System i

Programming
Simple Network Management Protocol APIs

Version 6 Release 1

System i

Programming
Simple Network Management Protocol APIs

Version 6 Release 1

Note

Before using this information and the product it supports, read the information in "Notices," on page 69.

This edition applies to version 6, release 1, modification 0 of IBM i5/OS (product number 5761-SS1) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

Contents

Simple Network Management Protocol APIs

The Simple Network Management Protocol (SNMP) APIs include the:

- “Simple Network Management Protocol (SNMP) Subagent APIs”
- “Simple Network Management Protocol (SNMP) Manager APIs” on page 35

Before using the SNMP APIs, read the Simple Network Management Protocol (SNMP) Support manual. It describes how to configure a System i™ product to use SNMP and discusses SNMP agents, subagents, managers, and management information base (MIBs). The manual also discusses “Using the SNMP Subagent DPI API.”

You can get more information about SNMP and Distributed Protocol Interface (DPI®) from Requests for Comment (RFC) on the Internet. A file called ways_to_get_rfcs has details about obtaining RFCs. To receive these details send an E-MAIL message as follows:

To: rfc-info@ISI.EDU
Subject: gettingrfcs
help: ways_to_get_rfcs

DPI is described in RFC 1592, ″Simple Network Management Protocol Distributed Protocol Interface,″ Version 2.0.

UNIX-Type APIs | APIs by category

APIs

These are the APIs for this category.

Simple Network Management Protocol (SNMP) Subagent APIs

The SNMP subagent APIs are:

- “connectSNMP()—Establish Connection with SNMP Agent” on page 4 (Establish connection with SNMP agent) establishes a logical connection between the SNMP subagent and the local (the same system) SNMP agent.
- “debugDPI()—Set DPI Packet Trace” on page 6 (Set DPI packet trace) sets the level of the Distributed Protocol Interface (DPI) packet trace.
- “disconnectSNMP()—End Connection with SNMP Agent” on page 8 (End connection with SNMP agent) ends the logical connection between the SNMP subagent and the SNMP agent.
- “DPI_PACKET_LEN()—Get Length of DPI Packet” on page 10 (Get length of DPI packet) returns the length (number of bytes) of a Distributed Protocol Interface (DPI) packet.
- “fDPIparse()—Free Storage from DPI Packet Parse” on page 11 (Free storage from DPI packet parse) frees storage that was previously allocated by a call to pDPIpacket() to store the DPI packet.
- “fDPIset()—Free Storage from DPI Set Packet” on page 11 (Free storage from DPI set packet) frees storage that was previously allocated for snmp_dpi_set_packet structures.
- “mkDPIAreYouThere()—Make a DPI AreYouThere Packet” on page 13 (Make a DPI AreYouThere packet) makes a DPI AreYouThere packet and returns a pointer to the packet.
- “mkDPIclose()—Make a DPI Close Packet” on page 14 (Make a DPI close packet) makes a DPI close packet and returns a pointer to the packet.

- “mkDPIopen()—Make a DPI Open Packet” on page 15 (Make a DPI open packet) makes a Distributed Protocol Interface (DPI) open packet and returns a pointer to the packet.
- “mkDPIregister()—Make a DPI Register Packet” on page 18 (Make a DPI register packet) makes a Distributed Protocol Interface (DPI) register packet and returns a pointer to the packet.
- “mkDPIresponse()—Make a DPI Response Packet” on page 20 (Make a DPI response packet) makes a DPI response packet and returns a pointer to the packet.
- “mkDPIset()—Make a DPI Set Packet” on page 22 (Make a DPI set packet) makes a DPI set structure and adds it to a chained list of set structures if previous calls have been made.
- “mkDPItrap()—Make a DPI Trap Packet” on page 24 (Make a DPI trap packet) makes a DPI trap packet and returns a pointer to the packet.
- “mkDPIunregister()—Make a DPI Unregister Packet” on page 26 (Make a DPI unregister packet) makes a DPI unregister packet and returns a pointer to the packet.
- “pDPIpacket()—Parse a DPI Packet” on page 27 (Parse a DPI packet) parses a serialized Distributed Protocol Interface (DPI) packet to make it available for processing by the subagent.
- “receiveDPIpacket()—Receive a DPI Packet from the SNMP Agent” on page 28 (Receive a DPI packet from the SNMP agent) obtains a copy of a DPI packet sent by the SNMP agent to the subagent, and returns the DPI packet to the caller.
- “sendDPIpacket()—Send a DPI Packet to the SNMP Agent” on page 30 (Send a DPI packet to the SNMP agent) sends a copy of a Distributed Protocol Interface (DPI) packet to the SNMP agent (on the same system as the subagent).
- “waitDPIpacket()—Wait for a DPI Packet” on page 32 (Wait for a DPI packet) waits for a message on the data queue with which the subagent has previously connected (see connectSNMP()—Establish Connection with SNMP Agent).

