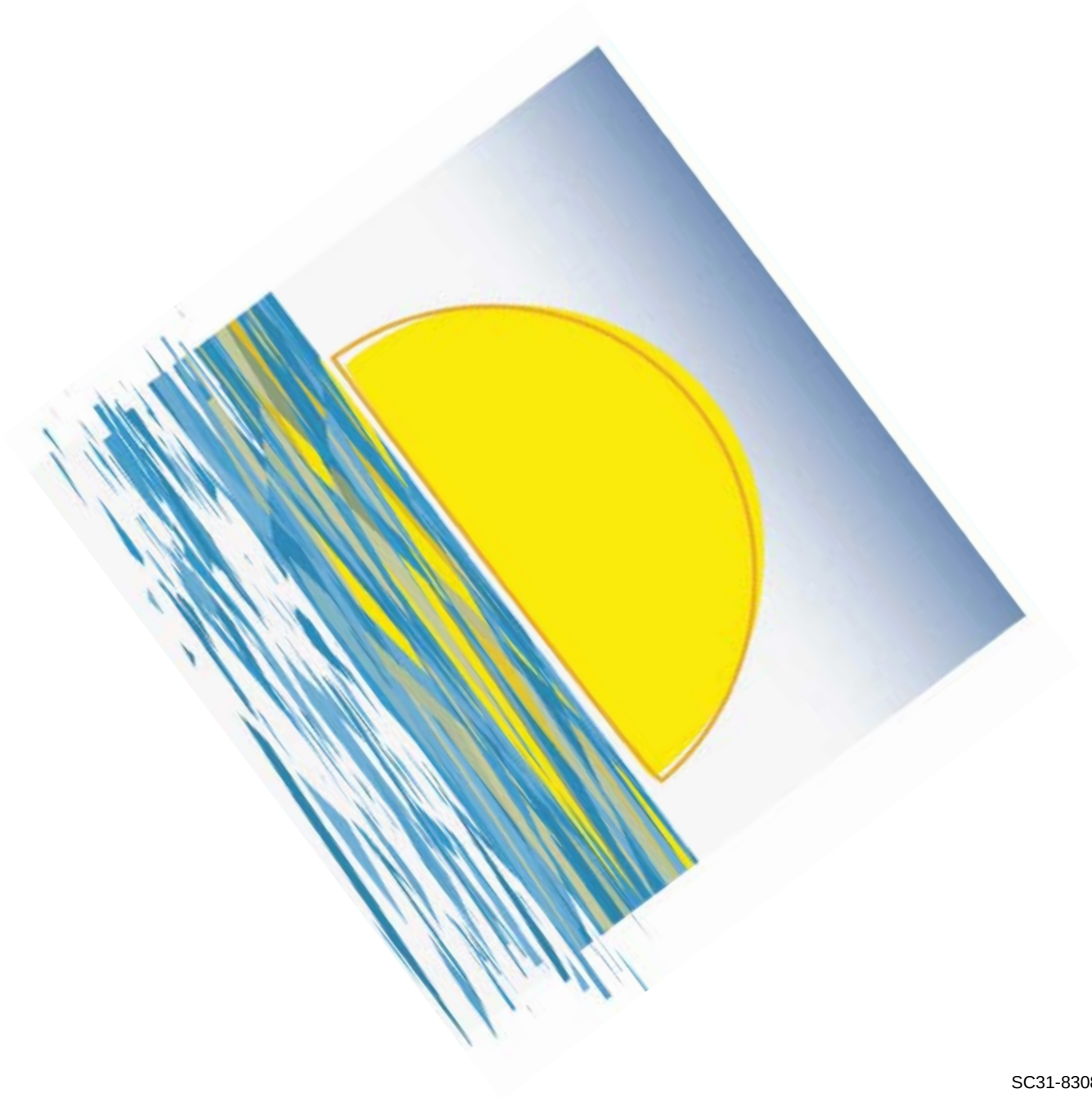
OS/390 TCP/IP Open Edition

Programmer's Reference



SC31-8308-00

OS/390 TCP/IP Open Edition

Programmer's Reference

SC31-8308-00

Note:

Before using this information and the product it supports, be sure to read the general information under Appendix D, “Notices” on page 115.

First Edition (June 1997)

This edition applies to OS/390 (5645-001) and OS/390 TCP/IP OpenEdition. See the “Summary of Changes” for a description of the changes made in this edition. Make sure you are using the correct edition for the level of the product.

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

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

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

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

Fax (USA and Canada):	1-800-227-5088
Internet e-mail:	usib2hpd@vnet.ibm.com
World Wide Web:	http://www.s390.ibm.com/os390
IBMLink:	CIBMORCF at RALVM13
IBM Mail Exchange:	USIB2HPD at IBMMAIL

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

© **Copyright International Business Machines Corporation 1989, 1997. All rights reserved.**
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

© Copyright IBM Corp. 1989, 1997

Tables

© Copyright IBM Corp. 1989, 1997

About This Book

This book describes the syntax and semantics of a set of high-level application functions that you can use to program your own applications in a TCP/IP environment. These functions provide support for application facilities, such as user authentication, distributed databases, distributed processing, network management, and device sharing.

The function included in this version of this book is limited to the Simple Network Management Protocol (SNMP) agent distributed protocol interface (DPI), the X Window Interface, and RPC for Open Edition.

For information about other function, refer to *TCP/IP for MVS: Programmer's Reference*, which supports the previous level of this product.

Please use the Reader's Comment Form located at the back of this book for instructions about how to submit your comments by mail, fax, or electronically.

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

How to Use this Book

This book is a companion to *TCP/IP for MVS: Programmer's Reference* (SC31-7135-02), which describes high-level application functions that you can use to program your own applications in a TCP/IP environment. These functions involve user authentication, distributed data bases, distributed processing, network management, and device sharing.

Who Should Use This Book

This book is intended for use by an experienced programmer familiar with MVS, the IBM Multiple Virtual Storage (MVS) operating system commands, and the TCP/IP protocols.

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

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

In addition,OS/390 TCP/IP OpenEdition and any required programming products should already be installed and customized for your network.

Where to Find Related Information on the Internet

You may find the following information helpful.

For current updates to the TCP/IP Version 3 Release 2 for MVS documentation described in "Bibliography" on page 117, check out the TCP/IP for MVS home page :

© Copyright IBM Corp. 1989, 1997

http://www.networking.ibm.com/tcm/tcmprod.html

To keep in close touch with OS/390, we suggest you look at the OS/390 home page :

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

To keep abreast of new products and technologies from IBM Networking, take a look at the IBM Networking home page :

http://www.networking.ibm.com/

The IBM Networking Software Glossary is now available in HTML format as well as PDF. You can access it directly at the following URL:

http://www.networking.ibm.com/nsg/nsggls.htm

How to Contact IBM Service

For telephone assistance in problem diagnosis and resolution (in the United States or Puerto Rico), call the IBM Software Support Center anytime (1-800-237-5511). You will receive a return call within 8 business hours (Monday – Friday, 8:00 a.m. – 5:00 p.m., local customer time).

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

Summary of Changes

Summary of Changes
for SC31–8308–00

This is the first edition of this book. It contains information previously presented in *TCP/IP for MVS: Programmer's Reference (SC31-7135-02),* which supports TCP/IP Version 3 Release 2 for MVS. This book is new for OS/390 TCP/IP OpenEdition, which provides OpenEdition function for TCP/IP in the OS/390 environment. For information about previously available TCP/IP function, continue to use the TCP/IP Version 3 Release 2 for MVS library.

This book describes:

- DPI support in an OpenEdition environment.
- Information on running X Windows in an OpenEdition environment.
- Information on running RPC in an OpenEdition environment.

© Copyright IBM Corp. 1989, 1997

Chapter 1. General Programming Information

For the fundamental, technical information you need to know before you attempt to work with the application program interfaces (APIs) provided with TCP/IP, please be sure you read the “before you begin” information in the *TCP/IP for MVS: Application Programming Interface Reference*.

© Copyright IBM Corp. 1989, 1997

Chapter 2. Simple Network Management Protocol Agent Distributed Protocol Interface

The Simple Network Management Protocol (SNMP) agent Distributed Protocol Interface (DPI) permits you to dynamically add, delete, or replace management variables in the local Management Information Base (MIB). The SNMP DPI protocol is also supported with the SNMP agent on OS/2, VM, and AIX. This makes it easier to port subagents between those platforms and OS/390 as well as connect agents and subagents across these platforms.

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

The following RFCs are related to SNMP and will be helpful when you are programming an SNMP API:

- RFC1592 is the SNMP DPI 2.0 RFC.
- RFC1901 through RFC1908 are the SNMP Version 2 RFCs.

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

To provide an environment that is generally platform independent, RFC 1592 strongly suggests that you also define a DPI API. There is a sample DPI API available in the RFC. The document describes the same sample API as the IBM supported DPI Version 2.0 API, see A DPI Subagent Example (see page 64).

The information about DPI is divided into the following topics:

Introduction	Includes: • Agents and Subagents • SNMP DPI Version 2.0 • SNMP DPI Version 1.1
Understanding DPI	• Subagent Programming Concepts • How to Specify the DPI API • Multi-threading Programming Considerations
Functions, Structures, and Values	• Basic functions • Transport-related functions • Data Structures • Constants and Values
Example	The DPI Subagent Example

© Copyright IBM Corp. 1989, 1997

SNMP Agents and Subagents

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

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

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

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

- A TCP connection
- A AF_UNIX streams connection

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

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

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

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

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

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

DPI Agent Requests

The SNMP agent can initiate several DPI requests:

- GET
- GETNEXT
- SET, COMMIT, and UNDO
- UNREGISTER
- CLOSE

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

The GETBULK requests are translated into multiple GETNEXT requests by the agent. According to RFC 1592, a subagent may request that the GETBULK be passed to it, but the MVS version of DPI does not yet support that request.

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

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

Related Information

- Overview of Subagent Processing (see page 64)
- Connecting to the Agent (see page 67)
- Registering a Sub-tree with the Agent (see page 69)
- Processing Requests from the Agent (see page 71)
- Processing a GET Request (see page 74)
- Processing a GETNEXT Request (see page 16)
- Processing a SET/COMMIT/UNDO Request (see page 81)
- Processing an UNREGISTER Request (see page 18)
- Processing an CLOSE Request (see page 18)
- Generating a TRAP (see page 17)

SNMP DPI Version 2.0 Library

OS/390 TCP/IP OpenEdition provides the following DPI library routines:

Table 1. Components of DPI 2.0

Name	Contents	Location
snmp_dpi.h	header file	/usr/lpp/tcpip/snmp/include
snmp_lDPI.o snmp_mDPI.o snmp_qDPI.o	• OE object files • DPI 2.0 library functions	/usr/lpp/tcpip/snmp/build/libdpi20
dpi_mvs_sample.c	SNMP DPI 2.0 C sample source	/usr/lpp/tcpip/samples
dpiSimpl.mi2	SNMP DPI 2.0 sample MIB definitions	/usr/lpp/tcpip/samples

SNMP DPI Version 2.0 API

DPI 2.0 is intended for use with OpenEdition sockets and is not for use with other socket libraries. A DPI Subagent must include the snmp_dpi.h header in any C part that intends to use DPI. The HFS path for snmp_dpi.h is /usr/lpp/tcpip/snmp/include. By default, when you include the snmp_dpi.h include file, you will be exposed to the DPI 2.0 API. For a list of the functions provided, read more about the snmp_dpi.h include file on page 63 . This is the recommended use of the SNMP DPI API. .

When you prelink your object code into an executable file, you must use the DPI 2.0 functions as provided in the `snmp_lDPI.o, snmp_mDPI.o, snmp_qDPI.o` object files in /usr/lpp/tcpip/snmp/build/libdpi20.

Usage Notes:

1. The object files are only located in OE HFS. HFS files can be accessed from JCL using the path parameter on an explicit DD definition.
2. Together the snmp_dpi.h include file and the dpi_mvs_sample.c file comprise an example of the DPI 2.0 API.
3. Debugging information (resulting from the DPIdebug function) is routed to SYSLOGD. Ensure the SYSLOG daemon is active.

   For more information about SYSLOGD, seeOS/390 TCP/IP OpenEdition Configuration Guide.
4. Compile your subagent code using the DEF(MVS) compiler option.
5. Waiting for a DPI packet depends on the platform and how the chosen transport protocol is implemented. In addition, some subagents want to control the sending of and waiting for packets themselves, because they may need to be driven by other interrupts as well.
6. There is a set of DPI transport-related functions that are implemented on all platforms to hide the platform-dependent issues for those subagents that do not need detailed control for the transport themselves.

For more information about SNMP, see the *OS/390 TCP/IP OpenEdition Configuration Guide* or the *OS/390 TCP/IP OpenEdition User's Guide*.

Compiling and Linking

DPI 2.0 is installed in HFS only. You can build a subagent for either the OpenEdition shell (using HFS and c89) or MVS (using JCL).

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

From an OE Environment

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

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

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

Use the - I option to add the HFS directory where snmp_dpi.h resides to the compiler's include search path.

See the *OS/390 OpenEdition Programming: Assembler Callable Services Reference* for information about building an application.

From an MVS Environment

C programs that use DPI must:

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

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

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

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

INCLUDE   DPI1
INCLUDE   DPI2
INCLUDE   DPI3
```

Then linkedit the prelink output as usual.

DPI 1.x Base Code Considerations

Use the DPI 1.1 API as described in the *TCP/IP for MVS: Programmer's Reference*.

The DPI 2.0 API provided with OS/390 TCP/IP OpenEdition is for OE (POSIX) sockets use only. Earlier versions of DPI were supported on C sockets.

See “Migrating Your SNMP DPI Subagent to Version 2.0” on page 8 for more detail about the changes that you must make to your DPI 1.x source.

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

You can keep your existing DPI 1.1 subagent and communicate with a DPI capable agent that supports DPI 1.1 in addition to DPI 2.0. For example, the MVS agent for

TCP/IP provides support for multiple versions of DPI, namely Version 1.0, Version 1.1 and Version 2.0.

SNMP DPI API Version 1.1 Considerations

Migrating Your SNMP DPI Subagent to Version 2.0

The information presented in this section **must be taken as guidelines and not exact procedures**. Your specific implementation will vary from the guidelines presented.

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

Required Actions

- Add a mkDPIopen() call and send the created packet to the agent. This opens your "DPI connection" with the agent. Wait for the response and ensure that the open is accepted. You need to pass a subagent ID (Object Identifier) which must be a unique ASN.1 OID.

  See The mkDPIopen() Function (see page 27) for more information.

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

  See The mkDPIregister() Function (see page 30) for more information.

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

  See The mkDPIset() Function (see page 33) for more information.

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

  See The mkDPItrap() Function (see page 35) for more information.

- Add code to recognize DPI RESPONSE packets, which should be expected as a result of OPEN, REGISTER, UNREGISTER requests.
- Add code to expect and handle the DPI UNREGISTER packet from the agent. It may send such packets if an error occurs or if a higher priority subagent registers the same sub-tree as you have registered.
- Add code to unregister your sub-tree(s) and close the "DPI connection" when you want to terminate the subagent.

  See The mkDPIunregister() Function (see page 37) and The mkDPIclose() Function (see page 26) for more information.

- Change your code to use the new SNMP Version 2 error codes as defined in the snmp_dpi.h include file.
- When migrating DPI 1.1 subagents to DPI 2.0, remove the include for manifest.h.
- Change your code that handles a GET request. It should return a varBind with SNMP_TYPE_noSuchObject value or SNMP_TYPE_noSuchInstance value instead of an error SNMP_ERROR_noSuchName if the object or the instance

do not exist. This is not considered an error any more. Therefore, you should return an SNMP_ERROR_noError with an error index of zero.

- Change your code that handles a GETNEXT request. It should return a varBind with SNMP_TYPE_endOfMibView value instead of an error SNMP_ERROR_noSuchName if you reach the end of your MIB or sub-tree. This is not considered an error any more. Therefore, you should return an SNMP_ERROR_noError with an error index of zero.
- Change your code that handles SET requests to follow the two phase SET/COMMIT scheme as described in SET Processing (see page 15) .

  See the sample handling of SET/COMMIT/UNDO in Processing a SET/COMMIT/UNDO Request (see page 81) .

Recommended Actions

- Do not reference the object ID pointer (object_p) in the snmp_dpi_xxxx_packet structures anymore. Instead start using the group_p and instance_p pointers. The object_p pointer may be removed in a future version of the DPI API.
- Check Transport-Related DPI API Functions (see page 39) to see if you want to use those functions instead of using your own code for those functions.
- Consider using more than 1 varBind per DPI packet. You can specify this on the REGISTER request. You must then be prepared to handle multiple varBinds per DPI packet. The varBinds are chained via the various snmp_dpi_xxxx_packet structures.

  See The mkDPIopen() Function (see page 27) for more information.
- Consider specifying a time out when you issue a DPI OPEN or DPI REGISTER.

  See The mkDPIopen() Function (see page 27) and The mkDPIregister() Function (see page 30) for more information.
- Ensure SYSLOGD is active. The result of using DPIdebug is routed to SYSLOGD. For information on how to configure SYSLOGD, see OS/390 TCP/IP OpenEdition Configuration Guide.

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

Name Changes

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

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

Old Name	New Name	Data Structure(XXXX)
group_id	group_p	getnext
object_id	object_p	get, getnext, set
value	value_p	set
type	value_type	set
next	next_p	set
enterprise	enterprise_p	trap
packet_body	data_u	dpi_hdr
dpi_get	get_p	hdr (packet_body)
dpi_getnext	next_p	hdr (packet_body)
dpi_set	set_p	hdr (packet_body)
dpi_trap	trap_p	hdr (packet_body)

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

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

However, the names may conflict with other definitions that you have, in which case you must change your code.

Subagent Programming Concepts

When implementing a subagent use the DPI Version 2 approach.

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

  VarBinds, or variable binding(s) refer to the number of objects specified in the SNMP PDU with respect to the requested operation. For example, using the SNMP Command Line Interface (CLI), a user can request the retrieval of multiple objects in the same request (GET or GETNEXT). The varBind portion of the PDU sent would include multiple object identifiers (OIDs). From the subagent perspective, it tells the agent via the max_varBinds parm on the mkDPIopen call on what its limitations are. When the subagent receives a request from the agent, it needs to handle multiple OIDs per request if it specified a max_varBinds value other than 1.

- Use the SNMP Version 2 approach for GETNEXT, and pass back endOfMibView value if appropriate. Continue to process all remaining varBinds.
- Specify the timeout period in the OPEN and REGISTER packets, when you are processing a request from the agent (GET, GETNEXT, SET, COMMIT, or UNDO).

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

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

Related Information

A DPI Subagent Example (see page 64)

Specifying the SNMP DPI API

The following section describes each type of DPI processing in this order:

1. Connect
2. Open
3. Register
4. Get, Set, Next, Trap, Are You There
5. Unregister
6. Close

Connect Processing

There are various connect functions that allow connections through either TCP or UNIXstream. Determine which is appropriate for you by evaluating whether you are connecting to the same machine or a different machine. If the agent and the subagent are using the same machine, use the UNIXstream connection for better performance. If the agent and the subagent are using different machines, you must use the TCP connection. There are two connect processing parameters:

- hostname—name or the IP address of the agent
- community name—password that allows the DPI connect function to obtain the port (for TCP) or path name (for UNIX) that allows the socket connect to occur.

Related Information

Connecting to the Agent (see page 67)

OPEN Request

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

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

  The agent may have an absolute maximum timeout value which will be used if the subagent asks for too large a timeout value. A value of zero can be used to indicate that the agent's own default timeout value should be used. A subagent is advised to use a reasonably short interval of a few seconds or so. If a specific sub-tree needs a (much) longer time, a specific REGISTER can be done for that sub-tree with a longer timeout value.

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

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

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

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

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

  For example: 1.3.5.1.2.3.4.5

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

  For example: "DPI sample subagent Version 2.0"

Once a subagent has sent a DPI OPEN packet to an agent, it should expect a DPI RESPONSE packet that informs the subagent about the result of the request. The packet ID of the RESPONSE packet should be the same as that of the OPEN request to which the RESPONSE packet is the response. See DPI RESPONSE Error Codes (see page 59) for a list of valid codes that may be expected.

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

If the OPEN is accepted, the next step is to REGISTER one or more MIB sub-trees.

Related Information

Connecting to the Agent (see page 67)

REGISTER Request

Before a subagent will receive any requests for MIB objects, it must first register the variables or sub-tree it supports with the SNMP agent. The subagent must specify a number of parameters in the REGISTER request:

- The sub-tree to be registered. This is a null terminated string in the selected character set. The sub-tree must have a trailing dot.

  For example: "1.3.6.1.2.3.4.5."

- The requested priority for the registration. The values are:

  -1 Request for the best available priority.

  0 Request for the next best available priority than the highest (best) priority currently registered for this sub-tree.

  NNN Any other positive value requests that specific priority if available or the next best priority that is available.

- The maximum timeout value in seconds. The agent is requested to wait this long for a response to any request for an object in this sub-tree. The agent may have an absolute maximum timeout value which will be used if the subagents asks for too large a timeout value. A value of zero can be used to indicate that the DPI OPEN value should be used for timeout.

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

If the response is successful, the error_index field in the RESPONSE packet contains the priority that the agent assigned to the sub-tree registration. See DPI RESPONSE Error Codes (see page 59) for a list of valid codes that may be expected.

Error Code: higherPriorityRegistered

The response to a REGISTER request may return the error code "higherPriorityRegistered". This may be caused by:

- Another subagent already registered the same sub-tree at a better priority than what you are requesting.
- Another subagent already registered a sub-tree at a higher level (at any priority). For instance, if a registration already exists for sub-tree 1.2.3.4.5.6 and you try to register for sub-tree 1.2.3.4.5.6.<anything> then you will get "higherPriorityRegistered" error code.

If you receive this error code, your sub-tree will be registered, but you will not see any requests for the sub-tree. They will be passed to the sub-agent which registered with a better priority. If you stay connected, and the other sub-agent goes away, then you will get control over the sub-tree at that point in time.

Related Information

Registering a Sub-tree with the Agent (see page 69)

GET Processing

The DPI GET packet holds one or more varBinds that the subagent has taken responsibility for.

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

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

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

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

Related Information

Processing a GET Request (see page 74)
The mkDPIresponse() Function (see page 31)

SET Processing

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

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

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

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

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

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

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

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

If there are no errors and the SET and COMMIT have been carried out with success, the subagent creates a DPI RESPONSE packet in which the error_code is

set to SNMP_ERROR_noError (zero) and error_index is set to zero. No name, type, length, or value information is needed.

So far we have discussed a successful SET and COMMIT sequence. However, after a successful SET, the subagent may receive a DPI UNDO packet. The subagent must now undo any preparations it made during the SET processing, such as free allocated memory.

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

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

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

Related Information

Processing a SET/COMMIT/UNDO Request (see page 81)

GETNEXT Processing

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

In this case, the instance ID may not be present (NULL) in the incoming DPI packet implying that the NEXT object must be the first instance of the first object in the sub-tree that was registered.

It is important to realize that a given subagent may support several discontinuous sections of the MIB tree. In that situation, it would be incorrect to jump from one section to another. This problem is correctly handled by examining the group ID in the DPI packet. This group ID represents the "reason" why the subagent is being called. It holds the prefix of the tree that the subagent had indicated it supported (registered).

If the next variable supported by the subagent does not begin with that prefix, the subagent must return the same object instance as in the request, for example the group ID and instance ID with a value of SNMP_TYPE_endOfMibView (implied NULL value). This endOfMibView is not considered an error, so the error_code and

error_index should be zero. If required, the SNMP agent will call upon the subagent again, but pass it a different group ID (prefix). This is illustrated in the discussion below.

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

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

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

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

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

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

Related Information

None.

GETBULK Processing Request

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

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

Related Information

Processing a GETNEXT Request (see page 16)

TRAP Request

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

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

Related Information

Generating a TRAP (see page86).

ARE_YOU_THERE Request

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

UNREGISTER Request

A subagent may unregister a previously registered sub-tree. The subagent must specify a few parameters in the UNREGISTER request:

- The sub-tree to be unregistered. This is a null terminated string in the selected character set. The sub-tree must have a trailing dot.

  For example: "1.3.6.1.2.3.4.5."

- The reason for the unregister. See DPI UNREGISTER Reason Codes (see page 60) for a list of valid reason codes.

Once a subagent has sent a DPI UNREGISTER packet to the agent, it should expect a DPI RESPONSE packet that informs the subagent about the result of the request. The packet ID of the RESPONSE packet should be the same as that of the REGISTER packet to which the RESPONSE packet is the response. See DPI RESPONSE Error Codes (see page 59) for a list of valid codes that may be expected.

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

Related Information

Processing an UNREGISTER request (see page84).

CLOSE Request

When a subagent is finished and wants to end processing, it should first UNREGISTER its sub-trees and then close the "connection" with the agent. To do so, it must send a DPI CLOSE packet, which specifies a reason for the closing. See DPI CLOSE Reason Codes (see page 58) for a list of valid codes. You should not expect a response to the CLOSE request.

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

Related Information

Processing a CLOSE request (see page85).

Multi-threading Programming Considerations

The DPI Version 2.0 program does not support multi-threaded subagents.

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

Use a Locking Mechanism

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

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

