

Each copy or any portion of these sample programs or any derivative work must
include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp.
Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights
reserved.

IBM is required to include the following statements in order to distribute portions
of this document and the software described herein to which contributions have
been made by The University of California. Portions herein © Copyright 1979,
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,

Some portions of this publication relating to X Window System** are Copyright ©
1987, 1988 by Digital Equipment Corporation, Maynard, Massachusetts, and the
Massachusetts Institute Of Technology, Cambridge, Massachusetts. All Rights
Reserved.

Some portions of this publication relating to X Window System are Copyright ©

Permission to use, copy, modify, and distribute the M.I.T., Digital Equipment
Corporation, and Hewlett-Packard Corporation portions of this software and its
documentation for any purpose without fee is hereby granted, provided that the
above copyright notice appears in all copies and that both that copyright notice
and this permission notice appear in supporting documentation, and that the
names of M.I.T., Digital, and Hewlett-Packard not be used in advertising or
publicity pertaining to distribution of the software without specific, written prior
permission. M.I.T., Digital, and Hewlett-Packard make no representation about the
suitability of this software for any purpose. It is provided "as is" without express
or implied warranty.


Copyright © 1988, 1993 The Regents of the University of California. All rights
reserved.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:
1. Redistributions of source code must retain the above copyright notice, this list
   of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this
   list of conditions and the following disclaimer in the documentation and/or
   other materials provided with the distribution.

3. All advertising materials mentioning features or use of this software must
   display the following acknowledgement:
   This product includes software developed by the University of
   California, Berkeley and its contributors.

4. Neither the name of the University nor the names of its contributors may be
   used to endorse or promote products derived from this software without
   specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS
IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF
THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.

This software program contains code, and/or derivatives or modifications of code
originating from the software program "Popper." Popper is Copyright ©1989-1991
The Regents of the University of California, All Rights Reserved. Popper was
created by Austin Shelton, Information Systems and Technology, University of
California, Berkeley.

Permission from the Regents of the University of California to use, copy, modify,
and distribute the "Popper" software contained herein for any purpose, without
fee, and without a written agreement is hereby granted, provided that the above
copyright notice and this paragraph and the following two paragraphs appear in
all copies. HOWEVER, ADDITIONAL PERMISSIONS MAY BE NECESSARY
FROM OTHER PERSONS OR ENTITIES, TO USE DERIVATIVES OR
MODIFICATIONS OF POPPER.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY
PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THE
POPPER SOFTWARE, OR ITS DERIVATIVES OR MODIFICATIONS, AND ITS
DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE. THE POPPER SOFTWARE PROVIDED HEREUNDER IS ON AN "AS
IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATIONS
TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR
MODIFICATIONS.

Copyright © 1983 The Regents of the University of California. All rights reserved.
Redistribution and use in source and binary forms are permitted provided that the above copyright notice and this paragraph are duplicated in all such forms and that any documentation, advertising materials, and other materials related to such distribution and use acknowledge that the software was developed by the University of California, Berkeley. The name of the University may not be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Copyright © 1991, 1993 The Regents of the University of California. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. All advertising materials mentioning features or use of this software must display the following acknowledgement:
   This product includes software developed by the University of California, Berkeley and its contributors.

4. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright © 1990 by the Massachusetts Institute of Technology

Export of this software from the United States of America may require a specific license from the United States Government. It is the responsibility of any person or organization contemplating export to obtain such a license before exporting.

WITHIN THAT CONSTRAINT, permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of M.I.T. not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. Furthermore
if you modify this software you must label your software as modified software and
not distribute it in such a fashion that it might be confused with the original M.I.T.
software. M.I.T. makes no representations about the suitability of this software for
any purpose. It is provided "as is" without express or implied warranty.

Copyright © 1998 by the FundsXpress, INC. All rights reserved.

Export of this software from the United States of America may require a specific
license from the United States Government. It is the responsibility of any person or
organization contemplating export to obtain such a license before exporting.

WITHIN THAT CONSTRAINT, permission to use, copy, modify, and distribute this
software and its documentation for any purpose and without fee is hereby granted,
provided that the above copyright notice appear in all copies and that both that
copyright notice and this permission notice appear in supporting documentation,
and that the name of FundsXpress not be used in advertising or publicity
pertaining to distribution of the software without specific, written prior
permission. FundsXpress makes no representations about the suitability of this
software for any purpose. It is provided "as is" without express or implied
warranty.

THIS SOFTWARE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE.


Permission to use, copy, modify, and distribute this software for any purpose with
or without fee is hereby granted, provided that the above copyright notice and this
permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND INTERNET SOFTWARE
CONSORTIUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS
SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS. IN NO EVENT SHALL INTERNET SOFTWARE CONSORTIUM
BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL
DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF
USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN
CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Copyright © 1995-1998 Eric Young (eay@cryptsoft.com) All rights reserved.

This package is an SSL implementation written by Eric Young (eay@cryptsoft.com).
The implementation was written so as to conform with Netscape’s SSL.

This library is free for commercial and non-commercial use as long as the
following conditions are adhered to. The following conditions apply to all code
found in this distribution, be it the RC4, RSA, lhash, DES, etc., code; not just the
SSL code. The SSL documentation included with this distribution is covered by the
same copyright terms except that the holder is Tim Hudson (tjh@cryptsoft.com).

Copyright remains Eric Young’s, and as such any Copyright notices in the code are
not to be removed. If this package is used in a product, Eric Young should be
given attribution as the author of the parts of the library used. This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement: "This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)". The word 'cryptographic' can be left out if the routines from the library being used are not cryptographic related.
4. If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include acknowledgement:
   "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"

THIS SOFTWARE IS PROVIDED BY ERIC YOUNG "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The license and distribution terms for any publicly available version or derivative of this code cannot be changed. i.e. this code cannot simply be copied and put under another distribution license [including the GNU Public License.]

