z/OS V1R4 Communications Server
Network Management
User's Guide

Preface

This document applies to z/OS ™ Communications Server (5694-A01).

How this document is organized

This document is organized by function:

Chapter 1 - Planning for Network Management
Chapter 2 - Application interfaces for network monitoring
Chapter 3 - Application interface for formatting packet and data trace records
Chapter 4 - Application interface for monitoring TCP/UDP end points and TCP/IP storage
Chapter 5 - Application interface for SNA network monitoring data
Chapter 6 - Diagnosis
Appendix A - Record formats
Appendix B - PTF information
Appendix C - File storage locations

Who should read this document

This document is intended to be used by programmers who want to use z/OS Communications Server network management interfaces. Before you use this document, you should have an understanding of z/OS Communications Server IP and SNA (VTAM) components.

Related information

You may need to refer to these documents as you implement this function:

z/OS MVS ™ Interactive Problem Control System (IPCS) Customization, SA22-7595

z/OS MVS Programming: Assembler Services Reference, Volume 1 (ABEND-HSPSERV), SA22-7609

z/*OS Communications Server: IP Configuration Reference*, SC31-8776

z/OS Communications Server: IP System Administrator's Commands, SC31-3881

z/OS Communications Server: IP Diagnosis, SC31-8782

z/OS Communications Server: SNA Network Implementation, SC31- *8777*

z/OS Security Server RACF Security Administrator's Guide, SA22-7683-04

UNIX™ System Services Messages and Codes, SA22-7807

z/OS C/C++ Run-Time Library Reference, SA22-7821

z/OS UNIX System Services Programming: Assembler Callable Services Reference, *SA22-7803-03*

Notices

Any reference to an IBM licensed program in this document does not imply that IBM intends to make them available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of the intellectual property rights of IBM may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, New York
USA 10504-1785

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement or other agreement between us.

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.

Interface information

This document describes the attachment to z/OS Communications Server's Network Management Interfaces. These interfaces will generally be upward compatible. In other words, applications that are successfully using these interfaces on a given release should expect that they will be able to execute on higher releases without any requirement for code changes or recompilation. However, because of the dependencies on detailed design and implementation, it is to be expected that the interfaces described in this document may need to be changed in order to run with new product releases or new system platforms or as a result of service.

Unique attachment content

This document indicates only **unique** actions required when attaching a z/OS Communications Server image via the interfaces described in this document and does **not** provide or discuss z/OS Communications Server on a general level.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

APF
IBM
MVS
RACF
UNIX System Services
z/OS

Table of Contents

Chapter 1 - Planning for Network Management

z/OS Communications Server provides several interfaces that allow network monitor and management applications to obtain information about its network operations, for both TCP/IP and VTAM. These interfaces for z/OS Communications Server TCP/IP, provide the following:

- The capability to programmatically obtain copies of TCP/IP packet and data trace buffers, real-time, as the traces are collected.
- The capability to format the TCP/IP packet trace records collected.
- The capability to obtain:
  - Activation and deactivation events for TCP connections in SMF format and buffered
  - Event information for the FTP and TN3270 clients and servers in SMF format and buffered
- The capability to monitor
  - TCP connection and UDP endpoint activity using a callable API
  - TCP/IP storage usage using a callable API

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 described in this document provide an alternative for collecting some of the TCP/IP SMF records and are expected to perform efficiently. Most of the data provided by the application interface for monitoring TCP/UDP end points and TCP/IP storage described in Chapter 4 can be collected from supported SNMP MIBs. Storage usage information is available through displays and the VTAM Performance Monitor Interface (PMI). When used properly, the interfaces documented in this book provide well-defined and efficient APIs to be used for obtaining management information related to the IP and SNA (VTAM) components of z/OS Communication's Server. They also allow for easy application migration to subsequent z/OS Communication's Server releases. They are targeted for use by responsible network management applications.

The following chapters describe the programming interfaces for these functions in detail, and provide the information required to develop network management applications that use them.

These interfaces:

- Use a client/server model or a called interface.
- Require all 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 document, 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.

Chapter 2 - Application interfaces for TCP/IP network monitoring

The following z/OS Communications Server network management interfaces are described in this chapter:

Name	Description
SYSTCPDA	Network management interface for obtaining TCP/IP real-time packet trace data.
SYSTCPCN	Network management interface for obtaining TCP connection information
SYSTCPSM	Network management interface for obtaining real-time SMF data

These interfaces allow network management applications to obtain data in real-time as well as programmatically. Details for invoking these interfaces and the data provided from them are documented in the following sections. Programmers will 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.

Overview

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 may differ in format depending on the interface.

The information provided by each interface is as follows:

Network management interface for TCP/IP Real-Time Packet Tracing (SYSTCPDA)	This interface provides a means for applications to obtain a copy of network packets (for example, Packet trace records) and/or data trace records that are buffered by the TCP/IP stack's packet/data trace functions. The packet trace and/or data trace function must be enabled with the Vary TCPIP,,PKTTRACE command or Vary TCPIP,,DATTRACE command.
Network Management interface for obtaining TCP connection information (SYSTCPCN)	This interface provides a means for applications to 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 and 2 records). The interface also may be used to provide records describing existing TCP connections. Note that use of this interface does not require TCP/IP SMF recording to be active.
Network Management interface for obtaining real-time SMF data (SYSTCPSM)	The records provided through the interface are type 119 SMF records. The specific subtypes that are provided are: • TN3270 server session initiation and termination records (subtypes 20 and 21). • TSO telnet client connection initiation and termination records (subtypes 22 and 23). • FTP server transfer completion records (subtype 70). • FTP server logon failure records (subtype 72). • FTP client transfer completion records (subtype 3).

	The records above are identical in format to SMF records created by TCP/IP, however they offer several key advantages: 1. They do not require that TCP/IP SMF record capturing is activated. 2. 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.) In addition to the records above, two more records are available across this interface that are not currently available from TCP/IP SMF records processing: • FTP server transfer initiation records (subtype 100). • FTP client transfer initiation records (subtype 101). See Appendix A for the structures and mappings of records 100 and 101.

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 streams socket for each of the above interfaces 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 streams sockets, performing the listen() and waiting for incoming connection requests. Applications wishing to exploit this interface connect to the server's listening AF_UNIX stream socket. Each of the interfaces has its own, distinct AF_UNIX pathname that uniquely identifies the socket to be used by the interface. In the case of SYSTCPDA and SYSTCPSM, once connected, the application will immediately start receiving applicable data. In the case of SYSTCPCN, after connecting, the application must send a record to the server to indicate the type of data it desires, only after which will it start receiving applicable data.
2. Each notification record received by the application over the socket represents a buffer of up to 64K bytes of data being stored by the TCP/IP stack. It is important to understand that the actual SYSTCPDA, SYSTCPCN and SYSTCPSM data is not part of this notification record. After receiving the entire notification record from the AF_UNIX socket, the application must then pass this record along with a user-allocated storage buffer to the EZBTMIC1 API routine provided. EZBTMIC1 will populate the provided storage buffer with the output records related to the interface that the input notification record defines. Once the notification is received over the AF_UNIX socket, the application must invoke EZBTMIC1 (or TMI_CopyBuffer) right away since the buffers are stored in a circular queue by the TCP/IP stack, and may 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 copied using the EZBTMIC1 API call contains the actual data of interest to the application. The format of these buffers, and the records contained therein, are described in the section called "Understanding the common buffer output of TMI_CopyBuffer".

In summary, the application connects to open an AF_UNIX socket pathname that is defined for the network management interface for which it would like to collect information (for example, SYSTCPCN, SYSTCPDA, SYSTCPSN) and receives notification records. It passes these

records to EZBTMIC1 to copy the actual data of interest into the application's storage. The application will then parse 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.

Enabling during configuration

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 *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 may 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, or SYSTCPSM.

For more information refer to *z/OS Communications Server: IP Configuration Reference*.

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.

Connecting to the server

The application wishing to make use of one of the interfaces must connect to the appropriate AF_UNIX streams socket provided by TCP/IP, which acts as the server. The socket pathnames 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 obtaining real-time SMF data (SYSTCPSM)

  `/var/sock/SYSTCPSM.tcpipprocname`

Use either the LE C/C++ API or the UNIX System Services BPX callable services to open AF_UNIX sockets and connect to the given service.

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). For the other two services, the application need take no 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 will send an initial record to the client, identifying the server (see the tmi_init record). After that record is received, the client will be sent tmi_token records representing data buffers. A record will be sent for each data buffer 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 for extended periods of time for incoming tokens.

When the server needs to terminate the connection, it will attempt to send a special termination record (see the tmi_term record) over the socket to the connected application, after which it will close the socket. This termination record describes the reason for closure. In some cases, the server may 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 and SYSTCPCN interfaces, large amounts of data can be generated. Care should be taken in the case of SYSTCPDA not to activate too broad of a packet trace filter option, so as to avoid recording unnecessary data; see the *z/OS Communications Server IP Configuration Reference* and *z/OS Communications Server IP System Administrator's Commands* for details. In the case of SYSTCPCN, the NETMONitor MINLIFETIME TCP/IP profile configuration option may be used to restrict the collection of short-lived connections; see the *z/OS Communications Server IP Configuration Reference* for details.

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 will send a termination record with tmit_termcode = EPIPE to the client, and will close the connection.

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")              */
                                   /* TCP connection request record */
#define TmiHr_Init    0xC9D5C9E3   /* Constant("INIT")              */
                                   /* Connection initialization     */
#define TmiHr_Term    0xE3C5D9D4   /* Constant("TERM")              */
                                   /* Normal connection termination */
#define TmiHr_SmfTok  0xE2D4E3D2   /* Constant("SMTK")              */
                                   /* Token for SMF buffer          */
#define TmiHr_PktTok  0xE2D7D3E2   /* Constant("TPKT")              */
                                   /* Token for packettrc data      */
#define TmiHr_Version1 1           /* Version number                */
```

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):

```
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 the *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 below.
- Initialize all remaining fields to zero.

The two fields *tmicnrq_list* and *tmicnrq_smf* control the data that the SYSTCPCN server will send to the client. These fields should be set as follows:

- *tmicnrq_list*

  If set, the server will send the client zero or more tokens representing data buffers that contain a list of all established TCP connections at the time the client connected. These connections will be represented as type 119 TCP connection initiation SMF records. If this field is set to 0, no such list will be sent to the client.

- *tmicnrq_smf*

  If set, the server will send 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 will not send any tokens representing ongoing connection establishment and closure.

The SYSTCPCN server will wait 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 will never report 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 will send a termination record (see following) with *tmit_termcode* = EINVAL to the client, and will close the connection.

Records sent by the server to the client

For each of the three interfaces, the server sends three types of records to the client:

1. Initialization records
2. Termination records
3. Token records

Each record is described in the sections that follow.

Initialization record

After the client connects to the server, the server sends an initialization record to the client. The initialization record may be recognized as having a *TmiHr_Id* equal *TmiHr_CnRqst*. This record contains miscellaneous information about the server and the stack that the client may choose to use or ignore. This record has the following format, defined in ezbytmih.h (an assembler mapping for this structure is in EZBYTMIA):

```
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)    */
  char              tmii_rsvd[16];  /* Reserved                     */
};
```

The component name, *tmii_comp*, represents the server the client is connected to. This will be one of SYSTCPDA, SYSTCPCN, or SYSTCPSM, depending on the server being accessed.

Termination record

The termination record is sent when the server closes the connection. The termination record may be recognized as having a *TmiHr_Id* equal to *TmiHr_Term*. The connection may be closed as part of normal operation (for example the service is being disabled or the stack is terminating), or it may be closed due to 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 will close the connection. The stack will attempt to send the termination record before it closes the socket. However, under certain abnormal stack termination conditions, it may be unsuccessful; furthermore, if the client's receive buffer is full, it may 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):

```
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                     */
};
```

The possible values for *tmit_termcode* and their explanations are as follows, as defined in errno.h:

Value	Description
0	No error; planned termination. Either this function is being disabled or the TCP/IP stack is ending.
EACCES (111)	The client is not permitted to connect to the server.
EINVAL (121)	The client has sent invalid data to the server.
ENOMEM (132)	The server was unable to allocate necessary storage.
EPIPE (140)	The client has erroneously sent data to the server when the server was not expecting data.
EWOULDBLOCK (1102)	The server could not write to the client socket because the client's receive buffer is full (in which case it is possible that the server may not have been able to write this record and closed the connection).

See the z*/OS UNIX System Services Messages and Codes* for more detail.

The *tmit_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 will be closed by the server.

Token record

The server sends the tmi_token record when a 64k buffer has been filled with records for the given service. The token record may be recognized as having a *TmiHr_Id* equal to *TmiHr_PktTok* (in the case of SYSTCPDA) or *TmiHr_SmfTok* (in the case of SYSTCPCN and SYSTCPSM). 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):

```
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 the following section.

Copying the trace buffer

Use the EZBTMIC1 service to copy the data buffer to the client application's storage. EZBTMIC1 may be invoked through a C function, TMI_CopyBuffer, which calls the callable service. EZBTMIC1 uses the *tmi_token* record just read from the AF_UNIX socket as input to locate and copy the data buffer to the user-provided 64K byte buffer.

EZBTMIC1 - Copy TCP/IP Management Interface Data Buffer

Function

The EZBTMIC1 callable service uses a token provided over a TCP/IP management interface to copy a data buffer into application storage. This service is also referred to as the TMI copy buffer service.

Requirements

Authorization	Supervisor state or problem state, any PSW key Caller must be APF authorized
Dispatchable unit mode	Task
Cross memory mode	PASN = HASN
AMODE	31-bit
ASC mode	Primary mode
Interrupt status	Enabled for interrupts
Locks	Unlocked
Control parameters	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL EZBTMIC1,(Token,
               Bufptr,
               Return_value,
               Return_code,
               Reason_code)
```

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 will be copied.

Type Structure

Length 12

The bufptr parameter is a 12-byte structure describing the address of the buffer:

```
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 zero. 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 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*, the data buffer will contain the unformatted packet trace data records (SYSTCPDA).
- When the token is a *TmiHr_SmfTok*, the data buffer will contain SMF records (SYSTCPCN or SYSTCPSM).

Return_value

Returned parameter

Type Integer

Length Fullword

The name of a fullword in which the TMI buffer copy service returns the results of the request:

- > 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. Refer to *Return_code* and *Reason_code* for more details.

Return_code

Returned parameter

Type Integer

Length Fullword

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:

Return_value	*Return_code*	Meaning
>0	**0**	The request was succesful.
-1	**EACCES**	The application is not APF authorized.
-1	**EBADF**	The token provided to locate a buffer is not a valid token.
-1	**EFAULT**	The address is incorrect.

-1	**EINVAL**	The token provided to locate a buffer does not specify a valid data buffer.
-1	**EILSEQ**	The data buffer described by token has been overwritten and is no longer available.

Reason_code

The name of a full word 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*.

Usage Notes

Compiling and Linking

Assembler mappings for the various records that flow over the AF_UNIX socket may be found in macro EZBYTMIA.

This routine will be in SYS1.CSSLIB as the callable stub EZBTMIC1.

TMI_CopyBuffer - Copy TCP/IP Management Interface Data Buffer

The TMI_CopyBuffer() function copies the 64K byte TMI data buffer described in ***token*** to the application-provided buffer pointed to by *bufptr*. Ezbytmih.h contains this definition.

Format

```
void Tmi_CopyBuffer (struct tmi_header *token,
                     struct bufptr_t *bufptr,
                     int  *retval,
                     int  *retcode,
                     int  *rsncode);
```

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 64K byte buffer provided by the user. The indicated buffer will be overwritten with the contents of the TMI data buffer if the call is successful.

The *tmi_bufptr* structure is a twelve-byte structure describing the address of the buffer.

```
       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      */
};
```

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 sets *retcode* to one of the following values:

retcode

A pointer to a full word 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.

Error code description

Return_code	**Meaning**
EACCES	The application is not APF authorized.
EBADF	The Token provided to locate a buffer is not a valid token.
EFAULT	Using the Buffer parameter as specified would result in an attempt to access storage outside the caller's address space.
EINVAL	The Token provided to locate a buffer does not specify a valid data buffer.
EILSEQ	The data buffer described by Token has been overwritten and no longer available.

rsncode

The address of a full word 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*.

Usage Notes

Character data

Some of the data contained in the TMI data buffer may be system data, such as job names. Such data will be in EBCDIC and the application should be prepared to process it appropriately.

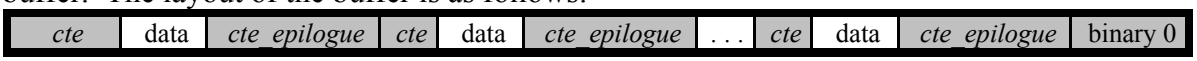
Compiling and linking

The callable service routine that provides this service is provided as a callable stub located in SYS1.CSSLIB.

Understanding the common buffer output of TMI copy buffer service

Upon successful completion of the EZBTMIC1 call of the TMI_CopyBuffer(), the user-supplied 64-k 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 64K data buffers. The *cte* describes the length of the data record. The data record is immediately followed by a *cteeplg (*cte epilogue) structure. The first *cte* structure begins at the beginning of the buffer. The last *cteeplg* 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:



The *cte* is a 16-byte descriptor whose format is as follows (as defined in ezbytmih.h, and in ITTCTE in SYS1.MACLIB):

```
struct cte
{
     unsigned    short     ctelenp;   /* Length of CTE
                                         and cte_epilogue.          */
                 short     cteoff;    /* Offset from start of CTE   */
     unsigned    long      ctefmtid;  /* Format ID of record        */
     unsigned    long long ctetime;   /* STCK timestamp of record
                                         creation                   */
};
```

ctelenp holds the total length of the record, including the *cte*, the data record, and the *cte_epilogue*. *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 a following section. The *ctetime* is an 8-byte STCK timestamp of the time the record was written.

The format of the two-byte *cteeplg* is as follows (as defined in ezbytmih.h, and in ITTCTE in SYS1.MACLIB):

```
struct cteeplg
{
  unsigned    short     ctelene;      /* Length of CTE, data, and
                                         cte_epilogue.              */
};
```

The field *ctelene* holds the same value as the *ctelenp* field in the *cte*.

Format of service-specific data

The sections below describe how to process CTE records for SYSTCPDA, SYSTCPCN, and SYSTCPSM**.**

Processing the CTE records for SYSTCPDA

The following *ctefmtid* values are supported for the SYSTCPDA interface:

ctefmtid	Data Area	Description	Command to Start
1 (TRCIDPKT)	Described by the GtCntl structure	IPv4 packet trace record	**Command to Start**
2 (TRCIDX25)	Described by the GtCntl structure	IPv4 packet trace record	**Command to Start**
3 (TRCIDDAT)	Described by the GtCntl structure	IPv4 data trace record	**Command to Start**
4 (PTHIdPkt)	Described by the PTHDR_T structure	IPv4 or IPV6 packet trace record	**Command to Start**
5 (PTHIdDat)	Described by the PTHDR_T structure	IPv4 or IPV6 data trace record	**Command to Start**

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 GtCntl is defined in EZBCTHDR and contains the following information:

GTCNTL

```
gtseqnum    One byte sequence number
gtsflg      Flag byte

      GTSPKT      0x80  Packet trace request
      GTSX25      0x40  X.25 Data trace request
      GTSDAT      0x20  Data trace request
      GTSVERS     0x10  Version number always 1
      GTSIUTL     0x08  Data from multiple PDUs
      GTSADJ      0x04  Record size adjust by +1
      GTSABBR     0x02  IP pkt was abbreviated
      GTSPOUT     0x01  IP pkt was sent = 1 rcvd = 0

gtslrcd     Lost record count
gtsrect     Record type (device type)

      GTSLCSE     1     IFPETH - Ethernet
      GTSLCS8     2     IFP8023 - 802.3 Ethernet
      GTSLCSE8    3     IFPETHOR - Ether|802.3
      GTSLCSTR    4     IFPTR - Token Ring
      GTSLCSFD    5     IFPFDDI - FDDI
      GTSLU62     6     IFPSNA62 - SNA LU6.2
      GTSHCH      10    IFPHCH - HyperChannel
      GTSCLWRS    21    IFPCLIP - CLAW
      GTSCTC      29    IFPCTC - CTC
      GTSCDLC     30    IFPCDIP - CDLC IP
      GTSATM      32    IFPATM - ATM
      GTSVIPA     33    IFPVIPA - VIPA
      GTSLOOPB    34    IFPLOOPB - LoopBack
      GTSMpc      35    IFPMPC - MPC
      GTSX25C     36    IFPX25 - X.25
      GTSSNALN    37    IFPSNALINK - SNA LINK
      GTSMPCIE    39    IFPIPAQENET - MPC IP AQENET link
      GTSMPCOD    40    IFPOSAFDDI - MPC OSAFDDI link
```

```
      GTSMPCON    41    IFPOSAENET - MPC OSAFNET link
      GTSMPCIH    42    IFPIPAQTR - MPC IPAQTR link
      GTSQIDIO    43    IFPIPAQIDO - iQdio

gtstlen     Total length of IP packet
gtslknm     Link name, or data trace job name
gtssipad    Source IPv4 address
gtsdpad     Destination IPv4 address
gtstod      Time of Day timestamp
gtssport    Source port number (data trace)
Gtsdport    Destination port number (data trace)
Gtstcb      MVS TCB address (data trace)
Gtsasid     ASID (data trace)
```

The PTHDR_T is defined in EZBYPTHA and contains the following information:

PTHDR_T

```
pth_len      Length of the PTHDR_T structure
pth_seqnum   Sequence number of this packet
pth_flag     Flag indicators

        PTH_Adj     0x04            Record size was adjusted by +1 (reflected
                                         in the ctelene and ctelenp). The
data length was odd and a single pad byte was added.
        PTH_Abbr    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.