```
mkDPIAreYouThere()
mkDPIopen()
mkDPIregister()
mkDPIunregister()
mkDPItrap()
mkDPIresponse()
mkDPIpacket()
mkDPIclose ()
```

After you have called the DPIsend_packet_to_agent() function for the buffer, which is pointed to by the pointer returned by one of the preceeding functions, it is free to use again.

There is one function that reads the static input buffer:

```
pDPIpacket()
```

The input buffer gets filled by the DPIawait_packet_from_agent() function. Upon return from the await, you receive a pointer to the static input buffer. The pDPIpacket() function parses the static input buffer and returns a pointer to dynamically allocated memory. Therefore, after the pDPIpacket() call the buffer is available for use again.

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

or delete data in those static structures. The functions that change those internal static structures are:

```
DPIconnect_to_agent_TCP()          /* everyone has this one    */
DPIconnect_to_agent_UNIXstream()   /* supported */
DPIdisconnect_from_agent()         /* everyone has this one    */
```

The following are other functions that access those static structures which must be assured that the structure is not being changed while they are referencing it during their execution are:

```
DPIawait_packet_from_agent()
DPIsend_packet_to_agent()
DPIget_fd_for_handle()
```

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

Functions, Data Structures, and Constants

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

Basic DPI Functions:

DPI Transport-Related Functions:

Data Structures:

Constants and Values:

Related Information:

Basic DPI API Functions

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

The Basic DPI Functions are:

The DPIdebug() Function

Syntax

```
#include <snmp_dpi.h>

void DPIdebug(int level);
```

Parameters

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

1 Display packet creation and parsing.

2 Display hex dump of incoming and outgoing DPI packets.

Description

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

Examples

```
#include <snmp_dpi.h>

DPIdebug(2);
```

Related Information

The snmp_dpi.h Include File (see page 63)

The DPI_PACKET_LEN() Macro

Syntax

```
#include <snmp_dpi.h>

int DPI_PACKET_LEN(unsigned char *packet_p)
```

Parameters

packet_p A pointer to a serialized DPI packet.

Return Values

An integer representing the total DPI packet length.

Description

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

Examples

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

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

The fDPIparse() Function

Syntax

```
#include <snmp_dpi.h>

void fDPIparse(snmp_dpi_hdr *hdr_p);
```

Parameters

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

Description

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

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

Examples

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

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

Related Information

The snmp_dpi_hdr Structure (see page 50)
The pDPIpacket() Function (see page 38)
The snmp_dpi.h Include File (see page 63)

The fDPIset() Function

Syntax

```
#include <snmp_dpi.h>

void fDPIset(snmp_dpi_set_packet *packet_p);
```

Parameters

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

Description

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

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

Examples

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

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

set_p = mkDPIset(snmp_dpi_NULL_p,        /* create first one  */
            "1.3.6.1.2.3.4.5.","1.0",    /* OID=1, instance=0 */
            SNMP_TYPE_Integer32,
            sizeof(num1), &num1);
if (set_p) {                             /* if success, then  */
   first_p = set_p;                      /* save ptr to first */
   set_p   = mkDPIset(set_p,             /* chain next one    */
               "1.3.6.1.2.3.4.5.","1.1", /* OID=1, instance=1 */
               SNMP_TYPE_Integer32,
               sizeof(num2), &num2);
   if (set_p) {                          /* success 2nd one   */
      pack_p = mkDPIresponse(hdr_p,      /* make response     */
                    SNMP_ERROR_noError,  /* It will also free */
                    0L, first_p);        /* the set_p tree    */
      /* send DPI response to agent */
   } else {                              /* 2nd mkDPIset fail */
      fDPIset(first_p);                  /* must free chain   */
   } /* endif */
} /* endif */
```

Related Information

The fDPIparse() Function (see page 23)
The snmp_dpi_set_packet Structure (see page 53)
The mkDPIresponse() Function (see page 31)

The mkDPIAreYouThere() Function

Syntax

```
#include <snmp_dpi.h>

unsigned char *mkDPIAreYouThere(void);
```

Parameters

none

Return Values

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

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

Description

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

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

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

Examples

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

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

Related Information

The snmp_dpi_resp_packet Structure (see page 52)
The DPIawait_packet_from_agent() Function (see page 39)

The mkDPIclose() Function

Syntax

```
#include <snmp_dpi.h>

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

Parameters

reason_code The reason for closing the DPI connection. See DPI CLOSE Reason Codes (see page 58) for a list of valid reason codes.

Return Values

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

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

Description

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

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

Examples

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

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

Related Information

The snmp_dpi_close_packet Structure (see page 48)
DPI CLOSE Reason Codes (see page 58)

The mkDPIopen() Function

Syntax

```
#include <snmp_dpi.h>

unsigned char *mkDPIopen(       /* Make a DPI open packet     */
  char          *oid_p,         /* subagent Identifier (OID) */
  char          *description_p, /* subagent descriptive name */
  unsigned long  timeout,       /* requested default timeout  */
  unsigned long  max_varBinds,  /* max varBinds per DPI packet*/
  char           character_set, /* selected character set     */
  #define DPI_NATIVE_CSET  0    /*   0 = native character set */
  #define DPI_ASCII_CSET   1    /*   1 = ASCII  character set */

  unsigned long  password_len,  /* length of password (if any)*/
  unsigned char *password_p);   /* ptr to password (if any)   */
```

Parameters

oid_p A pointer to a NULL terminated character string representing the OBJECT IDENTIFIER which uniquely identifies the subagent. The OID valued pointed to by oid_p must be in the EBCDIC character set when communicating with a TCP/IP OpenEdition SNMP agent. The agent will add the OID passed in the mkDPIopen call to the sysORTable as sysORID in a corresponding new entry. By convention, sysORID should match a capabilities statement's OID to refer to the MIBs supported by the subagent.

For a list of MIB variables, refer to theOS/390 TCP/IP OpenEdition User's Guide.

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

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

max_varBinds The maximum number of varBinds per DPI packet that the subagent is prepared to handle. It must be a positive number or zero.

- If a value greater than 1 is specified, the agent will try to combine as many varBinds which belong to the same sub-tree per DPI packet as possible up to this value.
- If a value of zero is specified, the agent will try to combine up to as many varBinds as are present in the SNMP packet and belong to the same sub-tree; there is no limit on the number of varBinds present in the DPI packet.

character_set The character set that you want to use for string-based data fields in the DPI packets and structures. The choices are:

DPI_NATIVE_CSET Specifies that you want to use the native character set of the platform on which the agent that you connect to is running.

See Character Set Selection (see page 56) for more information.

password_len The length in octets of an optional password. It depends on the agent implementation if a password is needed.

If coded, this parameter is ignored with the MVS agent.

password_p A pointer to an octet string representing the password for this subagent. A password may include any character value, including the NULL character. If the password_len is zero, this can be a NULL pointer.

If coded, this parameter is ignored with the MVS agent.

Return Values

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

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

Description

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

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

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

Examples

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

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

Related Information

Character Set Selection (see page 56)

The mkDPIregister() Function

Syntax

```
#include <snmp_dpi.h>

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

Parameters

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

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

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

group_p A pointer to a NULL terminated character string that represents the sub-tree to be registered. This group ID must have a trailing dot.

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

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

Return Values

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

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

Description

The mkDPIregister() function creates a serialized DPI REGISTER packet that can then be sent to the DPI peer which is a DPI capable SNMP agent.

Normally, the SNMP agent sends a DPI RESPONSE packet back. This packet identifies if the register was successful or not.

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

Examples

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

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

Related Information

The snmp_dpi_resp_packet Structure (see page 52)

The mkDPIresponse() Function

Syntax

```
#include <snmp_dpi.h>

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

Parameters

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

error_code The error code.

See DPI RESPONSE Error Codes (see page 59) for a list of valid codes.

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

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

Return Values

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

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

Description

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

Examples

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

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

Related Information

The pDPIpacket() Function (see page 38)
The snmp_dpi_hdr Structure (see page 50)
The snmp_dpi_set_packet Structure (see page 53)

The mkDPIset() Function

Syntax

```
#include <snmp_dpi.h>

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

Parameters

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

group_p A pointer to a NULL terminated character string that represents the registered sub-tree that caused this GET request to be passed to this DPI subagent. The sub-tree must have a trailing dot.

instance_p A pointer to a NULL terminated character string that represents the rest, which is the piece following the sub-tree part, of the OBJECT IDENTIFIER of the variable instance being accessed. Use of the term *instance_p* here should not be confused with an OBJECT instance because this string may consist of a piece of the OBJECT IDENTIFIER plus the INSTANCE IDENTIFIER.

value_type The type of the value.

See DPI SNMP Value Types (see page 60) for a list of currently defined value types.

value_len This is the value that specifies the length in octets of the value pointed to by the *value* field. The length may be zero if the value is of type SNMP_TYPE_NULL.

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

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

Return Values

If successful and a chain of one or more packets was passed in the *packet_p* parameter, the same pointer that was passed in *packet_p* is returned. A new dynamically allocated structure has then been added to the end of that chain of snmp_dpi_get_packet structures.
If successful and a NULL pointer was passed in the *packet_p* parameter, a pointer to a new dynamically allocated structure is returned.
If not successful, a NULL pointer is returned.

Description

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

The chain of snmp_dpi_set_packet structures can also be used to create a DPI TRAP packet that includes varBinds as explained in The mkDPItrap() Function (see page 35) .

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

Examples

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

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

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

Related Information

The pDPIpacket() Function (see page 38)
The mkDPIresponse() Function (see page 31)
The mkDPItrap() Function (see page 35)
The snmp_dpi_hdr Structure (see page 50)
The snmp_dpi_set_packet Structure (see page 53)
DPI SNMP Value Types (see page 60)
Value Representation (see page 61)

The mkDPItrap() Function

Syntax

```
#include <snmp_dpi.h>

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

Parameters

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

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

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

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

Return Values

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

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

Description

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

Examples

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

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

Related Information

The mkDPIset() Function (see page 33)

The mkDPIunregister() Function

Syntax

```
#include <snmp_dpi.h>

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

Parameters

reason_code The reason for the unregister.

See DPI UNREGISTER Reason Codes (see page page 60) for a list of the currently defined reason codes.

group_p A pointer to a NULL terminated character string that represents the sub-tree to be unregistered. The sub-tree must have a trailing dot.

Return Values

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

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

Description

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

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

Examples

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

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

Related Information

The snmp_dpi_ureg_packet Structure (see page 55)

The pDPIpacket() Function

Syntax

```
#include <snmp_dpi.h>

snmp_dpi_hdr *pDPIpacket(unsigned char *packet_p);
```

Parameters

packet_p A pointer to a serialized DPI packet.

Return Values

If successful, a pointer to a DPI parse tree (snmp_dpi_hdr) is returned. Memory for the parse tree has been dynamically allocated, and it is the callers responsibility to free it when no longer needed. You can use the fDPIparse() function to free the parse tree.
If not successful, a NULL pointer is returned.

Description

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

Examples

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

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

Related Information

The snmp_dpi_hdr Structure (see page 50)
The snmp_dpi.h Include File (see page 63)
The fDPIparse() Function (see page 23)

Transport-Related DPI API Functions

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

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

The Transport-Related DPI API Functions are:

The DPIawait_packet_from_agent() Function

Syntax

```
#include <snmp_dpi.h>

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

Parameters

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

timeout A timeout value in seconds. There are two special values:

- **-1** Causes the function to wait forever until a packet arrives.
- **0** Means that the function will only check if a packet is waiting. If not, an immediate return is made. If there is a packet, it will be returned.

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

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

Return Values

If successful, a zero (DPI_RC_OK) is returned. The buffer pointer and length of the caller will be set to point to the received DPI packet and to the length of that packet.
If not successful, a negative integer is returned, which indicates the kind of error that occurred. See Return Codes from DPI Transport-Related Functions (62) for a list of possible error codes.

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

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

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

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

DPI_RC_TIMEOUT No packet was received during the specified timeout period.

DPI_RC_PACKET_TOO_LARGE The packet received was too large.

Description

The DPIawait_packet_from_agent() function is used at the subagent side to await a DPI packet from the DPI capable SNMP agent. The programmer can specify how long to wait.

Examples

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

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

Related Information

The DPIconnect_to_agent_TCP() Function (see page 41)
The DPIconnect_to_agent_UNIXstream() Function (see page 42)

The DPIconnect_to_agent_TCP() Function

Syntax

```
#include <snmp_dpi.h>

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

Parameters

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

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

Return Values

If successful, a non-negative integer that represents the connection is returned. It is to be used as a handle in subsequent calls to DPI transport-related functions.
If not successful, a negative integer is returned, which indicates the kind of error that occurred. See Return Codes from DPI Transport-Related Functions (see page 62) for a list of possible error codes.

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

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

Description

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

As part of the connection processing, the DPIconnect_to_agent_TCP() function sends an SNMP GET request to the SNMP agent to retrieve the port number of the DPI port to be used for the TCP connection. By default, this SNMP GET request is sent to the well— known SNMP port 161. If the SNMP agent is listening on a port other than well— known port 161, the SNMP_PORT environment variable can be

set to the port number of the SNMP agent prior to issuing the DPIconnect_to_agent_TCP(). Use setenv() to override port 161 before using this function.

Examples

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

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

Related Information

Return Codes from DPI Transport-Related Functions (see page 62)
The DPIconnect_to_UNIXstream() Function (see page 42)

The DPIconnect_to_agent_UNIXstream() Function

Syntax

```
#include <snmp_dpi.h>

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

Parameters

hostname_p A pointer to a NULL terminated character string representing the local host name or IP address in dot notation of the local host where the DPI capable SNMP agent is running.

community_p A pointer to a NULL terminated character string representing the community name that is required to obtain the UNIX path name from the SNMP agent via an SNMP GET request.

Return Values

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

If not successful, a negative integer is returned, which indicates the kind of error that occurred. See Return Codes from DPI Transport-Related Functions (see page 62) for a list of possible error codes.

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

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

Description

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

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

Establishing Permission

This function uses a path name in the HFS as the name of the socket for connect. This path name is available at the snmp agent via the MIB object 1.3.6.1.4.1.2.2.1.1.3, which has the name dpiPathNameForUnixStream. The MVS snmp agent has a default name that it uses (/tmp/dpi_socket) if you do not supply another name in the agent's startup parameter (-s). Whatever name is chosen, it has to live in the HFS as a character special file.

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

Examples

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

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

Related Information

Return Codes from DPI Transport-Related Functions (see page 62)
The DPIconnect_to_agent_TCP() Function (see page 41)

The DPIdisconnect_from_agent() Function

Syntax

```
#include <snmp_dpi.h>

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

Parameters

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

Description

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

Examples

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

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

Related Information

The DPIconnect_to_agent_TCP() Function (see page 41)
The DPIconnect_to_UNIXstream() Function (see page 42)

The DPIget_fd_for_handle() Function

Syntax

```
#include <snmp_dpi.h>

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

Parameters

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

Return Values

If successful, a positive integer representing the file descriptor associated with the specified handle.
If not successful, a negative integer is returned, which indicates the error that occurred. See Return Codes from DPI Transport-Related Functions (see page 62) for a list of possible error codes.

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

Description

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

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

Examples

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

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

Related Information

The DPIconnect_to_agent_TCP() Function (see page 41)
The DPIconnect_to_UNIXstream() Function (see page 42)

The DPIsend_packet_to_agent() Function

Syntax

```
#include <snmp_dpi.h>

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

Parameters

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

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

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

Return Values

If successful, a zero (DPI_RC_OK) is returned.
If not successful, a negative integer is returned, which indicates the kind of error that occurred. See Return Codes from DPI Transport-Related Functions (see page 62) for a list of possible error codes.

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

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

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

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

Description

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

Examples

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

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

Related Information

The DPIconnect_to_agent_TCP() Function (see page 41)
The DPIconnect_to_UNIXstream() Function (see page 42)
The DPI_PACKET_LEN() Macro (see page 22)

The lookup_host() Function

Syntax

```
#include <snmp_dpi.h>

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

Parameters

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

Return Values

If successful, the IP address is returned in network byte order, so it is ready to be used in a sockaddr_in structure.
If not successful, a value of 0 is returned.

Description

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

Related Information

The DPIconnect_to_agent_TCP() Function (see page 41)

DPI Structures

This section describes each data structure that is used in the SNMP DPI API.

The Data Structures are:

- The snmp_dpi_close_packet Structure (see page 48)
- The snmp_dpi_get_packet Structure (see page 49)
- The snmp_dpi_next_packet Structure (see page 51)
- The snmp_dpi_hdr Structure (see page 50)
- The snmp_dpi_resp_packet Structure (see page 52)
- The snmp_dpi_set_packet Structure (see page 53)
- The snmp_dpi_ureg_packet Structure (see page 55)
- The snmp_dpi_u64 Structure (see page 56)

The snmp_dpi_close_packet Structure

Structure Definition

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

Structure Members

reason_code The reason for the close.

See DPI CLOSE Reason Codes (see page 58) for a list of valid reason codes.

Description

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

The snmp_dpi_close_packet structure may be created as a result of a call to pDPIpacket(). This is the case if the DPI packet is of type SNMP_DPI_CLOSE.

The snmp_dpi_hdr structure then contains a pointer to a snmp_dpi_close_packet structure.

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

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

Related Information

The pDPIpacket() Function (see page 38)
The mkDPIclose() Function (see page 26)
The snmp_dpi_hdr Structure (see page 50)

The snmp_dpi_get_packet Structure

Structure Definition

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

Structure Members

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

group_p A pointer to a NULL terminated character string that represents the registered sub-tree that caused this SET request to be passed to this DPI subagent. The sub-tree must have a trailing dot.

instance_p A pointer to a NULL terminated character string that represents the rest which is the piece following the sub-tree part of the OBJECT IDENTIFIER of the variable instance being accessed.

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

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

Description

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

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

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

Related Information

The pDPIpacket() Function (see page 38)
The snmp_dpi_hdr Structure (see page 50)

The snmp_dpi_hdr Structure

Structure Definition

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

Structure Members

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

proto_version The DPI version.

proto_release The DPI release.

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

packet_type The type of DPI packet (parse tree) which you are dealing with.

See DPI Packet Types (see page 58) for a list of currently defined DPI packet types

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

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

Description

The snmp_dpi_hdr structure is the anchor of a DPI parse tree. At the subagent side, the snmp_dpi_hdr structure is normally created as a result of a call to pDPIpacket().

The DPI subagent programmer uses this structure to interrogate packets. Depending on the *packet_type*, the pointer to the chain of one or more packet_type specific structures that contain the actual packet data can be picked.

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

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

Related Information

The fDPIparse() Function (see page 23)
The pDPIpacket() Function (see page 38)
The snmp_dpi_close_packet Structure (see page 48)
The snmp_dpi_get_packet Structure (see page 49)
The snmp_dpi_next_packet Structure (see page 51)
The snmp_dpi_resp_packet Structure (see page 52)
The snmp_dpi_set_packet Structure (see page 53)
The snmp_dpi_ureg_packet Structure (see page 55)

The snmp_dpi_next_packet Structure

Structure Definition

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

Structure Members

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

group_p A pointer to a NULL terminated character string that represents the registered sub-tree that caused this GETNEXT request to be passed to this DPI subagent. This sub-tree must have a trailing dot.

instance_p A pointer to a NULL terminated character string that represents the rest which is the piece following the sub-tree part of the OBJECT IDENTIFIER of the variable instance being accessed.

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

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

Description

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

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

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

Related Information

The pDPIpacket() Function (see page 38)
The snmp_dpi_hdr Structure (see page 50)

The snmp_dpi_resp_packet Structure

Structure Definition

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

Structure Members

error_code The return code or the error code.

See DPI RESPONSE Error Codes (see page page 59) for a list of valid codes.

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

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

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

Description

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

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

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

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

Related Information

The pDPIpacket() Function (see page 38)
The mkDPIresponse() Function (see page 31)
The snmp_dpi_set_packet Structure (see page 53)
The snmp_dpi_hdr Structure (see page 50)

The snmp_dpi_set_packet Structure

Structure Definition

```
struct dpi_set_packet {
  char             *object_p;    /* ptr to Object ID (string) */
  char             *group_p;     /* ptr to sub-tree (group)   */
  char             *instance_p;  /* ptr to rest of OID        */
  unsigned char     value_type;  /* value type: SNMP_TYPE_xxx */
  unsigned short    value_len;   /* value length              */
  char             *value_p;     /* ptr to the value itself   */
  struct dpi_set_packet *next_p; /* ptr to next in chain      */
};
typedef struct dpi_set_packet       snmp_dpi_set_packet;
#define snmp_dpi_set_packet_NULL_p  ((snmp_dpi_set_packet *)0)
```

Structure Members

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

group_p A pointer to a NULL terminated character string that represents the registered sub-tree that caused this SET, COMMIT, or UNDO request to be passed to this DPI subagent. The sub-tree must have a trailing dot.

instance_p A pointer to a NULL terminated character string that represents the rest, which is the piece following the sub-tree part, of the OBJECT IDENTIFIER of the variable instance being accessed.

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

value_type The type of the value.

See DPI SNMP Value Types (see page 60) for a list of currently defined value types.

value_len This is an unsigned 16-bit integer that specifies the length in octets of the value pointed to by the *value* field. The length may be zero if the value if of type SNMP_TYPE_NULL.

value_p A pointer to the actual value. This field may contain a NULL pointer if the value if of type SNMP_TYPE_NULL.

See Value Representation (see page 61) for information on how the data is represented for the various value types.

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

Description

The snmp_dpi_set_packet structure represents a parse tree for a DPI SET request.

The snmp_dpi_set_packet structure may be created as a result of a call to pDPIpacket(). This is the case if the DPI packet is of type SNMP_DPI_SET, SNMP_DPI_COMMIT or SNMP_DPI_UNDO. The snmp_dpi_hdr structure then contains a pointer to a chain of one or more snmp_dpi_set_packet structures.

This structure can also be created with a mkDPIset() call, which is typically used when preparing varBinds for a DPI RESPONSE packet.

Related Information

The pDPIpacket() Function (see page 38)
The mkDPIset() Function (see page 33)
DPI SNMP Value Types (see page 60)
Value Representation (see page 61)
The snmp_dpi_hdr Structure (see page 50)

The snmp_dpi_ureg_packet Structure

Structure Definition