This product includes cryptographic software written by Eric Young.


Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND INTERNET SOFTWARE CONSORTIUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL INTERNET SOFTWARE CONSORTIUM BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.


Copyright © 2004 IBM Corporation and its licensors, including Sendmail, Inc., and the Regents of the University of California. All rights reserved.

Copyright © 1999,2000,2001 Compaq Computer Corporation

Copyright © 1999,2000,2001 Hewlett-Packard Company

Copyright © 1999,2000,2001 IBM Corporation

Copyright © 1999,2000,2001 Hummingbird Communications Ltd.


Copyright © 1999,2000,2001 Sun Microsystems, Inc.

Copyright © 1999,2000,2001 The Open Group

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

X Window System is a trademark of The Open Group.
Programming interface information

This publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of z/OS Communications Server.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, HCD, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

The following terms are trademarks of other companies:

Intel is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

PostScript is a registered trademark of Adobe Systems Incorporated in the United States, and/or other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.
Bibliography

This bibliography contains descriptions of the documents in the z/OS Communications Server library.

z/OS Communications Server documentation is available in the following forms:

- In softcopy on CD-ROM collections. See “Softcopy information” on page xxviii.

z/OS Communications Server library updates


z/OS Communications Server information

z/OS Communications Server product information is grouped by task in the following tables.

### Planning

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS Communications Server: New Function Summary</td>
<td>GC31-8771</td>
<td>This document is intended to help you plan for new IP for SNA function, whether you are migrating from a previous version or installing z/OS for the first time. It summarizes what is new in the release and identifies the suggested and required modifications needed to use the enhanced functions.</td>
</tr>
<tr>
<td>z/OS Communications Server: IPv6 Network and Application Design Guide</td>
<td>SC31-8885</td>
<td>This document is a high-level introduction to IPv6. It describes concepts of z/OS Communications Server’s support of IPv6, coexistence with IPv4, and migration issues.</td>
</tr>
</tbody>
</table>

### Resource definition, configuration, and tuning

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS Communications Server: IP Configuration Guide</td>
<td>SC31-8775</td>
<td>This document describes the major concepts involved in understanding and configuring an IP network. Familiarity with the z/OS operating system, IP protocols, z/OS UNIX System Services, and IBM Time Sharing Option (TSO) is recommended. Use this document in conjunction with the z/OS Communications Server: IP Configuration Reference.</td>
</tr>
<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               | SC31-8776| This document presents information for people who want to administer and maintain IP. Use this document in conjunction with the z/OS Communications Server: IP Configuration Guide. The information in this document includes:  
  • TCP/IP configuration data sets  
  • Configuration statements  
  • Translation tables  
  • SMF records  
  • Protocol number and port assignments                              |
| z/OS Communications Server: SNA Network Implementation Guide         | SC31-8777| This document presents the major concepts involved in implementing an SNA network. Use this document in conjunction with the z/OS Communications Server: SNA Resource Definition Reference. |
| z/OS Communications Server: SNA Resource Definition Reference        | SC31-8778| This document describes each SNA definition statement, start option, and macroinstruction for user tables. It also describes NCP definition statements that affect SNA. Use this document in conjunction with the z/OS Communications Server: SNA Network Implementation Guide. |
| z/OS Communications Server: SNA Resource Definition Samples          | SC31-8836| This document contains sample definitions to help you implement SNA functions in your networks, and includes sample major node definitions.         |
| z/OS Communications Server: IP Network Print Facility                | SC31-8833| This document is for system programmers and network administrators who need to prepare their network to route SNA, JES2, or JES3 printer output to remote printers using TCP/IP Services. |

**Operation**

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS Communications Server: IP User’s Guide and Commands</td>
<td>SC31-8780</td>
<td>This document describes how to use TCP/IP applications. It contains requests that allow a user to log on to a remote host using Telnet, transfer data sets using FTP, send and receive electronic mail, print on remote printers, and authenticate network users.</td>
</tr>
<tr>
<td>z/OS Communications Server: IP System Administrator’s Commands</td>
<td>SC31-8781</td>
<td>This document describes the functions and commands helpful in configuring or monitoring your system. It contains system administrator’s commands, such as TSO NETSTAT, PING, TRACERTE and their UNIX counterparts. It also includes TSO and MVS commands commonly used during the IP configuration process.</td>
</tr>
<tr>
<td>z/OS Communications Server: SNA Operation</td>
<td>SC31-8779</td>
<td>This document serves as a reference for programmers and operators requiring detailed information about specific operator commands.</td>
</tr>
<tr>
<td>z/OS Communications Server: Quick Reference</td>
<td>SX75-0124</td>
<td>This document contains essential information about SNA and IP commands.</td>
</tr>
</tbody>
</table>
## Customization

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
</table>
| z/OS Communications Server: SNA Customization    | SC31-6854 | This document enables you to customize SNA, and includes the following:  
  • Communication network management (CNM) routing table  
  • Logon-interpret routine requirements  
  • Logon manager installation-wide exit routine for the CLU search exit  
  • TSO/SNA installation-wide exit routines  |