        PTHLCSE    1                 - Ethernet
        PTHLCS8    2                 - 802.3 Ethernet
        PTHLCSE8   3                 - Ether|802.3
        PTHLCSTR   4                 - Token Ring
        PTHLCSFD   5                 - FDDI
        PTHLU62    6                 - SNA LU6.2
        PTHHCH     10                - HyperChannel
        PTHCLWRS   21                - CLAW
        PTHCTC     29                - CTC
        PTHCDLC    30                - CDLC IP
        PTHATM     32                - ATM
        PTHVIPA    33                - VIRTUAL
        PTHLOOPB   34                - LoopBack
        PTHMpc     35                - MPC
        PTHX25C    36                - X.25
        PTHSNALN   37                - SNA LINK
        PTHMPCIG   38                - MPC giga
        PTHMPCIE   39                - MPC IPAQENET
        PTHMPCOD   40                - MPC OSAFDDI
        PTHMPCON   41                - MPC OSAFNET
        PTHMPCIH   42                - MPC IPAQTR
        PTHQIDIO   43                - iQdio
        PTH6loopb  51                - IPv6 loopback
        PTH6vipa   52                 ifp6vipa
        PTH6ipaqenet  53              ifp6ipaqenet
        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_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 Ascbasid of the sender of the outbound packet. On inbound, this will usually be the asid of the TCP/IP stack

pth_lost Total lost record count

pth_plen Payload length

The fields in the GtCntl and PTHDR_T with the same suffix serve the same purpose in both headers. IPv4 address in pth_src and pth_dst are prefixed with x'000000000000', x'00000000FFFE' or x'00000000FFFF'.

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 two byte *ctelente* field, which has the same value as the *ctelenp.* 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.
In some cases, to obtain the entire IP packet, multiple trace records must be processed. These trace records could span multiple 64K buffers and will probably not be contiguous. In this case, several fields must be examined. See the example of IP record X below. The *ctelenp* 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_tlen* and *pth_time* will be the same for each record in the sequence.

Example of split buffers for IP packet X:

First TMI_CopyBuffer() issued:

64K buffer received

Trace Records			
Record1 for IP packet X **CTE** structure ctelenp=1K	**PTHDR_T** structure pth_seqnum=0 pth_tlen=64K	*trace data* *(IP packet X)* *contains IP headers*	**ctelente=1K**

	pth_time=Time X		

Second TMI_CopyBuffer issued:
64K buffer received

Trace Records			
Record1 for IP packet Y **CTE** structure ctelenp=1K	**PTHDR_T** structure pth_seqnum=0 pth_tlen=ip payload length (less than 1K)	*trace data* *(IP packet Y)* *contains IP headers*	**ctelente=1K**
Record2 for IP packet X **CTE** structure ctelenp=32K	**PTHDR_T** structure pth_seqnum=1 pth_tlen=64K pth_time=Time X	*trace data* *(IP packet X* *continued).* *No headers.*	**ctelente=32K**
Trace Records			

Third TMI_CopyBuffer() issued:
64K buffer received

Trace Records			
Record3 for IP packet X **CTE** structure ctelenp=31K	**PTHDR_T** structure pth_seqnum=2 pth_tlen=64K pth_time=Time X	*trace data* *(IP packet X* *continued).* *No headers.*	**ctelente=31K**

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 will be a complete type 119 SMF record, of subtype 1 or 2.

Additionally, if requested, the server will fill 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 will be indistinguishable from newly established connections (except that

the connection establishment timestamp will be 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 will always be equal to the *subtype* of the SMF record (either 1 or 2) following the CTE in the data buffer.

Applications may use this interface to dynamically maintain a list of active TCP connections. Note that due to 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 may be found in ezasmf.h. Assembler mappings for the structures may 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 server transfer initialization (subtype 100).
- FTP server transfer completion (subtype 70).
- FTP server logon failure (subtype 72).
- TN3270 server session initialization (subtype 20).
- TN3270 server session termination (subtype 21).
- TSO telnet client connection initialization (subtype 22).
- TSO telnet client connection termination (subtype 23).

For the real-time SMF data server, the *ctefmtid* for the CTE will always be equal to the *subtype* of the SMF record (one of the values listed above) following the CTE in the data buffer. The structures and macros for mapping the SMF 119 record subtypes delivered by these interfaces are as follows:

Subtype	**C/C++ header**	**Assembler macro**
3, 20, 21, 22, 23, 70 and 72	*hlq*.SEZANMAC(EZASMF)	*sys*1.maclib(EZASMF77)
100 and 101	*hlq.SEZANMAC*(EZANMFTC)	*hlq.SEZANMAC*(EZANMFTA) Refer to Appendix A for the layout of the FTP Client and Server Transfer Initialization records.

- *hlq* represents the z/OS Communications Server data set high level qualifier.

The FTP client/server transfer initiation records are available only across this interface.

See z/*OS Communications Server: IP Configuration Reference* Appendix D for the formats of SMF type 119 records.

The header files and macros are described in the following table:

Header files for C/C++ programs	**Macros for Assembler programs**	**Contents**
EZBYTMIH	EZBYTMIA	Request and response headers containing the common headers, connection requests, initialization, termination and token records.
EZANMFTC	EZANMFTA	Structures and mappings for the SMF 119 records, subtype 100 and 101. See Appendix A.

These header files and macros are shipped in the *hlq.SEZANMAC* data set (*hlq* refers to the High Level qualifier used when the product was installed on your system). This data set must be available in the concatenation when compiling or assembling a part that makes use of these definitions.

Chapter 3 - Application interface for formatting packet and data trace records

Records collected from the SYSTCPDA interface described in the previous chapter may be formatted programmatically with the EZBCTAPI macro. This chapter describes how the EZBCTAPI interface may be used.

Name	Description
EZBCTAPI	Network management interface for formatting packet trace records

Overview

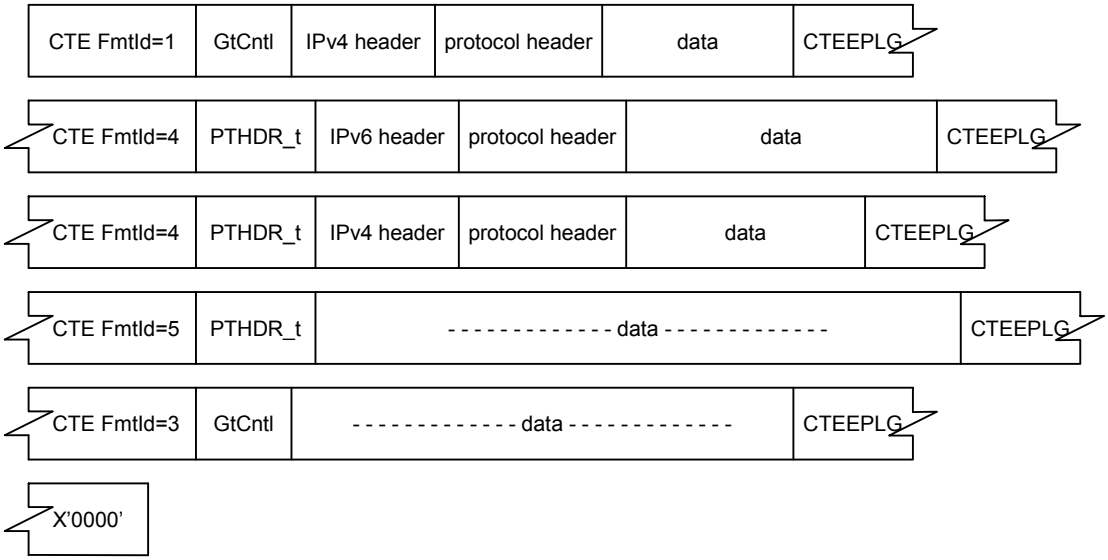
The interface to the formatter described in this chapter provides a means for network applications to format packet and data trace records. An application program can capture a copy of the packet and data trace buffers using the Network Management interface for TCP/IP real-time packet and data tracing (SYSTCPDA), described in Chapter 2.

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) described the EZBCTHDR macro (or the EZBYCTHH header). Types 4 and 5 contain a header (PTHDR_T) described by the EZBYPTHA macro (or the EZBYPTHH header). The table below depicts the layout of the various records.

CteFmtId	Description	Header	IP Header	Protocol	Data	V1R4
1	Packet Trace	GTCNTL	IPv4	variable	variable	Y
2	X25 Trace	GTCNTL	IPv4	variable	variable	N
3	Data Trace	GTCNTL	N/A	N/A	variable	Y (EE only)*
4	Packet Trace	PTHDR_T	IPv6	variable	variable	Y
5	Data Trace	PTHDR_T	N/A	N/A	variable	Y

* EE stands for Enterprise Extender. Read about Enterprise Extender *in z/OS Communications Server: SNA Network Implementation.*

The ABBREV value of the PKTTRACE or DATTRACE command determines the amount of data available. The layout of CTEs in the 64K buffer is below.



Configuration and enablement

There is no formal configuration required to enable this interface.

EZBCTAPI Network management interface for formatting packet trace records

Function

The EZBCTAPI macro accepts parameters to format component trace records from the TCP/IP packet trace 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.

The EZBCTAPI macro allows 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

Requirement: High Level Assembler Language, Version 1 Release 4 or higher is required to use this macro.

Requirements

The requirements for the caller are:

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.

Input register information

Before issuing the EZBCTAPI macro, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register contents
13 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).

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register Contents
0 Reason code, if GPR 15 contains a non-zero return code; otherwise, used as a work register by the system.
1 Used as a work register by the system
2-13 Unchanged
14 Used as a work register by the system
15 Return code

When control returns to the caller, the access registers (ARs) contain:

Register contents
0-1 Used as work registers by the system
2-13 Unchanged
14-15 Used as a work register by the system

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.

Performance implications

None.

Syntax

The EZBCTAPI macro is written as follows:

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).
,LDTO=**stcktime**	**stcktime** : RX-type address or register (2) - (12).
,LSO=**stcktime**	**stcktime**: RX-type address or register (2) - (12).
,MAXLINE=**number**	**number**: RX-type address or register (2) - (12).
,NMCTF=**epaaddr**	**epaaddr**: RX-type address of register (2) - (12)
,OBTAIN=**epaaddr**	**epaaddr** : RX-type address or register (2) - (12).
,OPTIONS=**options**	**options** : RX-type address or register (2) - (12).
,PRTSRV=**epaaddr**	**epaaddr** : RX-type address or register (2) - (12).
,RELEASE=**epaaddr**	**epaddr** : RX-type address or register (2) - (12).
,RETCODE=**retcode**	**retcode**: RX-type address or register (2) - (12), or (15).
,REPORT=**FULL**	**Default:** REPORT=FULL

,REPORT=**SHORT** ,REPORT=**SUMMARY** ,REPORT=**TALLY**	
,RSNCODE=**rsncode**	**rsncode**: RX-type address or register (2) - (12) or (0).
,TABLE=**name**	**name**: RX-type address or register (2) - (12)
,TIME=**GMT** ,TIME=**LOCAL**	**Default:** TIME=LOCAL
,USERTOKEN=**token**	**token**: RX-type address or register (2) - (12).
,MF=(L,**list_addr**) ,MF=(L,**list_addr** ,**attr)** ,MF=G ,MF=(M,**list_addr**)	**list_addr** : RX-type address or register (1) - (12) **Default:** MF=(L,**list_addr** ,0D)
,MF=(M,**list_addr**,COMPLETE) ,MF=(E,**list_addr**)	
,MF=(E,**list_addr**,COMPLETE)	

Parameters

The parameters are explained below. First select one of the four required parameters that define the function that the interface is to perform.

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 calling the filter routine one last time to release all the allocated resources acquired by the formatter.

Next select the optional parameters that you need:

,API=epiaddr

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 may 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 will be loaded by the SETUP function and deleted by the TERM or QUIT function.

,COMP=name

Specifies the location of a eight 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.

,LDTO=stcktime

Specifies the location of eight 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 eight 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=epaddr

Specifies the location of a word that contains the location of the EZBNMCTF stub routine. This may 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 will be 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 four word user token (see USERTOKEN)
- The word where the location of the obtained storage is returned.
- The word with the length of the storage to be obtained.

These 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 the "Passing options to the Packet Trace Formatter" section 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 four word user token (see USERTOKEN)

These 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 *MVS Programming: Assembler Services Reference, Volume 1 (ABEND-HSPSERV).*

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 will be 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 four word user token (see USERTOKEN)
- The word with the location of the storage to be released
- The word with the length of the storage to be obtained.

These 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

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 one line per trace record. Key fields from each qualifying trace record will be printed following the date, time, and entry description.

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.

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 eight (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 may be useful to avoid the overhead of loading and deleting this format table.

,TIME=GMT
,TIME=LOCAL

Specifies the conversion of the time field in the component trace records. The default is TIME=LOCAL.

GMT: The time is shown as Greenwich Mean Time
LOCAL: The time is shown as local time.

,USERTOKEN=token

Specifies the location of a four (4) word field that is copied and passed to the print service routine and the storage functions. The default is four words of zeros.

,WORKAREA=workarea

The location of a 16K work area used by the interface for its control blocks, work area, and save areas. The work area will be 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 16K work area is obtained.

,MF=(L,list_addr)
,MF=(L,list_addr,attr)

Requests that a EZBCTAPI parameter list be defined. **List_addr** is the name assigned to the list. **attr** is an optional attribute used to define the parameter list. The default is **0D**. No other keywords may be used with this macro format.

,MF=G

Requests that the EZBCTAPI_t parameter list description be generated. No other keywords may be used with this macro format.

,MF=(M,list_addr)
,MF=(M,list_addr,COMPLETE)

Request that the EZBCTAPI parameter list be modified. **COMPLETE** requests that the parameter list be zeroed before any modifications.

,MF=(E,list_addr)
,MF=(E,list_addr,COMPLETE)

Requests that the EZBCTAPI parameter list be modified. **COMPLETE** requests that the parameter list be zeroed 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.
Note: COMPLETE does not apply to TERM and QUIT functions.

This matrix shows supported functions and keyword combinations.

Keyword	Input /Output	MF(E) SETUP	MF(E) FORMAT	MF(E) TERM	MF(E) QUIT	MF(M)	MF(L)	MF(G)
WORKAREA	**I**	**X**				**X**		
API	**I**	**X**		**X**	**X**	**X**		
COMP	**I**	**X**	**X**			**X**		
CTE	**I**		**R**			**X**		
LDTO	**I**	**X**				**X**		
LSO	**I**	**X**				**X**		
NMCTF	**I**	**X**	**X**	**X**	**X**			
MAXLINE	**I**	**X**				**X**		
OBTAIN	**I**	**X**				**X**		
OPTIONS	**I**	**X**				**X**		
PRTSRV	**I**	**R**				**X**		
RELEASE	**I**	**X**				**X**		
REPORT	**I**	**X**				**X**		
RETCODE	**O**	**X**	**X**	**X**	**X**			
RSNCODE	**O**	**X**	**X**	**X**	**X**			
TABLE	**I**	**X**				**X**		
TIME	**I**	**X**				**X**		
USERTOKEN	**I**	**X**				**X**		

Legend:

I Input parameter
O Output parameter
R Required parameter
X Optional parameter

ABEND codes

None

Return and reason codes

When control returns from EZBCTAPI, GPR 15 (and retcode, if you coded RETCODE) contains one of the following return codes. GPR 0 (and rsncode, if you coded RSNCODE) might contain one of the following reason codes. The following table displays interface return and reason codes and their meaning.

Hexadecimal return code (CtApi_IRtnCd)	Hexadecimal reason code (CtApi_IRsnCd)	Meaning
00		Function was successful
04		The FORMAT function was not successful
04	10	The SETUP function was not done or did not complete.
04	11	The trace record is not the correct format
04	18	The trace record could not be identified
04	1B	The filter/analysis routine failed.
08		The SETUP function was not successful
08	01	The SETUP function has already initialized the interface
08	02	Print callback function was not provided
08	03	Unable to load format table
08	04	Unable to allocate storage for tables
08	05	Unable to load analysis/format exit
0C	xx	Unknown function code xx
10		Unable to load the function interface
10	04	The EZBCTAPI interface routine could not be found
10	08	An error occurred loading the EZBCTAPI interface routine
14		Unable to obtain storage for a work area
14	04	The program was not able to obtain storage for the work area
18	xxxxxxxx	The interface routine or the analysis routine abended. *xxxxxxxx* is the abend code.

Formatter return and reason codes

Hexadecimal return code (CtApi_FRtnCd)	Hexadecimal reason code (CtApi_FRsnCd)	Meaning
00	-	Normal processing of the entry
04	-	Reread the records from the first
08	-	The current entry is bypassed
0C	-	No further calls to the format/analysis routine
10	-	Ending of the subcommand

These are the return codes described in *z/OS MVS Interactive Problem Control System (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').

To capture trace records, perform the following steps:

1. Update the TCP/IP profile to allow the copying of trace data: 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
3. Start the application program
4. Issue Vary Tcpip,,PKTTRACE or Vary Tcpip,,DATTRACE commands to collect the data of interest.

In the expansion of step 3 above, the application program will do the following:

- Define the format options in the EZBYPTO control block, passed to EZBCTAPI.
- Use the EZBCTAPI macro to setup the packet trace formatter interface.
- Connect an AF_UNIX socket to the SYSTCPDA service (described in the preceding chapter).
- Allocate a 64K buffer.
- In a loop, read 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.
- Call EZBTMIC1 to copy the TCP/IP trace buffer to the application 64K buffer.
- For a return value of zero or negative, read 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.
- Process 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.
- Loop to read the next record from the socket.

At termination, free the 64K buffer, close the socket, and call the TERM function of EZBCTAPI.