```
struct dpi_ureg_packet {
  char                   reason_code;/* reason for unregister */
  char                  *group_p;    /* ptr to sub-tree(group)*/
  struct dpi_ureg_packet *next_p;     /* ptr to next in chain  */
};
typedef struct dpi_ureg_packet       snmp_dpi_ureg_packet;
#define snmp_dpi_ureg_packet_NULL_p  ((snmp_dpi_ureg_packet *)0)
```

Structure Members

reason_code The reason for the unregister.

See DPI UNREGISTER Reason Codes (see page page 60) for a list of the currently defined reason codes.

group_p A pointer to a NULL terminated character string that represents the sub-tree to be unregistered. This sub-tree must have a trailing dot.

next_p A pointer to a possible next snmp_dpi_ureg_packet structure. If this next field contains the NULL pointer, this is the end of the chain. Currently we do not support multiple unregister requests in one DPI packet, so this field should always be zero.

Description

The snmp_dpi_ureg_packet structure represents a parse tree for a DPI UNREGISTER request.

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

The DPI programmer is advised to use the mkDPIunregister() function to create a DPI UNREGISTER packet.

Related Information

The pDPIpacket() Function (see page 38)
The mkDPIunregister() Function (see page 37)
The snmp_dpi_hdr Structure (see page 50)

The snmp_dpi_u64 Structure

Structure Definition

```
struct snmp_dpi_u64 {           /* for unsigned 64-bit int */
  unsigned long high;           /* - high order 32 bits    */
  unsigned long low;            /* - low order  32 bits    */
};
typedef struct snmp_dpi_u64     snmp_dpi_u64;
```

Note: This structure is supported only in SNMP Version 2.

Structure Members

high The high order, most significant, 32 bits

low The low order, least significant, 32 bits

Description

The snmp_dpi_u64 structure represents an unsigned 64-bit integer as need for values with a type of SNMP_TYPE_Counter64.

The snmp_dpi_u64 structure may be created as a result of a call to pDPIpacket(). This is the case if the DPI packet is of type SNMP_DPI_SET and one of the values has a type of SNMP_TYPE_Counter64. The value_p pointer of the snmp_dpi_set_packet structure will then point to an snmp_dpi_u64 structure.

The DPI programmer must also use an snmp_dpi_u64 structure as the parameter to a mkDPIset() call if you want to create a value of type SNMP_TYPE_Counter64.

Related Information

The pDPIpacket() Function (see page 38)
The snmp_dpi_set_packet Structure (see page 53)
DPI SNMP Value Types (see page 60)
Value Representation (see page 61)

Character Set Selection

The version of DPI 2.0 shipped with TCP/IP for MVS requires use of the EBCDIC character set. Any DisplayString MIB objects known to the agent (in its compiled MIB) supplied with TCP/IP for MVS will have ASCII conversion handled by the agent. The subagent will always deal with the values of these objects in EBCDIC. Any portion of an instance identifier that is a DisplayString must be in ASCII. The agent does not handle instance IDs.

When the DPI subagent sends a DPI OPEN packet, it must specify the character set that it wants to use. The subagent here needs to know or determine in an implementation dependent manner if the agent is running on a system with the same character set as the subagent. If you connect to the agent at loopback, localhost, or your own machine, you might assume that you are using the same character set.

The DPI subagent has two choices:

DPI_NATIVE_CSET Specifies that you want to use the native character set of the platform on which the agent that you connect to is running.

DPI_ASCII_CSET Specifies that you want to use the ASCII character set. The agent will not translate between ASCII and the native character set.

Although you can specify ASCII, the MVS agent does not support it.

The DPI packets have a number of fields that are represented as strings. The fields that must be represented in the selected character set are:

- The null terminated string pointed to by the *description_p, enterprise_p, group_p, instance_p,* and *oid_p* parameters in the various *mkDPIxxxx(...)* functions.
- The string pointed to by the *value_p* parameter in the *mkDPIset(...)* function, that is if the *value_type* parameter specifies that the value is an *SNMP_TYPE_DisplayString* or an *SNMP_TYPE_OBJECT_IDENTIFIER*.
- The null terminated string pointed to by the *description_p, enterprise_p, group_p, instance_p,* and *oid_p* pointers in the various *snmp_dpi_xxxx_packet* structures.
- The string pointed to by the *value_p* pointer in the *snmp_dpi_set_packet* structure, that is if the *value_type* field specifies that the value is an *SNMP_TYPE_DisplayString* or an *SNMP_TYPE_OBJECT_IDENTIFIER*.

Related Information

The mkDPIopen() Function (see page 27)

Constants, Values, Return Codes, and Include File

This section describes all the constants and names for values as they are defined in the snmp_dpi.h include file (see page 63) .

The Constants and Values are:

DPI CLOSE Reason Codes

The currently defined DPI CLOSE reason codes as defined in the snmp_dpi.h include file are:

```
#define SNMP_CLOSE_otherReason                1
#define SNMP_CLOSE_goingDown                  2
#define SNMP_CLOSE_unsupportedVersion         3
#define SNMP_CLOSE_protocolError              4
#define SNMP_CLOSE_authenticationFailure      5
#define SNMP_CLOSE_byManager                  6
#define SNMP_CLOSE_timeout                    7
#define SNMP_CLOSE_openError                  8
```

These codes are used in the *reason_code* parameter for the *mkDPIclose()* function and in the *reason_code* field in the *snmp_dpi_close_packet* structure.

Related Information

The snmp_dpi_close_packet Structure (see page 48)
The mkDPIclose() Function (see page 26)

DPI Packet Types

The currently defined DPI packet types as defined in the snmp_dpi.h include file are:

```
#define SNMP_DPI_GET             1
#define SNMP_DPI_GET_NEXT        2  /* old DPI 1.x style */
#define SNMP_DPI_GETNEXT         2
#define SNMP_DPI_SET             3
#define SNMP_DPI_TRAP            4
#define SNMP_DPI_RESPONSE        5
#define SNMP_DPI_REGISTER        6
#define SNMP_DPI_UNREGISTER      7
#define SNMP_DPI_OPEN            8
#define SNMP_DPI_CLOSE           9
#define SNMP_DPI_COMMIT         10
#define SNMP_DPI_UNDO           11
#define SNMP_DPI_GETBULK        12
#define SNMP_DPI_TRAPV2         13  /* reserved, not ....    */
#define SNMP_DPI_INFORM         14  /* reserved, implemented */
#define SNMP_DPI_ARE_YOU_THERE  15
```

These packet types are used in the *type* parameter for the *packet_type* field in the *snmp_dpi_hdr* structure.

Related Information

The snmp_dpi_hdr Structure (see page 50)

DPI RESPONSE Error Codes

In case of an error on an SNMP request like GET, GETNEXT, SET, COMMIT, or UNDO, the RESPONSE can have one of these currently defined error codes. They are defined in the snmp_dpi.h include file:

```
#define SNMP_ERROR_noError               0
#define SNMP_ERROR_tooBig                1
#define SNMP_ERROR_noSuchName            2
#define SNMP_ERROR_badValue              3
#define SNMP_ERROR_readOnly              4
#define SNMP_ERROR_genErr                5
#define SNMP_ERROR_noAccess              6
#define SNMP_ERROR_wrongType             7
#define SNMP_ERROR_wrongLength           8
#define SNMP_ERROR_wrongEncoding         9
#define SNMP_ERROR_wrongValue           10
#define SNMP_ERROR_noCreation           11
#define SNMP_ERROR_inconsistentValue    12
#define SNMP_ERROR_resourceUnavailable  13
#define SNMP_ERROR_commitFailed         14
#define SNMP_ERROR_undoFailed           15
#define SNMP_ERROR_authorizationError   16
#define SNMP_ERROR_notWritable          17
#define SNMP_ERROR_inconsistentName     18
```

In case of an error on a DPI only request (OPEN, REGISTER, UNREGISTER, ARE_YOU_THERE), the RESPONSE can have one of these currently defined error codes. They are defined in the snmp_dpi.h include file:

```
#define SNMP_ERROR_DPI_noError                        0
#define SNMP_ERROR_DPI_otherError                   101
#define SNMP_ERROR_DPI_notFound                     102
#define SNMP_ERROR_DPI_alreadyRegistered            103
#define SNMP_ERROR_DPI_higherPriorityRegistered     104
#define SNMP_ERROR_DPI_mustOpenFirst                105
#define SNMP_ERROR_DPI_notAuthorized                106
#define SNMP_ERROR_DPI_viewSelectionNotSupported    107
#define SNMP_ERROR_DPI_getBulkSelectionNotSupported 108
#define SNMP_ERROR_DPI_duplicateSubAgentIdentifier  109
#define SNMP_ERROR_DPI_invalidDisplayString         110
#define SNMP_ERROR_DPI_characterSetSelectionNotSupported  111
```

These codes are used in the *error_code* parameter for the *mkDPIresponse()* function and in the *error_code* field in the *snmp_dpi_resp_packet* structure.

Related Information

The snmp_dpi_resp_packet Structure (see page 52)
The mkDPIresponse() Function (see page 31)

DPI UNREGISTER Reason Codes

These are the currently defined DPI UNREGISTER reason codes. They are define in the snmp_dpi.h include file:

```
#define SNMP_UNREGISTER_otherReason               1
#define SNMP_UNREGISTER_goingDown                 2
#define SNMP_UNREGISTER_justUnregister            3
#define SNMP_UNREGISTER_newRegistration           4
#define SNMP_UNREGISTER_higherPriorityRegistered  5
#define SNMP_UNREGISTER_byManager                 6
#define SNMP_UNREGISTER_timeout                   7
```

These codes are used in the *reason_code* parameter for the *mkDPIunregister()* function and in the *reason_code* field in the *snmp_dpi_ureg_packet* structure.

Related Information

The snmp_dpi_ureg_packet Structure (see page 55)
The mkDPIunregister() Function (see page 37)

DPI SNMP Value Types

These are the currently defined value types as as defined in the snmp_dpi.h include file:

```
#define SNMP_TYPE_MASK           0x7f  /* mask to isolate type*/
#define SNMP_TYPE_Integer32    (128|1) /* 32-bit INTEGER      */
#define SNMP_TYPE_OCTET_STRING      2  /* OCTET STRING        */
#define SNMP_TYPE_OBJECT_IDENTIFIER 3  /* OBJECT IDENTIFIER   */
#define SNMP_TYPE_NULL              4  /* NULL, no value      */
#define SNMP_TYPE_IpAddress         5  /* IMPLICIT OCTETSTRING*/
#define SNMP_TYPE_Counter32    (128|6) /* 32-bit Counter      */
#define SNMP_TYPE_Gauge32      (128|7) /* 32-bit Gauge        */
#define SNMP_TYPE_TimeTicks    (128|8) /* 32-bit TimeTicks in */
                                       /* hundredths of a sec */
#define SNMP_TYPE_DisplayString     9  /* DisplayString (TC)  */
#define SNMP_TYPE_BIT_STRING       10  /* BIT STRING          */
#define SNMP_TYPE_NsapAddress      11  /* IMPLICIT OCTETSTRING*/
#define SNMP_TYPE_UInteger32  (128|12) /* 32-bit INTEGER      */
#define SNMP_TYPE_Counter64        13  /* 64-bit Counter      */
#define SNMP_TYPE_Opaque           14  /* IMPLICIT OCTETSTRING*/
#define SNMP_TYPE_noSuchObject     15  /* IMPLICIT NULL       */
#define SNMP_TYPE_noSuchInstance   16  /* IMPLICIT NULL       */
#define SNMP_TYPE_endOfMibView     17  /* IMPLICIT NULL       */
```

These value types are used in the *value_type* parameter for the *mkDPIset()* function and in the *value_type* field in the *snmp_dpi_set_packet* structure.

Related Information

The snmp_dpi_set_packet Structure (see page 53)
The mkDPIset() Function (see page 33)
Value Representation (see page 61)
Value Ranges and Limits (see page 62)

Value Representation

Values in the snmp_dpi_set_packet structure are represented as follows:

- 32-bit integers are defined as long int or unsigned long int. We assume that a long int is 4 bytes.
- 64-bit integers are represented as an snmp_dpi_u64.

  We only deal with unsigned 64 bit integers in SNMP. In a structure that has two fields, the high order piece and the low order piece, each is of type unsigned long int. We assume these are 4-bytes.
- Object Identifiers are NULL terminated strings in the selected character set, representing the OID in ASN.1 dotted notation. The length includes the terminating NULL.

  An ASCII example:

  ```
  '312e332e362e312e322e312e312e312e3000'h
  ```

  represents "1.3.6.1.2.1.1.1.0" which is sysDescr.0.

  An EBCDIC example:

  ```
  'f14bf34bf64bf14bf24bf14bf14bf14bf000'h
  ```

  represents "1.3.6.1.2.1.1.1.0" which is sysDescr.0.
- DisplayStrings are in the selected character set. The length specifies the length of the string.

  An ASCII example:

  ```
  '6162630d0a'h
  ```

  represents "abc\r\n", no NULL.

  An EBCDIC example:

  ```
  '8182830d25'h
  ```

  represents "abc\r\n", no NULL.
- IpAddress and Opaque are implicit OCTET_STRING, so they are a sequence of octets/bytes. This means, for instance, that the IP address is in network byte order.
- NULL has a zero length for the value, no value data, so a NULL pointer in the *value_p* field.
- noSuchObject, noSuchInstance, and endOfMibView are implicit NULL and represented as such.
- BIT_STRING is an OCTET_STRING of the form uubbbb...bb, where the first octet (uu) is 0x00-0x07 and indicates the number of unused bits in the last octet (bb). The bb octets represent the bit string itself, where bit zero (0) comes first and so on.

Related Information

Value Ranges and Limits (see page 62)

Value Ranges and Limits

The following rules apply to object IDs in ASN.1 notation:

- The object ID consists of 1 to 128 subIDs, which are separated by dots.
- Each subID is a positive number. No negative numbers are allowed.
- The value of each number cannot exceed 4294967295 (4,294,967,295). This value is 2 to the power of 32 minus 1.
- The valid values of the first subID are: 0, 1, or 2.
- If the first subID has a value of 0 or 1, the second subID can only have a value of 0 through 39.

The following rules apply to DisplayString:

- A DisplayString (Textual Convention) is basically an OCTET STRING in SNMP terms.
- The maximum size of a DisplayString is 255 octets/bytes.

More information on the DPI SNMP value types can be found in the SNMP SMI (Structure of Management Information) and SNMP TC (Textual Conventions) RFCs. At the time of this publication, these two RFCs are RFC1902 and RFC1903.

Return Codes from DPI Transport-Related Functions

These are the currently defined values for the return codes from DPI transport-related functions. They are defined in the snmp_dpi.h include file:

```
#define DPI_RC_OK                   0 /* all OK, no error         */
#define DPI_RC_NOK                 -1 /* some other error         */
#define DPI_RC_NO_PORT             -2 /* can't determine DPIport  */
#define DPI_RC_NO_CONNECTION       -3 /* no connection to DPIagent*/
#define DPI_RC_EOF                 -4 /* EOF receivd on connection*/
#define DPI_RC_IO_ERROR            -5 /* Some I/O error on connect*/
#define DPI_RC_INVALID_HANDLE      -6 /* unknown/invalid handle   */
#define DPI_RC_TIMEOUT             -7 /* timeout occurred         */
#define DPI_RC_PACKET_TOO_LARGE    -8 /* packed too large, dropped*/
#define DPI_RC_UNSUPPORTED_DOMAIN -9 /*unsupported domain for connect*/
#define DPI_RC_INVALID_ARGUMENT  -10 /*invalid argument passed*/
```

These values are used as return codes for the transport-related DPI functions.

Related Information

The DPIconnect_to_agent_TCP() Function (see page 41)
The DPIconnect_to_agent_UNIXstream() Function (see page 42)
The DPIawait_packet_from_agent() Function (see page 39)
The DPIsend_packet_to_agent() Function (see page 45)

The snmp_dpi.h Include File

```
#include <snmp_dpi.h>
```

Parameters

None

Description

The snmp_dpi.h include file defines the SNMP DPI API to the DPI subagent programmer. It has all the function prototype statements, and it also has the definitions for the snmp_dpi structures.

The same include file is used at the agent side, so you will see some definitions which are unique to the agent side. Also there may be other functions or prototypes of functions not implemented on MVS. Therefore, you should only use the API as far as it is documented in this manual.

Related Information

Macros, functions, structures, constants and values defined in the snmp_dpi.h include file are:

- The DPIawait_packet_from_agent() Function (see page 39)
- The DPIconnect_to_agent_TCP() Function (see page41)
- The DPIconnect_to_agent_UNIXstream() Function (see page42)
- The DPIdebug() Function (see page 21)
- The DPIdisconnect_from_agent() Function (see page 43)
- The DPI_PACKET_LEN() Macro (see page 22)
- The DPIsend_packet_to_agent() Function (see page 45)
- The fDPIparse() Function (see page 23)
- The fDPIset() Function (see page 24)
- The mkDPIAreYouThere() Function (see page 25)
- The mkDPIclose() Function (see page 26)
- The mkDPIopen() Function (see page 27)
- The mkDPIregister() Function (see page 30)
- The mkDPIresponse() Function (see page 31)
- The mkDPIset() Function (see page 33)
- The mkDPItrap() Function (see page 35)
- The mkDPIunregister() Function (see page 37)
- The pDPIpacket() Function (see page 38)
- The snmp_dpi_close_packet Structure (see page 48)
- The snmp_dpi_get_packet Structure (see page 49)

- The snmp_dpi_next_packet Structure (see page 51)
- The snmp_dpi_hdr Structure (see page 50)
- The lookup_host() Function (see page 47)
- The snmp_dpi_resp_packet Structure (see page 52)
- The snmp_dpi_set_packet Structure (see page 53)
- The snmp_dpi_ureg_packet Structure (see page 55)
- DPI CLOSE Reason Codes (see page 58)
- DPI Packet Types (see page 58)
- DPI RESPONSE Error Codes (see page 59)
- DPI UNREGISTER Reason Codes (see page 60)
- DPI SNMP Value Types (see page 60)
- Character Set Selection (see page 56)

A DPI Subagent Example

This is an example of a DPI subagent. The code is called dpi_mvs_sample.c in the /usr/lpp/tcpip/samples directory.

Note: The example code in this document was copied from the sample file at the time of the publication. There may be differences in the code presented and the code that is shipped with the product. Always use the code provided in the/usr/lpp/tcpip/samples directory as the authoritative sample code.

The DPI subagent example includes:

- Overview of Subagent Processing (see page 64)
- Connecting to the Agent (see page 67)
- Registering a Sub-tree with the Agent (see page 69)
- Processing Requests from the Agent (see page 71)
- Processing a GET Request (see page 74)
- Processing a GETNEXT Request (see page 77)
- Processing a SET/COMMIT/UNDO Request (see page 81)
- Processing an UNREGISTER Request (see page 84)
- Processing an CLOSE Request (see page 85)
- Generating a TRAP (see page 86)

Related Information

Subagent Programming Concepts (see page 10)

Overview of Subagent Processing

This overview assumes that the subagent communicates with the agent over a TCP connection. Other connection implementations are possible and, in that case, the processing approach may be a bit different.

We also take a simple approach in the sense that we will request the agent to send us at most one varBind per DPI packet, so we do not need to loop through a list of varBinds. Potentially, you may gain performance improvements if you allow for multiple varBinds per DPI packet on GET, GETNEXT, SET requests, but to do so, your

code will have to loop through the varBind list and so it becomes somewhat more complicated. We assume that the DPI subagent programmer can handle that once you understand the basics of the DPI API.

The following are the supported MIB variable definitions for DPI_SIMPLE:

```
DPISimple-MIB DEFINITIONS ::= BEGIN

   IMPORTS
        MODULE-IDENTITY, OBJECT-TYPE, snmpModules, enterprises
                    FROM SNMPv2-SMI
        DisplayString
                    FROM SNMPv2-TC

   ibm      OBJECT IDENTIFIER ::= { enterprises 2 }
   ibmDPI   OBJECT IDENTIFIER ::= { ibm 2 }
   dpi20MIB OBJECT IDENTIFIER ::= { ibmDPI 1 }

-- dpiSimpleMIB MODULE-IDENTITY
--   LAST-UPDATED "9401310000Z"
--   ORGANIZATION "IBM Research - T.J. Watson Research Center"
--   CONTACT-INFO "           Bert Wijnen
--                Postal:    IBM International Operations
--                           Watsonweg 2
--                           1423 ND Uithoorn
--                           The Netherlands
--                Tel:       +31 2975 53316
--                Fax:       +31 2975 62468
--                E-mail:    wijnen@vnet.ibm.com
--                           (IBM internal: wijnen at nlvm1)"
--   DESCRIPTION
--          "The MIB module describing DPI Simple Objects for
--           the dpi_samp.c program"
--   ::= { snmpModules x }

   dpiSimpleMIB OBJECT IDENTIFIER ::= { dpi20MIB 5 }

   dpiSimpleInteger         OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
            "A sample integer32 value"
        ::= { dpiSimpleMIB 1 }

   dpiSimpleString          OBJECT-TYPE
        SYNTAX  DisplayString
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
            "A sample Display String"
        ::= { dpiSimpleMIB 2 }

   dpiSimpleCounter32       OBJECT-TYPE
        SYNTAX  Counter     -- Counter32 is SNMPv2
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
```

```
            "A sample 32-bit counter"
        ::= { dpiSimpleMIB 3 }

   dpiSimpleCounter64       OBJECT-TYPE
        SYNTAX  Counter     -- Counter64 is SNMPv2,
                            -- No SMI support for it yet
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
            "A sample 64-bit counter"
        ::= { dpiSimpleMIB 4 }
END
```

To make the code more readable, we have defined the following names in our dpi_mvs_sample.c source file.

```
#define DPI_SIMPLE_SUBAGENT   "1.3.6.1.4.1.2.2.1.5"
#define DPI_SIMPLE_MIB        "1.3.6.1.4.1.2.2.1.5."
#define DPI_SIMPLE_INTEGER    "1.0"  /* dpiSimpleInteger.0   */
#define DPI_SIMPLE_STRING     "2.0"  /* dpiSimpleString.0    */
#define DPI_SIMPLE_COUNTER32  "3.0"  /* dpiSimpleCounter32.0 */
#define DPI_SIMPLE_COUNTER64  "4.0"  /* dpiSimpleCounter64.0 */
```

In addition, we have defined the following variables as global variable in our dpi_mvs_sample.c source file.