## Writing application programs

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS Communications Server: IP Sockets Application Programming Interface Guide and Reference</td>
<td>SC31-8788</td>
<td>This document describes the syntax and semantics of program source code necessary to write your own application programming interface (API) into TCP/IP. You can use this interface as the communication base for writing your own client or server application. You can also use this document to adapt your existing applications to communicate with each other using sockets over TCP/IP.</td>
</tr>
<tr>
<td>z/OS Communications Server: IP CICS Sockets Guide</td>
<td>SC31-8807</td>
<td>This document is for programmers who want to set up, write application programs for, and diagnose problems with the socket interface for CICS using z/OS TCP/IP.</td>
</tr>
<tr>
<td>z/OS Communications Server: IP IMS Sockets Guide</td>
<td>SC31-8830</td>
<td>This document is for programmers who want application programs that use the IMS TCP/IP application development services provided by IBM’s TCP/IP Services.</td>
</tr>
<tr>
<td>z/OS Communications Server: IP Programmer’s Guide and Reference</td>
<td>SC31-8787</td>
<td>This document describes the syntax and semantics of a set of high-level application functions that you can use to program your own applications in a TCP/IP environment. These functions provide support for application facilities, such as user authentication, distributed databases, distributed processing, network management, and device sharing. Familiarity with the z/OS operating system, TCP/IP protocols, and IBM Time Sharing Option (TSO) is recommended.</td>
</tr>
<tr>
<td>z/OS Communications Server: SNA Programming</td>
<td>SC31-8829</td>
<td>This document describes how to use SNA macroinstructions to send data to and receive data from (1) a terminal in either the same or a different domain, or (2) another application program in either the same or a different domain.</td>
</tr>
<tr>
<td>z/OS Communications Server: SNA Programmer’s LU 6.2 Guide</td>
<td>SC31-8811</td>
<td>This document describes how to use the SNA LU 6.2 application programming interface for host application programs. This document applies to programs that use only LU 6.2 sessions or that use LU 6.2 sessions along with other session types. (Only LU 6.2 sessions are covered in this document.)</td>
</tr>
<tr>
<td>z/OS Communications Server: SNA Programmer’s LU 6.2 Reference</td>
<td>SC31-8810</td>
<td>This document provides reference material for the SNA LU 6.2 programming interface for host application programs.</td>
</tr>
<tr>
<td>z/OS Communications Server: CSM Guide</td>
<td>SC31-8808</td>
<td>This document describes how applications use the communications storage manager.</td>
</tr>
<tr>
<td>Title</td>
<td>Number</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>z/OS Communications Server: CMIP Services and Topology Agent Guide</td>
<td>SC31-8828</td>
<td>This document describes the Common Management Information Protocol (CMIP) programming interface for application programmers to use in coding CMIP application programs. The document provides guide and reference information about CMIP services and the SNA topology agent.</td>
</tr>
</tbody>
</table>

**Diagnosis**

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS Communications Server: IP Diagnosis Guide</td>
<td>GC31-8782</td>
<td>This document explains how to diagnose TCP/IP problems and how to determine whether a specific problem is in the TCP/IP product code. It explains how to gather information for and describe problems to the IBM Software Support Center.</td>
</tr>
<tr>
<td>z/OS Communications Server: ACF/TAP Trace Analysis Handbook</td>
<td>GC23-8588</td>
<td>This document explains how to gather the trace data that is collected and stored in the host processor. It also explains how to use the Advanced Communications Function/Trace Analysis Program (ACF/TAP) service aid to produce reports for analyzing the trace data information.</td>
</tr>
<tr>
<td>z/OS Communications Server: SNA Diagnosis Vol 1, Techniques and Procedures and z/OS Communications Server: SNA Diagnosis Vol 2, FFST Dumps and the VIT</td>
<td>GC31-6850, GC31-6851</td>
<td>These documents help you identify an SNA problem, classify it, and collect information about it before you call the IBM Support Center. The information collected includes traces, dumps, and other problem documentation.</td>
</tr>
<tr>
<td>z/OS Communications Server: SNA Data Areas Volume 1 and z/OS Communications Server: SNA Data Areas Volume 2</td>
<td>GC31-6852, GC31-6853</td>
<td>These documents describe SNA data areas and can be used to read an SNA dump. They are intended for IBM programming service representatives and customer personnel who are diagnosing problems with SNA.</td>
</tr>
</tbody>
</table>

**Messages and codes**

<table>
<thead>
<tr>
<th>Title</th>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
</table>
| z/OS Communications Server: SNA Messages                              | SC31-8790| This document describes the ELM, IKT, IST, IUT, IVT, and USS messages. Other information in this document includes:  
  • Command and RU types in SNA messages  
  • Node and ID types in SNA messages  
  • Supplemental message-related information |
| z/OS Communications Server: IP Messages Volume 1 (EZA)               | SC31-8783| This volume contains TCP/IP messages beginning with EZA.                                                                                     |
| z/OS Communications Server: IP Messages Volume 2 (EZB, EZD)          | SC31-8784| This volume contains TCP/IP messages beginning with EZB or EZD.                                                                               |
| z/OS Communications Server: IP Messages Volume 3 (EZY)               | SC31-8785| This volume contains TCP/IP messages beginning with EZY.                                                                                     |
| z/OS Communications Server: IP Messages Volume 4 (EZZ, SNM)          | SC31-8786| This volume contains TCP/IP messages beginning with EZZ and SNM.                                                                              |
| z/OS Communications Server: IP and SNA Codes                         | SC31-8791| This document describes codes and other information that appear in z/OS Communications Server messages.                                       |
Index

Special characters
<rapi.h> header 180
adspec definitions 182
filterspec definitions 184
flowspec definitions 181
function interface definitions 185
general definitions 180
policy definitions 184
reservation style definitions 184
tspec definitions 180