Example

Initialize the EZBCTAPI exit environment.

```
*        COPY  EZBCTAPI
EZBCTSMP CSECT
         SAVE  (14,12),,*
         LR    12,15                  SET A BASE REGISTER
         USING EZBCTSMP,12
         LA    15,MAINSA              CHAIN THE SAVE AREA
```

```
         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
         LA    0,EZBYPTO_SZ-4
         STH   0,PTO.PTO_OFFSET
         MVC   PTO.PTO_CBID,=A(PTO_EYEC)
*             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.PTO_REASM,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
*
* INTIALIZE THE EZBCTAPI PARAMETER LIST
         EZBCTAPI WORKAREA=(8),                                        C
               COMP==CL8'SYSTCPDA',                                    C
               PRTSRV==A(PRINTSRV),                                    C
               OPTIONS=APTO,                                           C
               REPORT=FULL,                                            C
               TIME=LOCAL,                                             C
               USERTOKEN=PRINTTKN,                                     C
               MAXLINE==A(L'PRINTBUF-1),                               C
               MF=(M,CTAPIL,COMPLETE)
*
* GET A BUFFER FOR READING BUFFERS
*
         STORAGE OBTAIN,LENGTH=65635
         ST    1,ABUFFER31
*
* SET UP THE FORMATTER INTERFACE
*
         EZBCTAPI SETUP,MF=(E,CTAPIL), SET UP THE INTERFACE            C
               RETCODE=RETCDE,RSNCODE=RETRSN
         LTR   15,15                   DID THIS WORK
         BNZ   ERROR
*
* READ IN A TOKEN
*
LOOP1    DS    0H
         CALL  BPX1RED,(SOCKET,                                        C
               ABUFFER,PRIMARYALET,LBUFTKN,                            C
               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                    WAS DATA MOVED?
         BNZ   LOOP1                    NO, GET NEXT ONE
*
         L     3,ABUFFER31              GET ADDRESS THE BUFFER
         USING CTE,3                    MAP THE BUFFERS
*
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,                                       C
               MF=(E,CTAPIL)
         ALR   3,2                      POINT TO THE NEXT CTE
         B     LOOP2                    DO THE NEXT RECORD
*
EOF      DS    0H
         STORAGE RELEASE,LENGTH=CTAPI_WKSIZE,ADDR=(8)
*                                       GET STORAGE ABDPL WORK AREA
         EZBCTAPI TERM,MF=(E,CTAPIL)
         CLOSE (PRINTDCB)
         L     13,4(13)
         RETURN (14,12),RC=0
*
*
ERROR    DS   0H
*
*
*   DATA
*
         LTORG
MAINSA   DC    18A(0)
         EZBCTAPI MF=(L,CTAPIL)
         EZBCTAPI MF=G
SOCKET   DC    F'0'                     FILE SYSTEM SOCKET NUMBER
ABUFFER  DC    A(BUFTOKEN)
PRIMARYALET    DC    F'0'
LBUFTKN  DS    F                        LENGTH OF BUFFER TOKEN
BUFTOKEN DS    CL64                     A BUFFER TOKEN
RETVAL   DS    F
RETCDE   DS    F
RETRSN   DS    F
BUFPTR   DC    0F
         DC    A(0,0)                   ALET, HI64BITS
ABUFFER31 DC   A(0)                     ADDRESS OF THE BUFFER
*
APTO     DS    CL(EZBYPTO_SZ)           SPACE FOR THE OPTIONS
*
PRINTTKN DC    0F                       TOKEN FOR PRINT SERVICE
         DC    A(PRINTDCB)
         DC    A(PRINTSA)
         DC    A(PRINTBUF)
         DC    A(0)
*
PRINTDCB DCB   DDNAME=SYSPRINT,DSORG=PS,MACRF=PM,                       C
               RECFM=FBA,LRECL=133
*
PRINTBUF DS    0CL133                   A PRINT BUFFER
PRINTCC  DC    C' '
PRINTDAT DC    CL132' '
*
PRINTSA  DC    18A(0)                   A SAVE AREA FOR PRINT SERVICE
*
```

```
*
*
         EJECT
PRINTSRV CSECT
         SAVE  (14,12),,*              SAVE REGISTERS
         LR    12,15                   SET BASE REGISTER
         USING PRINTSRV,12             MAP IT
         LR    2,1                     COPY PARM LIST POINTER
         USING PLIST,2
         LM    2,3,PLIST               GET PLIST POINTERS
         USING PPR2,2
         USING PTKN,3
         LM    4,6,PTKN                GET POINTERS TO STUFF
*              4 ===> DCB
*              5 ===> SAVE AREA
*              6 ===> PRINT BUFFER
         USING PBUF,6                  .
         ST    5,8(,13)                CHAIN THE SAVE AREAS
         ST    13,4(,5)                .
         LR    5,13                    .
*
         L     7,PPR2BUF               GET ADDRESS OF THE BUFFER
         L     8,PPR2BUFL              GET ITS LENGTH
*
         MVI   PBUFLNE-1,C' '          BLANK IT ALL OUT
         MVC   PBUFLNE,PBUFLNE-1       .
         LTR   8,8                     IS THERE A LINE
         BNP   PSRV0001                NO, JUST DO A BLANK LINE
         BCTR  8,0                     TO EXECUTE LENGTH
         EX    8,COPYLINE              COPY LINE OF TEXT
*
PSRV0001 DS    0H
*        L     4,PTKNDCB               GET ADDRESS OF PRINT DCB
         PUT   (4),PBUF                PRINT THE LINE OF TEXT
         L     13,4(,13)               UNCHAIN THE SAVE AREAS
         RETURN (14,12),RC=0           RETURN TO CALLER
*                                      INDICATE PRINT WAS OK
COPYLINE MVC   PBUFLNE(0),0(7)         COPY THE PRINT LINE
*
*
PPR2     BLSUPPR2 DSECT=YES            PPR2 PARAMETER LIST
PLIST    DSECT ,
PLPR2    DS    A                       POINTER TO PPR2 PARM LIST
PLTKN    DS    A                       POINTER TO OUR TOKEN
*
PTKN     DSECT ,                       OUR TOKEN
PTKNDCB  DS    A                       POINTER TO THE DCB
PTKNSA   DS    A                       POINTER TO SAVE AREA
PTKNBUF  DS    A                       POINTER TO BUFFER AREA
*
PBUF     DSECT ,                       OUTPUT BUFFER
PBUFCC   DS    C                       CARRIAGE CONTROL
PBUFLNE  DS    CL132                   OUTPUT LINE
*
         ITTCTE  ,
         EZBYPTO                  COPY FORMAT OPTIONS
         END
```

Passing options to the Packet Trace formatter

The EZBYPTO macro describes a data area that may be passed using the EZBCTAPI OPTIONS keyword. This data area contains flags, values, and pointers that describe packet trace formatter options. The table below 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 *z/OS Communications Server: IP Diagnosis*.

Available EZBYPTO options

Option	**Field setting**
ASCII	Pto_Dump=1;Pto_DmpCd=PtoAscii;
BASIC(DETAIL)	Pto_Basic=1;Pto_BasDtl=1;
BASIC(SUMMARY)	Pto_Basic=1;Pto_BasDtl=0;
BOTH	Pto_Dump=1;Pto_DmpCd=PtoBoth;
CLEANUP(nnnnn)	Pto_Cleanup=1;Pto_GcIntvl=nnnnn;
DUMP	Pto_Dump=1;
DUMP(nnnnn)	Pto_Dump=1;Pto_MaxDmp=nnnnn;
EBCDIC	Pto_Dump=1;Pto_DmpCd=PtoEbcdic;
FORMAT(DETAIL)	Pto_Format=1;Pto_FmtDtl=1;
FORMAT(SUMMARY)	Pto_Format=1;Pto_FmtDtl=0;
FULL	Pto_Dump=1,Pto_Format=1,Pro_FmtDtl=1;
HEX	Pto_Dump=1;Pto_DmpCd=PtoHex;
INTERFACE	Pto_Links@=Addr(list),Pto_Links#=nn
IPADDR(list)	Pto_Addr@=Addr(list);Pto_Addr#=nn
PORT(list)	Pto_Port@=Addr(list);Pto_Port#=nn
REASSEMBLY(nnnnn)	Pto_ReAsm=1;Pto_MaxRsm=nnnnn
REASSEMBLY(DETAIL)	Pto_ReAsm=1;Pto_RsmSum=0
REASSEMBLY(SUMMARY)	Pto_ReAsm=1;Pto_RsmSum=1
NOREASSEMBLY	Pto_ReAsm=0;
SEGMENT	Pto_Segment=1;
NOSEGMENT	Pto_Segment=0;
SESSION(DETAIL)	Pto_SesRpt=Pto_SesDetail; Pto_Session=1;
SESSION(SUMMARY)	Pto_SesRpt=Pto_SesSummary; Pto_Session=1;
SESSION(STATE)	Pto_SesRpt=Pto_SesState; Pto_Session=1;
STATISTICS(DETAIL)	Pto_Stats=1;Pto_StcSum=0;
STATISTICS(SUMMARY)	Pto_Stats=1;Pto_StcSum=1;
STREAMS(nnn)	Pto_Streams=1;Pto_StrmBuf=nnn
STREAMS(DETAIL)	Pto_Streams=1;Pto_StmSum=0;
STREAMS(SUMMARY)	Pto_Streams=1;Pto_StmSum=1;
SUMMARY	Pto_Summary=1;
TALLY	Pto_Stats=1;Pto_StcSum=0;

Notes:

1. A packet may span multiple trace records. When segmented records are encountered, the SEGMENT option recreates 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 will free all unprocessed segments.
2. When the NOSEGMENT option is used only the first segment has the IP header and protocol headers.
3. A packet may 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 zero, 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.

Using the formatter

There are two ways of passing the formatter truncated records so that trace records contain only headers.

1. 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.
2. Shorten the data passed to the formatter. Use these steps:
   a. Determine if the trace record is the first segment of packet. The sequence number field of the header (PTH_SeqNum) will be zero. The record contains the IP header and protocol header (if any). Otherwise the record just contains data.
   b. 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.
   c. Set the PTO_SEGMENT flag to zero. The length also includes the two 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.

The header files and macros are described in the following tables.

Header files for	Macros for	Contents

C/C++ programs	Assembler programs	
n/a	EZBCTAPI	Used to format the records created by the SYSTCPDA interfaces.
EZBYCTHH	EZBCTHDR	Packet trace header describing the TCP/IP packet for types 1, 2, and 3 trace records.
EZBYPTHH	EZBYPTHA	Packet trace header describing the TCP/IP packets for types 4 and 5 trace records.
n/a	EZBYPTO	Describes packet trace options for the formatter.

These header files and macros are shipped in the *hlq.SEZANMAC* data set (*hlq* refers to the High Level qualifier used when the product was installed on your system). This data set must be available in the concatenation when compiling or assembling a part that makes use of these definitions.

Chapter 4 - Application interface for monitoring TCP/UDP end points and TCP/IP storage

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.

Name	Description
EZBNMIFR	Interface to request network management data from TCP/IP

The EZBNMIFR interface can be invoked to obtain the following types of information:

- Active TCP connections.
- Active UDP end points.
- Active TCP listeners.
- TCP/IP storage utilization.

This chapter describes the details for invoking the EZBNMIFR interface with the defined input parameters and for processing the output it provides.

Overview

EZBNMIFR is a callable interface: a program makes a call specifying a request buffer with caller storage allocated to accommodate the returned response buffer. This callable interface to collect data about TCP and UDP end points is a polling type of interface that will show status at a given point in time for selected or all end points, as opposed to an asynchronous interface that will present all state changes. The caller can specify filters to limit the returned data to a specific set of information.

Configuration and enablement

There is no configuration required to enable this interface, as this is a polling interface.

EZBNMIFR - Request network management data from TCP/IP

Function

Request network management data from TCP/IP.

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

Format

Invoke EZBNMIFR, as follows:

For C/C++ callers:

```
EZBNMIFR(TcpipJobName,
                RequestResponseBuffer,
                &RequestResponseBufferAlet,
                &RequestResponseBufferLength,
                &ReturnValue,
                &ReturnCode,
                &ReasonCode);
```

For Assembler callers:

```
CALL EZBNMIFR,(TcpipJobName,
                        RequestResponseBuffer,
                        RequestResponseBufferAlet,
                        RequestResponseBufferLength,
                        ReturnValue,
                        ReturnCode,
                        ReasonCode)
```

Parameters

TcpipJobName

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. 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.

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 EZBNMRHC 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 which 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 which contains the length of request/response buffer.

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 zero indicates the number of output data bytes copied to the response buffer.
- -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:

ReturnValue	ReturnCode	ReasonCode	Meaning
0	0	0	The request was successful.
-1	ENOBUFS	JRBuffTooSmall	The request was not successful. The request/response buffer is too small to contain all of the requested information. Some of the requested information may be returned.
-1	EACCES	JRSAFNotAuthorized	The request was not successful. The caller is not authorized.
-1	EAGAIN	JRTCPNOTUP	The request was not successful. The target TCP/IP stack was not active.
-1	EFAULT	JRReadUserStorageFailed	The request was not successful. A program check occurred while copying input parameters, or while copying input data from the request/response buffer.
-1	EFAULT	JRWriteUserStorageFailed	The request was not successful. A program check occurred while copying output parameters, or while copying output data to the request/response buffer.
-1	EINVAL	JRInvalidValue	The request was not successful. An invalid value was specified in the request/response header.
-1	ETCPERR	JRTcpError	The request was not successful. An unexpected error occurred.

Network Management applications can use any of the following methods to invoke the EZBNMIFR service:

1. 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 (e.g. TCP/IP's SEZALOAD load library), or in LPA.
2. Issue a LINK macro to invoke the EZBNMIFR service. The EZBNMIFR load module must reside in a linklist dataset (for example, TCP/IP's SEZALOAD load library), or in LPA.
3. 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.

TCP/IP Network Management Interface Request/Response Format
The general format of the request is:

- The request header and the request section descriptors (triplets). A triplet 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 a request section element. The following requests can be made:
  - **GetTCPListeners** - obtain information about active TCP listeners.
  - **GetUDPTable** - obtain information about active UDP sockets.
  - **GetConnectionDetail** - obtain information about active TCP connections.
  - **GetStorageStatistics** - obtain information about TCP/IP storage utilization.
- The request sections. The following types of request sections can be specified:

- Filters (optional) - A subset of TCP or UDP end points can be selected based on any combination of the following items:

Filter item	Filter item value
ASID	A 16-bit address space number of a socket application address space.
Resource Name	An EBCDIC job name, right-padded with blanks if less then 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 0 or more characters. e.g. a value of A?C* will match all names with a first character "A" and a third character "C", but will not match two-character names or names beginning with "B" through "Z", etc.
Resource ID	A 32-bit unsigned binary TCP/IP resource identifier ("Client ID" in NETSTAT displays).
Server Resource ID	A 32-bit unsigned binary TCP/IP resource identifier of the related server listening connection.
Local IP Address	A 32-bit IPv4 address or a 128-bit IPv6 address. The local 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 may be specified as either an IPv4 address (e.g., 9.1.2.3) or as an IPv4-mapped IPv6 address (e.g., ::FFFF:9.1.2.3). For all connections, a null address may 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 (::).
Local IP Address Prefix	A 16-bit signed binary value specifying the number of local IP address bits to use, e.g., a value of 12 means that the first 12 bits of a connection's local IP address will be compared to the first 12 bit of the local IP address filter value. A value of 0 means that all address bits will be compared. A value greater than 32 for an IPv4 address, or greater than 128 for an IPv6 address, means that all address bits will be compared.
Local Port	A 16-bit unsigned binary port number.
Remote IP Address	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 filter value is an IPv4 address or an IPv6 address. For IPv4 connections, the remote IP address filter value may be specified as either an IPv4 address (e.g. 9.1.2.3) or as an IPv4-mapped IPv6 address (e.g. ::FFFF:9.1.2.3). For all connections, a null address may 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 (::).
Remote IP Address Prefix	A 16-bit signed binary value specifying the number of remote IP address bits to use, e.g. a value of 12 means that the first 12 bits of a connection's remote IP address will be compared to the first 12 bits of the remote IP address filter value. A value of 0 means that all address bits will be compared. A value greater than 32 for an IPv4 address, or greater than 128 for an IPv6 address, means that all address bits will be compared.
Remote Port	A 16-bit unsigned binary port number.

You can specify up to a maximum of 4 filter elements. Each filter element can contain any combination of the items listed in the table above. A filter element with no applicable items matches any connection. A connection must match all items specified in a filter element to pass that filter check; a connection must pass at least one filter check to be selected. For example:

Filter definition example

Two filters are defined:

1. Local IP Address = 9.0.0.1, Local Port = 5000
2. Resource Name = FTP*

The following TCP connections exist:

1. Resource Name = FTP1, Local IP Address = 9.0.0.2, Local Port = 5001
2. Resource Name = FTP2, Local IP Address = 9.0.0.1, Local Port = 5000
3. 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.

Specifying no filters (triplet offset field is zero, or triplet element count field is zero, or triplet element length field is zero) means that the caller is requesting information for all end points.

The following table shows which filter items are applicable for each request type. If you specify inapplicable filters for a particular request type, they are ignored.

Filter items	**GetTCPListeners**	**GetUDPTable**	**GetConnectionDetail**	**GetStorageStatistics**
ASID	Yes	Yes	Yes	No
Resource name	Yes	Yes	Yes	No
Resource ID	Yes	Yes	Yes	No
Server resource ID	No	No	Yes	No
Local IP address	Yes	Yes	Yes	No
Local IP address prefix	Yes	Yes	Yes	No
Local port	Yes	Yes	Yes	No
Remote IP address	No	No	Yes	No
Remote IP address prefix	No	No	Yes	No
Remote port	No	No	Yes	No

The general format of the response is:

- The response header, 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 buffer, the number of elements in the response section, the length of a response section element, and the total number of connections that passed the request filter checks.
  - The response header has the number of bytes required to contain all the requested data. When the return code is ENOBUFS, use this value to allocate a larger request/response buffer and reissue the request.
- The request sections.
- The response sections. One of the following types of response section will be returned:
  - TCP Connection Information
  - TCP Listener Information
  - UDP Connection Information
  - TCP/IP Storage Statistics

The EZBNMRHC header file contains the NMI request and response data structure definitions for C/C++ programs. The EZBNMRHA macro contains the NMI request and response data structure definitions for Assembler programs.

The C/C++ data structure definitions are:

Typedefs

```
typedef unsigned int           NWM_uint;
typedef unsigned short         NWM_ushort;
typedef unsigned char          NWM_uchar;
typedef unsigned long long     NWM_ull;
```

Triplet

```
typedef struct        /* Network Management Section Triplets        */
{
    NWM_uint   NWMTOffset; /* Offset to section                     */
    NWM_uint   NWMTLength; /* Length of each section element        */
    NWM_uint   NWMTNumber; /* Number of section elements            */
} NWMTriplet;
```

Quadruplet

```
typedef struct        /* Network Management Output Quadruplet       */
{
    NWM_uint   NWMQOffset; /* Offset to section                     */
    NWM_uint   NWMQLength; /* Length of each section element        */
    NWM_uint   NWMQNumber; /* Number of section elements returned   */
    NWM_uint   NWMQMatch;  /* Number section elements that
                              matched filters                       */
/* Number < Match implies that the output buffer is too small    */
} NWMQuadruplet;
```

Request/Response Header

```
typedef struct  {             /* Header overlay NWMHeader           */
   NWM_uint  NWMHeaderIdent;  /* Header Identifier
                                 (required input)                    */
#define NWMHEADERIDENTIFIER 0xD5E6D4C8 /* EyeCatcher "NWMH"          */
   NWM_uint   NWMHeaderLength; /* Length of record header
                                 (required input)                    */
   NWM_ushort NWMVersion;      /* NetWork Monitor Version
                                 (required input)                    */
#define NWMVERSION1    1            /* First version                */
#define NWMCURRENTVER  NWMVERSION1  /* Current version              */
   NWM_ushort NWMType;         /* NetWork Monitor Type
                                 (required input)                    */
#define NWMTCPCONNTYPE   1          /* TCP Connection Rec Type      */
#define NWMTCPLISTENTYPE 2          /* TCP Listen Record Type       */
#define NWMUDPCONNTYPE   3          /* UDP Connection Rec Type      */
#define NWMSTGSTATSTYPE  4          /* Storage Rec Type             */

NWM_uint   NWMBytesNeeded;  /* Length of buffer required to
                               contain all requested data
                               (output)                            */
   char       NWMHeaderRsvd01[20];  /* Reserved                    */
   union {
     NWMTriplet  NWMFiltersDesc;
#define NWM_FILTERNUMBER_MAX 4      /* Maximum number of filters   */
   } NWMInputDataDescriptors;      /* Input section descriptors
                                      (optional input)             */
   union {
   /* The TCP Connection, TCP Listen, UDP Connection, and Storage  */
   /* Statistics sections are only available on output             */
     NWMQuadruplet NWMTCPConnDesc;
     NWMQuadruplet NWMTCPListenDesc;
     NWMQuadruplet NWMUDPConnDesc;
   } NWMOutputDataDescriptors;     /* Output section descriptors
                                      (output)                     */
 } NWMHeader;                      /* NWMHeader                    */
```

Filter Element

```
 typedef struct {
   NWM_uint   NWMFilterIdent;       /* Identifier                  */
#define NWMFILTERIDENTIFIER 0xD5E6D4C6 /* EyeCatcher "NWMF"        */
   NWM_uint   NWMFilterFlags;       /* Bit flags indicating the
                                       items included in the filter */
#define NWMFILTERRESNAMEMASK 0x80000000 /* Resource name included  */
#define NWMFILTERRESIDMASK   0x40000000 /* Resource ID included    */
#define NWMFILTERLCLADDRMASK 0x20000000 /* Local address included  */
#define NWMFILTERLCLPORTMASK 0x10000000 /* Local port included     */
#define NWMFILTERLCLPFXMASK  0x08000000 /* Local prefix included   */
#define NWMFILTERRMTADDRMASK 0x04000000 /* Remote address included */
#define NWMFILTERRMTPORTMASK 0x02000000 /* Remote port included    */
#define NWMFILTERRMTPFXMASK  0x01000000 /* Remote prefix included  */
#define NWMFILTERASIDMASK    0x00800000 /* ASID included           */
#define NWMFILTERLSRESIDMASK 0x00400000 /* Listening server
                                           resource ID included    */
   char       NWMFilterResourceName.8.; /* Resource name           */
   /* All non-blank characters after a * wildcard are ignored      */
   NWM_uint   NWMFilterResourceId;  /* Resource ID                 */
   NWM_uint   NWMFilterListenerId;  /* Listener resource ID        */
   union {
      struct sockaddr_in  NWMFilterLocalAddr4;
```

```
                                  /* AF INET address and port
                                     (sin_family=AF_INET)          */
      struct sockaddr_in6 NWMFilterLocalAddr6;
                                  /* AF_INET6 address, port and
                                     scope
                                     (sin6_family=AF_INET6)        */
   } NWMFilterLocal;              /* Local Address Filter          */
   union {
      struct sockaddr_in  NWMFilterRemoteAddr4;
                                  /* AF_INET address and port
                                     (sin_family=AF_INET)          */
      struct sockaddr_in6 NWMFilterRemoteAddr6;
                                  /* AF_INET6 address, port and
                                     scope
                                     (sin6_family=AF_INET6)        */
   } NWMFilterRemote;             /* Remote Address Filter         */
   NWM_ushort NWMFilterLocalPrefix; /* Local Address prefix number  */
   NWM_ushort NWMFilterRemotePrefix;/* Remote Address prefix number */
   NWM_ushort NWMFilterAsid;      /* ASID                          */
   char       NWMFilterRsvd01[42];  /* Reserved                    */
 } NWMFilter;
```

The following table displays which filter element fields contain filter item data.