```
static int handle;                        /* handle has global scope */
static long int      value1      = 5;
#define              value2_p      cur_val_p   /* writable object    */
#define              value2_len    cur_val_len /* writable object    */
static char         *cur_val_p   = (char *)0;
static char         *new_val_p   = (char *)0;
static char         *old_val_p   = (char *)0;
static unsigned long cur_val_len = 0;
static unsigned long new_val_len = 0;
static unsigned long old_val_len = 0;
static unsigned long value3      = 1;
static snmp_dpi_u64  value4      = {0x80000000,1L};
```

Connecting to the Agent

Before a subagent can receive or send any DPI packets from/to the SNMP DPI capable agent, it must "connect" to the agent and identify itself to the agent.

The following example code returns a response. We assume that there are no errors in the request, but proper code should do the checking for that. We do proper checking for lexicographic next object, but we do no checking for ULONG_MAX, or making sure that the instance ID is indeed valid (digits and dots). If we get to the end of our dpiSimpleMIB, we must return an endOfMibView as defined by the SNMP Version 2 rules. You will need to specify:

- A host name or IP address in dot notation that specifies where the agent is running. Often the name "loopback" or "localhost" can be used if the subagent runs on the same system as the agent.
- A community name which is used to obtain the dpi TCP port from the agent. Internally that is done by sending a regular SNMP GET request to the agent. In an open environment, we probably can use the well known community name "public".

The function returns a negative error code if an error occurs. If the connection setup is successful, it returns a handle which represents the connection and which we must use on subsequent calls to send or await DPI packets.

The second step is to identify the subagent to the agent. This is done by making a DPI-OPEN packet, sending it to the agent, and then awaiting the response from the agent. The agent may accept or deny the OPEN request. Making a DPI-OPEN packet is done by calling mkDPIopen() which expects the following parameters:

- A unique subagent identification (an Object Identifier).
- A description which can be the NULL string ("").
- Overall subagent timeout in seconds. The agent uses this value as a timeout value for a response when it sends a request to the subagent. The agent may have a maximum value for this timeout that will be used if you exceed it.
- The maximum number of varBinds per DPI packet that the subagent is willing or is able to handle.
- The character set we want to use. In most cases you want to use the native character set.
- Length of a password. A zero means no password.
- Pointer to the password or NULL if no password. It depends on the agent if subagents must specify a password to open up a connection.

The function returns a pointer to a static buffer holding the DPI packet if successful. If it fails, it returns a NULL pointer.

Once the DPI-OPEN packet has been created, you must send it to the agent. You can use the DPIsend_packet_to_agent() function which expects the following parameters:

- The handle of a connection from DPIconnect_to_agent_TCP.
- A pointer to the DPI packet from mkDPIopen.

- The length of the packet. The snmp_dpi.h include file provides a macro DPI_PACKET_LEN that calculates the packet length of a DPI packet.

This function returns DPI_RC_OK (value zero) if successful. Otherwise, an appropriate DPI_RC_xxxx error code as defined in snmp_dpi.h is returned.

Now we must wait for a response to the DPI-OPEN. To await such a response, you call the DPIawait_packet_from_agent() function which expects the following parameters:

- The handle of a connection from DPIconnect_to_agent_TCP.
- A timeout in seconds, which is the maximum time to wait for response.
- A pointer to a pointer, which will receive a pointer to a static buffer containing the awaited DPI packet. If the system fails to receive a packet, a NULL pointer is stored.
- A pointer to a long integer (32-bit), which will receive the length of the awaited packet. If it fails, it will be set to zero.

This function returns DPI_RC_OK (value zero) if successful. Otherwise, an appropriate DPI_RC_xxxx error code as defined in snmp_dpi.h is returned.

The last step is to ensure that we received a DPI-RESPONSE back from the agent. If we did, then we must ensure that the agent accepted us as a valid subagent. This will be shown by the error_code field in the DPI response packet.

The following example code establishes a connection and "opens" it by identifying yourself to the agent.

```
static void do_connect_and_open(char *hostname_p, char *community_p)
{
       unsigned char *packet_p;
       int            rc;
       unsigned long  length;
       snmp_dpi_hdr  *hdr_p;

#ifdef INCLUDE_UNIX_DOMAIN_FOR_DPI
        handle =  DPIconnect_to_agent_UNIXstream(hostname_p,
           community_p);
#else
          handle =
                DPIconnect_to_agent_TCP(  /* (TCP) connect to agent  */
                   hostname_p,            /* on this host            */
                   community_p);          /* snmp community name     */
#endif /* INCLUDE_UNIX_DOMAIN_FOR_DPI */
       } /* endif */

       if (handle < 0) exit(1);           /* If it failed, exit      */

       packet_p = mkDPIopen(              /* Make DPI-OPEN packet    */
                    DPI_SIMPLE_SUBAGENT,  /* Our identification      */
                   "Simple DPI subAgent", /* description             */
                    10L,                  /* Our overall timeout     */
                    1L,                   /* max varBinds/packet     */
                    DPI_NATIVE_CSET,      /* native character set    */
                    0L,                   /* password length         */
                    (unsigned char *)0);  /* ptr to password         */
```

```
      if (!packet_p) exit(1);            /* If it failed, exit      */

      rc  = DPIsend_packet_to_agent(     /* send OPEN packet        */
               handle,                   /* on this connection      */
               packet_p,                 /* this is the packet      */
               DPI_PACKET_LEN(packet_p));/* and this is its length  */

      if (rc != DPI_RC_OK) exit(1);      /* If it failed, exit      */

      rc  = DPIawait_packet_from_agent(  /* wait for response       */
               handle,                   /* on this connection      */
               10,                       /* timeout in seconds      */
               &packet_p,                /* receives ptr to packet  */
               &length);                 /* receives packet length  */

      if (rc != DPI_RC_OK) exit(1);      /* If it failed, exit      */

      hdr_p = pDPIpacket(packet_p);      /* parse DPI packet        */
      if (hdr_p == snmp_dpi_hdr_NULL_p)  /* If we fail to parse it  */
         exit(1);                        /* then exit               */

      if (hdr_p->packet_type != SNMP_DPI_RESPONSE) exit(1);

      rc = hdr_p->data_u.resp_p->error_code;
      if (rc != SNMP_ERROR_DPI_noError) exit(1);

} /* end of do_connect_and_open() */
```

Registering a Sub-tree with the Agent

After we have setup a connection to the agent and after we have identified ourselves, we must register one or more MIB sub-trees for which we want to be responsible to handle all SNMP requests.

To do so, the subagent must create a DPI-REGISTER packet and send it to the agent. The agent will then send a response to indicate success or failure of the register request.

To create a DPI-REGISTER packet, the subagent uses a call to the mkDPIregister() function, which expects these parameters:

- A timeout value in seconds for this sub-tree. If you specify zero, your overall timeout value that was specified in DPI-OPEN is used. You can specify a different value if you expect longer processing time for a specific sub-tree.
- A requested priority. Multiple subagents may register the same sub-tree at different priorities. For example, 0 is better than 1 and so on. The agent considers the subagent with the best priority to be the active subagent for the sub-tree. If you specify -1, you are asking for the best priority available. If you specify 0, you are asking for a better priority than any existing subagent may already have.
- The MIB sub-tree which you want to control. You must specify this parameter with a trailing dot.

- You have no choice in GETBULK processing. You must ask the agent to map a GETBULK into multiple GETNEXT packets.

The function returns a pointer to a static buffer holding the DPI packet if successful. If it fails, it returns a NULL pointer.

Now we must send this DPI-REGISTER packet to the agent with the DPIsend_packet_to_agent() function. This is similar to sending the DPI_OPEN packet. We then wait for a response from the agent. Again, we use the DPIawait_packet_from_agent() function in the same way as we awaited a response on the DPI-OPEN request. Once we have received the response, we must check the return code to ensure that registration was successful.

The following code example demonstrates how to register one MIB sub-tree with the agent.

```
static void do_register(void)
{
       unsigned char *packet_p;
       int            rc;
       unsigned long  length;
       snmp_dpi_hdr  *hdr_p;

       packet_p = mkDPIregister(          /* Make DPIregister packet */
                    timeout,              /* timeout in seconds      */
                    0,                    /* requested priority      */
                    DPI_SIMPLE_MIB,       /* ptr to the subtree      */
                    DPI_BULK_NO);         /* Map GetBulk into GetNext*/

       if (!packet_p) exit(1);            /* If it failed, exit      */

       rc  = DPIsend_packet_to_agent(     /* send REGISTER packet    */
                handle,                   /* on this connection      */
                packet_p,                 /* this is the packet      */
                DPI_PACKET_LEN(packet_p));/* and this is its length  */

       if (rc != DPI_RC_OK) exit(1);      /* If it failed, exit      */

       rc  = DPIawait_packet_from_agent(  /* wait for response       */
                handle,                   /* on this connection      */
                10,                       /* timeout in seconds      */
                &packet_p,                /* receives ptr to packet  */
                &length);                 /* receives packet length  */

       if (rc != DPI_RC_OK) exit(1);      /* If it failed, exit      */

       hdr_p = pDPIpacket(packet_p);      /* parse DPI packet        */
       if (hdr_p == snmp_dpi_hdr_NULL_p)  /* If we fail to parse it  */
          exit(1);                        /* then exit               */

       if (hdr_p->packet_type != SNMP_DPI_RESPONSE) exit(1);

       rc = hdr_p->data_u.resp_p->error_code;
       if (rc != SNMP_ERROR_DPI_noError) exit(1);
} /* end of do_register() */
```

Processing Requests from the Agent

After we have registered our sample MIB sub-tree with the agent, we must expect that SNMP requests for that sub-tree will be passed for processing by us. Since the requests will arrive in the form of DPI packets on the connection that we have established, we go into a while loop to await DPI packets from the agent.

Since the subagent cannot know in advance which kind of packet arrives from the agent, we await a DPI packet (forever), then we parse the packet, check the packet type, and process the request based on the DPI packet type. A call to pDPIpacket, which expects as parameter a pointer to the encoded/serialized DPI packet, returns a pointer to a DPI parse tree. The pointer points to a snmp_dpi_hdr structure which looks as follows:

```
struct snmp_dpi_hdr {
  unsigned char  proto_major;
  unsigned char  proto_version;
  unsigned char  proto_release;
  unsigned short packet_id;
  unsigned char  packet_type;
  union {
     snmp_dpi_reg_packet      *reg_p;
     snmp_dpi_ureg_packet     *ureg_p;
     snmp_dpi_get_packet      *get_p;
     snmp_dpi_next_packet     *next_p;
     snmp_dpi_next_packet     *bulk_p;
     snmp_dpi_set_packet      *set_p;
     snmp_dpi_resp_packet     *resp_p;
     snmp_dpi_trap_packet     *trap_p;
     snmp_dpi_open_packet     *open_p;
     snmp_dpi_close_packet    *close_p;
     unsigned char            *any_p;
  } data_u;
};
typedef struct snmp_dpi_hdr    snmp_dpi_hdr;
#define snmp_dpi_hdr_NULL_p    ((snmp_dpi_hdr *)0)
```

With the DPI parse tree, we decide how to process the DPI packet. The following code example demonstrates the high level process of a DPI subagent.

```
#include <snmp_dpi.h>              /* DPI 2.0 API definitions */
static int handle;                 /* handle has global scope */

main(int argc, char *argv[], char *envp[
{
  unsigned char *packet_p;
  int            rc = 0;
  unsigned long  length;
  snmp_dpi_hdr  *hdr_p;

  if (argc>1) {                     /* if use passed one parm */
     if (strcmp(argv[1],"-d")==0)   /* being -d, then we      */
         DPIdebug(2);               /* turn on DPI debugging  */
  } /* endif */                     /* which shows us things  */

  do_connect_and_open();            /* connect and DPI-OPEN   */

  do_register();                    /* register our sub-tree  */

  while (rc == 0) {                 /* do forever             */
   rc = DPIawait_packet_from_agent( /* wait for a DPI packet  */
           handle,                  /* on this connection     */
           -1,                      /* wait forever           */
           &packet_p,               /* receives ptr to packet */
           &length);                /* receives packet length */

   if (rc != DPI_RC_OK) exit(1);    /* If it failed, exit     */

   hdr_p = pDPIpacket(packet_p);    /* parse DPI packet       */
   if (hdr_p == snmp_dpi_hdr_NULL_p)/* If we fail to parse it */
      exit(1);                      /* then exit              */

   switch(hdr_p->packet_type) {     /* handle by DPI type     */
   case SNMP_DPI_GET:
     rc = do_get(hdr_p,
                 hdr_p->data_u.get_p);
     break;
   case SNMP_DPI_GETNEXT:
     rc = do_next(hdr_p,
                  hdr_p->data_u.next_p);
     break;
```

```
   case SNMP_DPI_SET:
   case SNMP_DPI_COMMIT:
   case SNMP_DPI_UNDO:
     rc = do_set(hdr_p,
                 hdr_p->data_u.set_p);
     break;
   case SNMP_DPI_CLOSE:
     rc = do_close(hdr_p,
                   hdr_p->data_u.close_p);
     break;
   case SNMP_DPI_UNREGISTER:
     rc = do_unreg(hdr_p,
                   hdr_p->data_u.ureg_p);
     break;
   default:
     printf("Unexpected DPI packet type %d\n",
            hdr_p->packet_type);
     rc = -1;
   } /* endswitch */
   if (rc) exit(1);
  } /* endwhile */

  return(0);
} /* end of main() */
```

Processing a GET Request

When the DPI packet is parsed, the snmp_dpi_hdr structure will show in the *packet_type* that this is a SNMP_DPI_GET packet. In that case, the *packet_body* contains a pointer to a GET-varBind, which is represented in an snmp_dpi_get_packet structure:

```
struct dpi_get_packet {
  char                  *object_p;   /* ptr to OIDstring     */
  char                  *group_p;    /* ptr to sub-tree      */
  char                  *instance_p; /* ptr to rest of OID   */
  struct dpi_get_packet  *next_p;    /* ptr to next in chain */
};
typedef struct dpi_get_packet       snmp_dpi_get_packet;
#define snmp_dpi_get_packet_NULL_p  ((snmp_dpi_get_packet *)0)
```

Assuming we have registered example sub-tree 1.3.6.1.4.1.2.2.1.5 and a GET request comes in for one variable 1.3.6.1.4.1.2.2.1.5.1.0 so that it is object 1 instance 0 in our sub-tree, the fields in the snmp_dpi_get_packet would have pointers to:

```
  object_p   ->  "1.3.6.1.4.1.2.2.1.5.1.0"
  group_p    ->  "1.3.6.1.4.1.2.2.1.5."
  instance_p ->  "1.0"
  next_p     ->  snmp_dpi_get_packet_NULL_p
```

If there are multiple varBinds in a GET request, each one is represented in a snmp_dpi_get_packet structure and all the snmp_dpi_get_packet structures are chained via the next pointer. As long as the next pointer is not the snmp_dpi_get_packet_NULL_p pointer, there are more varBinds in the list.

Now we can analyze the varBind structure for whatever checking we want to do. Once we are ready to make a response that contains the value of the variable, we prepare a SET-varBind which is represented in an snmp_dpi_set_packet structure:

```
struct dpi_set_packet {
  char                  *object_p;   /* ptr to OIDstring     */
  char                  *group_p;    /* ptr to sub-tree      */
  char                  *instance_p; /* ptr to rest of OID   */
  unsigned char          value_type; /* SNMP_TYPE_xxxx       */
  unsigned short         value_len;  /* value length         */
  char                  *value_p;    /* ptr to value itself  */
  struct dpi_set_packet  *next_p;    /* ptr to next in chain */
};
typedef struct dpi_set_packet       snmp_dpi_set_packet;
#define snmp_dpi_set_packet_NULL_p  ((snmp_dpi_set_packet *)0)
```

We can use the mkDPIset() function to prepare such a structure. This function expects the following parameters:

- A pointer to an existing snmp_dpi_set_packet structure if the new varBind must be added to an existing chain of varBinds. If this is the first or the only varBind in the chain, pass the snmp_dpi_set_packet_NULL_p pointer to indicate this.
- A pointer to the sub-tree that we registered.
- A pointer to the rest of the OID; in other words, the piece that follows the sub-tree.

- The value type of the value to be bound to the variable name. This must be one of the SNMP_TYPE_xxxx values as defined in the snmp_dpi.h include file.
- The length of the value. For integer type values, this must be a length of 4. Work with 32-bit signed or unsigned integers except for the Counter64 type. For the Counter64 type, point to an snmp_dpi_u64 structure and pass the length of that structure.
- A pointer to the value.

Memory for the varBind is dynamically allocated and the data itself is copied. So upon return we can dispose of our own pointers and allocated memory as we please. If the call is successful, a pointer is returned as follows:

- To a new snmp_dpi_set_packet if it is the first or only varBind.
- To the existing snmp_dpi_set_packet that we passed on the call. In this case, the new packet has been chained to the end of the varBind list.

If the mkDPIset() call fails, a NULL pointer is returned.

Once we have prepared the SET-varBind data, we can create a DPI RESPONSE packet using the mkDPIresponse() function which expects these parameters:

- A pointer to an snmp_dpi_hdr. We should use the header of the parsed incoming packet. It is used to copy the *packet_id* from the request into the response, such that the agent can correlate the response to a request.
- A return code which is an SNMP error code. If successful, this should be SNMP_ERROR_noError (value zero). If failure, it must be one of the SNMP_ERROR_xxxx values as defined in the snmp_dpi.h include file.

  A request for a non-existing object or instance is not considered an error. Instead, we must pass a value type of SNMP_TYPE_noSuchObject or SNMP_TYPE_noSuchInstance respectively. These two value types have an implicit value of NULL, so we can pass a zero length and a NULL pointer for the value in this case.
- The index of the varBind in error starts counting at 1. Pass zero if no error occurred, or pass the proper index of the first varBind for which an error was detected.
- A pointer to a chain of snmp_dpi_set_packets (varBinds) to be returned as response to the GET request. If an error was detected, an snmp_dpi_set_packet_NULL_p pointer may be passed.

The following code example returns a response. We assume that there are no errors in the request, but proper code should do the checking for that. For instance, we return a noSuchInstance if the instance is not exactly what we expect and a noSuchObject if the object instance_ID is greater than 3. However, there might be no instance_ID at all and we should check for that too.

```
static int do_get(snmp_dpi_hdr *hdr_p, snmp_dpi_get_packet *pack_p)
{
       unsigned char       *packet_p;
       int                  rc;
       snmp_dpi_set_packet *varBind_p;

       varBind_p =                        /* init the varBind chain  */
          snmp_dpi_set_packet_NULL_p;     /* to a NULL pointer       */
```

```
        if (pack_p->instance_p &&
            (strcmp(pack_p->instance_p,"1.0") == 0))
        {
          varBind_p = mkDPIset(            /* Make DPI set packet     */
                      varBind_p,           /* ptr to varBind chain    */
                      pack_p->group_p,     /* ptr to subtree          */
                      pack_p->instance_p,  /* ptr to rest of OID      */
                      SNMP_TYPE_Integer32, /* value type Integer 32   */
                      sizeof(value1),      /* length of value         */
                      &value1);            /* ptr to value            */
        } else if (pack_p->instance_p &&
            (strcmp(pack_p->instance_p,"2.0") == 0))
        {
          varBind_p = mkDPIset(               /* Make DPI set packet  */
                      varBind_p,              /* ptr to varBind chain */
                      pack_p->group_p,        /* ptr to subtree       */
                      pack_p->instance_p,     /* ptr to rest of OID   */
                      SNMP_TYPE_DisplayString,/* value type           */
                      value2_len,             /* length of value      */
                      value2_p);              /* ptr to value         */
        } else if (pack_p->instance_p &&
            (strcmp(pack_p->instance_p,"3.0") == 0))
        {
          varBind_p = mkDPIset(               /* Make DPI set packet  */
                      varBind_p,              /* ptr to varBind chain */
                      pack_p->group_p,        /* ptr to subtree       */
                      pack_p->instance_p,     /* ptr to rest of OID   */
                      SNMP_TYPE_Counter32,    /* value type           */
                      sizeof(value3),         /* length of value      */
                      &value3);               /* ptr to value         */

#ifndef EXCLUDE_SNMP_V2_SUPPORT
        } else if (pack_p->instance_p &&      /*                *Apr23*/
            (strcmp(pack_p->instance_p,"4.0") == 0))
        {
          varBind_p = mkDPIset(               /* Make DPI set packet  */
                      varBind_p,              /* ptr to varBind chain */
                      pack_p->group_p,        /* ptr to subtree       */
                      pack_p->instance_p,     /* ptr to rest of OID   */
                      SNMP_TYPE_Counter64,    /* value type           */
                      sizeof(value4),         /* length of value      */
                      &value4);               /* ptr to value   *Apr23*/
        } else if (pack_p->instance_p &&
            (strcmp(pack_p->instance_p,"4")>0))
        {
#else
        } else if (pack_p->instance_p &&
            (strcmp(pack_p->instance_p,"3")>0))
        {
#endif /* ndef EXCLUDE_SNMP_V2_SUPPORT */
          varBind_p = mkDPIset(               /* Make DPI set packet  */
                     varBind_p,               /* ptr to varBind chain */
                     pack_p->group_p,         /* ptr to subtree       */
                     pack_p->instance_p,      /* ptr to rest of OID   */
                     SNMP_TYPE_noSuchObject,  /* value type           */
                     0L,                      /* length of value      */
                     (unsigned char *)0);     /* ptr to value         */
```

```
      } else {
        varBind_p = mkDPIset(              /* Make DPI set packet  */
                    varBind_p,             /* ptr to varBind chain */
                    pack_p->group_p,       /* ptr to subtree       */
                    pack_p->instance_p,    /* ptr to rest of OID   */
                    SNMP_TYPE_noSuchInstance,/* value type         */
                    0L,                    /* length of value      */
                    (unsigned char *)0);   /* ptr to value         */
      } /* endif */

      if (!varBind_p) return(-1);        /* If it failed, return    */

      packet_p = mkDPIresponse(          /* Make DPIresponse packet */
                  hdr_p,                 /* ptr parsed request      */
                  SNMP_ERROR_noError,    /* all is OK, no error     */
                  0L,                    /* index is zero, no error */
                  varBind_p);            /* varBind response data   */

      if (!packet_p) return(-1);         /* If it failed, return    */

      rc  = DPIsend_packet_to_agent(     /* send RESPONSE packet    */
               handle,                   /* on this connection      */
               packet_p,                 /* this is the packet      */
               DPI_PACKET_LEN(packet_p));/* and this is its length  */

      return(rc);                        /* return retcode          */
} /* end of do_get() */
```

Processing a GETNEXT Request

When a DPI packet is parsed, the snmp_dpi_hdr structure shows in the packet_type that this is a SNMP_DPI_GETNEXT packet, and so the packet_body contains a pointer to a GETNEXT-varBind, which is represented in an snmp_dpi_next_packet structure:

```
struct dpi_next_packet {
  char                   *object_p;    /* ptr to OIDstring    */
  char                   *group_p;     /* ptr to sub-tree     */
  char                   *instance_p;  /* ptr to rest of OID  */
  struct dpi_next_packet *next_p;      /* ptr to next in chain*/
};
typedef struct dpi_next_packet      snmp_dpi_next_packet;
#define snmp_dpi_next_packet_NULL_p ((snmp_dpi_next_packet *)0)
```

Assuming we have registered example sub-tree dpiSimpleMIB and a GETNEXT arrives for one variable, dpiSimpleInteger.0, so that is object 1 instance 0 in our sub-tree, the fields in the snmp_dpi_get_packet structure would have pointers to:

```
object_p    ->  "1.3.6.1.4.1.2.2.1.5.1.0"
group_p     ->  "1.3.6.1.4.1.2.2.1.5."
instance_p  ->  "1.0"
next_p      ->  snmp_dpi_next_packet_NULL_p
```

If there are multiple varBinds in a GETNEXT request, each one is represented in a snmp_dpi_next_packet structure and all the snmp_dpi_next_packet structures are

chained via the next pointer. As long as the next pointer is not the snmp_dpi_next_packet_NULL_p pointer, there are more varBinds in the list.

Now we can analyze the varBind structure for whatever checking we want to do. We must find out which OID is the one that lexicographically follows the one in the request. It is that OID with its value that we must return as a response. Therefore, we must now also set the proper OID in the response. Once we are ready to make a response that contains the new OID and the value of that variable, we must prepare a SET-varBind which is represented in an snmp_dpi_set_packet:

```
struct dpi_set_packet {
  char                   *object_p;   /* ptr to OIDstring     */
  char                   *group_p;    /* ptr to sub-tree      */
  char                   *instance_p; /* ptr to rest of OID   */
  unsigned char           value_type; /* SNMP_TYPE_xxxx       */
  unsigned short          value_len;  /* value length         */
  char                   *value_p;    /* ptr to value itself  */
  struct dpi_set_packet  *next_p;     /* ptr to next in chain */
};
typedef struct dpi_set_packet        snmp_dpi_set_packet;
#define snmp_dpi_set_packet_NULL_p   ((snmp_dpi_set_packet *)0)
```

We can use the mkDPIset() function to prepare such a structure. This function expects the following parameters:

- A pointer to an existing snmp_dpi_set_packet structure if the new varBind must be added to an existing chain of varBinds. If this is the first or only varBind in the chain, we pass the snmp_dpi_set_packet_NULL_p pointer to indicate this.
- A pointer to the sub-tree that we registered.
- A pointer to the rest of the OID, in other words the piece that follows the sub-tree.
- The value type of the value to be bound to the variable name. This must be one of the SNMP_TYPE_xxxx values as defined in the snmp_dpi.h include file.
- The length of the value. For integer type values, this must be a length of 4. Work with 32-bit signed or unsigned integers except for the Counter64 type. For Counter 64 type, point to a snmp_dpi_u64 structure and pass the length of that structure.
- A pointer to the value.

Memory for the varBind is dynamically allocated and the data itself is copied. Upon return, we can dispose of our own pointers and allocated memory as we please. If the call is successful, a pointer is returned as follows:

- A new snmp_dpi_set_packet if it is the first or only varBind.
- The existing snmp_dpi_set_packet that we passed on the call. In this case, the new packet has been chained to the end of the varBind list.

If the mkDPIset() call fails, a NULL pointer is returned.

Once we have prepared the SET-varBind data, we can create a DPI RESPONSE packet using the mkDPIresponse() function, which expects these parameters:

- A pointer to an snmp_dpi_hdr. We should use the header of the parsed incoming packet. It is used to copy the packet_id from the request into the response, such that the agent can correlate the response to a request.
- A return code which is an SNMP error code. If successful, this should be SNMP_ERROR_noError (value zero). If failure, it must be one of the SNMP_ERROR_xxxx values as defined in the snmp_dpi.h include file.

  A request for a non-existing object or instance is not considered an error. Instead, we must pass the OID and value of the first OID that lexicographically follows the non-existing object and/or instance.

  Reaching the end of our sub-tree is not considered an error. For example, if there is no NEXT OID, this is not an error. In this situation we must return the original OID as received in the request and a value_type of SNMP_TYPE_endOfMibView. This value_type has an implicit value of NULL, so we can pass a zero length and a NULL pointer for the value.
- The index of the first varBind in error starts counting at 1. Pass zero if no error occurred, or pass the proper index of the first varBind for which an error was detected.
- A pointer to a chain of snmp_dpi_set_packet(s) (varBinds) to be returned as response to the GETNEXT request. If an error was detected, an snmp_dpi_set_packet_NULL_p pointer may be passed.

The following code example returns a response. We assume that there are no errors in the request, but proper code should do the checking for that. We do proper checking for lexicographic next object, but we do no checking for ULONG_MAX, or making sure that the instance ID is indeed valid (digits and dots). If we get to the end of our dpiSimpleMIB, we must return an endOfMibView as defined by the SNMP Version 2 rules.