A
accessibility 761
addr parameter on RPC call
on clnttcp_create() 245
on clntudp_create() 247
on get_myaddress() 249
on pmap_getmaps() 251
on pmap_getport() 252
on pmap_rmtcall() 253
on xdrmem_create() 319
adspec definitions 182
adspec pieces 192
adspecs 174
AF_INET6 683
agent distributed protocol interface (DPI) 3, 41
ap parameter on RPC call, on xdr_opaque_auth() 299
application resources, X Windows 733, 742
Application Transparent Transport Layer Security (AT-TLS) 637
applications, functions and protocols
Network Computing System (NCS) 341
remote procedure calls (RPC) 207
SNMP DPI 3, 41
X Window system interface 195, 685
ar parameter on RPC call, on xdr_accepted_reply() 280
areas, clearing and copying, X Windows 705
areas, filling, X Windows 706
arrp parameter on RPC call, on xdr_array() 281
associate table functions, X Windows 719
asynchronous event handling, RAPI 175
AT-TLS
CICS transaction considerations 638
coding the SIOCTTLSCTL ioctl 643
defined 637
sample code for building and issuing the SIOCTTLSCTL ioctl 654
starting on a connection 639
steps for implementing a controlling server application 640
steps for implementing an aware server application 640
steps for starting an aware or controlling server application 641
using the SIOCTTLSCTL ioctl 639
athena widget set headers 690
Athena Widget Support 733
aup_gids parameter on RPC call, on authunix_create() 225
aupp parameter on RPC call, on xdr_authunix_parms() 282
auth parameter on RPC call, on auth_destroy() 223
auth_destroy(), RPC call 223
authnone_create()(RPC) 224
authorization routines, X Windows 722
authunix_create_default() 226
authunix_create() (RPC) 225
B
BANK sample program data sets, NCS 362
basep parameter on RPC call, on xdr_vector() 316
BINOP sample program
Data sets, NCS 353
bitmaps, manipulating 714
bp parameter on RPC call, on xdr_bool() 283
buffers, cut and paste, X Windows 713
building X client modules 699
C
c89 utility options 741
callrpc() 227
CC CLIST, processed by RPCGEN 215
changing window attributes 702
character set selection 101
character string sizes, X Windows 707
chdr parameter on RPC call, on xdr_callhdr() 285
choices parameter on RPC call, on xdr_union() 314
Client

cleanup 207
free resources 207
initialize 207
port numbers 210
process call 207
remote procedure call 207
clint parameter on RPC call
on clnt_call() 231
on clnt_control() 233
on clnt_destroy() 235
on clnt_freeres() 236
on clnt_geterr() 237
on clnt_perror() 238
on clnt_perror() 239
on clnt_perror() 240
on clnt_perror() 243
clint_broadcast() 229
clint_call() 231
clint_control() 233
clint_create() 234
clint_destroy() 235
clint_freeres() 236
clint_geterr() 237
clint_perror() 238
clint_perror() 239
clint_perror() 240
clint_perror() 241
clint_perror() 242
clint_perror() 243
clint_stat enumerated type 216
clntraw_create() 244
clntraw_create() 245
clnudp_create() 247
cmsg parameter on RPC call, on xdr_callmsg() 286

© Copyright IBM Corp. 1989, 2009
777
cnt parameter on RPC call, on xdr_opaque() 298
color cells, manipulating, X Windows 704
colormaps, manipulating, X Windows 703
Communications Server for z/OS, online information xxix
compiler nidl 347
compiling and linking
C sockets 6
Kerberos 6
NCS 346, 350
RPC 217
SNMP 6, 44
UNIX System Services 741
X Windows 691, 739
connecting to an agent through UNIX 82
controlled-load services formats 190, 191
COPY 432
cp parameter on RPC call
on xdr_char() 287
on xdr_opaque() 298
creating and destroying windows 701
cursors, manipulating, X Windows 708
D
data structures
header files for RPCs 217
header files for X Window system 344, 689
MANIFEST.H, to remap names 216
pascal include data set 344
default parameter on RPC call, on xdr_union() 314
diagnosis, network management 634
Differentiated Services Policies 679
Digital Certificate Access Server (DCAS), configuring 661
Digital Certificate Access Server (DCAS), defining request and response specifications 658
Digital Certificate Access Server (DCAS), interfacing with 657
disability 761
dispatch(), on svc_register() 265
display functions, X Windows 715
DNS, online information xxxi
dp parameter on RPC call, on xdr_double.parms() 289
DPI requests, processing 4
DPI_CLOSE_reason_codes 102
DPI_PACKET_LEN() 60
DPI_RC_values 105
DPI_UNREGISTER_reason_codes 103
DPI, packet types 102
DPI, value types 103
DPlawait_packet_from_agent() 78
DPlconnect_to_agent_TCP() 80
DPlconnect_to_agent_UNIXstream() 82
DPldebug() 59
DPldisconnect_from_agent() 84
DPlget_fd_for_handle() 85
DPlsend_packet_to_agent() 86
don TCP connections or UDP endpoints interfaces configuration and enablement 600
EZBNMIFR 600
request and response data structures 615
request format 603
response format 612
dscmp parameter on RPC call, on xdr_union() 314
E
eachresult parameter on RPC call, on clnt_broadcast() 229
elemsize parameter on RPC call, on xdr_vector() 316
elproc parameter on RPC call, on xdr_array() 281
elsize parameter on RPC call, on xdr_array() 281
ep parameter on RPC call, on xdr_enum.parms() 290, 316
error code, DPI RESPONSE error codes 102
error handling, default, X Windows 711
errp parameter on RPC call, on clnt_geterr() 237
events handling, X Windows 710
extension routines, X Windows 717
EZBCTAPI 583
EZBNMIFR 600
EZBNMIFR interface 598
EZBTMIC1 549
EZBTMIC4 549
F
FCAL_IE_InternalErr 469
FCAL_IE_LengthInvalid 467
FCAL_POLLWait 468
FCAL_ReqTimer 468
FCAL_Status_TraceFailed 467
FCAL_TraceStatus 467
fDPlparse() 61
fDPlparse(), SNMP 7
fDPlset() 62
file parameter on RPC call, on xdrstdio_create() 324
Files, Motif, location 203
filter spec definitions 184
filter specs 175
filter, mail 367
FIND 432
flowspecs 174
fonts, loading and freeing, X Windows 706
formats for controlled-load services 190, 191
formatting packet and data trace records interfaces 581
configuration and enablement 583
EZBCTAPI 583
passing options 596
using the formatter 598
fp parameter on RPC call, on xdr_float() 292
FTP client API 397
additional output 461
call formats 401
compatibility 398
control block 404
controlling requests 468
converting parameter descriptions 402
exceptional conditions 469
FCAL_IE_InternalErr 469
FCAL_IE_LengthInvalid 467
FCAL_POLLWait 468
FCAL_ReqTimer 468
FCAL_Status_TraceFailed 467
FCAL_TraceStatus 467
field values 404
guidelines and requirements 398
interpreting results 464
messages and replies 463
output register 460
programming notes 466
prompts 462
reporting failures 467
FTP client API (continued)  
  requests 418  
  GETL 427  
  INIT 419  
  POLL 425  
  SCMD 422  
  TERM 433  
  sample programs 474  
  sending requests 418  
  specifying a wait time 468  
  tracing 470  
  unanticipated conditions 469  
  z/OS FTP client behavior 402  
FTP client login failure record 573  
FTP client session record 578  
FTP client transfer initialization record 568  
FTP server session record 565  
FTP server transfer initialization record 559  
function  
  DPI_PACKET_LEN() 60  
  DPI_wait_packet_from_agent() 78  
  DPIconnect_to_agent_TCP() 80  
  DPIdebug() 59  
  DPI_disconnect_from_agent() 84  
  DPI_get_fd_for_handle() 85  
  DPI_send_packet_to_agent() 86  
  fDPI_parse() 61  
  fDPI_set() 62  
  lookup_host() 88  
  lookup_host6() 89  
  mkDPI_Are_You_There() 63  
  mkDPI_Close() 64  
  mkDPI_open() 65  
  mkDPI_register() 67  
  mkDPI_response() 69  
  mkDPI_trap() 73  
  mkDPI_unregister() 75  
  pDPI_packet() 76  