Data item	Filter item
NWMFilterResourceName	Resource name
NWMFilterResourceId	Resource ID
NWMFilterListenerId	Server resource ID
NWMFilterLocal	Local IP address and local port
NWMFilterRemote	Remote IP address and remote port
NWMFilterLocalPrefix	Local IP address prefix
NWMFilterRemotePrefix	Remote IP address prefix
NWMFilterAsid	ASID

TCP Connection Element

```
 typedef struct {
   NWM_uint   NWMConnIdent;       /* Identifier                    */
#define NWMTCPCONNIDENTIFIER   0xD5E6D4C3 /* EyeCatcher "NWMC"      */
   union {
      struct sockaddr_in  NWMConnLocalAddr4;
                                  /* AF_INET address               */
      struct sockaddr_in6 NWMConnLocalAddr6;
                                  /* AF_INET6 address              */
   } NWMConnLocal;                /* Local Address                 */
   union {
      struct sockaddr_in  NWMConnRemoteAddr4;
                                  /* AF_INET address               */
      struct sockaddr_in6 NWMConnRemoteAddr6;
                                  /* AF_INET6 address              */
   } NWMConnRemote;               /* Remote Address                */
   NWM_ull    NWMConnStartTime;   /* Connection start time         */
   NWM_ull    NWMConnLastActivity;/* Last time of connection
                                     activity                      */
   NWM_ull    NWMConnBytesIn;     /* Bytes received                */
   NWM_ull    NWMConnBytesOut;    /* Bytes sent                    */
   NWM_ull    NWMConnInSegs;      /* Segments received             */
   NWM_ull    NWMConnOutSegs;     /* Segments sent                 */
```

```
   NWM ushort NWMConnState;         /* State of the TCP Connection   */
#define NWMTCPSTATEKLOSED      1
#define NWMTCPSTATELISTEN      2
#define NWMTCPSTATESYNSENT     3
#define NWMTCPSTATESYNRCVD     4
#define NWMTCPSTATEESTAB       5
#define NWMTCPSTATEFINWAIT1    6
#define NWMTCPSTATEFINWAIT2    7
#define NWMTCPSTATECLOSWAIT    8
#define NWMTCPSTATELASTACK     9
#define NWMTCPSTATECLOSING     10
#define NWMTCPSTATETIMEWAIT    11
#define NWMTCPSTATEDELETTCB    12
   NWM_uchar  NWMConnActiveOpen;    /* 0->Passive open (remote end
                                       issued connect())
                                    1->Active open (local end
                                       issued connect())          */
   NWM_uchar  NWMConnRsvd01;        /* Reserved                   */
   NWM_uint   NWMConnOutBuffered;   /* Number output bytes buffered  */
   NWM_uint   NWMConnInBuffered;    /* Number incoming bytes buffered*/
   NWM_uint   NWMConnMaxSndWnd;     /* Max send window size       */
   NWM_uint   NWMConnReXmtCount;    /* Number retransmitted segments */
   NWM_uint   NWMConnCongestionWnd; /* Congestion window size     */
   NWM_uint   NWMConnSSThresh;      /* Slow-start threshold       */
   NWM_uint   NWMConnRoundTripTime; /* RTT average                */
   NWM_uint   NWMConnRoundTripVar;  /* RTT variance               */
   NWM_uint   NWMConnSendMSS;       /* Max send segment size      */
   NWM_uint   NWMConnSndWnd;        /* Send window                */
   NWM_uint   NWMConnRcvBufSize;    /* Receive buffer size        */
   NWM_uint   NWMConnSndBufSize;    /* Send buffer size           */
   NWM_uint   NWMConnOutOfOrderCount; /* Number out-of-order segments
                                       received                   */
   NWM_uint   NWMConnLcl0WindowCount; /* Number of times local
                                       window size set to 0       */
   NWM_uint   NWMConnRmt0WindowCount; /* Number of times remote
                                       window size set to 0       */
   NWM_uint   NWMConnDupacks;       /* Number of duplicate ACKs
                                       received                   */
   NWM_ushort MWNConnRsvd02;        /* Reserved                   */
   NWM_ushort MWNConnAsid;          /* ASID of address space
                                       that opened connection     */
   char       NWMConnResourceName.8.; /* Jobname of address space
                                       that opened connection     */
   NWM_uint   NWMConnResourceId;    /* TCP/IP connection ID       */
   NWM_uint   NWMConnSubtask;       /* Address of TCB in address space
                                       that opened connection     */
   NWM_uchar  NWMConnSockOpt;       /* Socket options             */
#define NWMConnSOCKOPT_SO_REUSERADDR  0x80 /* SO_REUSERADDR        */
#define NWMConnSOCKOPT_SO_OOBONLINE   0x40 /* SO_OOBONLINE         */
#define NWMConnSOCKOPT_SO_LINGER      0x20 /* SO_LINGER            */
#define NWMConnSOCKOPT_T_MSGDONTROUTE 0x10 /* T_MSG_DONTROUTE      */
#define NWMConnSOCKOPT_NO_DELAY       0x08 /* No delay (Nagle off) */
#define NWMConnSOCKOPT_SO_KEEPALIVE   0x04 /* SO_KEEPALIVE         */
#define NWMConnSOCKOPT_TIMING_LINGER  0x02 /* Current timing linger  */
#define NWMConnSOCKOPT_TIMING_KEEPALI 0x01 /* Current timing keep ali*/
   NWM_uchar  NWMConnSockOpt6;      /*                            */
#define NWMConnSOCKOPT_UNICAST_HOPS   0x80 /* Unicast Hops set     */
#define NWMConnSOCKOPT_USEMINMTU      0x40 /* UseMinMtu set        */
#define NWMConnSOCKOPT_RCVHOPLIM      0x20 /* RcvHopLim set        */
#define NWMConnSOCKOPT_V6ONLY         0x10 /* v6Only sock opt      */
   NWM_uchar  NWMConnClusterConnFlag;   /* Sysplex socket flags    */
#define NWMConnINTERNALCLUSTER        0x08 /* Internal             */
#define NWMConnSAMEIMAGE              0x04 /* Same image           */
#define NWMConnSAMECLUSTER            0x02 /* Same cluster         */
```

```
#define NWMConnNOCLUSTER              0x01 /* None                    */
   NWM_uchar  NWMConnProto;          /* 0=Non-TELNET connection      */
#define NWMConnPROTO_TN3270E      0x04     /* TN3270E mode           */
#define NWMConnPROTO_TN3270       0x02     /* TN3270 mode            */
#define NWMConnPROTO_LINE_MODE    0x01     /* Line mode              */
   char       NWMConnTargetAppl[8]; /* TELNET target application name*/
   char       NWMConnLuName[8];     /* TELNET LU name               */
   char       NWMConnClientUserId[8]; /* TELNET user client name    */
   char       NWMConnLogMode[8];   ]/* TELNET LOGMODE name          */
   NWM_uint   NWMConnTimeStamp;     /* Most recent timestamp from
                                       partner                      */
   NWM_uint   NWMConnTimeStampAge;  /* When most recent timestamp
                                       from partner was updated     */
   NWM_uint   NWMConnServerResourceId; /* Resource ID of related
                                          load-balancing server     */
   char       NWMConnIntfName[16];       /* Interface name          */
 } NWMConnEntry;
```

The following table displays the data returned in TCP connection element fields.

Data Item	***Description***
NWMConnLocal	The local IP address and port, in sockaddr format, for this TCP connection.
NWMConnRemote	The remote IP address and port, in sockaddr format, for this TCP connection.
NWMConnStartTime	The time, in MVS TOD clock format, when this connection was started.
NWMConnLastActivity	The time, in MVS TOD clock format, of the last activity on this connection.
NWMConnBytesIn	The number of bytes received from IP for this connection.
NWMConnBytesOut	The number of bytes sent to IP for this connection.
NWMConnInSegs	The number of segments received from IP for this connection.
NWMConnOutSegs	The number of segments sent to IP for this connection.
NWMConnState	The state of the TCP Connection: - 3=Syn sent - 4=Syn received - 5=Established - 6=FIN wait 1 - 7=FIN wait 2 - 8=Close wait - 9=Last Ack - 10=Closing
NWMConnActiveOpen	Type of open performed: - 0=Passive open (remote end initiated the connection) - 1=Active open (local end initiated the connection)
NWMConnOutBuffered	Number of outgoing bytes buffered.
NWMConnInBuffered	Number of incoming bytes buffered.
NWMConnMaxSndWnd	Maximum send window size.
NWMConnReXmtCount	Number of times segments have been retransmitted.
NWMConnCongestionWnd	Congestion window size.
NWMConnSSThresh	Slow start threshold.
NWMConnRoundTripTime	The amount of time that has elapsed, in milliseconds, from when the last TCP segment was transmitted by the TCP Stack until the ACK was received.
NWMConnRoundTripVar	Round trip time variance.
NWMConnSendMSS	Maximum Segment Size we can send.
NWMConnSndWnd	Send Window size**.**

NWMConnRcvBufSize	Receive buffer size.
NWMConnOutOfOrderCount	The number of out-of-order segments received.
NWMConnLcl0WindowCount	The number of times local window size closed to 0.
NWMConnRmt0WindowCount	The number of times remote window size closed to 0.
NWMConnDupacks	Number of duplicate ACKs received for this connection.
NWMConnAsid	The MVS Address space ID of the address space that opened the socket..
NWMConnResourceName	Resource Name is the text identification of this resource. It represents the user who opened the socket and is updated again during the bind processing.
NWMConnResourceID	Resource ID is the numeric identification of this resource. This value is also known as the connection ID.
NWMConnSubtask	The address of the TCB in the address space that opened the socket.
NWMConnSockOpt	Socket option flags: -bit(1) = SO_REUSEADDR option -bit(2) = SO_OOBINLINE option -bit(3) = SO_LINGER option -bit(4) = T_MSGDONTROUTE -bit(5) = No delay (Nagle off) option -bit(6) = SO_Keepalive option -bit(7) = Currently timing linger -bit(8) = Currently timing keep alive
NWMConnSockOpt6	Socket option flags: -bit(1) = Unicast Hops set -bit(2) = UseMinMtu set -bit(3) = RcvHopLim set -bit(4) = v6Only
NWMConnClusterConnFlag	This flag contains sysplex cluster connection types for this connection: -bit(1) = getsockopt(clusterconntype) requested -bit(2 - 4) = <reserved> -bit(5) = cluster internal -bit(6) = same image -bit(7) = same cluster -bit(8) = none
NWMConnProto	This flag will indicates the following Telnet modes: -bit(1 - 5) = <reserved> -bit(6) = TN3270E mode -bit(7) = TN3270 mode -bit(8) = line mode
NWMConnTargetAppl	The Target VTAM Application name if the TCP connection is for a TN3720 or TN3270E session.
NWMConnLuName	The VTAM LU name if the TCP connection is for a TN3270 or TN3270E session.
NWMConnClientUserId	The Client's userid if the TCP connection is for a TN3720 or TN3270E session.
NWMConnLogMode	The VTAM Logmode if the TCP connection is for a TN3270 or TN3270E session.
NWMConnTimeStamp	Most recent timestamp value, in milliseconds, received from the remote side of the connection.
NWMConnTimeStampAge	Time, in milliseconds, when most recent timestamp from partner was updated.
NWMConnServerResourceId	The numeric identification of the server (i.e., listener connection) associated with this client connection, if any.
NWMConnIntfName	Name of the interface over which the last outbound segment was sent.

TCP Listener Element

```
 typedef struct {
   NWM_uint   NWMTCPLIdent;         /* Identifier                    */
#define NWMTCPLISTENIDENTIFIER 0xD5E6D4E3 /* EyeCatcher "NWMT"       */
   union {
      struct sockaddr_in  NWMTCPLLocalAddr4;
                                    /* AF_INET address               */
      struct sockaddr_in6 NWMTCPLLocalAddr6;
                                    /* AF_INET6 address              */
   } NWMTCPLLocal;                  /* Local Address                 */
   NWM_ushort NWMTCPLRsvd01;        /* Reserved                      */
   NWM_ushort NWMTCPLAsid;          /* ASID                          */
   char       NWMTCPLResourceName.8.; /* Resource name              */
   NWM_uint   NWMTCPLResourceID;    /* Resource ID                   */
   NWM_uint   NWMTCPLSubtask;       /* Address of TCB in address space
                                       that opened connection        */
   NWM_uint   NWMTCPLAcceptCount;   /* Number connections accepted   */
   NWM_uint   NWMTCPLExceedBacklog; /* Number connections dropped    */
   NWM_uint   NWMTCPLCurrBacklog;   /* Current connections in backlog*/
   NWM_uint   NWMTCPLMaxBacklog;    /* Max backlogs allowed          */
   NWM_uint   NWMTCPLCurrActive;    /* Number of current connections */
   NWM_ull    NWMTCPLStartTime;     /* Listener start time           */
   NWM_ull    NWMTCPLLastActivity;  /* Last time connection processed*/
   NWM_ull    NWMTCPLLastReject;    /* Last time connection rejected
                                       due to backlog exceeded       */
 } NWMTCPListenEntry ;
```

The following table displays the data returned in TCP Listener element fields.

Data Item	***Description***
NWMTCPLLocal	The local IP address and port, in sockaddr format, for this TCP connection. In the case of a listener which is willing to accept connections for any IP interface associated with the node, an IP address of INADDR_ANY or IN6ADDR_ANY is used.
NWMTCPLAsid	The MVS Address space ID of the address space that opened the socket..
NWMTCPLResourceName	Resource Name is the text identification of this resource. It represents the user who opened the socket and is updated again during the bind processing.
NWMTCPLResourceID	Resource ID is the numeric identification of this resource. This value is also known as the connection ID.
NWMTCPLSubtask	The address of the TCB in the address space that opened the socket.
NWMTCPLAcceptCount	The total number of connections accepted by this listener.
NWMTCPLExceedBacklog	The total number of connections dropped by this listener due to backlog exceeded.
NWMTCPLCurrBacklog	The current number of connections in backlog.
NWMTCPLMaxBacklog	The maximum number of connections allowed in backlog at one time.
NWMTCPLCurrActive	The number of current connections.
NWMTCPLStartTime	The time, in MVS TOD clock format, that the listener started.
NWMTCPLLastActivity	The time, in MVS TOD clock format, that a connection was last processed.
NWMTCPLLastReject	The time, in MVS TOD clock format, that a connection was last rejected due to backlog exceeded.

UDP Connection Element

```
 typedef struct {
```

```
   NWM uint   NWMUDPCIdent;         /* Identifier                    */
#define NWMUDPCONNIDENTIFIER   0xD5E6D4E4 /* EyeCatcher "NWMU        */
   union {
      struct sockaddr_in  NWMUDPCLocalAddr4;
                                    /* AF_INET address               */
      struct sockaddr_in6 NWMUDPCLocalAddr6;
                                    /* AF_INET6 address              */
   } NWMUDPCLocal;                  /* Local Address                 */
   union {
      struct sockaddr_in  NWMUDPCRemoteAddr4;
                                    /* AF_INET address               */
      struct sockaddr_in6 NWMUDPCRemoteAddr6;
                                    /* AF_INET6 address              */
   } NWMUDPCRemote;                 /* Remote Address                */
   NWM_ull    NWMUDPCStartTime;     /* Connection start time         */
   NWM_ull    NWMUDPCLastActivity;  /* Last time of connection
                                       activity                      */
   NWM_ull    NWMUDPCDgramIn;       /* Number datagrams received     */
   NWM_ull    NWMUDPCBytesIn;       /* Number byptes received        */
   NWM_ull    NWMUDPCDgramOut;      /* Number datagrams sent         */
   NWM_ull    NWMUDPCBytesOut;      /* Number byptes sent            */
   NWM_ushort NWMUDPCRsvd01;        /* Reserved                      */
   NWM_ushort NWMUDPCAsid;          /* ASID                          */
   char       NWMUDPCResourceName[8]; /* Resource name              */
   NWM_uint   NWMUDPCResourceId;    /* Resource Identifier           */
   NWM_uint   NWMUDPCSubtask;       /* Hexadecimal subtask number    */
   NWM_uchar  NWMUDPCSockOpt;       /* Socket options                */
#define NWMUDPCSOCKOPT_BROADCAST    0x80 /* Allow broadcast          */
#define NWMUDPCSOCKOPT_LOOPBACK     0x40 /* Allow loopback           */
#define NWMUDPCSOCKOPT_BYPASSRTE    0x20 /* Bypass Normal Routing    */
#define NWMUDPCSOCKOPT_ICMPFWD      0x10 /* Forward ICMP (PASCAL)    */
#define NWMUDPCSOCKOPT_SENDMULTI    0x08 /* Allow outgoing multicast */
#define NWMUDPCSOCKOPT_RECVMULTI    0x04 /* Allow incoming multicast */
    NWM_uchar  NWMUDPC6SockOpt1;        /* Socket option1           */
#define NWMUDPC6SOCKOPT1_AF_INET6    0x80 /* AF_INET6 family         */
#define NWMUDPC6SOCKOPT1_V6ONLY      0x40 /* IPV6_V6ONLY             */
#define NWMUDPC6SOCKOPT1_RCVPKT      0x20 /* IPV6_RECVPKTINFO        */
#define NWMUDPC6SOCKOPT1_RCVHOP      0x10 /* IPV6_RECVHOPLIMIT       */
#define NWMUDPC6SOCKOPT1_MINMTU      0x08 /* IPV6_USE_MIN_MTU        */
#define NWMUDPC6SOCKOPT1_SENDPKTADDR 0x04 /* IPV6_PKTINFO src IP@    */
#define NWMUDPC6SOCKOPT1_SENDPKTINTF 0x02 /* IPV6_PKTINFO PIF index  */
#define NWMUDPC6SOCKOPT1_HOPLIMIT    0x01 /* IPV6_UNICAST_HOPS       */
   NWM_uchar  NWMUDPC6SockOpt2;     /* Socket option2                */
#define NWMUDPC6SOCKOPT1_USEMINMTU   0x80 /* IPV6_USE_MIN_MTU        */
   NWM_uchar  NWMUDPCRsvd02;        /* Reserved                      */
   NWM_uint   NWMUDPCSendLim;       /* Send limit                    */
   NWM_uint   NWMUDPCRecvLim;       /* Receive Limit                 */
   NWM_uint   NWMUDPCReadQueueCount;     /* Number of datagrams on
                                            read queue               */
   NWM_uint   NWMUDPCReadQueueByteCount; /* Number of data bytes
                                            read queue               */
   NWM_uint   NWMUDPCReadQueueLimit;     /* Maximum number of
                                            datagrams allowed on
                                            read queue               */
   NWM_uint   NWMUDPCReadQueueByteLimit; /* Maximum number of
                                            data bytes allowed on
                                            read queue               */
   NWM_uint   NWMUDPCReadQueueLimitDiscards; /* Number of datagrams
                                                discarded due to
                                                queue limits         */
 } NWMUDPConnEntry;
```

The following table displays the data returned in UDP connection element fields.