```
static int do_next(snmp_dpi_hdr *hdr_p, snmp_dpi_next_packet *pack_p)
{
       unsigned char       *packet_p;
       int                  rc;
       unsigned long        subid;        /* subid is unsigned       */
       unsigned long        instance;     /* same with instance      */
       char                *cp;
       snmp_dpi_set_packet *varBind_p;

       varBind_p =                        /* init the varBind chain  */
          snmp_dpi_set_packet_NULL_p;     /* to a NULL pointer       */

       if (pack_p->instance_p) {          /* we have an instance ID  */
          cp = pack_p->instance_p;        /* pick up ptr             */
          subid = strtoul(cp, &cp, 10);   /* convert subid (object)  */
          if (*cp == '.') {               /* followed by a dot ?     */
             cp++;                        /* point after it if yes   */
             instance=strtoul(cp,&cp,10); /* convert real instance   */
                                          /* not that we need it, we */
             subid++;                     /* only have instance 0,   */
                                          /* so NEXT is next object  */
             instance = 0;                /* and always instance 0   */
          } else {                        /* no real instance passed */
             instance = 0;                /* so we can use 0         */
             if (subid == 0) subid++;     /* if object 0, start at 1 */
          } /* endif */
```

```
        } else {                          /* no instance ID passed   */
           subid = 1;                     /* so do first object      */
           instance = 0;                  /* instance 0 (all we have)*/
        } /* endif */

        /* we have set subid and instance such that we can basically  */
        /* process the request as a GET now. Actually, we don't even  */
        /* need instance, because all out object instances are zero.  */

        if (instance != 0) printf("Strange instance: %lu\n",instance);

        switch (subid) {
        case 1:
          varBind_p = mkDPIset(           /* Make DPI set packet     */
                      varBind_p,          /* ptr to varBind chain    */
                      pack_p->group_p,    /* ptr to subtree          */
                      DPI_SIMPLE_INTEGER, /* ptr to rest of OID      */
                      SNMP_TYPE_Integer32,/* value type Integer 32   */
                      sizeof(value1),     /* length of value         */
                      &value1);           /* ptr to value            */
          break;
        case 2:
          varBind_p = mkDPIset(              /* Make DPI set packet  */
                      varBind_p,             /* ptr to varBind chain */
                      pack_p->group_p,       /* ptr to subtree       */
                      DPI_SIMPLE_STRING,     /* ptr to rest of OID   */
                      SNMP_TYPE_DisplayString,/* value type          */
                      value2_len,            /* length of value      */
                      value2_p);             /* ptr to value         */
          break;
        case 3:
          varBind_p = mkDPIset(              /* Make DPI set packet  */
                      varBind_p,             /* ptr to varBind chain */
                      pack_p->group_p,       /* ptr to subtree       */
                      DPI_SIMPLE_COUNTER32,  /* ptr to rest of OID   */
                      SNMP_TYPE_Counter32,   /* value type           */
                      sizeof(value3),        /* length of value      */
                      &value3);              /* ptr to value         */
          break;
#ifndef EXCLUDE_SNMP_V2_SUPPORT
        case 4:                              /*               *Apr23*/
          varBind_p = mkDPIset(              /* Make DPI set packet  */
                      varBind_p,             /* ptr to varBind chain */
                      pack_p->group_p,       /* ptr to subtree       */
                      DPI_SIMPLE_COUNTER64,  /* ptr to rest of OID   */
                      SNMP_TYPE_Counter64,   /* value type           */
                      sizeof(value4),        /* length of value      */
                      &value4);              /* ptr to value         */
          break;                             /*               *Apr23*/
#endif /* ndef EXCLUDE_SNMP_V2_SUPPORT */
        default:
          varBind_p = mkDPIset(              /* Make DPI set packet  */
                      varBind_p,             /* ptr to varBind chain */
                      pack_p->group_p,       /* ptr to subtree       */
                      pack_p->instance_p,    /* ptr to rest of OID   */
                      SNMP_TYPE_endOfMibView,/* value type           */
                      0L,                    /* length of value      */
                      (unsigned char *)0);   /* ptr to value         */
```

```
        break;
      } /* endswitch */

      if (!varBind_p) return(-1);        /* If it failed, return    */

      packet_p = mkDPIresponse(          /* Make DPIresponse packet */
                  hdr_p,                 /* ptr parsed request      */
                  SNMP_ERROR_noError,    /* all is OK, no error     */
                  0L,                    /* index is zero, no error */
                  varBind_p);            /* varBind response data   */

      if (!packet_p) return(-1);         /* If it failed, return    */

      rc  = DPIsend_packet_to_agent(     /* send RESPONSE packet    */
               handle,                   /* on this connection      */
               packet_p,                 /* this is the packet      */
               DPI_PACKET_LEN(packet_p));/* and this is its length  */

      return(rc);                        /* return retcode          */
} /* end of do_next() */
```

Processing a SET/COMMIT/UNDO Request

These three requests can come in one of these sequences:

- SET, COMMIT
- SET, UNDO
- SET, COMMIT, UNDO

The normal sequence is SET and then COMMIT. When we receive a SET request, we must make preparations to accept the new value. For example, check that it is for an existing object and instance, check the value type and contents to be valid, allocate memory, but we must not yet make the change.

If there are no SET errors, the next request we receive will be a COMMIT request. It is then that we must make the change, but we must also keep enough information such that we can UNDO the change later if we get a subsequent UNDO request. The latter may happen if the agent discovers any errors with other subagents while processing requests that belong to the same original SNMP SET packet. All the varBinds in the same SNMP request PDU must be processed "as if atomic".

When the DPI packet is parsed, the snmp_dpi_hdr structure shows in the *packet_type* that this is an SNMP_DPI_SET, SNMP_DPI_COMMIT, or SNMP_DPI_UNDO packet. In that case, the packet_body contains a pointer to a SET-varBind, represented in an snmp_dpi_set_packet structure. COMMIT and UNDO have same varBind data as SET upon which they follow:

```
struct dpi_set_packet {
  char                   *object_p;   /* ptr to OIDstring     */
  char                   *group_p;    /* ptr to sub-tree      */
  char                   *instance_p; /* ptr to rest of OID   */
  unsigned char           value_type; /* SNMP_TYPE_xxxx       */
  unsigned short          value_len;  /* value length         */
  char                   *value_p;    /* ptr to value itself  */
  struct dpi_set_packet  *next_p;     /* ptr to next in chain */
};
typedef struct dpi_set_packet        snmp_dpi_set_packet;
#define snmp_dpi_set_packet_NULL_p   ((snmp_dpi_set_packet *)0)
```

Assuming we have registered example sub-tree dpiSimpleMIB and a SET request comes in for one variable dpiSimpleString.0 so that is object 1 instance 0 in our sub-tree, and also assuming that the agent knows about our compiled dpiSimpleMIB so that it knows this is a DisplayString as opposed to just an arbitrary OCTET_STRING, the pointers in the snmp_dpi_set_packet structure would have pointers and values like:

```
object_p    ->  "1.3.6.1.4.1.2.2.1.5.2.0"
group_p     ->  "1.3.6.1.4.1.2.2.1.5."
instance_p  ->  "2.0"
value_type  ->  SNMP_TYPE_DisplayString
value_len   ->  8
value_p     ->  pointer to the value to be set
next_p      ->  snmp_dpi_get_packet_NULL_p
```

If there are multiple varBinds in a SET request, each one is represented in a snmp_dpi_set_packet structure and all the snmp_dpi_set_packet structures are chained via the next pointer. As long as the next pointer is not the snmp_dpi_set_packet_NULL_p pointer, there are more varBinds in the list.

Now we can analyze the varBind structure for whatever checking we want to do. Once we are ready to make a response that contains the value of the variable, we may prepare a new SET-varBind. However, by definition, the response to a successful SET is exactly the same as the SET request. So there is no need to return any varBinds. A response with SNMP_ERROR_noError and an index of zero will do. If there is an error, a response with the SNMP_ERROR_xxxx error code and an index pointing to the varBind in error (counting starts at 1) will do.

The following code example returns a response. We assume that there are no errors in the request, but proper code should do the checking for that. We also do not check if the varBind in the COMMIT and/or UNDO is the same as that in the SET request. A proper agent would make sure that that is the case, but a proper subagent may want to verify that for itself. We only do one check that this is dpiSimpleString.0, and if it is not, we return a noCreation. This may not be correct, the mainline does not even return a response.

```
static int do_set(snmp_dpi_hdr *hdr_p, snmp_dpi_set_packet *pack_p)
{
       unsigned char       *packet_p;
       int                  rc;
       int                  index       = 0;
       int                  error       = SNMP_ERROR_noError;
       snmp_dpi_set_packet *varBind_p;

       varBind_p =                        /* init the varBind chain  */
```

```
   snmp_dpi_set_packet_NULL_p;     /* to a NULL pointer       */

if (!pack_p->instance_p ||
    (strcmp(pack_p->instance_p,"2.0") != 0))
{

   if (pack_p->instance_p &&
      (strncmp(pack_p->instance_p,"1.",2) == 0))
   {
      error = SNMP_ERROR_notWritable;
   } else if (pack_p->instance_p &&
      (strncmp(pack_p->instance_p,"2.",2) == 0))
   {
      error = SNMP_ERROR_noCreation;
   } else if (pack_p->instance_p &&
      (strncmp(pack_p->instance_p,"3.",2) == 0))
   {
      error = SNMP_ERROR_notWritable;
   } else {
      error = SNMP_ERROR_noCreation;
   } /* endif */

   packet_p = mkDPIresponse(       /* Make DPIresponse packet */
             hdr_p,                /* ptr parsed request      */
             error,                /* all is OK, no error     */
             1,                    /* index is 1, 1st varBind */
             varBind_p);           /* varBind response data   */

   if (!packet_p) return(-1);      /* If it failed, return    */

   rc  = DPIsend_packet_to_agent(  /* send RESPONSE packet    */
         handle,                   /* on this connection      */
         packet_p,                 /* this is the packet      */
         DPI_PACKET_LEN(packet_p));/* and this is its length  */

   return(rc);                     /* return retcode          */
}

switch (hdr_p->packet_type) {
case SNMP_DPI_SET:
  if ((pack_p->value_type != SNMP_TYPE_DisplayString) &&
      (pack_p->value_type != SNMP_TYPE_OCTET_STRING))
  {  /* check octet string in case agent has no compiled MIB  */
     error = SNMP_ERROR_wrongType;
     break;                        /* from switch             */
  } /* endif */
  if (new_val_p) free(new_val_p);  /* free these memory areas */
  if (old_val_p) free(old_val_p);  /* if we allocated any     */
  new_val_p   = (char *)0;
  old_val_p   = (char *)0;
  new_val_len = 0;
  old_val_len = 0;

  new_val_p =                     /* allocate memory for     */
      malloc(pack_p->value_len);  /* new value to set        */
  if (new_val_p) {                /* If success, then also   */
     memcpy(new_val_p,            /* copy new value to our   */
            pack_p->value_p,      /* own and newly allocated */
```

```
                pack_p->value_len);    /* memory area.            */
            new_val_len = pack_p->value_len;
          } else {                     /* Else failed to malloc,  */
            error = SNMP_ERROR_genErr; /* so that is a genErr     */
            index = 1;                 /* at first varBind        */
          } /* endif */
          break;
        case SNMP_DPI_COMMIT:
          old_val_p = cur_val_p;       /* save old value for undo */
          cur_val_p = new_val_p;       /* make new value current  */
          new_val_p = (char *)0;       /* keep only 1 ptr around  */
          old_val_len = cur_val_len;   /* and keep lengths correct*/
          cur_val_len = new_val_len;
          new_val_len = 0;
          /* may need to convert from ASCII to native if OCTET_STRING */
          break;
        case SNMP_DPI_UNDO:
          if (new_val_p) {             /* free allocated memory   */
            free(new_val_p);
            new_val_p   = (char *)0;
            new_val_len = 0;
          } /* endif */
          if (old_val_p) {
            if (cur_val_p) free(cur_val_p);
            cur_val_p   = old_val_p;   /* reset to old value      */
            cur_val_len = old_val_len;
            old_val_p   = (char *)0;
            old_val_len = 0;
          } /* endif */
          break;
        } /* endswitch */

        packet_p = mkDPIresponse(      /* Make DPIresponse packet */
                     hdr_p,            /* ptr parsed request      */
                     error,            /* all is OK, no error     */
                     index,            /* index is zero, no error */
                     varBind_p);       /* varBind response data   */

        if (!packet_p) return(-1);     /* If it failed, return    */

        rc  = DPIsend_packet_to_agent( /* send RESPONSE packet    */
                 handle,               /* on this connection      */
                 packet_p,             /* this is the packet      */
                 DPI_PACKET_LEN(packet_p));/* and this is its length  */

        return(rc);                    /* return retcode          */
      } /* end of do_set() */
```

Processing an UNREGISTER Request

An agent can send an UNREGISTER packet if some other subagent does a register for the same sub-tree at a higher priority. An agent can also send an UNREGISTER if, for example, an SNMP manager tells it to "invalidate" the subagent connection or the registered sub-tree.

Here is an example of how to handle such a packet.

```
static int do_unreg(snmp_dpi_hdr *hdr_p, snmp_dpi_ureg_packet *pack_p)
{
       printf("DPI UNREGISTER received from agent, reason=%d\n",
               pack_p->reason_code);
       printf("    subtree=%s\n",pack_p->group_p);
       if (pack_p->reason_code ==
           SNMP_UNREGISTER_higherPriorityRegistered)
       {
          return(0); /* keep waiting, we may regain subtree later */
       } /* endif */

       DPIdisconnect_from_agent(handle);
       return(-1); /* causes exit in main loop */
} /* end of do_unreg() */
```

Processing a CLOSE Request

An agent can send a CLOSE packet if it encounters an error or for some other reason. It can also do so if an SNMP MANAGER tells it to "invalidate" the subagent connection.

Here is an example of how to handle such a packet.

```
static int do_close(snmp_dpi_hdr *hdr_p, snmp_dpi_close_packet *pack_p)
{
       printf("DPI CLOSE received from agent, reason=%d\n",
              pack_p->reason_code);

       DPIdisconnect_from_agent(handle);
       return(-1); /* causes exit in main loop */
} /* end of do_close() */

/*
\end{verbatim}
*/
```

Generating a TRAP

Issue a trap any time after a DPI OPEN was successful. To do so, you must create a trap packet and send it to the agent. With the TRAP, you can pass different kinds of varBinds, if you want. In this example, we pass three varBinds; one with integer data, one with an octet string, and one with a counter. You can also pass an Enterprise ID, but with DPI 2.0, the agent will use your subagent ID as the enterprise ID if you do not pass one with the trap. In most cases that will probably not cause problems.

We must first prepare a varBind list chain that contains the three variables that we want to pass along with the trap. To do so we must prepare a chain of three snmp_dpi_set_packet structures, which looks like:

```
struct dpi_set_packet {
  char                   *object_p;   /* ptr to OIDstring     */
  char                   *group_p;    /* ptr to sub-tree      */
  char                   *instance_p; /* ptr to rest of OID   */
  unsigned char           value_type; /* SNMP_TYPE_xxxx       */
  unsigned short          value_len;  /* value length         */
  char                   *value_p;    /* ptr to value itself  */
  struct dpi_set_packet  *next_p;     /* ptr to next in chain */
};
typedef struct dpi_set_packet        snmp_dpi_set_packet;
#define snmp_dpi_set_packet_NULL_p   ((snmp_dpi_set_packet *)0)
```

We can use the mkDPIset() function to prepare such a structure. This function expects the following parameters:

- A pointer to an existing snmp_dpi_set_packet structure if the new varBind must be added to an existing chain of varBinds. If this is the first or the only varBind in the chain, pass the snmp_dpi_set_packet_NULL_p pointer to indicate this.
- A pointer to the sub-tree that we registered.
- A pointer to the rest of the OID, in other words, the piece that follows the sub-tree.
- The value type of the value to be bound to the variable name. This is must be one of the SNMP_TYPE_xxxx values as defined in the snmp_dpi.h include file.
- The length of the value. For integer type values, this must be a length of 4. We always work with 32-bit signed or unsigned integers except for the Counter64 type. For the Counter64 type, we must point to a snmp_dpi_u64 structure and pass the length of that structure.
- A pointer to the value.

Memory for the varBind is dynamically allocated and the data itself is copied. Upon return, we can dispose of our own pointers and allocated memory as we please. If the call is successful, a pointer is returned as follows:

- To a new snmp_dpi_set_packet if it is the first or only varBind.
- To the existing snmp_dpi_set_packet that we passed on the call. In this case, the new packed has been chained to the end of the varBind list.

If the mkDPIset() call fails, a NULL pointer is returned.

Once we have prepared the SET-varBind data, we can create a DPI TRAP packet. To do so we can use the mkDPItrap() function which expects these parameters:

- The generic trap code. Use 6 for enterprise specific trap type.
- The specific trap type. This is a type that is defined by the MIB which we are implementing. In our example we just use a 1.
- A pointer to a chain of varBinds or the NULL pointer if no varBinds need to be passed with the trap.
- A pointer to the enterprise OID if we want to use a different enterprise ID than the OID we used to identify ourselves as a subagent at DPI-OPEN time.

The following code creates an enterprise— specific trap with specific type 1 and passes three varBinds. The first varBind with our object 1, instance 0, Integer32 value; the second varBind with our object 2, instance 0, Octet String; the third with Counter32. We pass no enterprise ID.

```
static int do_trap(void)
{
       unsigned char       *packet_p;
       int                  rc;
       snmp_dpi_set_packet *varBind_p;

       varBind_p =                        /* init the varBind chain  */
          snmp_dpi_set_packet_NULL_p;     /* to a NULL pointer       */

       varBind_p = mkDPIset(              /* Make DPI set packet     */
                     varBind_p,           /* ptr to varBind chain    */
                     DPI_SIMPLE_MIB,      /* ptr to subtree          */
                     DPI_SIMPLE_INTEGER,  /* ptr to rest of OID      */
                     SNMP_TYPE_Integer32, /* value type Integer 32   */
                     sizeof(value1),      /* length of value         */
                     &value1);            /* ptr to value            */

       if (!varBind_p) return(-1);        /* If it failed, return    */

       varBind_p = mkDPIset(                 /* Make DPI set packet  */
                     varBind_p,              /* ptr to varBind chain */
                     DPI_SIMPLE_MIB,         /* ptr to subtree       */
                     DPI_SIMPLE_STRING,      /* ptr to rest of OID   */
                     SNMP_TYPE_DisplayString,/* value type           */
                     value2_len,             /* length of value      */
                     value2_p);              /* ptr to value         */

       if (!varBind_p) return(-1);        /* If it failed, return    */

       varBind_p = mkDPIset(                 /* Make DPI set packet  */
                     varBind_p,              /* ptr to varBind chain */
                     DPI_SIMPLE_MIB,         /* ptr to subtree       */
                     DPI_SIMPLE_COUNTER32,   /* ptr to rest of OID   */
                     SNMP_TYPE_Counter32,    /* value type           */
                     sizeof(value3),         /* length of value      */
                     &value3);               /* ptr to value         */

       if (!varBind_p) return(-1);        /* If it failed, return    */

#ifndef EXCLUDE_SNMP_V2_SUPPORT
```

```
                                          /*               *Apr23*/
      varBind_p = mkDPIset(               /* Make DPI set packet  */
                   varBind_p,             /* ptr to varBind chain */
                   DPI_SIMPLE_MIB,        /* ptr to subtree       */
                   DPI_SIMPLE_COUNTER64,  /* ptr to rest of OID   */
                   SNMP_TYPE_Counter64,   /* value type           */
                   sizeof(value4),        /* length of value      */
                   &value4);              /* ptr to value         */

      if (!varBind_p) return(-1);        /* If it failed, return    */
                                         /*               *Apr23*/

#endif /* ndef EXCLUDE_SNMP_V2_SUPPORT */

      packet_p = mkDPItrap(              /* Make DPItrap packet     */
                  6,                     /* enterpriseSpecific      */
                  1,                     /* specific type = 1       */
                  varBind_p,             /* varBind data, and use   */
                  (char *)0);            /* default enterpriseID    */

      if (!packet_p) return(-1);         /* If it failed, return    */

      rc  = DPIsend_packet_to_agent(     /* send TRAP packet        */
               handle,                   /* on this connection      */
               packet_p,                 /* this is the packet      */
               DPI_PACKET_LEN(packet_p));/* and this is its length  */

      return(rc);                        /* return retcode          */
} /* end of do_trap() */
```

Chapter 3. Sample SNMP DPI Client Program

This section explains the sample SNMP DPI client program, dpi_mvs_sample.c, installed in /usr/lpp/tcpip/samples. It can be run using the SNMP agents that support the SNMP-DPI interface as described in RFC 1228.

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

The sample implements a set of variables in the dpiSample table, which consists of a set of objects in the IBM Research tree (13.6.1.2.2.1.5). See “dpiSample Table MIB Descriptions” on page 90 for the objectID and type of each object.

Using the Sample Program

The dpi_mvs_sample.c program accepts the following arguments:

? Explains the usage.

-d *n* Sets the debug at level *n*. The range is 0 (for no messages) to 2 (for the most verbose). The default is 1, if you specify —d with no value.

- **0** no debug messages
- **1** packet creation debug messages.
- **2** packet creation debug messages, and traces of packets sent and received. The debug output goes to syslogd because the debug used is dpi.

-h hostname
Specifies the host name or IP address where an SNMP DPI-capable agent is running. The default is localhost.

-c community_name
Specifies the community name for the SNMP agent, which is required to get the dpiPort. The default is public.

The sample uses TCP as the default connect type. In order to get an AF_UNIX connection, define INCLUDE_UNIX_DOMAIN_FOR _DPI before compiling the sample.

Compiling and Linking the dpi_mvs_sample.c Source Code

The dpi_mvs_sample.c program is located in /usr/lpp/tcpip/samples.

You can specify the following compile time flags:

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

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

© Copyright IBM Corp. 1989, 1997

dpiSample Table MIB Descriptions

The following shows the MIB descriptions for DPI sample table.