G  
general definitions, RAPI 180  
GET request processing 4  
get_myaddress() 249  
GET-NEXT request processing 4  
GETL 427  
  Assembler example 432  
  COPY operation 427, 432  
  example 429  
  FIND operation 427, 432  
GETL request  
  result guidelines 431  
getreq() (RPC) 263  
getrpctcpport() 250  
gid parameter on RPC call, on authunix_create() 225  
images, manipulating, X Windows 714  
images, transferring 707  
in parameter on RPC call  
  on callrpc() 227  
  on clnt_broadcast() 229  
  on clnt_call() 231  
  on pmap_rmtcall() 253  
  on svc_freeargs() 260  
  on svc_getargs() 261  
  include, snmp_dpi.h 106  
info parameter on RPC call, on clnt_control() 233  
Information APARs xxvii  
INIT 419  
  example 419  
INIT request  
  result guidelines xxvii  
inproc parameter on RPC call  
  on callrpc() 227  
  on clnt_broadcast() 229  
  on clnt_call() 231  
  on pmap_rmtcall() 253  
  on registerrpc() 257  
  on svc_freeargs() 260  
  on svc_getargs() 261  
integrated services adspec 194  
integrated services data structures and macros 186  
adspec pieces 192  
  formats for controlled-load services 190, 191  
  general definitions 187  
  generic tspec format 189  
  integrated services adspec 194  
  integrated services flowspec 193  
  integrated services tspec 193  
  integrated services flowspec 193  
  integrated services tspec 193  
interfaces  
  drop TCP connections or UDP endpoints  
    configuration and enablement 600  
    EZBNMIFR 600  
  request and response data structures 615  
  request format 603  
  response format 612  
  formatting packet and data trace records 581  
    configuration and enablement 583  
    EZBCTAPI 583  
  monitor TCP/UDP endpoints, TCP/IP storage, and TN3270 performance  
    configuration and enablement 600  
    EZBNMIFR 600  

H  
handle parameter on RPC call, on xdrrec_create() 320  
header files  
  NCS 344  
  NCS C 344  
  remote procedure calls 217  
  SNMP DPI 5  
  header files (continued)  
    X Window system  
      Athena Widget Set 690  
      OSF/Motif 691  
    X Window system and Xt Intrinsics 690  
  header files, RAPI 179  
  high_vers parameter on RPC call, on svcerr_progvers() 274  
  host parameter on RPC call  
    on authunix_create() 225  
    on callrpc() 227  
    on clnt_create() 234  
    on getrpctcpport() 250  
  hosts and access control, X Windows 709  

I  
IBM Software Support Center, contacting xxiii  
identifying the target display, X Windows 688, 739  
images, manipulating, X Windows 714  
images, transferring 707  
in parameter on RPC call  
  on callrpc() 227  
  on clnt_broadcast() 229  
  on clnt_call() 231  
  on pmap_rmtcall() 253  
  on svc_freeargs() 260  
  on svc_getargs() 261  
  include, snmp_dpi.h 106  
info parameter on RPC call, on clnt_control() 233  
Information APARs xxvii  
INIT 419  
  example 419  
INIT request  
  result guidelines xxvii  
inproc parameter on RPC call  
  on callrpc() 227  
  on clnt_broadcast() 229  
  on clnt_call() 231  
  on pmap_rmtcall() 253  
  on registerrpc() 257  
  on svc_freeargs() 260  
  on svc_getargs() 261  
integrated services adspec 194  
integrated services data structures and macros 186  
adspec pieces 192  
  formats for controlled-load services 190, 191  
  general definitions 187  
  generic tspec format 189  
  integrated services adspec 194  
  integrated services flowspec 193  
  integrated services tspec 193  
  integrated services flowspec 193  
  integrated services tspec 193  
interfaces  
  drop TCP connections or UDP endpoints  
    configuration and enablement 600  
    EZBNMIFR 600  
  request and response data structures 615  
  request format 603  
  response format 612  
  formatting packet and data trace records 581  
    configuration and enablement 583  
    EZBCTAPI 583  
  monitor TCP/UDP endpoints, TCP/IP storage, and TN3270 performance  
    configuration and enablement 600  
    EZBNMIFR 600  