Note: These functions use header (include) files from the library QSYSINC, which is optionally installable. Make sure QSYSINC is installed on your system before using any of the functions. All of the SNMP subagent APIs use header file **qtossapi.h**. You can see this source in source file H, member name QTOSSAPI, in the QSYSINC library.

The Simple Network Management Protocol (SNMP) subagent APIs can be used to dynamically extend the management information base (MIB) that the system SNMP agent is aware of. The MIB is extended, without any change to the SNMP agent itself, while the system is running. Dynamically added MIB subtrees (as defined and supported by a program known as a subagent) provide this capability. You may now extend the remote and automated system management capabilities of the system within the SNMP framework. So, for example, you could define an SNMP MIB group for your RPG and SQL application, and then use SNMP protocol data units (PDUs), such as get and set, to determine status information or to make changes in control variables.

The *Distributed Protocol Interface (DPI®) packet* is used throughout this information. The DPI is an extension to SNMP agents that permit users to dynamically add, delete, or replace management variables in the local MIB without requiring recompilation of the SNMP agent.

The diagram below shows typical DPI API call sequences that are used to accomplish the SNMP subagent functions that are listed.

(A) Subagent initiation

(B) Subagent registration (loop for multiple subtrees)

(C) Normal processing loop for a subagent, starting with a wait for a (get, getnext, set...) packet from the SNMP agent (`other` may be an unregister or close packet)

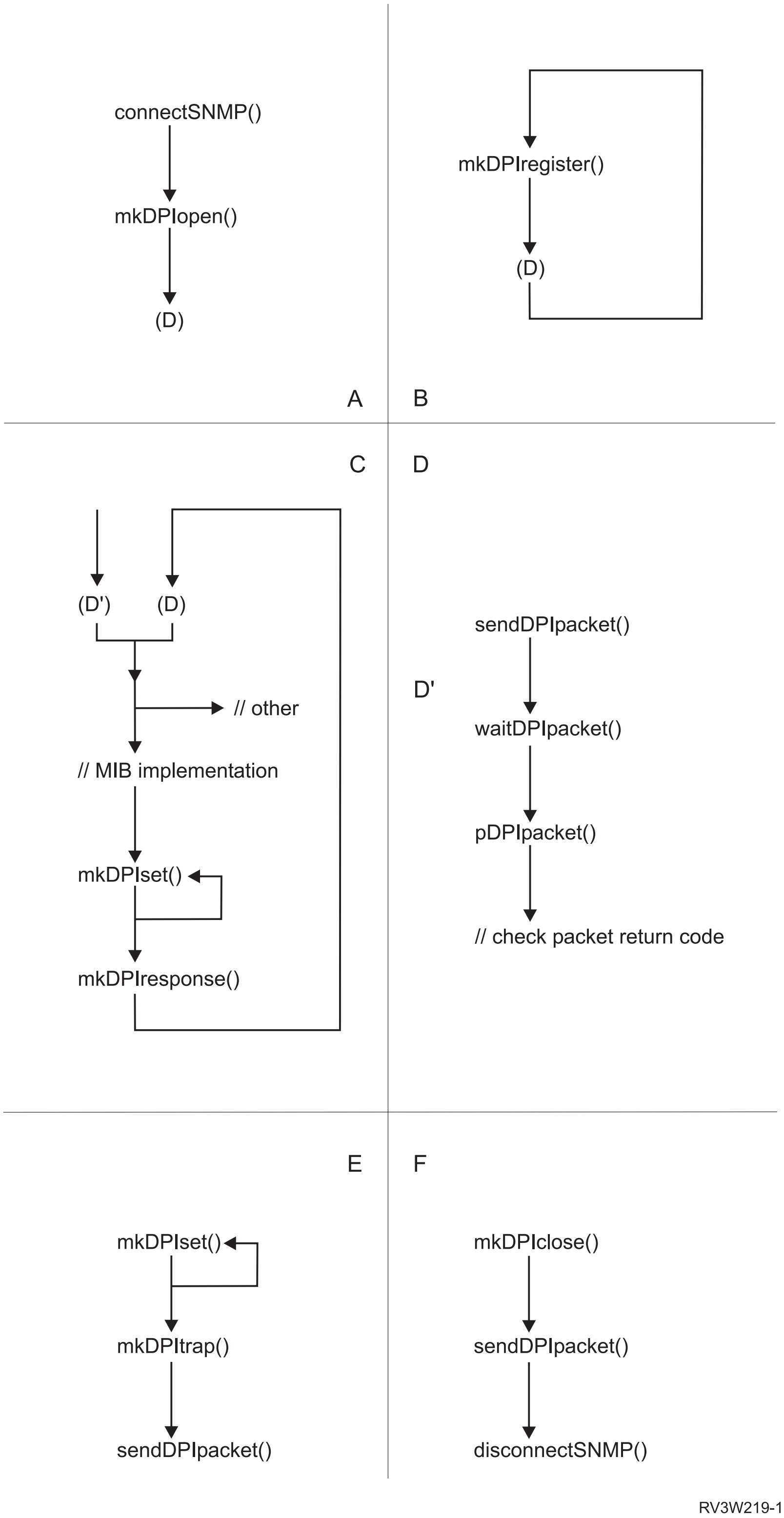
(D) A common call sequence that might be consolidated