```
# dpi_mvs_sample.c supports these variables as an SNMP DPI sample sub-agent
# it also generates enterprise specific traps via DPI with these objects
Name                    OID                       Type        Value
----------------- ---------------------- --------- --------


dpiSimpleInteger         1.3.6.1.4.1.2.2.1.5.1.0  integer     5
dpiSimpleString          1.3.6.1.4.1.2.2.1.5.2.0  string      "Initial String"
dpiSimpleCounter32       1.3.6.1.4.1.2.2.1.5.3.0  counter32   1
dpiSimpleCounter64       1.3.6.1.4.1.2.2.1.5.4.0  counter64   X'8000000000000001'

Of the above, only dpiSimpleString can be changed with an SNMP SET
request.
```

Chapter 4. X Window System and OSF/Motif Interface for the OpenEdition Environment

This chapter describes the X Window System application program interface (API)that allows you to write applications in the OpenEdition MVS environment.

There are three X Windows libraries:

- Non-OE (X11R4)
- Open Sockets (X11R4)
- OE Applications Feature (X11R6)

Use of the first two libraries is explained in TCP/IP for MVS: Programmer's Reference. Use of X11R6 is explained here. IBM recommends migration to X11R6.

The X Window System support includes the following APIs from the X Window System Version 11, Release 6:

- X11 Core distribution routines (X11)
- Inter-Client Exchange routines (ICE)
- Session Manager routines (SM)
- X Window System extended routines (Xext) including:
  - XC-MISC - Allows clients to get back ID ranges from the server
  - Big-Requests - Allows large length value in protocol requests
  - Shape - Allows non-rectangular windows
  - Sync - Lets clients synchronize via the X Server
- Authentication functions (Xau)
- X10 compatibility routines (oldX)
- X Toolkit (Xt)
- Utility functions used by Xaw (Xmu)
- Athena Widget set (Xaw)
- PEX (PEX5) 3D Graphics
- Header files needed for compiling X clients
- Selection of standard MIT X clients
- Sample X demos

The X Window System support provided also includes the APIs based on OSF/Motif Release 1.2.4:

- OSF/Motif-based widget set (Xm library)
- OSF/Motif Resource Manager (Mrm library)
- OSF/Motif User Interface language (uil library)
- OSF/Motif User Interface Language Compiler
- Header files needed for compiling clients using the OSF/Motif-based widget set

© Copyright IBM Corp. 1989, 1997

HFS Files

The HFS files used by the X Window System and OSF/Motif and their location in the HFS files are as follows:

- /usr/include/X11 — X Window System header files
- /usr/include — OSF/Motif header files
- /usr/X11 — uil
- /usr/man/C/cat1/uil.1 — This file contains the associated manual page (man page) for the User Interface Language (uil) compiler. It provides online help for the user.
- /usr/lib
  - X Window System and OSF/Motif archive files
  - locales and data files

OpenEdition Application Resource File

The X Window System allows you to modify certain characteristics of an application at run time using application resources. Typically, application resources are set to tailor the appearance and possibly the behavior of an application. The application resources can specify information about an application's window sizes, placement, coloring, font usage, and other functional details.

In the OpenEdition environment, this information can be found in the file

`/u/user_id/.Xdefaults`

where

`/u/user_id`

is found from the environment variable *HOME*.

Identifying the Target Display in OpenEdition

The *DISPLAY* environment variable is used by the X Window System to identify the host name of the target display.

The following is the format of the *DISPLAY* environment variable:

```
host_name:target_server.target_screen
```

Value	Description
host_name	Specifies the host name or IP address of the host machine on which the X Window System server is running.
target_server	Specifies the number of the display server on the host machine.
target_screen	Specifies the screen to be used on the target server.

Programming Considerations

The X Window System toolkit includes files that define two macros for obtaining the offset of fields in an X Window System Toolkit structure, XtOffset, and XtOffsetOf. Programs written for, or ported to, OpenEdition MVS must use the XtOffsetOf macro for this purpose.

Using the X11R6 and OSF/Motif Libraries with DLLs

If the X11R6 and OSF/Motif libraries are used with applications utilizing DLLs, take care to ensure that all references to X11R6 or OSF/Motif functions are made from only one DLL.

Porting Motif Applications to OpenEdition MVS

Some OSF/Motif widget and gadget resources have the type 'KeySym'. In an ASCII-based system the KeySym is the same as the ASCII character value. For example, the character 'F' has the ASCII hexadecimal value 46 and a KeySym hexadecimal value of 46.

However, on OpenEdition MVS the character value of 'F' is hexadecimal C6, while the KeySym hexadecimal value is still 46. Remember to use true KeySym values when specifying resources of type KeySym, whether in a defaults file or in a function call.

In some cases, an X Window System server may have clients that are not running on OpenEdition MVS. If an OE MVS X Window System application sends non-standard properties that contain text strings to the X Window System server, and these properties might be accessed by clients that are not running on OpenEdition MVS, the strings should be translated. The translation should be to the server default character set before transmission to the server and to the appropriate host character set when retrieved from the server.

This translation is an application responsibility.

X Window System Environment Variables

The following is a list of the environment variables examined by the OE MVS support for X Window System Version 11, Release 6:

- DISPLAY - Contains the name of the display to be used. There is no default value.
- XENVIRONMENT - Contains the full pathname of a file containing resource defaults. There is no default value.
- XMODIFIERS - Used by the XSetLocaleModifiers function to specify additional modifiers. There is no default value.
- RESOURCE_NAME - Used by XtOpenDisplay as an alternative specification of an application name. There is no default value.
- XUSERFILEPATH - Used to specify the search paths for files containing application defaults. There is no default value.
- XAPPLRESDIR - Used to specify the directory to search for files containing application defaults. There is no default value.

- XFILESEARCHPATH - Used by XtResolvePathname as a default path. There is no default value.
- SESSION_MANAGER - If defined, causes a Session Shell widget to connect to a session manager. There is no default value.
- XLOCALEDIR - Specifies the directory to be searched for locale files. The default value is '/usr/lib/X11/locale'.
- XWTRACE - Controls the generation of traces of the socket level communications between Xlib and the X Window System server. It controls the traces as follows:
  - XWTRACE undefined or zero - No trace generated.
  - XWTRACE=1 - Error messages
  - XWTRACE>=2 - API function tracing for TRANS functions.

  There is no default value. The output is sent to stderr.
- XWTRACELC - If defined, causes a trace of certain locale sensitive routines. There is no default value. The output is sent to stderr.

EBCDIC/ASCII Translation in MVS OE X Windows

Because the X Window System was designed primarily for an ASCII-based environment, and OpenEdition MVS uses EBCDIC, it is necessary to provide translations between these and also between locale-based coded character sets in OpenEdition MVS and the coded character sets used at the X Window System server. The following sections describe how this is accomplished.

Locale Independent Translation

All arguments for X Window System functions that are specified to be in the Host Portable Character Set are translated between EBCDIC and ASCII by a translation between code page IBM-1047 and code page ISO8859-1. All single byte character set string arguments to X Window System function calls that are not locale dependent (do not have names starting with Xmb or Xwc) are also translated between EBCDIC and ASCII using code page IBM-1047 and ISO8859-1. In addition, properties of type STRING passed to XChangeProperty are translated to ASCII before transmission to the server.

These translations are performed on data being transmitted to the server and on data received from the server that is being returned to the application.

The arguments to X Window System functions of the type XChar2b are not translated. This includes such functions as XDraw16, XDrawText16, and XTextExtents16.

Locale Dependent Translation

The string arguments to X Window System functions with names starting with Xmb or Xwc are translated between the current MVS OE locale codeset (the value returned by nl_info(CODESET)) and the current XLocale. The MVS OE locale is mapped to the XLocale by an entry in /usr/lib/X11/locale/locale.alias. Properties passed to XChangeProperty with a type of the locale encoding-name atom are translated from the MVS OE locale coded character set to the XLocale coded character set.

XTextProperty with COMPOUND_TEXT Encoding

The XTextProperty structure returned by XmbTextListToProperty and XwcTextListToProperty has its property data translated from the MVS OE locale coded character set to the XLocale coded character set if the XTextProperty encoding is COMPOUND_TEXT. Similarly the reverse translation is performed for XmbTextPropertyToTextList and XwcTextPropertyToTextList if the XTextProperty has the encoding COMPOUND_TEXT.

Standard Clients Supplied with MVS OE X Window System Support

The following standard clients are provided in /usr/lpp/tcpip/ X11R6/Xamples/clients:

Client	*Description*
appres	Lists application resource database
atobm	Bitmap conversion utility
bitmap	Bitmap editor
bmtoa	Bitmap conversion utility
editres	Resouce editor
iceauth	ICE authority file utility
oclock	Displays time of day
xauth	X authority file utility
xclipboard	Clipboard utility
xcutsel	Clipboard utility
clock	Analog/digital clock for X
xdpyinfo	Display information utility for X
xfd	X font display utility
xlogo	Displays X logo
xlsatoms	Lists internned atoms defined on server
xlsclients	Lists client applications running on a display
xmag	Magnifies part of screen
xlsfonts	Lists Server fonts
xprop	Property displayer for X
xwininfo	Window information utility for X
xwd	Dumps an image of an X window
xwud	Displays dumped image for X

Use the *man* command to display information about these clients as shown below:

```
man -M /usr/lpp/tcpip/X11R6/Xamples/man client
```

Demo Programs Supplied with MVS OE X Window System Support

The following demo programs are supplied in /usr/lpp/tcpip/X11R6/ Xamples/demos:

xsamp1	Uses only Xlib
xsamp2	Uses Athena widget set
xsamp3	Uses OSF/Motif widget set
pexsamp	Uses PEX5 library

Where Files are Located

The following diagram shows X Window System and OSF/Motif locations in the HFS from a user perspective.

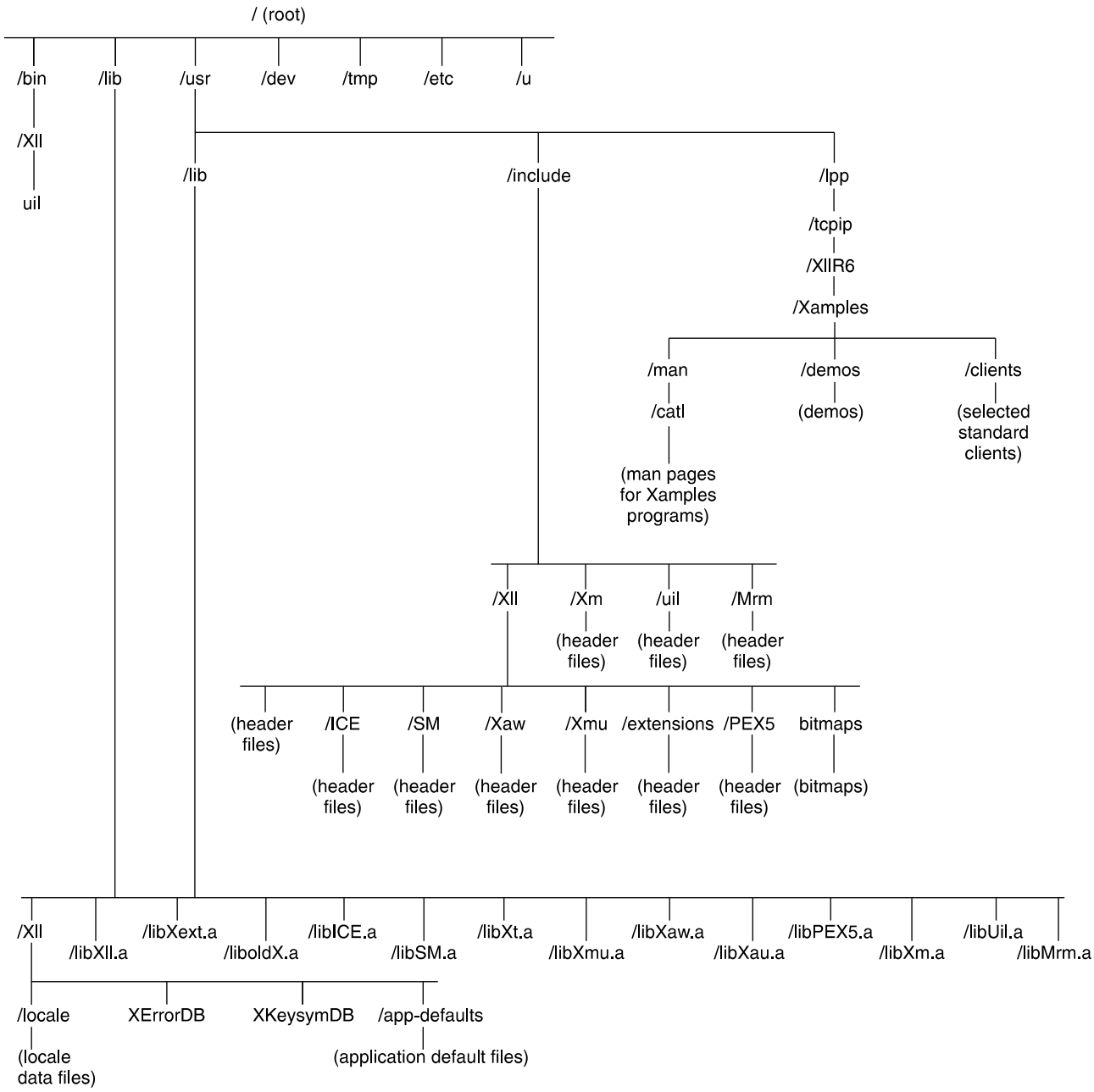


Figure 1. X-Window System and OSF/Motif HFS From a User Perspective

Compiling and Linking OSF/Motif and X Window System Programs

Use the OS/390 OpenEdition 'c89' or 'make' commands to compile and link X Window System and OSF/Motif programs. The following example shows how to use the 'c89' command to compile an X Window System program, xxx, which uses the Athena widget set, and creates the executable file xxx:

```
c89 -o xxx xxx.c -lXaw -lXmu -lXt -lSM -lICE -lX11 -lXau -loldX
```

The following example shows how to use the c89 command to compile an X Window System program yyy, which uses the OSF/Motif widget set, and creates the executable file, yyy.

```
c89 -o xxx xxx.c -lXm -lXt -lSM -lICE -lX11 -lXau -loldX
```

For examples of the input to the 'make' command examine the Makefile in each of the subdirectories of /usr/lpp/tcpip/X11R6/Xamples/demos and /usr/lpp/tcpip/X11R6/Xamples/clients. For more information on the OpenEdition MVS 'c89' and 'make' commands, refer to the OS/390 OpenEdition MVS Command Reference publication.

Chapter 5. RPC in the OpenEdition Environment

The HFS files used by OE RPC and their locations in the HFS are as follows:

- /usr/include/rpc — all header files are contained here
- /usr/lib/librpclib.a — rpc archive files
- orpcgen — ONC RPC protocol compiler
- orpcinfo — utility program for looking a portmaps of networked machines
- oportmap — network service program that maps ONC RPC program and version numbers to transport-specific port numbers.

For information about library functions, see TCP/IP for MVS: Programmer's Reference.

Deviations from Sun RPC 4.0

Source Margins

The source was modified to fit into 72 columns.

Functions

xdr_enum()

In OE rpc xdr_enum() is a macro. This is a change identical to the changes in TCP/IP Version 2 for MVS and VM, and Version 3.1 for MVS. It is necessary because enumerations in C/370 may have a length of one, two, or four bytes. enum_t is not defined, and xdr_enum() is replaced first by a call to _xdr_enum(), which returns the entry to the appropriate XDR routine (xdr_char(), xdr_short(), or xdr_long()) followed by a call to that routine. xdr_union() is also modified into a macro, which separates the call for the discriminant from the remainder. The discriminant is processed as an enumeration, and then passed as a value to _xdr_union() to process the remaining union.

xdr_string()

As with previous 370 versions of TCP/IP, xdr_string() translates from EBCDIC to ASCII or reverse. With OE the iconv() call is used, and data is translated directly into or out of the XDR buffers if sufficient buffer is available as indicated by an xdr_inline() call. With previous versions (or with OE if the entire string will not fit into the buffer) it is necessary to allocate an additional buffer. While encoding, if the length of the data changes in the translation, xdr_setpos() is used to adjust the XDR buffer to reflect the actual amount of translated data. realloc() is used while decoding or for the temporary buffer, which may be necessary while encoding.

The default translation is between ISO8859-1 and IBM-1047. This can be modified by iconv_open() calls during initialization, by specifying the external iconv_t variables xdr_hton_cd and xdr_ntoh_cd.

xdr_float(), xdr_double()

© Copyright IBM Corp. 1989, 1997

The format for S/370 floating point data differs from the IEEE format specified for XDR. The xdr_float() and xdr_double() routines are modified to make the necessary conversions. For OE, these routines utilize the C/370 library routines frexp() and ldexp() to extract and restore the exponent from the floating point number, rather than private subroutines.

Using OE RPC

For RPC, a Sun ONC sample program is provided in /usr/lpp/tcpip/rpc/samples. To run the sample, you can run the Makefile facility in the rpc samples directory. Running *make* produces three executable files.

- printmsg

  The command 'printmsg text' prints the message (text) on the local console. It can be displayed by viewing the system log.

- msg_svc

  msg_svc is an RPC server that enables the user at a remote station to put a message on the console of the server. The command "msg_svc &" starts this server.

- rprintmsg

  The command 'rprintmsg rhost text' prints a message (text) on the console of host "rhost".

  To run *make*, use:

  – cd/usr/lpp/tcpip/rpc/samples

  – ./make

  New cache call function for RPC