Index 779
monitor TCP/UDP endpoints, TCP/IP storage, and TN3270 performance

Real-time TCP/IP network monitoring

SNA network monitoring data

TCP/IP callable NMI (EZBNMIFR)

TCP/IP network monitoring

communications with the server

configuration and enablement

enabling and disabling

request/response format

SNA network monitoring NMI

TCP/IP network monitoring

configurations and enablement

EZBNMIFR

request and response data structures

request format

response format

X Window System

Internet, finding z/OS information online

intrinsics routines, X Windows

ip parameter on RPC call, on xdr_int()

keyboard

keyboard events, X Windows

Keyboard settings, manipulating, X Windows

Library

len parameter on RPC call

on authunix_create() 225

on xdr_inline() 295

lf_smpl.c sample mail filter

libraries

SNMP

X Window system

license, patent, and copyright information

limits

lines, drawing, X Windows

LookAt

lookup_host() 88

lookup_host6() 89

low_vers parameter on RPC call, on svcerr_progvers() 274

lp parameter on RPC call, on xdr_long()

macro, DPI_PACKET_LEN() 60

mail filter

callbacks

xxfi_abort 380

xxfi_body 379

xxfi_close 381

xxfi_connect 377

xxfi_envfrom 377

xxfi_envrcpt 378

mail filter (continued)

callbacks (continued)

xxfi_eoh 379

xxfi_eom 380

xxfi_header 379

xxfi_helo 377

compiling and linking sample source

data access functions

library control functions

message modification functions

running

Mainframe

education

management information base (MIB)

MANIFEST.H data set, long name remapping

manipulating window properties

manipulating windows

maxsize parameter on RPC call

on xdr_array() 281

on xdr_bytes() 284

on xdr_string() 308

messages and replies

FTP client API

MIT extensions to X

mkDPIAreYouThere() 63

mkDPIclose() 64

mkDPIopen() 65

mkDPIregister() 8, 67

mkDPIresponse() 8, 69

mkDPIset() 9, 71

mkDPItrap(10, 73

mkDPIunregister() 75

monitor TCP/UDP endpoints, TCP/IP storage, and TN3270

performance

interfaces

NCS

compiling, linking, and running sample program

IDL data sets

MVS limitations

NCSDEFS.H, defined

portability issues

redefines for sample program

RPC-RUNTIME library

sample programs

USERDEFS.H, user defined
procnum parameter on RPC call (continued) on registerpc() 257
prognnum parameter on RPC call
on callrpc() 227
on clnt_broadcast() 229
on clnt_create() 234
on clntcreate() 234
on clnt_rawcreate() 244
on clnttcp_create() 245
on clntudp_create() 247
on getrpcport() 250
on pmap_getport() 252
on pmap_rmtcall() 253
on pmap_set() 255
on pmap_unset() 256
on registerpc() 257
on svc_register() 265
reason code, DPI CLOSE reason codes 102
reason code, DPI UNREGISTER reason codes 103
record formats
Real-time SMF data NMI (SYSTCPSM) 558
FTP client login failure record 573
FTP client session record 578
FTP client transfer initialization record 568
FTP server session record 565
FTP server transfer initialization record 559
TCP/IP identification section 558
recv_bufsize parameter on RPC call
on svc_tcp_create() 278
on svcudp_create() 279
recvsize parameter on RPC call, on xdrrec_create() 320
recvsz parameter on RPC call, on clnttcp_create() 245
reference sections
well-known port assignments 675
regions, X Windows 713
REGISTER request processing 5
registerpc() 257
regs parameter on RPC call, on xdr_pmap() 300
remote Procedure and external data representation calls
auth_destroy() 223
authnone_create() 224
authunix_create_default() 226
authunix_create() 225
callrpc() 227
clnt_broadcast() 229
clnt_call() 231
clnt_control() 233
clnt_create() 234
clnt_destroy() 235
clnt_freeres() 236
clnt_geterr() 237
clnt_pcreateerror() 238
clnt_errno() 239
clnt_perror() 240
clnt_pcreateerror() 241
clnt_sperror() 242
clnt_sperror() 243
clntraw_create() 244
clnttcp_create() 245
clntudp_create() 247
get_myaddress() 249
getrpcport() 250
pmap_getmaps() 251
pmap_getport() 252
pmap_rmtcall() 253
pmap_set() 255
pmap_unset() 256
registerpc() 257
rpc_createerr 219
svc_destroy() 259
svc_fds() 220
svc_fds() 220
svc_freeargs() 260
svc_getargs() 261
svc_getcaller() 262
svc_getreq() 263
svc_register() 265
svc_run() 267
svc_sendreply() 268
svc_unregister() 269
svcerr_auth() 270
svcerr_decode() 271
svcerr_noproc() 272
svcerr_noprog() 273
svcerr_progvers() 274

Q
query_DPI_port() 13

R
RAPI (Resource Reservation Setup Protocol API) 161
RAPI error codes 178
RAPI error handling 177
RAPI function interface definitions 185
RAPI objects 173
adspecs 174
filter specs 175
flowspecs 174
sender templates 175
sender tspecs 174
RAPI policy definitions 184
RAPI reservation style definitions 184
rapi_dispatch() 176
rapi_event_rtn() 163
rapi_fmt_adspec() 170
rapi_fmt_filtspec() 171
rapi_fmt_flowspec() 172
rapi_fmt_tspec() 173
rapi_getfd() 177
rapi_release() 166
rapi_reserve() 166
rapi_sender() 167
rapi_session() 169
rapi_version() 170
rc values, DPI_RC_values 105
rdfs parameter on RPC call, on svc_getreq() 263
readit() parameter, on xdrrec_create() 320
Real-time SMF data NMI (SYSTCPSM)
record formats 558
FTP client login failure record 573
FTP client session record 578
FTP client transfer initialization record 568
FTP server session record 565
FTP server transfer initialization record 559
TCP/IP identification section 558