(E) Subagent initiated trap

(F) Subagent termination

A loop around **mkDPIset()** represents building a packet with multiple varbinds.

DPI API Call Sequences—Example



Top | UNIX-Type APIs | APIs by category

connectSNMP()—Establish Connection with SNMP Agent

Syntax

```
#include <qtossapi.h>

int             connectSNMP(
          char        *queue_name,
          char        *lib_name,
          long int    timeout );
```

Service Program Name: QTOSSAPI
Default Public Authority: *USE
Threadsafe: No

The **connectSNMP()** function establishes a logical connection between the SNMP subagent and the local (the same system) SNMP agent. The data queue named by the *queue_name* parameter is used by the SNMP agent as the target data queue in a call to the Send Data Queue (QSNDDTAQ) API when it sends a message to the subagent. Only a single connection is allowed per data queue and library, hence a subagent may have only a single data queue. (Of course, a subagent may have multiple registrations. See "mkDPIregister()—Make a DPI Register Packet" on page 18.)

Authorities

So that the subagent can receive messages from the SNMP agent, the following conditions must be met:

- The library and data queue whose names are passed as a parameter in the **connectSNMP()** call must exist prior to the call.
- The SNMP agent job must have write access to the data queue. If you suspect a problem with the data queue, check the job log of the SNMP agent job (QTMSNMP in subsystem QSYSWRK) for TCP4041 messages with reason code 001.

Parameters

queue_name

(Input) The name of the data queue (as a null-terminated string) on which the subagent wants to receive Distributed Protocol Interface (DPI®) packets. The value must conform to i5/OS® rules for data queue names (such as using uppercase letters and starting with a letter, $, #, @, and so forth).

lib_name

(Input) The name of the i5/OS library (as a null-terminated string) to which the data queue belongs. QTEMP is not an allowed value. The value must conform to i5/OS rules for library names (such as using uppercase letters and starting with a letter, $, #, @, and so forth).

Note that the actual character representation of the specific library name must be used. Special values such as *LIBL and *CURLIB are not allowed.

timeout

(Input) The amount of time in seconds that the subagent is willing to wait for a connection. This field may contain the following values:

0	Unlimited wait
> 0	The number of seconds to wait (maximum is 2 147 483 647)

Any other values result in an error return code.

Return Value

The return values are defined in the **<qtossapi.h>** file in the QSYSINC library.

0 *snmpsa_RC_ok*

The call was successful.

-1 *snmpsa_RC_err*

An exception occurred. Check the subagent job log for the exception information, correct the condition, and resubmit the subagent job. (This return code is only used when a more specific return code is not available.)

-2 *snmpsa_RC_noagent*

The SNMP agent is not available.