```
svcudp_enablecache(transp, size)
SVCXPRT *transp;
u_long size;
```

  where:

  – ***transp*** is the UDP service transport for which caching is to be enabled.

  – ***size*** is the number of entries to be provided in the cache.

  – ***svcudp_enablecache*** enables the caching of replies to remote calls via UDP. When a request due to a retry is received, and there is a reply to an earlier attempt in the cache, the cached reply is immediately returned to the client without calling the remote procedure.

When issuing "rpcgen" for a specification file that contains a "%#", the following compiler error message may be displayed: "ERROR EDC0401 abc.x:n The character is not valid," where "abc.x" is the name of the file and "n" is the line number containing a "%#". This combination of characters is not accepted by the compiler.

For a description of the RPC interface, see TCP/IP for MVS: Programmer's Reference, SC31-7135-02.

Appendix A. Well-Known Port Assignments

This appendix lists the well-known port assignments for transport protocols TCP and UDP, and includes port number, keyword, and a description of the reserved port assignment. You can also find a list of these well-known port numbers in the *hlq*.ETC.SERVICES data set.

Table 2 lists the well-known port assignments for TCP.

Table 2 (Page 1 of 2). TCP Well-Known Port Assignments

Port Number	Keyword	Assigned to	Services Description
0		reserved	
5	**rje**	remote job entry	remote job entry
7	**echo**	echo	echo
9	**discard**	discard	sink null
11	**systat**	active users	active users
13	**daytime**	daytime	daytime
15	**netstat**	netstat	who is up or netstat
19	**chargen**	ttytst source	character generator
21	**ftp**	FTP	File Transfer Protocol
23	**telnet**	telnet	telnet
25	**smtp**	mail	Simple Mail Transfer Protocol
37	**time**	timeserver	timeserver
39	**rlp**	resource	Resource Location Protocol
42	**nameserver**	name	host name server
43	**nicname**	who is	who is
53	**domain**	name server	domain name server
57	**mtp**	private terminal access	private terminal access
69	**tftp**	TFTP	Trivial File Transfer protocol
77	**rje**	netrjs	any private RJE service
79	**finger**	finger	finger
87	**link**	ttylink	any private terminal link
95	**supdup**	supdup	SUPDUP protocol
101	**hostname**	hostname	nic hostname server, usually from SRI-NIC
109	**pop**	postoffice	Post Office Protocol
111	**sunrpc**	sunrpc	Sun remote procedure call
113	**auth**	authentication	authentication service
115	**sftp**	sftp	Simple File Transfer Protocol
117	**uucp-path**	UUCP path service	UUCP path service
119	**untp**	readnews untp	USENET News Transfer Protocol
123	**ntp**	NTP	Network Time Protocol
160–223		reserved	

© Copyright IBM Corp. 1989, 1997

Table 2 (Page 2 of 2). TCP Well-Known Port Assignments

Port Number	Keyword	Assigned to	Services Description
712	**vexec**	vice-exec	Andrew File System authenticated service
713	**vlogin**	vice-login	Andrew File System authenticated service
714	**vshell**	vice-shell	Andrew File System authenticated service
2001	**datasetsrv**		Andrew File System service
2106	**venus.itc**		Andrew File System service, for the Venus process

Well-Known UDP Port Assignments

Table 3 lists the well-known port assignments for UDP.

Table 3 (Page 1 of 2). Well-Known UDP Port Assignments

Port Number	Keyword	Assigned to	Services Description
0		reserved	
5	**rje**	remote job entry	remote job entry
7	**echo**	echo	echo
9	**discard**	discard	sink null
11	**users**	active users	active users
13	**daytime**	daytime	daytime
15	**netstat**	Netstat	Netstat
19	**chargen**	ttytst source	character generator
37	**time**	timeserver	timeserver
39	**rlp**	resource	Resource Location Protocol
42	**nameserver**	name	host name server
43	**nicname**	who is	who is
53	**domain**	nameserver	domain name server
69	**tftp**	TFTP	Trivial File Transfer Protocol
75			any private dial out service
77	**rje**	netrjs	any private RJE service
79	**finger**	finger	finger
111	**sunrpc**	sunrpc	Sun remote procedure call
123	**ntp**	NTP	Network Time Protocol
135	**llbd**	NCS LLBD	NCS local location broker daemon
160–223		reserved	
531	**rvd-control**		rvd control port
2001	**rauth2**		Andrew File System service, for the Venus process
2002	**rfilebulk**		Andrew File System service, for the Venus process

Table 3 (Page 2 of 2). Well-Known UDP Port Assignments

Port Number	Keyword	Assigned to	Services Description
2003	**rfilesrv**		Andrew File System service, for the Venus process
2018	**console**		Andrew File System service
2115	**ropcons**		Andrew File System service, for the Venus process
2131	**rupdsrv**		assigned in pairs; bulk must be **srv** +1
2132	**rupdbulk**		assigned in pairs; bulk must be **srv** +1
2133	**rupdsrv1**		assigned in pairs; bulk must be **srv** +1
2134	**rupdbulk1**		assigned in pairs; bulk must be **srv** +1

Appendix B. Related Protocol Specifications (RFCs)

This appendix lists the related protocol specifications for TCP/IP for MVS. The internet protocol suite is still evolving through Requests for Comments (RFC). New protocols are being designed and implemented by researchers, and are brought to the attention of the internet community in the form of RFCs. Some of these are so useful that they become a recommended protocol. That is, all future implementations for TCP/IP are recommended to implement this particular function or protocol. These become the *de facto* standards, on which the TCP/IP protocol suite is built.

Many features of TCP/IP for MVS are based on the following RFCs:

RFC	Title and Author
768	*User Datagram Protocol* J.B. Postel
791	*Internet Protocol* J.B. Postel
792	*Internet Control Message Protocol* J.B. Postel
793	*Transmission Control Protocol* J.B. Postel
821	*Simple Mail Transfer Protocol* J.B. Postel
822	*Standard for the Format of ARPA Internet Text Messages* D. Crocker
823	*DARPA Internet Gateway* R.M. Hinden, A. Sheltzer
826	*Ethernet Address Resolution Protocol: or Converting Network Protocol Addresses to 48.Bit Ethernet Address for Transmission on Ethernet Hardware* D.C. Plummer
854	*Telnet Protocol Specification* J.B. Postel, J.K. Reynolds
856	*Telnet Binary Transmission* J.B. Postel, J.K. Reynolds
857	*Telnet Echo Option* J.B. Postel, J.K. Reynolds
862	*Echo Protocol* J.B. Postel
863	*Discard Protocol* J.B. Postel
864	*Character Generator Protocol* J.B. Postel
877	*Standard for the Transmission of IP Datagrams over Public Data Networks* J.T. Korb
885	*Telnet End of Record Option* J.B. Postel
903	*Reverse Address Resolution Protocol* R. Finlayson, T. Mann, J.C. Mogul, M. Theimer
904	*Exterior Gateway Protocol Formal Specification* D.L. Mills
919	*Broadcasting Internet Datagrams* J.C. Mogul
922	*Broadcasting Internet Datagrams in the Presence of Subnets* J.C. Mogul
950	*Internet Standard Subnetting Procedure* J.C. Mogul, J.B. Postel
952	*DoD Internet Host Table Specification* K. Harrenstien, M.K. Stahl, E.J. Feinler
959	*File Transfer Protocol* J.B. Postel, J.K. Reynolds
974	*Mail Routing and the Domain Name System* C. Partridge

© Copyright IBM Corp. 1989, 1997

1009 *Requirements for Internet Gateways* R.T. Braden, J.B. Postel

1013 *X Window System Protocol, Version 11: Alpha Update* R.W. Scheifler

1014 *XDR: External Data Representation Standard* Sun Microsystems Incorporated

1027 *Using ARP to Implement Transparent Subnet Gateways* S. Carl-Mitchell, J.S. Quarterman

1032 *Domain Administrators Guide* M.K. Stahl

1033 *Domain Administrators Operations Guide* M. Lottor

1034 *Domain Names—Concepts and Facilities* P.V. Mockapetris

1035 *Domain Names—Implementation and Specification* P.V. Mockapetris

1042 *Standard for the Transmission of IP Datagrams over IEEE 802 Networks* J.B. Postel, J.K. Reynolds

1044 *Internet Protocol on Network System's HYPERchannel: Protocol Specification* K. Hardwick, J. Lekashman

1055 *Nonstandard for Transmission of IP Datagrams over Serial Lines: SLIP* J.L. Romkey

1057 *RPC: Remote Procedure Call Protocol Version 2 Specification* Sun Microsystems Incorporated

1058 *Routing Information Protocol* C.L. Hedrick

1091 *Telnet Terminal-Type Option* J. VanBokkelen

1094 *NFS: Network File System Protocol Specification* Sun Microsystems Incorporated

1118 *Hitchhikers Guide to the Internet* E. Krol

1122 *Requirements for Internet Hosts—Communication Layers* R.T. Braden

1123 *Requirements for Internet Hosts—Application and Support* R.T. Braden

1155 *Structure and Identification of Management Information for TCP/IP-Based Internets* M.T. Rose, K. McCloghrie

1156 *Management Information Base for Network Management of TCP/IP-based Internets* K. McCloghrie, M.T. Rose

1157 *Simple Network Management Protocol (SNMP),* J.D. Case, M. Fedor, M.L. Schoffstall, C. Davin

1179 *Line Printer Daemon Protocol* The Wollongong Group, L. McLaughlin III

1180 *TCP/IP Tutorial,* T.J. Socolofsky, C.J. Kale

1183 *New DNS RR Definitions* C.F. Everhart, L.A. Mamakos, R. Ullmann, P.V. Mockapetris, (Updates RFC 1034, RFC 1035)

1187 *Bulk Table Retrieval with the SNMP* M.T. Rose, K. McCloghrie, J.R. Davin

1188 *Proposed Standard for the Transmission of IP Datagrams over FDDI Networks* D. Katz

1198 *FYI on the X Window System* R.W. Scheifler

1207 *FYI on Questions and Answers: Answers to Commonly Asked :q.Experienced Internet User:eq. Questions* G.S. Malkin, A.N. Marine, J.K. Reynolds

1208 *Glossary of Networking Terms* O.J. Jacobsen, D.C. Lynch

1213 *Management Information Base for Network Management of TCP/IP-Based Internets: MIB-II,* K. McCloghrie, M.T. Rose

1215 *Convention for Defining Traps for Use with the SNMP* M.T. Rose

1228 *SNMP-DPI Simple Network Management Protocol Distributed Program Interface* G.C. Carpenter, B. Wijnen

1229 *Extensions to the Generic-Interface MIB* K. McCloghrie

1230 *IEEE 802.4 Token Bus MIB IEEE 802 4 Token Bus MIB* K. McCloghrie, R. Fox

1231 *IEEE 802.5 Token Ring MIB IEEE 802.5 Token Ring MIB* K. McCloghrie, R. Fox, E. Decker

1267 *A Border Gateway Protocol 3 (BGP-3)* K. Lougheed, Y. Rekhter

1268 *Application of the Border Gateway Protocol in the Internet* Y. Rekhter, P. Gross

1269 *Definitions of Managed Objects for the Border Gateway Protocol (Version 3)* S. Willis, J. Burruss

1270 *SNMP Communications Services*, F. Kastenholz, ed.

1325 *FYI on Questions and Answers: Answers to Commonly Asked :q.New Internet User:eq. Questions* G.S. Malkin, A.N. Marine

1340 *Assigned Numbers* J.K. Reynolds, J.B. Postel

1350 *TFTP Protocol* K.R. Sollins

1351 *SNMP Administrative Model* J. Davin, J. Galvin, K. McCloghrie

1352 *SNMP Security Protocols* J. Galvin, K. McCloghrie, J. Davin

1353 *Definitions of Managed Objects for Administration of SNMP Parties* K. McCloghrie, J. Davin, J. Galvin

1354 *IP Forwarding Table MIB* F. Baker

1356 *Multiprotocol Interconnect on X.25 and ISDN in the Packet Mode* A. Malis, D. Robinson, R. Ullmann

1374 *IP and ARP on HIPPI* J. Renwick, A. Nicholson

1381 *SNMP MIB Extension for X.25 LAPB* D. Throop, F. Baker

1382 *SNMP MIB Extension for the X.25 Packet Layer* D. Throop

1387 *RIP Version 2 Protocol Analysis* G. Malkin

1388 *RIP Version 2 — Carrying Additional Information* G. Malkin

1389 *RIP Version 2 MIB Extension* G. Malkin

1390 *Transmission of IP and ARP over FDDI Networks* D. Katz

1393 *Traceroute Using an IP Option* G. Malkin

1397 *Default Route Advertisement In BGP2 And BGP3 Versions of the Border Gateway Protocol* D. Haskin

1398 *Definitions of Managed Objects for the Ethernet-like Interface Types* F. Kastenholz

1540 *IAB Official Protocol Standards* J.B. Postel

1901 *Introduction to Community-based SNMPv2* J. Case, SNMP Research, Inc.; K. McCloghrie, Cisco Systems, Inc.; M. Rose, Dover Beach Consulting, Inc.; S. Waldbusser, International Network Services.

1902 *Structure of Management Information* J. Case, SNMP Research, Inc.; K. McCloghrie, Cisco Systems, Inc.; M. Rose, Dover Beach Consulting, Inc.; S. Waldbusser, International Network Services.

1903 *Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMP V2)* J. Case, SNMP Research, Inc.; K. McCloghrie, Cisco Systems, Inc.; M. Rose, Dover Beach Consulting, Inc.; S. Waldbusser, International Network Services.

1904 *Conformance Statements for Version 2 of the Simple Network Management Protocol (SNMP V2)* J. Case, SNMP Research, Inc.; K. McCloghrie, Cisco Systems, Inc.; M. Rose, Dover Beach Consulting, Inc.; S. Waldbusser, International Network Services.

1905 *Protocol Operations for the Simple Network Management Protocol (SNMP V2)* J. Case, SNMP Research, Inc.; K. McCloghrie, Cisco Systems, Inc.; M. Rose, Dover Beach Consulting, Inc.; S. Waldbusser, International Network Services.

1906 *Transport Mappings for Version 2 of the Simple Network Protocol (SNMPv2)* J. Case, SNMP Research, Inc.; K. McCloghrie, Cisco Systems, Inc.; M. Rose, Dover Beach Consulting, Inc.; S. Waldbusser, International Network Services.

1907 *Management Information Base for Version 2 of the Simple Network Managemetn Protocal (SNMP V2)* J. Case, SNMP Research, Inc.; K. McCloghrie, Cisco Systems, Inc.; M. Rose, Dover Beach Consulting, Inc.; S. Waldbusser, International Network Services.

1908 *Coexistence between Version 1 and Version 2 of the Internet-standard Network Management Framework (SNMP V2)* J. Case, SNMP Research, Inc.; K. McCloghrie, Cisco Systems, Inc.; M. Rose, Dover Beach Consulting, Inc.; S. Waldbusser, International Network Services.

1909 *An Administration Infrastructure for SNMPv2* K. McCloghrie, Cisco Systems, Inc.

1910 *User-based Security Model for SNMPv2* G. Waters, Bell-Northern Research Ltd.

These documents can be obtained from:

Government Systems, Inc.
Attn: Network Information Center
14200 Park Meadow Drive
Suite 200
Chantilly, VA 22021

Many RFCs are available online. Hard copies of all RFCs are available from the NIC, either individually or on a subscription basis. Online copies are available using FTP from the NIC at `nic.ddn.mil`. Use FTP to download the files, using the following format:

```
RFC:RFC-INDEX.TXT
RFC:RFCnnnn.TXT
RFC:RFCnnnn.PS
```

Where:

nnnn	Is the RFC number.
TXT	Is the text format.
PS	Is the PostScript** format.

You can also request RFCs through electronic mail, from the automated NIC mail server, by sending a message to `service@nic.ddn.mil` with a subject line of `RFC nnnn` for text versions or a subject line of `RFC nnnn.PS` for PostScript versions. To request a copy of the RFC index, send a message with a subject line of `RFC INDEX`.

For more information, contact `nic@nic.ddn.mil`.

Appendix C. Abbreviations and Acronyms

This appendix lists the abbreviations and acronyms used throughout this book.

AIX	Advanced Interactive Executive
ANSI	American National Standards Institute
API	Application Program Interface
APPC	Advanced Program-to-Program Communications
APPN	Advanced Peer-to-Peer Networking
ARP	Address Resolution Protocol
ASCII	American National Standard Code for Information Interchange
ASN.1	Abstract Syntax Notation One
AUI	Attachment Unit Interface
BIOS	Basic Input/Output System
BNC	Bayonet Neill-Concelman
CCITT	Comite Consultatif International Telegraphique et Telephonique. The International Telegraph and Telephone Consultative Committee
CETI	Continuously Executing Transfer Interface
CLAW	Common Link Access to Workstation
CLIST	Command List
CMS	Conversational Monitor System
CP	Control Program
CPI	Common Programming Interface
CREN	Corporation for Research and Education Networking
CSD	Corrective Service Diskette
CTC	Channel-to-Channel
CU	Control Unit
CUA	Common User Access
DASD	Direct Access Storage Device
DBCS	Double Byte Character Set
DLL	Dynamic Link Library
DNS	Domain Name System
DOS	Disk Operating System
DPI	Distributed Program Interface
EBCDIC	Extended Binary-Coded Decimal Interchange Code
ELANS	IBM Ethernet LAN Subsystem
EISA	Enhanced Industry Standard Adapter
ESCON	Enterprise Systems Connection

© Copyright IBM Corp. 1989, 1997

FAT	File Allocation Table
FDDI	Fiber Distributed Data Interface
FTAM	File Transfer Access Management
FTP	File Transfer Protocol
FTP API	File Transfer Protocol Applications Programming Interface
GCS	Group Control System
GDDM	Graphical Data Display Manager
GDDMXD	Graphics Data Display Manager Interface for X Window System
GDF	Graphics Data File
HCH	HYPERchannel device
HIPPI	High Performance Parallel Interface
HPFS	High Performance File System
ICAT	Installation Configuration Automation Tool
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronic Engineers
IETF	Internet Engineering Task Force
ILANS	IBM Token-Ring LAN Subsystem
IP	Internet Protocol
IPL	Initial Program Load
ISA	Industry Standard Adapter
ISDN	Integrated Services Digital Network
ISO	International Organization for Standardization
IUCV	Inter-User Communication Vehicle
JES	Job Entry Subsystem
JIS	Japanese Institute of Standards
JCL	Job Control Language
LAN	Local Area Network
LAPS	LAN Adapter Protocol Support
LCS	IBM LAN Channel Station
LPD	Line Printer Daemon
LPQ	Line Printer Query
LPR	Line Printer Client
LPRM	Line Printer Remove
LPRMON	Line Printer Monitor
LU	Logical Unit
MAC	Media Access Control
Mbps	Megabits per second

MBps Megabytes per second

MCA Micro Channel Adapter

MIB Management Information Base

MIH Missing Interrupt Handler

MILNET Military Network

MHS Message Handling System

MTU Maximum Transmission Unit

MVS Multiple Virtual Storage

MX Mail Exchange

NCP Network Control Program

NCS Network Computing System

NDIS Network Driver Interface Specification

NFS Network File System

NIC Network Information Center

NLS National Language Support

NSFNET National Science Foundation Network

OS/2 Operating System/2

OSF Open Software Foundation, Inc.

OSI Open Systems Interconnection

OSIMF/6000 Open Systems Interconnection Messaging and Filing/6000

OV/MVS OfficeVision/MVS

OV/VM OfficeVision/VM

PAD Packet Assembly/Disassembly

PC Personal Computer

PCA Parallel Channel Adapter

PDN Public Data Network

PDU Protocol Data Units

PING Packet Internet Groper

PIOAM Parallel I/O Access Method

POP Post Office Protocol

PROFS Professional Office Systems

PSCA Personal System Channel Attach

PSDN Packet Switching Data Network

PU Physical Unit

PVM Passthrough Virtual Machine

RACF Resource Access Control Facility

RARP Reverse Address Resolution Protocol

REXEC	Remote Execution
REXX	Restructured Extended Executor Language
RFC	Request For Comments
RIP	Routing Information Protocol
RISC	Reduced Instruction Set Computer
RPC	Remote Procedure Call
RSCS	Remote Spooling Communications Subsystem
SAA	System Application Architecture
SBCS	Single Byte Character Set
SDLC	Synchronous Data Link Control
SLIP	Serial Line Internet Protocol
SMI	Structure for Management Information
SMTP	Simple Mail Transfer Protocol
SNA	Systems Network Architecture
SNMP	Simple Network Management Protocol
SOA	Start of Authority
SPOOL	Simultaneous Peripheral Operations Online
SQL	IBM Structured Query Language
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TFTP	Trivial File Transfer Protocol
TSO	Time Sharing Option
TTL	Time-to-Live
UDP	User Datagram Protocol
VGA	Video Graphic Array
VM	Virtual Machine
VMCF	Virtual Machine Communication Facility
VM/SP	Virtual Machine/System Product
VM/XA	Virtual Machine/Extended Architecture
VTAM	Virtual Telecommunications Access Method
WAN	Wide Area Network
XDR	eXternal Data Representation

Appendix D. Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make them available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

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

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
USA

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Site Counsel
IBM Corporation
P.O. Box 12195
3039 Cornwallis Road
Research Triangle Park, NC 27709-2195
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

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

This document is not intended for production use and is furnished as is without any warranty of any kind, and all warranties are hereby disclaimed including the warranties of merchantability and fitness for a particular purpose.

Trademarks

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

© Copyright IBM Corp. 1989, 1997

ACF/VTAM
AD/Cycle
AIX
AIX/ESA
BookManager
C/370
CICS
DB2
DFSMS
DFSMS/MVS
ESCON
ES/9000
EtherStreamer
Extended Services
GDDM
Hardware Configuration Definition
IBM
LANStreamer
Library Reader
MVS/ESA
MVS/SP
MVS/XA
NetView
OpenEdition
OS/2
OS/390
PS/2
RACF
RISC System/6000
RS/6000
SAA
System/370
System/390
VTAM
3090

The following terms are trademarks of other companies:

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Microsoft, Windows, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

Bibliography

This bibliography lists the publications for IBM TCP/IP products.

IBM TCP/IP Publications

The following sections describe the books associated with IBM TCP/IP products.

OS/390 TCP/IP OpenEdition Publications

- *OS/390 TCP/IP OpenEdition Configuration Guide*, SC31-8304-00.

  This book is for people who want to configure, customize, administer, and maintain OS/390 TCP/IP OpenEdition. Familiarity with MVS operating system, TCP/IP protocols, and IBM Time Sharing Option (TSO) is recommended.

- *OS/390 TCP/IP OpenEdition Diagnosis Guide*, SC31-8492-00.

  This book explains how to diagnose TCP/IP problems and how to determine whether a specific problem is in the OS/390 TCP/IP OpenEdition product code. It explains how to gather information for and describe problems to the IBM Software Support Center.

- *OS/390 TCP/IP OpenEdition Messages and Codes*, SC31-8307-00.

  This book explains the informational and error messages issued by OS/390 TCP/IP OpenEdition. It can help users, operators, or system programmers to diagnose and fix problems identified by error messages.

- *OS/390 TCP/IP OpenEdition Planning and Release Guide*, SC31-8303-00.

  This book is intended to help you plan for OS/390 TCP/IP OpenEdition whether you are migrating from a previous version or installing TCP/IP for the first time. This book also identifies the suggested and required modifications needed to enable you to use the enhanced functions provided with OS/390 TCP/IP OpenEdition.

- *OS/390 TCP/IP OpenEdition Programmer's Reference*, SC31-8308-00

  This book describes the syntax and semantics of a set of high-level application functions that you can use to program your own applications in a TCP/IP environment. These functions provide support for application facilities, such as user authentication, distributed databases, distributed processing, network management, and device sharing.

  This book is for people who want to use the supplied interfaces while writing application programs that access OS/390 TCP/IP OpenEdition. Familiarity with the MVS operating system, TCP/IP protocols, and IBM Time Sharing Option (TSO) is recommended.

- *OS/390 TCP/IP OpenEdition User's Guide*, GC31-8305-00.

  This book is for people who want to use OS/390 TCP/IP OpenEdition for data communication. Familiarity with MVS operating system and IBM Time Sharing Option (TSO) is recommended.

TCP/IP for MVS Publications

- *TCP/IP Version 3 for OpenEdition MVS: Applications Feature Guide*, SC31-8069-00.

  This book explains how to plan for, install, customize, and use the OpenEdition MVS Applications Feature. The Feature consists of applications and interfaces for direct access to the OpenEdition MVS environment. For example, users of the Feature can use MVS, UNIX, or AIX commands to transfer files, log in to the OpenEdition environment without going through TSO, and run commands remotely. This book also explains how to improve performance and diagnose problems when using the Feature.

- *TCP/IP for MVS: Application Programming Interface Reference*, SC31-7187-02.

  This book describes the syntax and semantics of program source code necessary to write your own application programming interface (API) into TCP/IP. You can use this interface as the communication base for writing your own client or server application. You can also use this book to adapt your existing applications to communicate with each other using sockets over TCP/IP.

- *TCP/IP for MVS: CICS TCP/IP Socket Interface Guide and Reference*, SC31-7131-02.

  This book is for people who want to set up, write application programs for, and diagnose problems with the socket interface for CICS using TCP/IP for MVS.

- *TCP/IP for MVS: Customization and Administration Guide*, SC31-7134-03.

  This book is for people who want to con customize, administer, and maintain TCP/IP for MVS. Familiarity with MVS operating system, TCP/IP protocols,

© Copyright IBM Corp. 1989, 1997

and IBM Time Sharing Option (TSO) is recommended.

- *TCP/IP for MVS: Diagnosis Guide*, LY43-0105-02.

  This book explains how to diagnose TCP/IP problems and how to determine whether a specific problem is in the IBM TCP/IP for MVS product code. It explains how to gather information for and describe problems to the IBM Software Support Center.

- *TCP/IP for MVS: IMS TCP/IP Application Development Guide and Reference*, SC31-7186-02.

  This book is for programmers who want application programs that use the IMS TCP/IP application development services provided by IBM TCP/IP for MVS.

- *TCP/IP for MVS: Messages and Codes*, SC31-7132-03.

  This book explains the informational a nd error messages issued by IBM TCP/IP for MVS. It can help users, operators, or system programmers to diagnose and fix problems identified by TCP/IP for MVS error messages.

- *TCP/IP for MVS: Network Print Facility*, SC31-8074-03.

  This book is for system programmers an d network administrators who need to prepare their network to route VTAM, JES2, or JES3 printer output to remote printers using TCP/IP for MVS.

- *TCP/IP for MVS: Offloading TCP/IP Processing*, SC31-7133-02.

  This book is for people who want to install and configure the Offload feature on IBM 3172 Model 3 Interconnect Controllers. This book is also for people who want to use and customize the Offload feature of TCP/IP for MVS.

- *TCP/IP for MVS: Planning and Migration Guide*, SC31-7189-01.

  This book is intended to help you plan for TCP/IP for MVS whether you are migrating from a previous version or installing TCP/IP for MVS for the first time. This book also identifies the suggested and required modifications needed to enable you to use the enhanced functions provided with TCP/IP for MVS.

- *TCP/IP: Performance Tuning Guide*, SC31-7188-02.

  This book describes how to improve the performance of your network operations.

- *TCP/IP for MVS: Programmer's Reference*, SC31-7135-02.

  This book describes the syntax and semantics of a set of high-level application functions that you can use to program your own applications in a TCP/IP environment. These functions provide support for application facilities, such as user authentication, distributed databases, distributed processing, network management, and device sharing.

  This book is for people who want to use the supplied interfaces while writing application programs that access TCP/IP for MVS. Familiarity with the MVS operating system, TCP/IP protocols, and IBM Time Sharing Option (TSO) is recommended.

- *TCP/IP for MVS: User's Guide*, SC31-7136-02.

  This book is for people who want to use TCP/IP for MVS for data communication. Familiarity with MVS operating system and IBM Time Sharing Option (TSO) is recommended.

TCP/IP for VM Publications

The following list describes books in the IBM TCP/IP for VM library.

- *IBM TCP/IP Version 2 Release 4 for VM: Messages and Codes*, SC31-6151-03.

  This book is for system programmers who want to diagnose and fix problems identified by TCP/IP for VM error messages.

- *IBM TCP/IP Version 2 Release 4 for VM: Planning and Customization*, SC31-6082-03.

  This book is for system programmers who want to plan and customize the TCP/IP for VM environment.

- *IBM TCP/IP Version 2 Release 4 for VM: Programmer's Reference*, SC31-6084-03.

  This book is for application and system programmers who want to write application programs that use TCP/IP for VM. Application programmers should know the VM operating system.

- *IBM TCP/IP Version 2 Release 4 for VM: User's Guide*, SC31-6081-03.

  This book is for people who want to use TCP/IP for VM for data communication. Familiarity with VM operating system, IBM Command Processor (CP), and IBM Conversational Monitor System (CMS) is recommended.

TCP/IP for OS/2 Publication

IBM TCP/IP Version 3.0 for OS/2: Programmer's Reference, SC31-6077.

This book provides application and system programmers with the information required to write application programs that use TCP/IP for OS/2. Programmers should know the OS/2 operating system.

TCP/IP for DOS Publications

The following list describes books in the IBM TCP/IP for DOS library.

- *IBM TCP/IP Version 2.1.1 for DOS: Command Reference*, SX75-0083.

  This book is for people who use a workstation with TCP/IP for DOS, such as end users and system programmers. The people who use this book should be familiar with DOS and the workstation, understand DOS operating system concepts, and be familiar with the *IBM TCP/IP Version 2.1.1 for DOS: User's Guide*

- *IBM TCP/IP Version 2.1.1 for DOS: Installation and Administration*, SC31-7047.

  This book provides system programmers, network administrars, and workstation users responsible for installing TCP/IP for DOS with the information required to plan and implement the installation of TCP/IP for DOS. The topics include hardware and software requirements, pre-installation system performance considerations, instructions for installing TCP/IP for DOS, instructions for customizing the TCP/IP for DOS environment, and installation examples.

- *IBM TCP/IP Version 2.1.1 for DOS: Programmer's Reference*, SC31-7046.

  This book is for application and system programmers to aid them in writing application programs that use TCP/IP for DOS on a workstation. Application programmers should know the DOS operating system and multitasking operating system concepts. Application programmers should be knowledgeable in the C programming language.

- *IBM TCP/IP Version 2.1.1 for DOS: User's Guide*, SC31-745.

  This book is for people who use a workstation with TCP/IP for DOS, such as end users and system programmers. The people who use this book should be familiar with DOS and the workstation, and also understand DOS operating system concepts.

TCP/IP for AIX (RS/6001, PS/2, RT, 370) Publications

The following list shows books in the TCP/IP for AIX library.

- *AIX Operating System TCP/IP User's Guide*, SC23-2309.
- *AIX PS/2 TCP/IP User's Guide*, SC23-2047.
- *TCP/IP for IBM X-Windows on DOS 2.1*, SC23-2349.

TCP/IP for AS/400 Publications

The following list shows books in the TCP/IP for AS/400 library.

- *IBM AS/400 Communications: TCP/IP Guide*, SC41-9875.
- *IBM AS/400 Communications: User's Guide*, SC21-9601.

Other IBM TCP/IP Publications

The following list shows other available IBM TCP/IP books.

- *IBM Local Area Network Technical Reference*, SC30-3383.
- *IBM TCP/IP for VM and MVS: Diagnosis Guide*, LY43-0013.
- *TCP/IP and National Language Support*, GG24-3840.
- *TCP/IP Introduction*, GC31-6080.
- *TCP/IP Tutorial and Technical Overview*, GG24-3376.

IBM Operating System Publications

The following lists show books about various IBM operating systems.

AIX Publications

- *AIX Communications Concepts and Procedures for IBM RISC System/6001*, GC23-2203.
- *AIX Communications Programming Concepts*, SC23-2206.
- *IBM AIX Operating System Technical Reference, Volume 1*, SC23-2300.
- *IBM AIX Operating System Technical Reference, Volume 2*, SC23-2301.

AS/400 Publications

- *IBM AS/400 CL Reference Manual Volume 1*, SC21-9775.
- *IBM AS/400 CL Reference Manual Volume 2*, SC21-9776.
- *IBM AS/400 CL Reference Manual Volume 3*, SC21-9777.
- *IBM AS/400 CL Reference Manual Volume 4*, SC21-9778.

- *IBM AS/400 CL Reference Manual Volume 5*, SC21-9779.
- *IBM AS/400 Communications: APPN Network User's Guide*, SC21-8188.
- *IBM AS/400 Communications: Programmer's Guide*, SC21-9590.
- *IBM AS/400 Communications: User's Guide*, SC21-9601.
- *IBM AS/400 Device Configuration Guide*, SC21-8106.
- *IBM AS/400 Programming: Command Reference Summary*, SC21-8076.
- *IBM AS/400 Programming: Data Management Guide*, SC21-9658.
- *IBM AS/400 System Operations: Database Coordinator' Guide*, SC21-8086.
- *IBM AS/400 System Operations: Operator's Guide*, SC21-8082.

DOS Publications

- *DOS Getting Started Version 5.00*, SA40-0637.
- *DOS 5.02 Technical Reference*, S16G-4559.
- *DOS/Windows Client Getting Started*, SC09-3001.
- *PC DOS 6.1 Command Reference*, S71G-3634.

MVS Publications

For a complete description of the library for MVS/ESA Version 5, see *OS/390 Information Roadmap*, GC28-1727-02. See also “JES Publications” on page 122.

OS/2 Publications

- *IBM OS/2 Warp Server Up and Running!*, S25H-8004
- *IBM Official Guide to Using OS/2 Warp*, ISBN 1-56884-466-2 (Karla Stagray and Linda S. Rogers; Foster City, CA: An IBM Press Book published by IDG Books Worldwide, Inc., 1995)
- *IBM OS/2 Warp Internet Connection: Your Key to Cruising the Internet and the World Wide Web*, ISBN 1-56884-465-4 (Deborah Morrison; Foster City, CA: An IBM Press Book published by IDG Books Worldwide, Inc., 1995)

OS/390 Publications

- *OS/390 Information Roadmap*, GC28-1727-02

  This book describes the documentation for the specific elements included in OS/390.
- *OS/390 Planning for Installation Release 3*, GC28-1726-02

  This book is intended to help you plan for the installation of OS/390. It describes migration, installation, hardware and software requirements, and coexistence considerations.
- *OS/390 OpenEdition Introduction*, GC28-1889-01.
- *OS/390 OpenEdition Planning*, SC28-1890-02.
- *OS/390 OpenEdition User's Guide*, SC28-1891-02.
- *OS/390 OpenEdition Command Reference*, SC28-1892-02.
- *OS/390 OpenEdition Messages and Codes*, SC28-1908-02.
- *OS/390 Language Environment Programming Guide*, SC28-1939-02.
- *OS/390 Language Environment Programming Reference*, SC28-1940-02.
- *OS/390 OpenEdition Programming: Assembler Callable Services Reference*, SC28-1899-02.
- *OS/390 Open Systems Adapter Support Facility Users's Guide*, SC28-1855.
- *Planning for the System/390 Open Systems Adapter Feature*, GC23-3870.

VM Publications

- *VM/ESA CMS Command Reference Summary*, SX24-5249.
- *VM/ESA CP Planning and Administration for 370*, SC24-5430.
- *VM/ESA CP Programming Services for 370*, SC24-5435.
- *VM/ESA Group Control System Reference for 370*, SC24-5426.
- *VM/ESA: Library Guide and Master Index*, GC23-0367.
- *VM/ESA: Master Index for 370*, GC24-5436.
- *VM/ESA Service Introduction and Reference*, SC24-5444.
- *VM/SP CMS Command Reference*, ST00-1981.
- *VM/SP Group Control System Macro Reference*, SC24-5250.
- *VM/SP Installation Guide*, SC24-5237.
- *VM/SP High Performance Option:*

Library Guide and Master Index, GC23-0187.

- *VM/SP System Facilities for Programming*, SC24-5288.
- *VM/XA CP Programming Services*, SC23-0370.
- *VM/XA Diagnosis Reference*, LY27-8054.
- *VM/XA Installation and Service*, SC23-0364.
- *VM/XA SP Group Control System Command and Macro Reference*, SC23-0433.

IBM Software Publications

The following sections describe the books associated with IBM software products.

ACF/VTAM Publications

The following list shows books in the VTAM Version 4 Release 4 library.

- *VTAM Installation and Migration Guide*, GC31-8367-00.
- *VTAM Release Guide*, GC31-6545-00.
- *VTAM Network Implementation Guide*, SC31-8370-00.
- *VTAM Resource Definition Reference*, SC31-8377-00.
- *VTAM Resource Definition Samples*, SC31-8378-00.
- *VTAM Customization*, LY43-0075-00.
- *VTAM Operation*, SC31-8372-00.
- *VTAM Messages*, GC31-8368-00.
- *VTAM Codes*, GC31-8369-00.
- *VTAM Programming*, SC31-8373-00.
- *VTAM Guide to Programming for LU 6.2*, SC31-8374-00.
- *VTAM Programming Reference for LU 6.2*, SC31-8375-00.
- *VTAM Programming for CSM*, SC31-8420-00.
- *VTAM CMIP Services and Topology Agent Programming Guide*, SC31-8365-00.
- *VTAM Diagnosis*, LY43-0078-00.
- *VTAM Data Areas for MVS/ESA Volume 1*, LY43-0076-00.
- *VTAM Data Areas for MVS/ESA Volume 2*, LY40-0077-00.
- *APPC Application Suite User's Guide*, SC31-6532-00.
- *APPC Application Suite Administration*, SC31-6533-00.
- *APPC Application Suite Programming*, SC31-6534-00.
- *VTAM AnyNet Guide to Sockets over SNA*, SC31-8371-00.
- *VTAM AnyNet Guide to SNA over TCP/IP*, SC31-8376-00.
- *VTAM Glossary*, GC31-8366-00.
- *Planning for NetView, NCP, and VTAM*, SC31-8063-00.
- *Planning for Integrated Networks*, SC31-8062-00.
- *VTAM Licensed Program Specifications*, GC31-8379-00.
- *VTAM Operation Quick Reference*, SX75-0208-00.

DATABASE 2 Publications

The following lists show books in the DATABASE 2 library.

DATABASE 2 Version 2

- *IBM DATABASE 2 Version 2: Administration Guide*, SC26-4374.
- *IBM DATABASE 2 Version 2: Application Programming and SQL Guide*, SC26-4377.
- *IBM DATABASE 2 Version 2: Messages and Codes*, SC26-4379.
- *IBM DATABASE 2 Version 2: Reference Summary*, SX26-3771.
- *IBM DATABASE 2 Version 2: SQL Reference*, SC26-4380.

DATABASE 2 Version 3

- *IBM DATABASE 2 Version 3: DB2 Administration Guide*, SC26-4888.
- *IBM DATABASE 2 Version 3: DB2 Application Programming and SQL Guide*, SC26-4889.
- *IBM DATABASE 2 Version 3: DB2 Messages and Codes*, SC26-4892.
- *IBM DATABASE 2 Version 3: DB2 Reference Summary*, SX26-3801.
- *IBM DATABASE 2 Version 3: DB2 SQL Reference*, SC26-4890.

ISPF Publication

ISPF Dialog Management Guide and Reference, SC34-4266.

JES Publications

- *MVS/ESA Library Guide with JES2*, GC28-1423.
- *MVS/ESA Library Guide with JES3*, SC28-1424

MVS/DFP Publications

- *MVS/DFP Version 3 Release 3: Customizing and Operating the Network File System Server*, SC26-4832.
- *MVS/DFP Version 3 Release 3: Macro Instructions for Data Sets*, S26-4747.
- *MVS/DFP Version 3 Release 3: Using Data Sets*, SC26-4749.
- *MVS/DFP Version 3 Release 3: Using the Network File System Server*, SC26-4732.

Network Control Program (NCP) Publications

- *ACF/NCP V7R1 IP Router Planning and Installation Guide*, GG24-3974.
- *NCP and EP Reference*, LY43-0029.
- *NCP, SSP, and EP Generation and Loading Guide*, SC31-6221.
- *NCP, SSP, and EP Resource Definition Guide*, SC31-6223.
- *NCP, SSP, and EP Resource Definition Reference*, SC31-6224.

TME 10 NetView for OS/390 Publications

For a complete description of the TME 10 NetView for OS/390 library, see the *TME 10 NetView for OS/390 Library Reference*, SC31-8249.

Networking Systems Cross-Product Library

The following list shows books in the Networking Systems cross-product library.

- *Planning Aids: Pre-Installation Planning Checklist for NetView, NCP, and VTAM*, SX75-0092.
- *Planning for Integrated Networks*, SC31-8062.
- *Planning for NetView, NCP, and VTAM*, SC31-8063.

OpenEdition MVS Publications

The following list shows selected books in the OpenEdition MVS library.

- *OS/390 OpenEdition Introduction*, GC28-1889-01
- *OS/390 OpenEdition Planning*, SC28-1890-02

Programming Publications

The following list shows books about various programming applications.

- *IBM C/370 Diagnosis Guide and Reference* LY09-1804 (feature 8082).
- *IBM C/370 General Information Manual* GC09-1386.
- *IBM C/370 Installation and Customization Guide Version 2 Release 1.0*, GC09-1387.
- *IBM C/370 Programming Guide*, SC09-1384.
- *IBM C/370 Reference Summary*, SX09-1211.
- *IBM C/370 User's Guide*, SC09-1264.
- *OS/390 C/C++ Run-Time Library Reference*, SC28-1663-01.
- *IBM TSO Extensions CLISTs*, SC28-1876.
- *IBM TSO Extensions Command Language Reference* GX23-0015.
- *IBM TSO Extensions Interactive Data Transmission Facility: User's Guide*, SC28-1104.
- *IMS/ESA V3R1 Application Programming: DL/I Calls* SC26-4274.
- *HiPPI User's Guide and Programmer's Reference*, SA23-0369.
- *Parallel I/O Access Methods Programmer's Guide*, SC26-4648.
- *VS Pascal Application Programming Guide* SC26-4319.
- *VS Pascal Diagnosis Guide and Reference* LY27-9525.
- *VS Pascal General Information*, GT00-2664.
- *VS Pascal Installation and Customization for MVS* SC26-4321.
- *VS Pascal Installation and Customization for VM* SC26-4342.
- *VS Pascal Language Reference*, SC26-4320.

RACF Publications

The following list shows books in the RACF library.

- *IBM Resource Access Control Facility (RACF): General Information Manual*, GT00-2820.
- *IBM Resource Access Control Facility (RACF): User's Guide*, SC28-1341.
- *External Security Interface (RACROUTE) Macro Reference*, GC28-1366.
- *RACF Publications Order Guide*, GX22-0012.
- *Resource Access Control Facility (RACF) Security Administrator's Guide*, SC28-1340.
- *System Programming Library: RACF*, SC28-1343.

SMP/E Publications

The following list shows books in the SMP/E Release 8 library.

- *SMP/E Diagnosis Guide*, SC23-3130.
- *SMP/E Messages and Codes*, SC28-1107.
- *SMP/E Reference*, SC28-1107.
- *SMP/E Reference Summary*, SX22-0016.
- *SMP/E User's Guide*, SC28-1302.

VSAM Publication

MVS/370 VSAM Administration Guide, GC26-4066.

X.25 NPSI Publications

The following list shows books in the X.25 NPSI library.

- *X.25 Network Control Program Packet Switching Interface Diagnosis, Customization, and Tuning Version 3*, LY30-5610.
- *X.25 Network Control Program Packet Switching Interface Host Programming*, SC30-3502.
- *X.25 Network Control Program Packet Switching Interface Planning and Installation*, SC30-3470.

IBM Hardware Publications

The following sections describe the books associated with IBM hardware products.

System/370 and System/390 Publications

The following list shows the principles of operation manuals for the System/370 and System/390 processors.

- *IBM ESA/370 Principles of Operation*, SA22-7200.
- *IBM ESA/390 Principles of Operation*, SA22-7201.
- *IBM System/370 Extended Architecture Principles of Operation*, SA22-7085.
- *IBM System/370 Principles of Operation*, GA22-7001.
- *S/360, S/370, and S/390 I/O Interface Channel to Channel Control Unit OEMI*, GA22-6974.

3172 Interconnect Controller Publications

The following list shows books in the IBM 3172 Interconnect Controller library.

- *IBM Interconnect Controller Program User's Guide*, SC30-3525.
- *IBM 3172 Interconnect Controller Installation and Service Guide*, GA27-3861.
- *IBM 3172 Interconnect Controller Operator's Guide*, GA27-3860.
- *IBM 3172 Interconnect Controller Planning Guide*, GA27-3867.
- *IBM 3172 Interconnect Controller Status Codes*, GA27-3951.

3270 Information Display System Publication

3270 Information Display System: 3270 Data Stream Programmer's Reference, GA23-0059.

8232 LAN Channel Station Publications

The following list shows books in the IBM 8232 LAN Channel Station library.

- *IBM LAN Channel Support Program: Version 1.0 User's Guide*, SC30-3458.
- *IBM 8232 LAN Channel Station: Installation and Testing*, GA27-3796.
- *IBM 8232 LAN Channel Station: Operating Guide*, GA27-3785.

9370 Publications

The following list shows books in the 9370 library.

- *IBM 9370 Information System: Using the X.25 Communications Subsystem*, SA09-1742.
- *IBM 9370 Information System X.25 Communications Subsystem Description*, SA09-1743.
- *VM/ESA: Connectivity Planning, Administration, and Operation Release 1*, SC24-5448.

Other TCP/IP-Related Publications

The following sections describe other books associated with TCP/IP.

- *The Art of Distributed Application: Programming Techniques for Remote Procedure Calls* John R. Corbin, Springer-Verlog, 1991.
- *CAE Specification: X/Open Transport Interface (XTI),* X/Open Company Ltd., U. K., 1992, SC31-8005.
- *IEEE Network Magazine*, July 1990.
- *TCP/IP Illustrated Volume I: The Protocols*, W. Richard Stevens, Addison-Wesley Publishing Company, Inc., 1994, SR28-5586.
- *TCP/IP Illustrated Volume II: The Implementation*, Gary R. Wright and Richard Stevens, Addison-Wesley Publishing Company, Inc., 1995, SR28-5630.
- *TCP/IP Illustrated Volume III*, W. Richard Stevens, Addison-Wesley Publishing Company, Inc., 1996, SR23-7289
- *Interoperability Report*, Volume 3, No. 3, March 1989.
- "MIB II Extends SNMP Interoperability," C. Vanderberg, *Data Communications*, October 1990.
- *"Network Management and the Design of SNMP,"* J.D. Case, J.R. Davin, M.S. Fedor, M.L. Schoffstall.
- *"Network Management of TCP/IP Networks: Present and Future,"* A. Ben-Artzi, A. Chandna, V. Warrier.
- *The Simple Book: An Introduction to Management of TCP/IP-based Internets*, Marshall T Rose, Prentice Hall, Englewood Cliffs, New Jersey,1993.
- "Special Issue: Network Management and Network Security," *ConneXions-The Interoperability Report* Volume 4, No. 8, August 1990.
- *UNIX Programmer's Reference Manual* (4.3 Berkeley Software Distribution, Virtual VAX-11 Version). Department of Electrical Engineering and Computer Science. University of California, Berkeley, 1988.

OSF/Motif Publications

The following list shows OSF/Motif books.

- *OSF/Motif Application Environment Specifications* (AES), Open Software Foundation, Prentice Hall, Inc., 1990, ISBN 0-13-640483-9.
- *OSF/Motif Programmer's Guide* Open Software Foundation, Prentice Hall, Inc., 1990, ISBN 0-13-640509-6.
- *OSF/Motif Programmer's Reference* Open Software Foundation, Prentice Hall, Inc., 1990, ISBN 0-13-640517-7.
- *OSF/Motif Style Guide* Open Software Foundation, Prentice Hall, Inc., 1990, ISBN 0-13-640491-X.
- *OSF/Motif User's Guide* Open Software Foundation, Prentice Hall, Inc., 1990, ISBN 0-13-640525-8.

Sun (RPC) Publications

The following list shows Sun Microsystems books.

- *Networking on the Sun Workstation: Remote Procedure Call Programming Guide* (800-1324-03), Sun Microsystems, Inc.
- *Network Programming* (800-1779-10), Sun Microsystems, Inc.

X Window System Publications

The following list shows X Window System books.

- *Introduction to the X Window System*, Oliver Jones, Prentice-Hall, 1988, ISBN 0-13-499997-5.
- *PEXlib Specification and C Language Binding* Jeff Stevenson, Hewlett-Packard Company, 1992, SR28-5116.
- *The X Window System Series* (6 volumes), O'Reilly & Associates, 1988, 1989, 1990, ISBN 0-937175-40-4, 0-937175-27-7, 0-937175-28-5, 0-937175-35-6, 0-937175-33-1, 0-937175-35-8.
- *X Protocol Reference Manual* Adrian Nye, ed. O'Reilly & Associates, Inc., 1990, ISBN 0-937175-50-1.
- *X Window System: C Library and Protocol Reference* Robert Scheifler, James Gettys, and Ron Newman, DEC Press, 1988, ISBN 1-55558-012-2.
- *X Window System: Programming and Applications with Xt*, Douglas A. Young, Prentice-Hall, 1989, ISBN 0-13-972167-3.

- *X Window System: Programming and Applications with Xt, OSF/Motif Edition* Douglas A. Young, Prentice-Hall, 1990, ISBN 0-13-497074-8.
- *X Window System Technical Reference*, Steven Mikes, Addison-Wesley, 1990, ISBN 0-201-52370-1.
- *X Window System User's Guide* Valerie Quercia and Tim O'Reilly, O'Reilly & Associates, Inc., 1990, ISBN 0-937175-14-5.

Network Architecture Publications

The following sections list books associated with network architecture.

Open Systems Interconnection (OSI) Publication

Open Systems Interconnection, Z320-9757.

Systems Network Architecture (SNA) Publications

The following list shows books in the SNA library.

- *Systems Network Architecture: Sessions between Logical Units*, GC20-1868.
- *Systems Network Architecture Format and Protocol Reference Manual: Architecture Logic*, SC30-3112.
- *Systems Network Architecture Format and Protocol Reference Manual: Management Services*, SC30-3346.
- *Systems Network Architecture Formats* GA27-3136.
- *Systems Network Architecture Network Product Formats*, LY43-0081.

Index

© Copyright IBM Corp. 1989, 1997

T

U

V

W

Communicating Your Comments to IBM

OS/390 TCP/IP Open Edition
Programmer's Reference

Publication No. SC31-8308-00

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

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

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:

  United States and Canada: **1-800-227-5088**
- If you prefer to send comments electronically, use this network ID:
  - IBM Mail Exchange: **USIB2HPD at IBMMAIL**
  - IBMLink: **CIBMORCF at RALVM13**
  - Internet: **USIB2HPD@VNET.IBM.COM**

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies.

Help us help you!

OS/390 TCP/IP Open Edition
Programmer's Reference

Publication No. SC31-8308-00

If your concern is service related, you can reach Service at 1-800-992-4777 in the United States. Outside the United States, please check your phone listing for the IBM Service Center nearest you.

We hope you find this publication useful, readable and technically accurate, but only you can tell us! Please take a few minutes to let us know what you think by completing this form.

Overall, how satisfied are you with the information in this book?	Satisfied	Dissatisfied
	☐	☐

How satisfied are you that the information in this book is:	Satisfied	Dissatisfied
Accurate	☐	☐
Complete	☐	☐
Easy to find	☐	☐
Easy to understand	☐	☐
Well organized	☐	☐
Applicable to your task	☐	☐

Specific Comments or Problems:

Please tell us how we can improve this book:

Thank you for your response. When you send information to IBM, you grant IBM the right to use or distribute the information without incurring any obligation to you. You of course retain the right to use the information in any way you choose.

Your Internet Address: _____

Name

Address

Company or Organization

Phone No.

Help us help you!
SC31-8308-00

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

Information Development
Department CGMD
International Business Machines Corporation
PO BOX 12195
RESEARCH TRIANGLE PARK NC 27709-9990

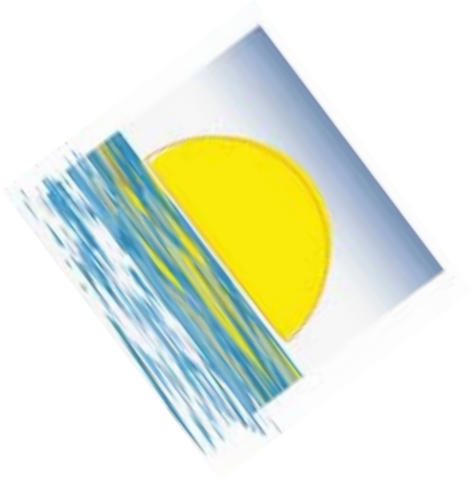
Fold and Tape **Please do not staple** Fold and Tape

SC31-8308-00

Cut or Fold
Along Line

File Number: S390-50
Program Number: 5645-001

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.



SC31-8308-00

Spine information:

OS/390 TCP/IP Open Edition Programmer's Reference