782 z/OS V1R11.0 Comm Svr: IP Programmer’s Guide and Reference
Remote Procedure and external data representation calls (continued)

- svcerr_systemerr() 275
- svcerr_weakauth() 276
- svcraw_create() 277
- svctcp_create() 278
- svcudp_create() 279
- xdr_accepted_reply() 280
- xdr_array() 281
- xdr_authunix_parms() 282
- xdr_bool() 283
- xdr_bytes() 284
- xdr_callhdr() 285
- xdr_callmsg() 286
- xdr_char() 287
- xdr_destroy() 288
- xdr_double() 289
- xdr_enum() 290
- xdr_float() 291
- xdr_free() 292
- xdr_getpos() 293
- xdr_inline() 294
- xdr_int() 295
- xdr_long() 296
- xdr_opaque_auth() 297
- xdr_opaque() 298
- xdr_pmap() 299
- xdr_pmaplist() 300
- xdr_pointer() 301
- xdr_reference() 302
- xdr_rejected_reply() 303
- xdr_replmsg() 304
- xdr_repos() 305
- xdr_short() 306
- xdr_string() 307
- xdr_u_char() 308
- xdr_u_int() 309
- xdr_u_long() 310
- xdr_u_short() 311
- xdr_union() 312
- xdr_vector() 313
- xdr_void() 314
- xdr_wrapstring() 315
- xdrmem_create() 316
- xdrnew_create() 317
- xdrnew_endofrecord() 318
- xdrnew_eof() 319
- xdrnew_skiprecord() 320
- xdrnew_create() 321
- xdrnew_eof() 322
- xdrnew_skiprecord() 323
- xdrstdio_create() 324
- xprt_register() 325
- xprt_unregister() 326
- remote procedure call (RPC) header files 217
- portmapper 209
- RPCGEN command 214
- RPCGEN sample programs 334
- sample programs
  - GENESEND, client 327
  - GENESERV, server 329
- RAWEX, raw data stream 331
- remote procedure call (RPC) global variables
  - global variables 218
  - rpc_createerr 219
  - svc_fds 220
- remote procedure call (RPC) protocol
  - clnt_stat enumerated type 216
SNMP DPI request 4
setting window selections 703
shortcut keys 761
simple network management protocol (SNMP) 3, 41
SOCSAPPLDATA IOCTL 663
size parameter on RPC call
  on xdr_pointer() 302
  on xdr_reference() 303
  on xdrmem_create() 319
sizep parameter on RPC call
  on xdr_array() 281
  on xdr_bytes() 284
smfi_addheader 373
smfi_addrcpt 374
smfi_chgheader 373
smfi_delrcpt 375
smfi_getpriv 371
smfi_getsymval 370
smfi_main 370
smfi_register 368
smfi_replacebody 375
smfi_setconn 369
smfi_setpriv 371
smfi_setreply 372
smfi_settimeout 369
SNA network monitoring data
  interfaces
    configuration 621
    data structures and records 627
    enabling and disabling 621
    request/response format 622
SNA network monitoring NMI interface 620
SNMP
  client program 13, 129
  compiling and linking 6, 44
  fDPIparse() 7
  GET-NEXT 4
  header files 5
  library routines 6
  mkDPIpacket() 11
  mkDPIregister() 8
  mkDPIresponse() 8
  mkDPIset() 9
  mkDPItrap() 10
  query_DPI_port() 13
  REGISTRATION request, processing 5
  TRAP request 5
SNMP agents 3, 41
SNMP manager API 131
SNMP subagents 3, 41
SNMP_CLOSE_reason_codes 102
snmp_dpi_close_packet 91
snmp_dpi_get_packet 92
snmp_dpi_hdr 93
snmp_dpi_next_packet 95
SNMP_DPI_packet_types 102
snmp_dpi_resp_packet 96
snmp_dpi_set_packet 97
snmp_dpi_u64 100
snmp_dpi_ureg_packet 99
snmp_dpi.h 106
SNMP_ERROR_codes 102
SNMP_TYPE_value_types 103
SNMP_UNREGISTER_reason_codes 103
snmpAddVarBind 133
snmpBuildPDU 133
snmpBuildSession 135
snmpBuildV1TrapPDU 154
snmpBuildV2TrapOrInformPDU 155
snmpCreateVarBinds 136
snmpFreeDecodedPDU 137
snmpFreeOID 137
snmpFreePDU 137
snmpFreeVarBinds 138
snmpGetErrorInfo 138
snmpGetNumberOfVarBinds 139
snmpGetOID 139
snmpGetRequestID 140
snmpGetSockFd 140
snmpGetValue 140
snmpGetVarbind 141
snmpInitialize 142
snmpSendRequest 143
snmpSetLogFunction 144
snmpSetLogLevel 145
snmpSetRequestID 145
snmpTerminate 146
snmpTerminateSession 146
snmpValueCreateBits 147
snmpValueCreateCounter32 147
snmpValueCreateCounter64 148
snmpValueCreateInteger 148
snmpValueCreateInteger32 149
snmpValueCreateNSAPAddr 150
snmpValueCreateNull 150
snmpValueCreateOctet 151
snmpValueCreateOID 151
snmpValueCreateOpaque 152
snmpValueCreateTimerTicks 152
snmpValueCreateUnsignedInteger32 153
sock parameter on RPC call, on svctcp_create() 278
sockets
  compiler restrictions 738
    UNIX System Services 738
      using 738
  sockp parameter on RPC call
    on clnttcp_create() 245
    on clntudp_create() 247
    on svcudp_create() 279
softcopy information xxvii
software requirements
    UNIX System Services 738
    X Windows 686
sp parameter on RPC call
  on xdr_bytes() 284
  on xdr_short() 307
  on xdr_string() 308
  on xdr_wrapstring() 318
stat parameter on RPC call
  on clnt_perrno() 239
  on clnt_sperrno() 242
structure
  snmp_dpi_close_packet 91
  snmp_dpi_get_packet 92
  snmp_dpi_hdr 93
  snmp_dpi_next_packet 95
  snmp_dpi_resp_packet 96
subroutines (X Window system) 701
svc_destroy() 259
tasks (continued)
  Run (continued)
    steps for the sample BINOP program 357
  Setup
    steps for the BANK program 362
    steps for the NCSSMP program 358
    steps for the sample BINOP program 354
TCP_KeepAlive socket option 668
TCP/IP
  online information xxix
  protocol specifications 743
  TCP/IP identification section 558
  TCP/IP network monitoring
    format of service-specific data 553
    interfaces 540
    communicating with the server 622
    configuration and enablement 543
    connecting to the server 544
    copying the trace buffer 548
    EZBTMIC1 549
    EZBTMIC4 549
    records 546
    requests 546
    TMI_CopyBuffer 551
SYSTCPDN 556
SYSTCPDA 553
SYSTCPSM 557
  record formats 558
SYSTCPSM interface 540
system toolkit, X Windows 723