Data Item	*Description*
NWMUDPCLocal	The local IP address and port, in sockaddr format, for this UDP connection.
NWMUDPCRemote	The remote IP address and port in sockaddr format, for this UDP connection. If no connect() was done, this sockaddr will contain all zeroes.
NWMUDPCStartTime	The time, in MVS TOD clock format, when this connection was started.
NWMUDPCLastActivity	The time, in MVS TOD clock format, of the last activity on this connection.
NWMUDPCDgramIn	Number of received datagrams.
NWMUDPCBytesIn	Number of bytes received.
NWMUDPCDgramOut	Number of sent datagrams.
NWMUDPCBytesOut	Number of bytes sent.
NWMUDPCAsid	The MVS Address space ID of the address space that opened the socket.
NWMUDPCResourceName	Resource Name is the text identification of this resource. It represents the user who opened the socket and is updated again during the bind processing.
NWMUDPCResourceID	Resource ID is the numeric identification of this resource. This value is also known as the connection ID.
NWMUDPCSubtask	The address of the TCB in the address space that opened the socket.
NWMUDPCSockOpt	IPv4 UDP Socket options: -bit(1) = allow broadcast address -bit(2) = allow loopback of datagrams -bit(3) = bypass normal routing -bit(4) = forward ICMP message (Pascal) -bit(5) = outgoing multicast datagrams -bit(6) = incoming multicast datagrams -bit(7) = <reserved> -bit(8) = <reserved>
NWMUDPC6SockOpt1	IPv6 UDP Socket options: -bit(1) = AF_INET6 family -bit(2) = IPV6_V6ONLY -bit(3) = IPV6_RCVPKT -bit(4) = IPV6_RCVHOP -bit(5) = IPV6_MINMTU -bit(6) = IPV6_SENDPKTADDR -bit(7) = IPV6_SENDPKTINTF -bit(8) = IPV6_HOPLIMIT
NWMUDPC6SockOpt2	IPv6 UDP Socket Options: -bit(1) = IPv6 Use minimum MTU
NWMUDPCSendLim	Maximum transmit datagram size.
NWMUDPCRecvLim	Maximum received datagram size.
NWMUDPCReadQueueCount	Number of datagrams on read queue.
NWMUDPCReadQueueByteCount	Number of data bytes on read queue.
NWMUDPCReadQueueLimit	Maximum number of datagrams allowed on read queue.
NWMUDPCReadQueueByteLimit	Maximum number of data bytes allowed on read queue.
NWMUDPCReadQueueLimitDiscards	Number of datagrams discarded due to queue limits.

TCP/IP Storage Statistics Element

```
 typedef struct {
   NWM_uint   NWMStgIdent;          /* Identifier                    */
#define NWMSTORAGESTATSIDENTIFIER   0xD5E6D4E2 /* EyeCatcher "NWMS"  */
   NWM_uint   NWMStgRsvd01;         /* Reserved                      */
   NWM_ull    NWMStgECSACurrent;    /* Current number of ECSA storage
                                       bytes allocated               */
   NWM_ull    NWMStgECSAMax;        /* Maximum number of ECSA storage
                                       bytes allocated since the
                                       TCP/IP stack was started      */
   NWM_ull    NWMStgECSALimit;      /* Maximum number of ECSA storage
                                       bytes allowed as specified on
                                       the GLOBALCONFIG statement in
                                       the TCP/IP profile
                                       A value of zero indicates that
                                       there is no limit             */
   NWM_ull    NWMStgPrivateCurrent; /* Current number of authorized
                                       private subpool storage bytes
                                       allocated                     */
   NWM_ull    NWMStgPrivateMax;     /* Maximum number of authorized
                                       private subpool storage bytes
                                       allocated since the TCP/IP
                                       stack was started             */
   NWM_ull    NWMStgPrivateLimit;   /* Maximum number of authorized
                                       private subpool storage bytes
                                       allowed as specified on the
                                       GLOBALCONFIG statement in the
                                       TCP/IP.profile. A value of zero
                                       indicates that there is no
                                       limit                         */
 } NWMStgStatEntry;
```

The following table displays the data returned in TC/IP storage statistics element fields.

Data Item	***Description***
NWMStgECSACurrent	Current number of ECSA storage bytes allocated.
NWMStgECSAMax	Maximum number of ECSA storage bytes allocated since the TCP/IP stack was started.
NWMStgECSALimit	Maximum number of ECSA storage bytes allowed as specified on the GLOBALCONFIG statement in the TCP/IP profile. A value of zero indicates that there is no limit.
NWMStgPrivateCurrent	Current number of authorized private subpool storage bytes allocated.
NWMStgPrivateMax	Maximum number of authorized private subpool storage bytes allocated since the TCP/IP stack was started.
NWMStgPrivateLimit	Maximum number of authorized private subpool storage bytes allowed as specified on the GLOBALCONFIG statement in the TCP/IP.profile. A value of zero indicates that there is no limit.

The header file and macro are described in the following table:

Header file for C/C++ programs	**Macros for Assembler programs**	**Contents**

EZBNMRHC	EZBNMRHA	The NMI request and response data structure definitions.

These header files and macros are shipped in the *hlq.SEZANMAC* data set (*hlq* refers to the High Level qualifier used when the product was installed on your system). This data set must be available in the concatenation when compiling or assembling a part that makes use of these definitions.

Example

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 (which starts on page 48):

```
/*********************************************************************/
/*                                                                   */
/* 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;
/*****************************************************************/
/*                                                               */
/* Format filter 2                                               */
/*                                                               */
/*****************************************************************/
NMIbuffer->NMIfilter[2].NWMFilterIdent=NWMFILTERIDENTIFIER;
NMIbuffer->NMIfilter[2].NWMFilterFlags=NWMFILTERRESNAMEMASK;
NMIbuffer->NMIfilter[2].NWMFilterResourceName="FTP*    ";
memcpy(NMIbuffer->NMIfilter[2].NWMFilterResourceName,"FTP*    ",8);
/*****************************************************************/
/*                                                               */
/* Invoke NMI service                                            */
/*                                                               */
/*****************************************************************/
EZBNMIFR(TcpipJobName,NMIbuffer,&NMIalet,&NMIlength,&RV,&RC,&RSN);
```

Chapter 5 - Application interface for SNA network monitoring data

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: information about all EE connections or a desired set of EE connections as specified by the application using the local IP address or hostname and/or the remote IP address or hostname.
- Enterprise Extender summary data: information comprising a summary of EE activity for this host.
- High Performance Routing (HPR) connection data: 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: CSM storage pool statistics and CSM summary information.

Overview

The SNA Network Management Interface is a client network management application polls for information through specific requests via an AF_UNIX streams socket connection using VTAM as the server for that socket. The requested data is provided to the application directly via the AF_UNIX streams socket connection. This interface does not support IPv6 addresses in z/OS Communications Server V1R4.

Configuration

The z/OS system administrator may restrict access to this interface by defining the RACF (or equivalent external security manager product) resource IST.NETMGMT.*sysname*.*vtamprocname*.SNAMGMT in the SERVAUTH class.

- *sysname* represents the MVS system name where the interface is being invoked.
- *vtamprocname* represents the job name associated with the VTAM started task procedure.

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:

1. The administrator must define an OMVS segment for VTAM if one is not already defined.
2. The VTAM OMVS user ID must have write access to the /var directory.

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 may be specified in any of the following ways:

- Using the START command for VTAM
  1. IBM default value is NO
  2. Within the supplemental VTAM Start list (ATCSTRxx, if LIST=xx entered) as SNAMGMT=YES or SNAMGMT=NO
  3. START command options entered by operator as SNAMGMT=YES or SNAMGMT=NO

- 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
DISPLAY NET,VTAMOPTS,OPTION=SNAMGMT
DISPLAY NET,VTAMOPTS,FUNCTION=VTAMINIT

Connecting to the server

The application wishing to make use of this interface must connect to the AF_UNIX streams socket provided by the VTAM server for this interface. The socket pathname is /var/sock/SNAMGMT.

Either the LE C/C++ API or the UNIX System Services BPX services may be used to create AF_UNIX sockets and connect to this service.

When an application connects to the socket, the VTAM server will send an Initialization record to the client application. When VTAM closes a client connection (reasons for doing so include severe errors in the format of data requests sent by the application to the VTAM server, the disabling of the interface by the VTAM operator, and VTAM termination), VTAM will attempt to send a termination record to the client application before closing the connection. Both the Initialization and Termination Records conform in structure to the solicited response records sent by VTAM to the application (see **SNA Network Management Interface (NMI) Request/Response Format**, below).

The initialization record contains the following information:

- VTAM level
- Time and date VTAM was started
- Flags indicating functions supported by this VTAM

The termination record contains the following information:

- Return Code
- Reason Code

SNA Network Management Interface (NMI) Request/Response Format

This interface uses a request/response method over the socket. The application will build and send an NMI request over the socket. The request specifies the type of information to be received and may contain data filters. The application must issue a receive to get the NMI response over the socket. The NMI response will provide 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 will result 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 application interfaces for network monitoring described in Chapter 2, 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.

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, which 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.

  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 the following table. z/OS Communications Server will not perform name resolution (to an IP address) on any supplied hostname, but will simply look for connections that were established using the given hostname.

  Restriction: Support for the Local Hostname and IPv6 filter parameters is release dependent. If the initialization record received by the client when the connection was opened specifies that Local Hostname and IPv6 addresses are not supported by this VTAM level, then the server rejects any request that contains a Local Hostname or IPv6-format addresses.

Local Hostname	An EBCDIC name, right-padded with nulls or blanks if less than 64 characters long. Local Hostname is ignored if Local IP Address is specified. This parameter is release dependent.
Local IP Address	A 32-bit IPv4 address.
Remote Hostname	An EBCDIC name, right-padded with nulls or blanks if less than 64 characters

	long. Remote Hostname is ignored if Remote IP Address is specified.
Remote IP Address	A 32-bit IPv4 address.

- 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 (between one and four filter elements may 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 provided. If a network identifier is not supplied, the partner CP Name is qualified with the host's network ID.
COS Name	An EBCDIC name, right-padded with nulls or blanks if less than 8 characters long. COS is ignored if RTP PU Name provided.

- An EE Summary Request is not permitted to contain any Filter Elements. No filters are applicable to an EE summary request.
- A CSM Statistics Request is not permitted to contain any Filter Elements. No filters are applicable to a CSM statistics request

The following table shows which filter parameters are required, optional, or not applicable (N/A) for each request type. If inapplicable filters are specified for a particular request type, an EE Connection Request or HPR Connection Request, they are ignored. EE Summary Requests or CSM Statistics Requests containing Filter Elements will be rejected by VTAM.

Request Type	**Local IP Address or Hostname**	**Remote IP Address or Hostname**	**RTP PU name or Partner CP name**	**COS name**
EE Connection Request	N/A	Optional, Remote Hostname ignored if Remote IP Address is given	N/A	N/A
EE Summary Request	N/A	N/A	N/A	N/A
HPR Connection Request	N/A	N/A	One is required, Partner CP Name ignored if RTP PU Name given	Optional, ignored if RTP PU Name given
CSM Statistics Request	N/A	N/A	N/A	N/A

Conceptually, every valid request record sent to VTAM by the client will look like this:

General Request format structure:

Common Request/Response Header
Input Triplet information - a single triplet is defined • Offset from start of request header to first input section • Length of each input section of this type • Number of input sections of this type
Start of input information (offset from start of request header to this data given in Input Triplet)

The C/C++ data structure definitions for SNA NMI Requests are contained in the ISTEEHNC header file, and are shown below. The assembler mappings for these structures are in ISTEEHNA.

Request/Response Header

```
/*********************************************************************/
/*                                                                   */
/* Overall EE/HPR Network Management Interface Request and Response  */
/* Header Mapping.                                                   */
/*                                                                   */
/* This header is provided in every EE/HPR NMI request from the      */
/* application client, and on every EE/HPR NMI response from VTAM.   */
/*                                                                   */
/*********************************************************************/

typedef struct {
  unsigned int   EEHNMHeaderIdent;  /* Header EyeCatcher: "EEHH"
                                       (required input/output)   @Q2C*/
  unsigned int   EEHNMHeaderLength; /* Record Header Length (required
                                       input/output)                */
  unsigned short EEHNMVersion;      /* EE/HPR Network Monitor Version
                                       (required input/output)      */
  unsigned short EEHNMType;         /* EE/HPR Network Monitor Type
                                       (required input/output)      */
  char           EEHMNCorrelator[16];   /* Request/response correlator
                                           supplied by client on
                                           request and returned by
                                           VTAM on the response     */
  unsigned int   EEHNMClientID;     /* Internal server identifier
                                       for requesting client        */
  unsigned int   EEHNMReturnCode;   /* Errno value (output)         */
  unsigned int   EEHNMReasonCode;   /* ErrnoJr value (output)       */
  unsigned int   EEHNMRecordLength; /* Overall length of the input
                                       request and/or output response
                                       (required input/output)      */
  unsigned long long EEHNMTimestamp;/* Timestamp when last piece of
                                       data was collected (output).
                                       Format is STCK value, time
                                       is GMT (not local).          */
  char           EEHNMRsvd1[16];    /* Reserved                     */
  EEHNMTriplet   EEHNMInputDataDescriptors; /* Input section
                                               (filter) descriptors  */
  EEHNMTriplet   EEHNMRsvd2;        /* Reserved                     */
  EEHNMQuadruplet EEHNMOutputDataDescriptors;   /* Output section
                                                   descriptors      */
  EEHNMQuadruplet EEHNMRsvd3;       /* Reserved                     */
} EEHNMHeader;
```

Triplets

```
/*****************************************************************/
/* Overall Record Triplet mapping, used for input data           */
/*****************************************************************/

typedef struct {
  unsigned int   EEHNMTOffset;      /* Offset to start of first
                                       section of this type       */
  unsigned short EEHNMTLength;      /* Length of this section     */
  unsigned short EEHNMTNumber;      /* Number of instances of this
                                       section                    */
} EEHNMTriplet;

/*****************************************************************/
/* Internal Record Triplet mapping                               */
/*****************************************************************/

typedef struct {
  unsigned int   EEHNM_RTOffset;    /* Offset from the start of the
                                       record to the first section of
                                       this particular type       */
  unsigned short EEHNM_RTLength;    /* Length of each section of this
                                       particular type            */
  unsigned short EEHNM_RTNumber;    /* Number of instances of this
                                       particular type of section */
} EEHNMRecordTriplet;
```

Quadruplet

```
/*****************************************************************/
/* Overall Record Quadruplet mapping, used for output data       */
/*****************************************************************/

typedef struct {
  unsigned int   EEHNMQOffset;      /* Offset to start of first
                                       record data within the
                                       response. Subsequent records
                                       are found using the length of
                                       the record being processed */
  unsigned int   EEHNMQRsvd1;       /* Reserved for EE/HPR NMI
                                       responses, since the length
                                       of the records are variable
                                       within the response data   */
  unsigned int   EEHNMQNumber;      /* Number of records returned
                                       on this response.  If less than
                                       EEHNMQMatch, then server is
                                       unable to return all of the
                                       record elements that matched
                                       the filters due to VTAM storage
                                       constraints.               */
  unsigned int   EEHNMQMatch;       /* Number of record elements
                                       that matched the filters.  */
} EEHNMQuadruplet;
```

Filter Element

```
Note: z/OS V1R$ CS EE does not support IPv6 address structures.

/*****************************************************************/
/*                                                               */
/* EE/HPR Network Management Interface Request Filter Mapping    */
```

```
/*                                                                   */
/* The client application includes these filters ("pointed" to by    */
/* the input triplet construct) on EE/HPR NMI requests to indicate   */
/* what specific information the server should return for this       */
/* request. The valid filters are request-specific.                  */
/*                                                                   */
/* The server will set the bit EEHNMFilter_FilterCheck on output     */
/* if any filter parameters were included that were inapplicable     */
/* to the request type and thus were ignored by the server.          */
/*                                                                   */
/*******************************************************************/

typedef struct {
  unsigned int EEHNMFilter_Eye;      /* Filter Eyecatcher (FLTR) @Q2C*/

  struct {                           /* Filter flags                */
    unsigned int EEHNMFilter_LocIPF     :1; /* Local IP Address
                                               included             */
    unsigned int EEHNMFilter_RemIPF     :1; /* Remote IP Address
                                               included             */
    unsigned int EEHNMFilter_IPv6F      :1; /* IP Address(es) in
                                               IPv6 format:
                                                 0 = IPv4 format
                                                 1 = IPv6 format    */
    unsigned int EEHNMFilter_LocHNF     :1; /* Local Hostname
                                               included             */
    unsigned int EEHNMFilter_RemHNF     :1; /* Remote Hostname
                                               included             */
    unsigned int EEHNMFilter_RTPPUF     :1; /* RTP PU Name included */
    unsigned int EEHNMFilter_PrtrCPF    :1; /* Partner CP Name
                                               included             */
    unsigned int EEHNMFilter_COSF       :1; /* APPN COS name
                                               included             */
    unsigned int EEHNMFilter_Rsvd1      :23;/* Reserved (set to 0)  */
    unsigned int EEHNMFilter_FilterCheck :1; /* Output indicator set
                                     by server if inapplicable
                                     filters were specified on
                                     request.  The server will
                                     ignore the inapplicable filters
                                     and return data matching the
                                     valid filters.                 */
  } EEHNMFilter_Flags;

  /*****************************************************************/
  /* Following is the specific filter data                         */
  /*                                                               */
  /* For EE Summary requests, no filters are expected.             */
  /*                                                               */
  /* For EE Connection requests, no filters are required.          */
  /*   Optional filters:                                           */
  /*     Local IP address or hostname                              */
  /*     Remote IP address or hostname                             */
  /*                                                               */
  /* For HPR Connection requests, RTP PU name or partner name      */
  /*   is required, and COS is optional when partner name is       */
  /*   also specified.                                             */
  /*****************************************************************/

  union {
    struct in6_addr EEHNMFilter_LocIPv6_ADDR; /* Local IPv6 address  */
    struct {
      char           Rsvd[12];             /* Pad                   */
      struct in_addr Address;              /* Local IPv4 address    */
    } EEHNMFilter_LocIPv4_ADDR;
```

```
  } EEHNMFilter Loc ADDR;

  char EEHNMFilter_Rsvd2[12];       /* Reserved for VTAM usage       */

  union {
    struct in6_addr EEHNMFilter_RemIPv6_ADDR; /* Remote IPv6 address */
    struct {
      char           Rsvd[12];             /* Pad                    */
      struct in_addr Address;              /* Remote IPv4 address    */
    } EEHNMFilter_RemIPv4_ADDR;
  } EEHNMFilter_Rem_ADDR;

  char EEHNMFilter_Rsvd3[12];       /* Reserved for VTAM usage       */

  unsigned char EEHNMFilter_LocHN_Len;      /* Hostname len in bytes */
  char          EEHNMFilter_LocHName[64];   /* Local hostname        */
  unsigned char EEHNMFilter_RemHN_Len;      /* Hostname len in bytes */
  char          EEHNMFilter_RemHName[64];   /* Remote hostname       */

  char          EEHNMFilter_RTPPU[8];       /* RTP PU name           */

  unsigned char EEHNMFilter_PrtrCP_Len;     /* CP name len in bytes  */
  char          EEHNMFilter_PrtrCPName[17]; /* FQ Partner CP Name    */

  char          EEHNMFilter_COS[8];         /* APPN COS name         */
} EEHNMFilter;
```

Constants

```
/*********************************************************************/
/* Eyecatcher constants for EE/HPR Network Management data           */
/*********************************************************************/

const unsigned int EEHNM_ID         /* EE/HPR NMI record data (EEHH) */
            = 0xC5C5C8C8;                                  /*@Q2C*/
const unsigned int EEHNM_FLTR       /* EE/HPR filter record (FLTR)   */
            = 0xC6D3E3D9;                                  /*@Q2C*/
const unsigned int EEHNM_INIT       /* EE/HPR init record (NMII)     */
            = 0xD5D4C9C9;                                  /*@Q2C*/
const unsigned int EEHNM_TERM       /* EE/HPR term record (NMIT)     */
            = 0xD5D4C9E3;                                  /*@Q2C*/

/*********************************************************************/
/* Constant for EEHNMI_Comp                                          */
/*********************************************************************/

const char EEHNMI_Comp_Name[7]      /* EE/HPR network mgmt server    */
            = { '\xE2', '\xD5', '\xC1', '\xD4',
                '\xC7', '\xD4', '\xE3' }; /* SNAMGMT           @Q1C*/

/*********************************************************************/
/* Equates for EEHNMVersion field                                    */
/*********************************************************************/

const int EEHNMVersion1       = 1;  /* Initial EE/HPR service version*/
const int EEHNMCurrentVersion = 1;  /* Current EE/HPR service version*/

/*********************************************************************/
/* Equates for EEHNMType field                                       */
/*********************************************************************/

const int EEHNMInitializationType = 1;  /* Socket conn init data     */
const int EEHNMTerminationType    = 2;  /* Socket conn term data     */
```

```
const int EEHNMEEConnType  = 3;     /* EE connection data processing */
const int EEHNMEESummType  = 4;     /* EE summary data processing    */
const int EEHNMHPRConnType = 5;     /* HPR connection data processing*/
const int EEHNMCSMGlobalType = 6;   /* CSM GLOBAL Statistical Data
                                       processing                @Q1A*/
/*****************************************************************/
/* Miscellaneous equates                                         */
/*****************************************************************/

const int EEHNM_FNumber_Min_EESumm  = 0;    /* Minimum number of
                                       of filter records in a valid
                                       EE Summary Request          */
const int EEHNM_FNumber_Min_EEConn  = 0;    /* Minimum number of
                                       of filter records in a valid
                                       EE Connection Request       */
const int EEHNM_FNumber_Min_HPRConn = 1;    /* Minimum number
                                       of filter records in a valid
                                       HPR Connection request      */
const int EEHNM_FNumber_Max_EESumm  = 0;    /* Maximum number
                                       of filter records in a valid
                                       EE Summary Request          */
const int EEHNM_FNumber_Max_EEConn  = 1;    /* Maximum number
                                       of filter records in a valid
                                       EE Connection Request       */
const int EEHNM_FNumber_Max_HPRConn = 4;    /* Maximum number
                                       of filter records in a valid
                                       HPR Connection request      */
```

All SNA NMI responses flow on the socket from the VTAM server to the client application. The general format of an NMI *response* is:

- 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.
- The request sections.
- The response sections.
  - Response sections of the following solicited response types will be 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” containing information that passed the request filter checks.

The general format of an NMI response section record is:

- 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 associated with this response section record.

An application wishing to navigate 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:

1. EE Summary Response
   - One EE Summary Global Data Section Record.
   - One or more EE Summary IP Address Data Section Records.
2. EE Connection Response
   - One or more EE Connection Data Section Records.
3. HPR Connection Response
   - One HPR Connection Global Data Section Record.
   - One or more HPR Connection Specific Data Section Records.
4. CSM Global Statistics Response
   - One CSM Global Pools Section Record.
   - One CSM Summary Section Record.

Conceptually, every response record sent by VTAM to the client will look like the format that follows.

General Response format structure:

Common Request/Response Header
Input Triplet information (copied from corresponding request, if any) -- a single triplet is defined • Offset from start of response data to first input section • Length of each input section of this type • Number of input sections of this type
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 (if any), then some data was not reported due to 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)

Initialization Record

The structure of the initialization record is below.

Enterprise Extender initialization 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 • 0 • Number of output records included in this response - 1
Start of output information (offset from start of response data saved in Output Quadruplet), specifically one: • Enterprise Extender initialization record

Enterprise Extender initialization record:

Record Identifier (4 chars) --- "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
• Reserved (6 bits) - '000000'B

Reserved (15 bytes) - 0

The C/C++ data structure definitions for the Initialization Response Record are contained in the ISTEEHNC header file, and are shown below. The assembler mappings for these structures are in ISTEEHNA.

```
/*****************************************************************/
/*                                                               */
/* EE/HPR Network Management Interface Initialization Record     */
/*                                                               */
/* This record is used to pass information about the VTAM EE/HPR */
/* Management Server to the client application. This is the first */
/* record written by the server to the client.                   */
/*                                                               */
/*****************************************************************/

typedef struct {
  unsigned int       EEHNMI_Eye;    /* Init record eyecatcher (NMII)
                                                          @Q2C*/
  char               EEHNMI_Level[8];   /* VTAM Level from ATCVT     */
  unsigned long long EEHNMI_Time;       /* TOD VTAM started          */
  char               EEHNMI_Comp[8];    /* Component name            */

  struct {                           /* Functions supported          */
    unsigned int EEHNMI_IPv6_Supp      :1;   /* IPv6 addrs supported */
    unsigned int EEHNMI_Local_Hostname :1;   /* Lcl Hostnm supported */
```

```
    unsigned int EEHNMI Rsvd1          :6;   /* Unused - available   */
  } EEHNMI_Supported;

  char               EEHNMI_Rsvd2[15];  /* Reserved                 */
} EEHNMInit;
```

Termination Record

The structure of the termination record is below.

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 • 0 • Number of output records included in this response - 0

EE Summary Response Record

The structure of the EE Summary response is below.

Enterprise Extender Summary Response format:

Common Request/Response Header
Input Triplet information (copied from request) -- a single triplet is defined • Offset from start of response data to first input section • Length of each input section of this type • Number of input sections of this type
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) • Enterprise Extender Summary IP Address Output Record(s) (one instance per IP address being reported)

Enterprise Extender Summary Global Output Record:

Record Identifier (4 chars) --- "EESG"

Length of overall record (4 bytes)
Reserved field (2 chars)
Number of triplets for this output record (2 bytes) -- 1
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:

Record Identifier (4 chars) --- "EESI"
Length of overall record (4 bytes)
Reserved field (2 chars)
Number of triplets for this output record (2 bytes) **-- 2**
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 hostname used to obtain this IP address, zero if no hostname resolution was performed)

The C/C++ data structure definitions for the EE Summary Response Record are contained in the ISTEESUC header file, and are shown below. The assembler mappings for these structures are in ISTEEHUA.

```
/*****************************************************************/
 /* Enterprise Extender Summary Global record                     */
 /*****************************************************************/
 typedef struct {
   unsigned int      EESumG_Eye;    /* EE Summary Global ID (EESG)
                                                            @Q1C*/
   unsigned int      EESumG_Len;    /* Overall length of this record */
   char              EESUMG_Rsv[2]; /* Reserved                      */
   unsigned short    EESumG_NumTriplets; /* Number of triplets defined
                                       for this record               */
   EEHNMRecordTriplet EESumG_Triplet; /* Only one triplet
                                       defined for this record       */
 } EESumGlobal;                     /* Enterprise Extender Summary
                                       Global record                 */
 /*****************************************************************/
 /* Enterprise Extender Summary Global record                     */
 /*****************************************************************/
 typedef struct {
   struct {
     unsigned char   EESumGD_Low_TOS;  /* Low priority              */
     unsigned char   EESumGD_Medium_TOS; /* Medium priority         */
     unsigned char   EESumGD_High_TOS; /* High priority             */
     unsigned char   EESumGD_Network_TOS; /* Network priority       */
     unsigned char   EESumGD_Signal_TOS; /* Signal priority         */
   } EESumGD_TOS_Info;              /* TOS Information (IPv4) or
                                       traffic class data (IPv6)     */

   char              EESumGD_Rsvd;  /* Reserved                      */
```

```
   struct {
     unsigned short  EESumGD_Port_Num_Low; /* Low priority data      */
     unsigned short  EESumGD_Port_Num_Medium; /* Medium priority data*/
     unsigned short  EESumGD_Port_Num_High;   /* High priority data  */
     unsigned short  EESumGD_Port_Num_Network; /* Network priority   */
     unsigned short  EESumGD_Port_Num_Signal; /* LDLC signals        */
   } EESumGD_Port_Numbers;          /* Port Numbers                 */

   struct {
     unsigned int    EESumGD_Timer_LIVTIME; /* LIVTIME               */
     unsigned int    EESumGD_Timer_SRQTIME; /* SRQTIME               */
     unsigned char   EESumGD_Timer_SRQRETRY; /* SRQRETRY             */
     char            EESumGD_Timer_Rsvd[3]; /* Reserved              */
   } EESumGD_Timer_Info;            /* EE Timer Information         */

 } EESumGlobalData;                 /* Enterprise Extender Summary
                                       Global data section          */
/*****************************************************************/
 /* Enterprise Extender Summary IP Address Record                 */
 /*****************************************************************/
 typedef struct {
   unsigned int      EESumI_Eye;    /* EE Summary IPAddress ID field
                                       (EESI)                 @Q1C*/
   unsigned int      EESumI_Len;    /* Overall length of this record */
   char              EESumI_Rsv[2]; /* Reserved                      */
   unsigned short    EESumI_NumTriplets; /* Number of triplets defined
                                       for this record              */
   EEHNMRecordTriplet EESumI_IPTriplet;  /* First triplet points to
                                       IP specific data             */
   EEHNMRecordTriplet EESumI_HNTriplet;  /* Second triplet
                                       points to hostname data      */
 } EESumIPAddress;                  /* Enterprise Extender Summary
                                       IP Address record format     */

 /*****************************************************************/
 /* Enterprise Extender Summary IP Address Specific               */
 /*****************************************************************/
 typedef struct {
   union {
     struct in6_addr EESumIP_Local_IPv6_Address; /* Local IPv6 address*/
     struct {
       char           Rsvd[12];             /* Pad                  */
       struct in_addr Address;              /* Local IPv4 address   */
     } EESumIP_Local_IPv4_Address;
   } EESumIP_Local_Address;

   char              EESumIP_Rsvd1[12]; /* Reserved for VTAM usage   */

   struct {
     unsigned int    EESumIP_IPv6_Address : 1; /* Local IPAddress
                                       is IPv6
                                       '1'B = IPv6 Address
                                       '0'B = IPv4 Address          */
     unsigned int    EESumIP_IPv6_Rsvd : 7; /* Reserved              */
   } EESumIP_Flags;                 /* Information Flags            */

   char              EESumIP_Rsvd2[3]; /* Reserved                  */

   unsigned int      EESumIP_Num_SRQRETRY_INOPS;  /* Count of the
                                       number of lines that have
                                       INOPed due to SRQRETRY       */

   unsigned int      EESumIP_Num_Active_Total_Conns;  /* Total active
                                     EE connections for this
```

```
                                    IP address                    */
  struct {
    unsigned short  EESumIP_Num_Avail_Lines_PreDefined; /* Total
                                    number of available lines for
                                    predefined connections        */
    unsigned short  EESumIP_Num_Active_PreDefined_Conns; /* Total
                                    number of active predefined
                                    connections                   */
  } EESumIP_PreDefined_Info;       /* Predefined Connection Info
                                    specific to this IP address   */

  struct {
    unsigned short  EESumIP_Num_Local_VRNs; /* Total number of Local
                                    VRNs defined with this
                                    local IP address              */
    unsigned short  EESumIP_Num_Avail_Lines_LVRN; /* Total number of
                                    available lines for Local
                                    VRN connections               */
    unsigned short  EESumIP_Num_Active_LVRN_Conns; /* Total number of
                                    active Local VRN connections  */
  } EESumIP_Local_VRN_Info;        /* Local VRN Info specific to this
                                    IP address                    */
  struct {
    unsigned short  EESumIP_Num_Global_VRNs; /* Total number of Global
                                    VRNs defined with this local
                                    IP address                    */
    unsigned short  EESumIP_Num_Avail_Lines_GVRN; /* Total number of
                                    available lines for Global
                                    VRN connections               */
    unsigned short  EESumIP_Num_Active_GVRN_Conns; /* Total number of
                                    active Global VRN connections */
  } EESumIP_Global_VRN_Info;       /* Global VRN Info for this
                                    specific IP address           */
} EESumIPAddressData;              /* Enterprise Extender Summary
                                    IP Address specific section   */

/*****************************************************************/
/* Enterprise Extender Summary Hostname Section                  */
/*****************************************************************/
typedef struct {
  unsigned char     EESumIH_HostLen;   /* Actual length of hostname */
  char              EESumIH_Hostname[64]; /* Hostname used to resolve
                                    to the local IP address
                                    reported in this EE
                                    Summary record                */
} EESumI_HostnameData;             /* Enterprise Extender Summary
                                    Hostname section mapping      */

/*****************************************************************/
/* Eyecatcher constants for EE Summary records                   */
/*****************************************************************/
const unsigned int EESumG_ID
  = 0xC5C5E2C7;                    /* 'EESG' EE summary global
                                    record                  @Q1C*/
const unsigned int EESumI_ID
  = 0xC5C5E2C9;                    /* 'EESI' EE summary IP
                                    address record          @Q1C*/

/*****************************************************************/
/* Constants for Triplet counts for the various records          */
/*****************************************************************/
const int EESumG_TripletCnt = 1;   /* EE summary global record has
                                    one triplet                   */
const int EESumI_TripletCnt = 2;   /* EE Summary IP address record
```

```
                                        record has two triplets       */
```

EE Connection Response Record

The structure of the response record is as follows:

Common Request/Response Header
Input Triplet information (copied from request) - a single triplet is defned • Offset from start of response data to first input section • Length of each input section of this type • Number of input sections of this type
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: • Enterprise Extender Connection Specific Output Record(s) (one instance per EE connection reported**)**

Enterprise Extender Connection Specific Output Record:

Record Identifier (4 chars) --- "EECO"
Length of overall record (4 bytes)
Reserved field (2 chars)
Number of triplets for this output record (2 bytes) -- 4
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 Enterprise Extender Connection Hostname section(s) (zero-two possible instances, one for local and one for remote hostname if applicable)
Start of Enterprise Extender Connection Associated VRN name section (one instance, only included if the EE connection is across a virtual routing node)
Start of Enterprise Extender Connection Associated RTP PU name section(s) (one instance per RTP PU that is using this EE connection)

The C/C++ data structure definitions for the EE Connection Response Record are contained in the ISTEECOC header file, and are shown below. The assembler mappings for these structures are in ISTEECOA.

```
/*********************************************************************/
```

```
/* Enterprise Extender Connection Data Block                          */
/*********************************************************************/

typedef struct {
  unsigned int   EEConn_Eye;        /* EE Connection ID (EECO)   @Q1C*/
  unsigned int   EEConn_Len;        /* Overall length of this record */
  short          EEConn_Rsvd;       /* Reserved                      */
  unsigned short EEConn_NumTriplets;/* Number of triplets defined
                                       for this record               */
  EEHNMRecordTriplet EEConn_StTriplet; /* First triplet
                                          defines the static section */
  EEHNMRecordTriplet EEConn_HnTriplet; /* Second triplet
                                          defines the associated
                                          hostname section(s)        */
  EEHNMRecordTriplet EEConn_VNTriplet; /* Third triplet
                                          defines the VRN section    */
  EEHNMRecordTriplet EEConn_PUTriplet; /* Fourth triplet
                                          defines the list of
                                          associated RTP PU names    */
} EEConnRecord;

/*********************************************************************/
/* Enterprise Extender Connection static information section         */
/*********************************************************************/

typedef struct {
  union {
    struct in6_addr EEConnS_Local_IPv6_Address; /* Local IPv6 address*/
    struct {
      char           Rsvd[12];             /* Pad                    */
      struct in_addr Address;              /* Local IPv4 address     */
    } EEConnS_Local_IPv4_Address;
  } EEConnS_Local_Address;

  char           EEConnS_Rsvd1[12];        /* Reserved               */

  union {
    struct in6_addr EEConnS_Remote_IPv6_Address; /* Remote IPv6 addr */
    struct {
      char           Rsvd[12];             /* Pad                    */
      struct in_addr Address;              /* Remote IPv4 address    */
    } EEConnS_Remote_IPv4_Address;
  } EEConnS_Remote_Address;

  char           EEConnS_Rsvd[12];        /* Reserved               */

  char           EEConnS_Stack_Name[8]; /* Enterprise Extender
                                           TCP/IP stack name         */
  char           EEConnS_Line_Name[8];  /* Enterprise Extender
                                           Line Name                 */
  char           EEConnS_PU_Name[8];/* Enterprise Extender PU Name   */
  unsigned char  EEConnS_Remote_SAP;/* Remote SAP                    */
  unsigned char  EEConnS_Local_SAP; /* Local SAP                     */

  struct {
    unsigned int EEConnS_IPv6_Address :1;   /* Local and Remote
                                               addresses are IPv6:
                                               1 - Both IPv6 Address
                                               0 - Both IPv4 Address */
    unsigned int EEConnS_Dynamic_PU   :1;   /* Dynamic PU indicator
                                               1 - Dynamic
                                               0 - Predefined        */
    unsigned int EEConnS_KEEPACT      :1;   /* KEEPACT boolean flag  */
    unsigned int EEConnS_DWINOP       :1;   /* DWINOP  boolean flag  */
```

```
  unsigned int EEConnS FlagsRsvd    :4;   /* Reserved             */
} EEConnS_Flags;

unsigned char  EEConnS_REDIAL_Cnt;/* EE Redial Count              */
short          EEConnS_REDIAL_Dly;/* EE Redial Delay in seconds   */
unsigned short EEConnS_Rsvd3;     /* Padding, available           */
unsigned int   EEConnS_Total_LULU_Count;  /* Count of LU-LU sessions
                                             on RTP pipes using this
                                             EE connection        */
  /*****************************************************************
    Outbound Data Transfer Information by Priority:
       Low
       Medium
       High
       Network
       Signal
  *****************************************************************/

struct {
  unsigned long long  EEConnS_SNA_Bytes_Sent_L;  /* Total number of
                                    bytes sent over this EE
                                    connection - LOW priority.
                                    This includes the data bytes
                                    along with NLH, THDR and FID5 */
  unsigned long long  EEConnS_NLPOut_Info_L;     /* Count of NLPs
                                    sent outbound - LOW priority  */
  unsigned long long  EEConnS_NLPOut_Rxmt_Info_L;/* Count of NLPs
                                    retransmitted outbound -
                                    LOW priority                  */

  unsigned long long  EEConnS_SNA_Bytes_Sent_M;  /* Total number of
                                    bytes sent over this EE
                                    connection - MEDIUM priority.
                                    This includes the data bytes
                                    along with NLH, THDR and FID5 */
  unsigned long long  EEConnS_NLPOut_Info_M;     /* Count of NLPs
                                    sent outbound - MEDIUM
                                    priority                      */
  unsigned long long  EEConnS_NLPOut_Rxmt_Info_M;/* Count of NLPs
                                    retransmitted outbound -
                                    MEDIUM priority               */

  unsigned long long  EEConnS_SNA_Bytes_Sent_H;  /* Total number of
                                    bytes sent over this EE
                                    connection - HIGH priority.
                                    This includes the data bytes
                                    along with NLH, THDR and FID5 */
  unsigned long long  EEConnS_NLPOut_Info_H;     /* Count of NLPs
                                    sent outbound - HIGH priority */
  unsigned long long  EEConnS_NLPOut_Rxmt_Info_H;/* Count of NLPs
                                    retransmitted outbound -
                                    HIGH priority                 */

  unsigned long long  EEConnS_SNA_Bytes_Sent_N;  /* Total number of
                                    bytes sent over this EE
                                    connection - NETWORK priority.
                                    This includes the data bytes
                                    along with NLH, THDR and FID5 */
  unsigned long long  EEConnS_NLPOut_Info_N;     /* Count of NLPs
                                    sent outbound - NETWORK
                                    priority                      */
  unsigned long long  EEConnS_NLPOut_Rxmt_Info_N;/* Count of NLPs
                                    retransmitted outbound -
                                    NETWORK priority              */
```

```
  unsigned long long  EEConnS_SNA_Bytes_Sent_S;  /* Total number of
                                     bytes sent over this EE
                                     connection - SIGNAL priority.
                                     This includes the data bytes
                                     along with NLH, THDR and FID5 */
  unsigned long long  EEConnS_NLPOut_Info_S;     /* Count of NLPs
                                     sent outbound - SIGNAL
                                     priority                      */
  unsigned long long  EEConnS_NLPOut_Rxmt_Info_S;/* Count of NLPs
                                     retransmitted outbound -
                                     SIGNAL priority               */

  unsigned long long  EEConnS_SNA_Bytes_Sent_A;  /* Total number of
                                     bytes sent over this EE
                                     connection - ALL priorities
                                     This includes the data bytes
                                     along with NLH, THDR and FID5 */
  unsigned long long  EEConnS_NLPOut_Info_A;     /* Count of NLPs
                                     sent outbound - ALL priorities*/
  unsigned long long  EEConnS_NLPOut_Rxmt_Info_A;/* Count of NLPs
                                     retransmitted outbound -
                                     ALL priorities                */
} EEConnS_Data_Transfer_Info_OutBound;

  /*****************************************************************
    Inbound Data Transfer Information by Priority:
       Low
       Medium
       High
       Network
       Signal
  *****************************************************************/
struct {
  unsigned long long  EEConnS_SNA_Bytes_Rcv_L;   /* Total number
                                     of bytes received over this
                                     EE connection - LOW priority.
                                     This includes data bytes along
                                     with NLH, THDR and FID5       */
  unsigned long long  EEConnS_NLPIn_Info_L;      /* Count of NLPs
                                     received inbound - LOW
                                     priority                      */

  unsigned long long  EEConnS_SNA_Bytes_Rcv_M;   /* Total number
                                     of bytes received over this
                                     EE connection - MEDIUM priority.
                                     This includes data bytes along
                                     with NLH, THDR and FID5       */
  unsigned long long  EEConnS_NLPIn_Info_M;      /* Count of NLPs
                                     received inbound - MEDIUM
                                     priority                      */

  unsigned long long  EEConnS_SNA_Bytes_Rcv_H;   /* Total number
                                     of bytes received over this
                                     EE connection - HIGH priority.
                                     This includes data bytes along
                                     with NLH, THDR and FID5       */
  unsigned long long  EEConnS_NLPIn_Info_H;      /* Count of NLPs
                                     received inbound - HIGH
                                     priority                      */

  unsigned long long  EEConnS_SNA_Bytes_Rcv_N;   /* Total number
                                     of bytes received over this
                                     EE connection - NETWORK
```

```
                                        priority.  This includes data
                                        bytes along with NLH, THDR and
                                        FID5                          */
    unsigned long long  EEConnS_NLPIn_Info_N;      /* Count of NLPs
                                        received inbound - NETWORK
                                        priority                      */

    unsigned long long  EEConnS_SNA_Bytes_Rcv_S;   /* Total number
                                        of bytes received over this
                                        EE connection - SIGNAL priority.
                                        This includes data bytes along
                                        with NLH, THDR and FID5       */
    unsigned long long  EEConnS_NLPIn_Info_S;      /* Count of NLPs
                                        received inbound - SIGNAL
                                        priority                      */

    unsigned long long  EEConnS_SNA_Bytes_Rcv_A;   /* Total number
                                        of bytes received over this
                                        EE connection - ALL priorities.
                                        This includes data bytes along
                                        with NLH, THDR and FID5       */
    unsigned long long  EEConnS_NLPIn_Info_A;      /* Count of NLPs
                                        received inbound - ALL
                                        priorities                    */
  } EEConnS_Data_Transfer_Info_InBound;

  unsigned long long EEConnS_Connection_Act_TOD;   /* TOD the EE
                                        connection was activated      */

  unsigned short EEConnS_Num_SRQRETRY_GT_One;  /* Number of times
                                        this connection has had signal
                                        responses require more than
                                        one retry.                    */
  unsigned short EEConnS_Num_SRQRETRY_EQ_Max;  /* Number of times
                                        this connection has had signal
                                        responses require the maximum
                                        allowable retries.            */
} EEConn_StaticData;

/*********************************************************************/
/* Enterprise Extender Connection Associated Hostname section        */
/*********************************************************************/

typedef struct {
  struct {                          /* Hostname indicators           */
    unsigned int EEConnH_Usage :1;  /* Local vs. Remote hostname
                                       '1'B - hostname was used to
                                          obtain remote IP address
                                       '0'B - hostname was used to
                                          obtain local IP address    */
    unsigned int EEConnH_Rsvd  :7;  /* Unused                        */
  } EEConnH_Flags;

  unsigned char EEConnH_Length;     /* Actual length of the hostname
                                       being reported. For convenience,
                                       the section will always have
                                       space for a maximum sized
                                       hostname                      */
  char          EEConnH_Host[64];   /* Hostname being reported       */
} EEConn_HostnameData;

/*********************************************************************/
/* Enterprise Extender Connection Associated VRN section             */
/*********************************************************************/
```

```
typedef struct {
  struct {                              /* Header data                  */
    struct {                            /* VRN data indicators          */
      unsigned int EEConnV_Type :1; /* VRN type being reported:
                                          '1'B - Global VRN
                                          '0'B - Local  VRN            */
      unsigned int EEConnV_Rsvd :7; /* Unused                         */
    } EEConnV_Flags;
  } EEConnV_Header;

  /* Following the header is the Virtual Routing Node Name.  The     */
  /* length of the name is obtained from the total length of the     */
  /* VRN section, as shown in the record triplet, less the length    */
  /* of the section header.                                          */

} EEConn_VRNData;

/*******************************************************************/
/* Enterprise Extender Connection Associated RTP PU name section   */
/*******************************************************************/

typedef struct {
  char EEConnP_Name[8];             /* RTP PU name, right-padded with
                                       blanks.                      */
} EEConn_RTPPUDATA;

/*******************************************************************/
/* Eyecatcher constants for EE Connection records                  */
/*******************************************************************/

const unsigned int EEConn_ID        /* EE connection record (EECO)   */
            = 0xC5C5C3D6;                                 /*@Q1C*/

/*******************************************************************/
/* Constants for Triplet counts for the EE Connection record       */
/*******************************************************************/

const int  EEConn_TripletCnt = 4;   /* EE conn rcd has 4 triplets    */
```