T

tasks
(noun, gerund phrase)
  steps 159
  aware or controlling server application, starting
  steps 641
  aware server application, implementing
  steps 640
  capturing trace records
  steps for 592
Compile
  steps for the BANK program 364
  steps for the NCSSMP program 359
  steps for the sample BINOP program 355
  controlling server application, implementing
  steps 640
  creating an ancillary socket
  steps 667
  issuing the GETL request
  steps for 431
  issuing the INIT request
  steps for 421
  issuing the POLL request
  steps for 426
  issuing the SCMD request
  steps for 424
  issuing the TERM request
  steps for 435
Link
  steps for the BANK program 365
  steps for the NCSSMP program 360
  steps for the sample BINOP program 356
Run
  steps for the BANK program 366
  steps for the NCSSMP program 361

U
ucp parameter on RPC call, on xdr_u_char() 310
uid parameter on RPC call, on authunix_create() 225
ulp parameter on RPC call, on xdr_u_long() 312
UNIX System Services
  compiling and linking 741
  sockets 738
  software requirements 738
  using 738
  what is provided 738
UNIXstream function 82
unp parameter on RPC call, on xdr_union() 314
up parameter on RPC call, on xdr_u_int() 311
using
Motif 203
X Window System 195
uspr parameter on RPC call, on xdr_u_short() 313
utility routines, X Windows 720
UUID@GEN identifier generator 352

V
value ranges 105
value types, SNMP_TYPE_value_types 103
versnum parameter on RPC call
on callrpc() 227
on clnt_broadcast() 229
on clnt_create() 234
on clnttcp_create() 245
on clntudp_create() 247
on getrpcport() 250
on pmap_getport() 252
on pmap_rmtcall() 253
on pmap_set() 255
on pmap_unset() 256
on registerrpc() 257
on svc_register() 265
on svc_unregister() 269
visual types 714
VTAM, online information xxix

W
wait parameter on RPC call, on clntudp_create() 247
well-known port assignments 675
what is provided, UNIX System Services 738
what is provided, X Windows 685
why parameter on RPC call, on ssvc_auth() 270
widgets, defining 723
window manager functions, X Windows 708
window manager, communicating with, X Window system 711
writeit() parameter on RPC, on xdrrec_create() 320

X
X Window system (continued)
hosts and access control 709
identifying target display 688, 739
images, manipulating 714
images, transferring 707
keyboard events, manipulating 712
keyboard settings, handling 709
lines, drawing 705
manipulating window properties 703
manipulating windows 702
obtaining properties and atoms 703
obtaining window information 703
opening and closing a display 701
pixmaps, creating and freeing 704
porting applications 723
regions, manipulating 713
resource manager 715
sample programs, X Windows 695
screen saver, controlling 709
setting window selections 703
synchronization, enable and disable 710
text, drawing 707
visual types 714
window manager functions 708
window managers, communicating 711
X client applications 697
X client modules, building 699
X defaults 688
X Window system Interface 195, 685, 686
X Window system Toolkit 723
Xt Intrinsics 733, 742
X Window system, application layer
Application Resources 733, 742
Athena Widget Support 733
Authorization Routines 722
Miscellaneous Utility Routines 720
MIT Extensions 718
Motif-Based Widget Support 737
Routines 701
Xt Intrinsics Routines 724
X Window system, what is provided 685
xdr_accepted_reply() 280
xdr_array() 281
xdr_authunix_parms() 282
xdr_bool() 283
xdr_bytes() 284
xdr_callhdr() 285
xdr_callmsg() 286
xdr_char() 287
xdr_destroy() 288
xdr_double() 289
xdr_elem parameter on RPC call, on xdr_vector() 316
xdr_enum() 290
xdr_float() 292
xdr_free() 293
xdr_getpos() 294
xdr_inline() 295
xdr_int() 296
xdr_long() 297
xdrOpaque_auth() 299
xdr_pointer() 298
xdr_pmaplist() 301
xdr_pmaplist() 302
xdr_reference() 303
xdr_replayed_reply() 304
xdr_repliedmsg() 305
Communicating Your Comments to IBM

If you especially like or dislike anything about this document, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this document. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

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

Please send your comments to us in either of the following ways:

- If you prefer to send comments by FAX, use this number: 1+919-254-1258
- If you prefer to send comments electronically, use this address: comsvrcf@us.ibm.com
- If you prefer to send comments by post, use this address:
  International Business Machines Corporation
  Attn: z/OS Communications Server Information Development
  P.O. Box 12195, 3039 Cornwallis Road
  Department AKCA, Building 501
  Research Triangle Park, North Carolina 27709-2195

Make sure to include the following in your note:

- Title and publication number of this document
- Page number or topic to which your comment applies.