HPR Connection Response Record

HPR Connection Response format:

Common Request/Response Header
Input Triplet information (copied from request) -- **a single triplet is defined** • Offset from start of response data to first input section • Length of each input section of this type • Number of input sections of this type
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 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: • HPR Connection Global Output Record (one instance) • HPR Connection Specific Output Record(s) (one instance per HPR connection reported)

HPR Connection Global Output Record:

Record Identifier (4 chars) -- "HPRG"
Length of overall record (4 bytes)
Reserved field (2 chars)
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 HPR Connection Global data

HPR Connection Specific Output Record:

Record Identifier (4 chars) --- "HPRC"
Length of overall record (4 bytes)
Reserved field (2 chars)
Number of triplets for this output record (2 bytes) -- 3
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 section (one instance, potentially none if connection is in the process of performing a pathswitch)
Start of HPR Connection Pathswitch information section (only present if pathswitch had ever occurred on this connection, one instance if present)

The C/C++ data structure definitions for the HPR Connection Response Record are contained in the ISTHPRCC header file, and are shown below. The assembler mappings for these structures are in ISTHPRCA.

```
/*****************************************************************/
/* HPR Connection Global record                                  */
/*****************************************************************/

 typedef struct {
   unsigned int      HPRConnG_Eye;  /* HPRConn EyeCatcher (HPRG) @Q3C*/
   unsigned int      HPRConnG_Len;  /* Overall length of this record */
   char              HPRConnG_Rsv[2]; /* Reserved                    */
   unsigned short    HPRConnG_NumTriplets; /* Number of triplets
                                      defined for this record       */
   EEHNMRecordTriplet HPRConnG_Triplet; /* Only one triplet
                                      defined for this record       */
 } HPRConnGlobal;                  /* HPR Connection Global Data
```

```
                                        record                          */

/*******************************************************************/
/* HPR Connection Global Data section                              */
/*******************************************************************/
 typedef struct {
   struct {
     char            HPRConnGD_CPNetId[8]; /* NETID                  */
     char            HPRConnGD_CPName[8]; /* CPNAME                  */
   } HPRConnGD_Endpoint_Name;       /* Node name of this VTAM        */
 } HPRConnGlobalData;               /* HPR Connection Global Data
                                        section                       */

/*******************************************************************/
/* HPR Connection Specific Data record                             */
/*******************************************************************/
 typedef struct {
   unsigned int      HPRConnD_Eye;  /* HPRConn EyeCatcher (HPRC) @Q3C*/
   unsigned int      HPRConnD_Len;  /* Overall length of this record */
   char              HPRConnD_Rsv[2]; /* Reserved                    */
   unsigned short    HPRConnD_NumTriplets; /* Number of triplets
                                        defined for this record       */
   EEHNMRecordTriplet HPRConnD_StTriplet; /* First triplet
                                        points to static data section */
   EEHNMRecordTriplet HPRConnD_CVTriplet; /* Second triplet
                                        points to RSCV data section   */
   EEHNMRecordTriplet HPRConnD_PSTriplet; /* Third triplet
                                        points to pathswitch data (only
                                        present if at least one switch
                                        has occurred)                 */
 } HPRConnData;                     /* HPR Connection Specific Data
                                        record                        */

/*******************************************************************/
/* HPR Connection Specific Data Section                            */
/*******************************************************************/
 typedef struct {
   char              HPRConnDS_Name[8]; /* RTP PU Name               */

   struct {
     char            HPRConnDS_CPNetId[8]; /* NETID - destination    */
     char            HPRConnDS_CPName[8]; /* CPNAME - destination    */
   } HPRConnDS_FQ_Partner_Name;     /* FullyQualified Partner CPName */

   char              HPRConnDS_Local_NCB_PUName[8]; /* Physical NCB
                                        PU Name                       */
   struct {
     char            HPRConnDS_First_Hop_CPNetId[8]; /* NETID        */
     char            HPRConnDS_First_Hop_CPName[8]; /* CPNAME        */
   } HPRConnDS_First_Hop;           /* First Hop CPName              */

   struct {
     struct {
       unsigned int  HPRConnDS_Routing_Mode : 3; /* Routing Mode     */
       unsigned int  HPRConnDS_Rsv1 : 2; /* Reserved                 */
       unsigned int  HPRConnDS_TPF : 2; /* Transmission Priority     */
       unsigned int  HPRConnDS_Rsv2 : 9; /* Reserved                 */
     } HPRConnDS_NET_HEAD_BIT;      /* Net header overlay            */
   } HPRConnDS_NET_Header;          /* NLH header                    */

   struct {
     unsigned int    HPRConnDS_ARB_Algorithm : 2; /* ARB Algorithm
                                          '00'B - Original
                                          '01'B - Responsive Mode     */
```

```
  unsigned int    HPRConnDS ARB Mode : 2; /* ARB Pacing Mode
                                          '00'B - Green mode
                                          '01'B - Yellow mode
                                          '10'B - Red mode          */
  unsigned int    HPRConnDS_Role : 2; /* Passive or Active
                                          '10'B - Active
                                          '01'B - Passive           */
  unsigned int    HPRConnDS_MNPS : 1; /* RTP connection is being
                                         used by an MNPS application  */
  unsigned int    HPRConnDS_DYNLU : 1; /* DYNLU support indicator */
  unsigned int    HPRConnDS_XNETALS : 1; /* XNETALS support
                                         indicator                  */
  unsigned int    HPRConnDS_Rsv3 : 7; /* Reserved                  */
} HPRConnDS_Flags;              /* Informational flags           */

unsigned char     HPRConnDS_State; /* RTP State                    */
char              HPRConnDS_COS_Original[8]; /* Original COS  @Q2C*/
char              HPRConnDS_Rsv5[3]; /* Reserved              @Q2A*/

struct {
  unsigned long long HPRConnDS_Data_Bytes_Sent;  /* Number of
                                         data bytes sent over this
                                         RTP connection.            */

  unsigned long long HPRConnDS_Total_Bytes_Sent;  /* Total number
                                         of bytes sent over this RTP
                                         connection. This includes data
                                         bytes along with NLH, THDR and
                                         FID5                       */

  unsigned long long HPRConnDS_NLPOut;    /* Count of NLPs sent
                                         outbound                   */

  unsigned short  HPRConnDS_Largest_NLPOut;/* Largest NLP sent    */
  unsigned short  HPRConnDS_Num_Rexmitted_NLPS; /* Number of
                                         retransmitted NLPs         */
} HPRConnDS_Data_Transfer_Info_OutBound; /* OutBound data transfer
                                         information                */

struct {
  unsigned long long HPRConnDS_Data_Bytes_Rcv;  /* Number of data
                                         bytes received over this RTP
                                         connection.                */

  unsigned long long HPRConnDS_Total_Bytes_Rcv;  /* Total number
                                         of bytes received over this RTP
                                         connection.  This includes data
                                         bytes along with NLH, THDR and
                                         FID5                       */

  unsigned long long HPRConnDS_NLPIn;    /* Count of NLPs received
                                         inbound                    */

  unsigned short  HPRConnDS_Largest_NLPIN; /* Largest NLP received*/
  unsigned short  HPRConnDS_Rsv4; /* Reserved                     */
} HPRConnDS_Data_Transfer_Info_InBound;  /* InBound data transfer
                                         information                */

struct {
  unsigned int    HPRConnDS_Initial_Send_Rate; /* Initial send
                                         Rate                       */
  unsigned int    HPRConnDS_Allowed_Send_Rate; /* Allowed Send
                                         Rate                       */
  /*************************************************************/
```

```
    /* The next five threshholds are valid only if                  */
    /* HPRConnDS_ARB_Algorithm indicates "ARB Responsive mode" is   */
    /* being used                                                   */
    /*--------------------------------------------------------------*/
    unsigned int    HPRConnDS_Maximum_Send_Rate; /* Highwater mark for
                                    HPRConnDS_Actual_Send_Rate@Q1M*/
    unsigned int    HPRConnDS_Actual_Send_Rate; /* Actual Send Rate
                                                          @Q1M*/
    int             HPRConnDS_ARB2_RCVR_THRESHOLD; /* Current
                                    receiver threshold
                                    (value in microsecs)         */
    int             HPRConnDS_ARB2_RCVR_THRESHOLD_MIN; /* Min
                                    receiver threshold
                                    (value in microsecs)         */
    int             HPRConnDS_ARB2_RCVR_THRESHOLD_MAX; /* Max receiver
                                    receiver threshold
                                    (value in microsecs)         */
  } HPRConnDS_ARB_Info;            /* ARB Information             */

  struct {
    unsigned int    HPRConnDS_Num_NLPs_On_Pending_Sends_Q; /* Number
                                of NLPs on RPNCB_PENDING_SENDS_Q */
    unsigned int    HPRConnDS_Num_NLPs_On_OOSQ; /* Number of NLPs on
                                    the RPN_OutOfSeq_Msg_Q       */
    unsigned int    HPRConnDS_Num_NLPs_On_In_Segments_Q; /* Number
                                    of NLPS on contained within the
                                    RPN_RCV_Segments_DaPtr       */
    struct {
      unsigned int  HPRConnDS_NLPs_On_Wait_For_Ack_Q; /* Number of
                                    NLPs on RPNCB_WAIT_FOR_ACK_Q  */
      unsigned int  HPRConnDS_Max_Num_NLPs_On_Wait_For_Ack_Q; /* High
                                    water mark for the number of
                                    NLPs on RPNCB_WAIT_FOR_ACK_Q  */
      unsigned long long HPRConnDS_Max_Num_NLPs_On_Wait_TOD; /* TOD
                                    clock of high water mark for
                                    number of NLPs on
                                    RPNCB_WAIT_FOR_ACK_Q         */
    } HPRConnDS_Wait_For_Ack_Q_Info;

  } HPRConnDS_Queue_Info;

  struct {
    int             HPRConnDS_Smooth_Deviation; /* Responsive mode
                                    ARB smoothing deviation.     */
    unsigned int    HPRConnDS_SRTT; /* Smoothed roundtrip time in ms*/
    unsigned int    HPRConnDS_Liveness_Time; /* Liveness time length
                                    in seconds                   */
  } HPRConnDS_Timer_Info;

  unsigned int      HPRConnDS_LULU_Session_Count; /* Active LU-LU
                                    sessions using this RTP
                                    connection                   */
  unsigned long long HPRConnDS_Activation_TOD; /* TOD HPR Pipe
                                    activated                    */
  unsigned long long HPRConnDS_Local_TCID;      /* Local TCID     */
  unsigned long long HPRConnDS_Remote_TCID;     /* Remote TCID    */

  unsigned char   HPRConnDS_Local_NCE_Len;  /* Local NCE length   */
  char            HPRConnDS_Local_NCE[8];   /* Local NCE          */

  unsigned char   HPRConnDS_NCE_ID_LEN; /* Remote NCEID length    */
  char            HPRConnDS_NCE_ID[8];  /* Remote NCEID           */

  unsigned char   HPRConnDS_Local_ANR_LEN; /* Local ANR length    */
```

```
  char            HPRConnDS Local ANR[8];  /* Local ANR label
                                   outbound                    */

} HPRConnSpecificData;             /* HPR Connection Specific Data
                                   section                     */

/*****************************************************************/
/* Mapping for the HPR Connection Pathswitch Data. This section  */
/* will only be supplied if a pathswitch had occurred in the life*/
/* of the HPR connection being reported.                         */
/*****************************************************************/

typedef struct {
  unsigned long long HPRConnDP_TOD_Last_Pathswitch; /* TOD when
                                   last path switch was initiated*/
  struct {
    unsigned int    HPRConnDP_In_PS : 1; /* RTP pipe in currently in
                                   the process of pathswitching
                                   '1'B - Pipe is pathswitching
                                   '0'B - Pipe is not switching  */
    unsigned int    HPRConnDP_Last_PS_Reason : 3; /* Last Path
                                   Switch Reason
                                   '001'B - TGINOP
                                   '010'B - SRT retries
                                   '011'B - No NCB
                                   '100'B - Modify RTP command
                                   '101'B - Auto Pathswitch
                                   '110'B - Partner Initiated
                                        last switch      @Q1C*/
    unsigned int    HPRConnDP_Rsv : 4; /* Reserved                */
  } HPRConnDP_PS_Flags;            /* Path Switch Flags           */

  char              HPRConnPathSwitchData_Rsvd;   /* Reserved     */
  unsigned short    HPRConnDP_Cnt_PS_Initiated_Rem; /* Number of Path
                                   switches initiated by the
                                   remote RTP partner            */
  unsigned short    HPRConnDP_Cnt_PS_Initiated_Loc; /* Total number
                                   of path switches initiated by
                                   this node                     */
  unsigned short    HPRConnDP_Cnt_PS_Due_To_Failure; /* Number of
                                  Path switches initiated by this
                                  node due to errors (i.e.,
                                  TGINOP, short response time
                                  retries, or no NCB)           */
  unsigned short    HPRConnDP_Cnt_PS_Due_To_PSRETRY; /* Number of
                                   Path switches initiated by this
                                   node due to PSRETRY.         */
} HPRConnPathSwitchData;         /* HPR Connection Pathswitch data
                                   section                     */

/*****************************************************************/
/* Eyecatcher constants for HPR Connection records               */
/*****************************************************************/
const unsigned int HPRConnG_ID
  = 0xC8D7D9C7;                  /* HPR connection global
                                    record 'HPRG'              @Q3C*/
const unsigned int HPRConnD_ID
  = 0xC8D7D9C3;                  /* HPR connection global
                                    record 'HPRC'              @Q3C*/

/*****************************************************************/
/* Constants for Triplet counts for the various records          */
/*****************************************************************/
const int HPRConnG_TripletCnt = 1; /* HPR connection global
```

```
                                        record has one triplet        */
const int HPRConnD_TripletCnt = 3; /* HPR conn specific
                                        record has three triplets     */

/*****************************************************************/
/* Constants for ARB Algorithm                                   */
/*****************************************************************/

const int HPRConnDS_ARB_Original = 0; /* ARB original mode          */
const int HPRConnDS_ARB_Responsive = 1; /* ARB responsive mode      */

/*****************************************************************/
/* Constants for ARB Mode                                        */
/*****************************************************************/
const int HPRConnDS_ARB_GreenMode = 0; /* Green                    */
const int HPRConnDS_ARB_YellowMode = 1; /* Yellow                  */
const int HPRConnDS_ARB_RedMode = 2; /* Red                        */

/*****************************************************************/
/* Constants for Endpoint Role                                   */
/*****************************************************************/
const int HPRConnDS_Role_Active = 2; /* Active, or the node that
                                        setup the pipe             */
const int HPRConnDS_Role_Passive = 1; /* Passive, or the partner
                                        endpoint that was told to set
                                        up the pipe                */

/*****************************************************************/
/* Constants for Pathswitch reason codes                         */
/*****************************************************************/
const int HPRConnDP_PS_TGINOP = 1; /* TG INOP condition was detected*/
const int HPRConnDP_PS_SRTRetry = 2; /* Short request timer
                                        expiration                 */
const int HPRConnDP_PS_NoNCB = 3;  /* No NCB was available to use   */
const int HPRConnDP_PS_Modify = 4; /* Operator issued pathswitch
                                        request                    */
const int HPRConnDP_PS_AutoSwtch = 5; /* Pathswitch driven due to
                                        automatic retry timer      */
const int HPRConnDP_PS_Partner = 6; /* Partner initiated last
                                        switch                  @Q1A*/
```

CSM Statistics Response Record

The structure of the CSM Statistics response is as follows:

CSM Statistics Response format:

Common Request/Response Header
Input Triplet information (copied from request) -- a single triplet is defined • Offset from start of response data to first input section • Length of each input section of this type • Number of input sections of this type
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 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: • CSM Global Pool Output Record containing multiple CSM Buffer Pool data records (CSMPoolGData), one per pool • CSM Global Summary Output Record containing a single CSM Summary Data record (CSMSummGData) representing CSM system wide summary info

CSM Global Output Pool Record:

Record Identifier (4 chars) -- "CSMP"
Length of overall record (4 bytes)
Reserved field (2 chars)
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 Global Pool data (CSMPoolGDdata) records - one per CSM pool

CSM Global Output Summary Record:

Record Identifier (4 chars) **-- "CSMS"**
Length of overall record (4 bytes)
Reserved field (2 chars)
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 Global Summary data (CSMSummGData) record one single system wide record

The C/C++ data structure definitions for the CSM Statistics Response Record are contained in the ISTCSMGC header file, and are shown below. The assembler mappings for these structures are in ISTCSMGA.

```
/*****************************************************************/
/* CSM Pool Global record                                        */
/*****************************************************************/

typedef struct {
  unsigned int   CSMPoolG_Eye;      /* CSM Pool Global ID (CSMP) @Q1C*/
  unsigned int   CSMPoolG_Len;      /* Overall length of this record */
  unsigned short CSMPoolG_Rsvd;     /* Reserved                      */
  unsigned short CSMPoolG_NumTriplets;  /* Number of triplets defined
                                           for this record          */
  EEHNMRecordTriplet CSMPoolG_Triplet;  /* Only one triplet
                                           defined for this record  */
} CSMPoolGlobal;

/*****************************************************************/
/* CSM Pool Global Data record (one per CSM pool)                */
/*****************************************************************/
```

```
typedef struct {
  int           CSMPoolGD_Size;     /* Pool size                    */
  unsigned char CSMPoolGD_Srce;     /* Buffer source flag           */
  char          CSMPoolGD_Rsvd[3];  /* Not used - available         */
  int           CSMPoolGD_InUse;    /* Number of buffers in
                                       pool that are in use         */
  int           CSMPoolGD_Free;     /* Number of buffers in
                                       pool that are available      */
} CSMPoolGData;

/*********************************************************************/
/* CSM Summary Global record                                         */
/*********************************************************************/

typedef struct {
  unsigned int   CSMSummG_Eye;      /* CSM Summary Global ID (CSMS)
                                                              @Q1C*/
  unsigned int   CSMSummG_Len;      /* Overall length of this record */
  unsigned short CSMSummG_Rsvd;     /* Reserved                     */
  unsigned short CSMSummG_NumTriplets;  /* Number of triplets defined
                                       for this record              */
  EEHNMRecordTriplet CSMSummG_Triplet;  /* Only one triplet
                                       defined for this record      */
} CSMSummGlobal;

/*********************************************************************/
/* CSM Summary Global Data record (one per system)                   */
/*********************************************************************/

typedef struct {
  unsigned int CSMSummGD_MaxFiNMeg :1;  /* When off value = bytes
                                       When on value  = megabytes*/
  unsigned int CSMSummGD_Rsvd1     :7;  /* Reserved                  */
  unsigned int CSMSummGD_MaxFixed  :24; /* Installation Max
                                       fixed storage                */
  unsigned int CSMSummGD_CurFiNMeg :1;  /* When off value = bytes
                                       When on value  = megabytes*/
  unsigned int CSMSummGD_Rsvd2     :7;  /* Reserved                  */
  unsigned int CSMSummGD_CurFixed  :24; /* Current fixed storage
                                       in use                       */
  unsigned int CSMSummGD_MaxECSA;       /* Installation max ECSA     */
  unsigned int CSMSummGD_CurECSA;       /* Current ECSA Storage      */
} CSMSummGData;

/*********************************************************************/
/* Eyecatcher constants for CSM Summary pool and summary records     */
/*********************************************************************/

const unsigned int CSMPool_ID
  = 0xC3E2D4D7;                     /* CSM Pool Global ID    @Q1C*/
const unsigned int CSMSumm_ID
  = 0xC3E2D4E2;                     /* CSM Summary Global ID @Q1C*/

/*********************************************************************/
/* Constants to describe CSMPoolGD_Srce (buffer source)              */
/*********************************************************************/

const int CSMPoolGD_SrceECSA = 0x80;    /* Indicates source is
                                       CSM ECSA                     */
const int CSMPoolGD_SrceDS   = 0x40;    /* Indicates source is
                                       CSM DS                       */
```

```
/*****************************************************************/
/* Constants for Triplet counts for the various records          */
/*****************************************************************/
const int CSMPoolG_TripletCnt = 1;  /* CSM Pool Global record has
                                       one triplet                */
const int CSMSummG_TripletCnt = 1;  /* CSM Summary Global record has
                                       one triplet                */
```

The following table describes the errors in an NMI Request for which VTAM will send a termination record with the given Return Code and Reason Code, and then close the connection

Return Code	Reason Code	Meaning
EINVAL	‘00007110’X	Request header too short.
EINVAL	‘00007111’X	Unsupported version number in request header.
EINVAL	‘00007112’X	Invalid triplet format: first request section is not contiguous to request header.
EINVAL	‘00007112’X	Invalid triplet format: length of filter element is not correct for given version.
EINVAL	‘00007113’X	Length of request header plus length of request sections does not equal total length of request.
EINVAL	‘00007114’X	Invalid eyecatcher in request header.

The following table describes the error in an NMI request for which VTAM will return a negative response of the same type as the request. VTAM will leave the connection active after returning the negative response for these errors.

Return Code	Reason Code	Meaning
EINVAL	‘00007115’X	Unrecognized request type.
EINVAL	‘00007116’X	Too many filter elements (request sections) included for request type.
EINVAL	‘00007117’X	Too few filter elements (request sections) included for request type.
EINVAL	‘00007118’X	Undefined filter parameter indicator set in filter element.
EINVAL	‘00007119’X	Required filter parameter missing from filter element.
EINVAL	‘0000711A’X	Unsupported filter parameter indicator set in filter element.

The header files and macros are described in the following table.

Header files for C/C++ programs	Macro for Assembler programs	Contents
ISTEEHNC	ISTEEHNA	The NMI request and response header, initialization record, and termination record structure definitions.
ISTEESUC	ISTEESUA	The EE summary response data structure definitions.
ISTEECOC	ISTEECOA	The EE connection response data structure definitions
ISTHPRCC	ISTHPRCA	The HPR connection response data structure definitions.

ISTCSMGC	ISTCSMGA	The CSM global statistics response data structure definitions.

These header files and macros are shipped in the *hlq.SEZANMAC* data set (*hlq* refers to the High Level qualifier used when the product was installed on your system). This data set must be available in the concatenation when compiling or assembling a part that makes use of these definitions.

Chapter 6 - Diagnosis

The interfaces described in this document are designed to return error information as either a return_value, return_code or reason_code, where applicable. This information should be used to further diagnosis the problem being reported.

When the return_value is -1, the return_code and reason_code will indicate the problem that was incurred by the interface. Refer to the chapter describing the interface 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 documenting the error and contact IBM Customer Support.

Interface	**Documentation**
Application interfaces for network monitoring	• Set the SYSTCPIP "MISC" trace as active. • Collect a dump of the TCP/IP address space and data space.
Application interface for formatting packet and data trace records	Collect a dump of the TCP/IP address space and data space.
Application interface for monitoring TCP/UDP end points and TCP/IP storage usage	Collect a dump of the TCP/IP address space and data space.
Application interface for SNA network monitoring data	Collect a dump of the VTAM address space.

Appendix A - Record Formats

FTP Server Transfer Initialization record

FTP Server Transfer Initialization self-defining section of SMF record:

Offset	Name	Length	Format	Description
0 (x'0')	Standard SMF header	24	N/A	Standard SMF header; subtype will be 100 (x'64')
	Self Defining Section			
24 (x'18')	SMF119SD_TRN	2	binary	Number of triplets in this record V1R4: 5
26 (x'1A')		2		reserved
28 (x'1C')	SMF119IDOff	4	binary	Offset to TCP/IP identification section
32 (x'20')	SMF119IDLen	2	binary	Length of TCP/IP identification section
34 (x'22')	SMF119IDNum	2	binary	Number of TCP/IP identification sections
36 (x'24')	SMF119S1Off	4	binary	Offset to FTP server section
40 (x'28')	SMF119S1Len	2	binary	Length of FTP server section
42 (x'2A')	SMF119S1Num	2	binary	Number of FTP server sections
44 (x'2C')	SMF119S2Off	4	binary	Offset to FTP server hostname section
48 (x'30')	SMF119S2Len	2	binary	Length of FTP server hostname section
50 (x'32')	SMF119S2Num	2	binary	Number of FTP server hostname sections
52 (x'34')	SMF119S3Off	4	binary	Offset to FTP server first associated data set name section
56 (x'38')	SMF119S3Len	2	binary	Length of FTP server first associated data set name section
58 (x'3A')	SMF119S3Num	2	binary	Number of FTP server first associated data set name sections
60 (x'3C')	SMF119S4Off	4	binary	Offset to FTP server second associated data set nam section
64 (x'40')	SMF119S4Len	2	binary	Length of FTP server second associated data set name section
66 (x'42')	SMF119S4Num	2	binary	Number of FTP server second associated data set name sections

The TCP/IP Identification section is the same as for the completion record:

Offset	Name	Length	Description
0	SMF119TI_SYSName	8	System name from SYSNAME in IEASYSxx
8	SMF119TI_SysplexName	8	Sysplex name from SYSPLEX in COUPLExx
16	SMF119TI_Stack	8	TCP/IP stack name
24	SMF119TI_ReleaseID	8	CS OS/390 TCP/IP Release Identifier
32	SMF119TI_Comp	8	TCP/IP subcomponent (right padded with blanks): FTPS: FTP server
40	SMF119TI_ASName	8	Started task qualifier or address space name of address space that writes this SMF record

48	SMF119TI_UserID	8	User ID of security context under which this SMF record is written
56	SMF119TI_ASID	4	ASID of address space that writes this SMF record
60	SMF119TI_Reason	4	Reason for writing this SMF record: **X'08'**: Event SMF record

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.

Offset	Name	Length	Description
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 RFC959+)
8	SMF119FT_FSIFType	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_FSIFType	1	Data type A: ASCII E: EBCDIC

			I: Image B: Double-byte U: UCS-2
93	SMF119FT_FSIMode	1	Transmission mode **B**: Block **C**: Compressed **S**: Stream
94	SMF119FT_FSIStruct	1	Data structure **F**: File **R**: Record
95	SMF119FT_FSIDsType	1	Data set type **S**: SEQ **P**: PDS **H**: HFS
96	SMF119FT_FSISTime	4	Data connection start time, formatted in 1/100 seconds since midnight (using Coordinated Universal Time (UTC))
100	SMF119FT_FSISDate	4	Data connection start date (format: **0cyydddF**). If the start date is not available, the value specified will be **x'0000000F'**.
104	SMF119FT_FSICSTime	4	Control connection start time in 1/100 seconds since midnight (using Coordinated Universal Time (UTC)). (FTP session start time)
108	SMF119FT_FSICSDate	4	Control connection start date (format: **0cyydddF**). If the end date is not available, the value specified will be **x'0000000F'**. (FTP sessions start date)
112	SMF119FT_FSIM1	8	PDS Member name
120	SMF119FT_FSIM2	8	Second PDS member name (if rename operation)

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 will only be present if a **gethostbyaddr** operation was performed for the Local IP address:

Offset	Name	Length	Description
0	SMF119FT_Hostname	n	Host Name

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 will be included for Rename File Transfer operations.

Offset	Name	Length	Description
0	SMF119FT_MVSDataSet	44	MVS Data Set Name

The FTP Server HFS Filename 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 may be included in this section:

Offset	Name	Length	Description
0	SMF119FT_HFSLen1	2	Length of first HFS File name
2	SMF119FT_HFSName1	n	HFS File Name
2+n	SMF119FT_HFSLen2	2	Length of second HFS File name (zero if only one HFS File name is being reported)
4+n	SMF119FT_HFSName2	m	HFS File Name

FTP Client Transfer Initialization record

FTP Client Transfer Initialization self-defining section of SMF record:

Offset	Name	Length	Format	Description
0 (x'0')	Standard SMF header	24	N/A	Standard SMF header; subtype will be 101 (x'64')
	Self Defining Section			
24 (x'18')	SMF119SD_TRN	2	binary	Number of triplets in this record V1R4: 4
26 (x'1A')		2		reserved
28 (x'1C')	SMF119IDOff	4	binary	Offset to TCP/IP identification section
32 (x'20')	SMF119IDLen	2	binary	Length of TCP/IP identification section
34 (x'22')	SMF119IDNum	2	binary	Number of TCP/IP identification sections
36 (x'24')	SMF119S1Off	4	binary	Offset to FTP client section
40 (x'28')	SMF119S1Len	2	binary	Length of FTP client section
42 (x'2A')	SMF119S1Num	2	binary	Number of FTP client sections
44 (x'2C')	SMF119S2Off	4	binary	Offset to FTP client associated data set name section
48 (x'30')	SMF119S2Len	2	binary	Length of FTP client associated data set name section
50 (x'32')	SMF119S2Num	2	binary	Number of FTP client associated data set name sections
52 (x'34')	SMF119S3Off	4	binary	Offset to FTP client SOCKS section
56 (x'38')	SMF119S3Len	2	binary	Length of FTP client SOCKS section
58 (x'3A')	SMF119S3Num	2	binary	Number of FTP client SOCKS sections

The TCP/IP Identification section is the same as for the completion record.

Offset	Name	Length	Description
0	SMF119TI_SYSName	8	System name from SYSNAME in IEASYSxx
8	SMF119TI_SysplexName	8	Sysplex name from SYSPLEX in COUPLExx
16	SMF119TI_Stack	8	TCP/IP stack name
24	SMF119TI_ReleaseID	8	CS OS/390 TCP/IP Release Identifier
32	SMF119TI_Comp	8	TCP/IP subcomponent (right padded with blanks): **FTPC**: FTP client

40	SMF119TI_ASName	8	Started task qualifier or address space name of address space that writes this SMF record
48	SMF119TI_UserID	8	User ID of security context under which this SMF record is written
56	SMF119TI_ASID	4	ASID of address space that writes this SMF record
60	SMF119TI_Reason	4	Reason for writing this SMF record: **X'08'**: Event SMF record

FTP Client Transfer Initialization record section (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.

Offset	Name	Length	Description
0	SMF119FT_FCICmd	4	FTP subcommand (according to RFC959)
4	SMF119FT_FCIFType	4	Local file type (SEQ or SQL)
8	SMF119FT_FCIDRIP	16	Remote IP address (data connection)
24	SMF119FT_FCIDLIP	16	Local IP address (data connection)
40	SMF119FT_FCIDRPort	2	Remote port number (data connection)
42	SMF119FT_FCIDLPort	2	Local port number (data connection)
44	SMF119FT_FCICRIP	16	Remote IP address (control connection)
60	SMF119FT_FCICLIP	16	Local IP address (control connection)
76	SMF119FT_FCICRPort	2	Remote port number (control connection)
78	SMF119FT_FCICLPort	2	Local port number (control connection)
80	SMF119FT_FCIRUser	8	User ID (login name) on server
88	SMF119FT_FCILUser	8	Local User ID
96	SMF119FT_FCIType	1	Data format A: ASCII E: EBCDIC I: Image B: Double-byte U: UCS-2
97	SMF119FT_FCIMode	1	Transfer mode **B**: Block **C**: Compressed **S**: Stream
98	SMF119FT_FCIStruct	1	Structure **F**: File **R**: Record
99	SMF119FT_FCIDSType	1	Data set type **S**: SEQ **P**: PDS **H**: HFS

100	SMF119FT_FCISTime	4	Start time of data connection in 1/100 seconds since midnight (using Coordinated Universal Time (UTC))
104	SMF119FT_FCISDate	4	Start date of data connection (format: **0cyydddF**). If the start date is not available, the value specified will be **x'0000000F'**.
108	SMF119FT_FCICSTime	4	Start time of control connection in 1/100 seconds since midnight (using Coordinated Universal Time (UTC)). FTP session start time.
112	SMF119FT_FCICSSDate	4	Start date of the control connection (format 0cyydddF). If the start date is not available, the value specified will be x'0000000F'. FTP session start date.
116	SMF119FT_FCIM1	8	PDS member name
124	SMF119FT_FCIActPas	1	Passive or active mode data connection: x'00' active using default ip and port x'01' active using PORT x'03' passive using PASV x'04' passive using EPSV
125	reserved	3	

The FTP Client Hostname section, physically located after the FTP Client Transfer Complete section. This section is optional and is identical to the one present in the transfer completion record - it will only be present if a **gethostbyaddr** operation was performed for the Local IP address:

Offset	Name	Length	Description
0	SMF119FTC_Hostname	n	Host Name

The FTP Client MVS Data Set Name section, physically located after the FTP Client Hostname section (if present) or the FTP Client Transfer Complete section and is identical to the one present in the transfer completion record. This section represents the MVS data set names associated with the file transfer:

Offset	Name	Length	Description
0	SMF119FTC_MVSDataSet	44	MVS Data Set Name

The FTP Client HFS Filename section, physically located after the FTP Client MVS Data Set Name section and is identical to the one present in the transfer completion record. This section will only be present if Data Set Type = HFS, and only one HFS filename will be present.

Offset	Name	Length	Description
0	SMF119FT_HFSName1	n	HFS File Name

The FTP client SOCKS section is only present if the connection passes through a SOCKS server and is identical to the one present in the transfer completion record.

Offset	Name	Length	Description
0	SMF119FT_FCCIP	16	SOCKS server IP address
16	SMF119FT_FCCPort	2	SOCKS Server port number
18	SMF119FT_FCCProt	1	SOCKS protocol version: x'01' SOCKS Version 4 x'02' SOCKS Version 5

Appendix B - PTF information

This appendix contains information about PTFs required for enabling this function on z/OS V1R4.

Interface	APAR	
IP	PQ77244	Stack related API code
IP	PQ77837	Netstat, Configuration, SNMP
IP	PQ77838	FTP
IP	PQ77840	API code that is not in the EZBTCPIP address space: EZBCTAPI, Packet trace formatter, NM Service, IPCS
IP	PQ79566	IP Headers/Macros
IP FTP SMF	PQ78753	FTP SMF data
SNA	OA04394	EE and CSM
SNA	OA05225	SNA Headers/Macros
	II13699	Informational APAR

Appendix C - File storage locations

The following table shows parts that are needed in order to compile Network Management Interface applications and their locations. Your compiler should be configured to have access to these libraries.

Function	***Filename***	***Type***	***Library***
Allow applications to capture data packets	EZBYTMIA (1) EZBYTMIH (1)	MACRO H	hlq.SEZANMAC
Allow applications to format CTRACE records	EZBCTAPI EZBYPTO EZBYPTHA EZBCTHDR EZBYCTHH EZBYPTHH	MACRO MACRO MACRO MACRO H H	hlq.SEZANMAC
Allow applications to obtain TCP connection information	EZBYTMIA (1) EZBYTMIH (1) EZASMF77 EZASMF	MACRO H MACRO H	hlq.SEZANMAC hlq.SEZANMAC SYS1.MACLIB hlq.SEZANMAC
Callable API to retrieve local TCP and UDP End Point Data	EZBNMRHA EZBNMRHC	MACRO H	hlq.SEZANMAC
Network Management - Callable API to retrieve new TCP/IP storage statistics details	Same files as "Callable API to retrieve local TCP and UDP End Point Data"		
Enterprise Extender Network Management	ISTEEHNC ISTEESUC ISTEECOC ISTHPRCC ISTCSMGC ISTEEHNA ISTEESUA ISTEECOA ISTHPRCA ISTCSMGA	H H H H H MACRO MACRO MACRO MACRO MACRO	SYS1.MACLIB
Network Management - Real Time Interface for SMF event records	EZBYTMIA (1) EZBYTMIH (1) EZANMFTA EZANMFTC	MACRO H MACRO H	hlq.SEZANMAC

Table Notes: (1) Part used for multiple functions.