-3 *snmpsa_RC_mismatch*

There is a code-level mismatch between the agent and the subagent. If this occurs, report the problem to the appropriate service organization.

-4 *snmpsa_RC_timedout*

The specified timeout value was exceeded.

-7 *snmpsa_RC_parmerr*

A parameter error occurred. This is more likely caused by errors in the value of a parameter (for example, a value was too large or too small) or by a pointer parameter that has a NULL value and should not. For char* parameters, it may also be caused if the length of the string exceeds some limit.

-8 *snmpsa_RC_lengtherr*

During an attempt to communicate with the agent, a length exception occurred.

-9 *snmpsa_RC_buffer*

An internal buffer was not obtained. See any messages in the job log and correct any errors that are indicated, then retry the operation.

-10 *snmpsa_RC_duplicate*

The agent already has a subagent with this queue and library name. The subagent may continue as usual with the **mkDPIopen()** and **mkDPIregister()** functions. If these fail, the subagent should use different library and queue names.

-13 *snmpsa_RC_alreadyconnected*

The subagent is already connected using the same data queue and library names passed on the call. If the SNMP agent still does not forward requests to the subagent properly, use the **disconnectSNMP()** function, then the **connectSNMP()** function.

For more information, see the Simple Network Management Protocol (SNMP) Support manual.

Usage Notes

The **connectSNMP()** function establishes a logical connection with the SNMP agent that is running on the same system as the subagent. This is normally the first subagent API that a subagent calls.

This API, like all the subagent APIs, checks to ensure that the pointers passed are generally valid for user data, for example, user domain. Such audits occur for all pointer parameters and for all pointers that appear in all C structures that are passed as parameters. If one of these checks fail, a CPF9872 exception is generated. This can occur from all the subagent APIs except **debugDPI()**, **DPI_PACKET_LEN()**, and **mkDPIAreYouThere()**.

Related Information

- The <**qtossapi.h**> file (see "Header Files for UNIX-Type Functions" on page 57)
- "disconnectSNMP()—End Connection with SNMP Agent" on page 8—End Connection with SNMP Agent
- "mkDPIregister()—Make a DPI Register Packet" on page 18—Make a DPI Register Packet

Example

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 68.

```
#include <qtossapi.h>
int      rc;

rc = connectSNMP( "QABCDEFG", "LIBABC", 0 );
if (rc) {
   /* Handle exception. */
}
```

API introduced: V3R6

Top | UNIX-Type APIs | APIs by category

debugDPI()—Set DPI Packet Trace

Syntax

```
#include <qtossapi.h>

void   debugDPI( int level );
```

Service Program Name: QTOSSAPI
Default Public Authority: *USE
Threadsafe: No

The **debugDPI()** function sets the level of the Distributed Protocol Interface (DPI®) packet trace. The trace consists of a representation (printed to STDERR) of DPI packets as they are parsed (by the **pDPIpacket()** function) or made (by one of the mkDPI*xxx*() APIs). The trace is written to ILE C standard error output.

Authorities and Locks

None.

Parameters

level (Input) The level of tracing to perform. 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. Possible values follow:

0	Turn off packet tracing
1	Display packet creation and parsing
2	Level 1, plus display the hexadecimal dump of incoming and outgoing DPI packets.

Usage Notes

The **debugDPI()** function is used to turn the DPI packet trace on or off.

Related Information

- The <**qtossapi.h**> file (see “Header Files for UNIX-Type Functions” on page 57)
- “mkDPIregister()—Make a DPI Register Packet” on page 18—Make a DPI Register Packet
- “mkDPIresponse()—Make a DPI Response Packet” on page 20—Make a DPI Response Packet
- “pDPIpacket()—Parse a DPI Packet” on page 27—Parse a DPI Packet

Example

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 68.

```
#include <qtossapi.h>

debugDPI(2);
```

Following are some examples of the DPI packet trace. A simple way to view STDERR is to run your program in batch in a submitted job.

The following is an example of a trace, with the *level* parameter set to 1, of a register packet made by the subagent’s call to **mkDPIregister()**. This is indicated in the trace by the letter c (for create) at the beginning of a trace line. Immediately following that is the parse of the response packet that the subagent got back from the SNMP agent. This is indicated in the trace by the letter p (for parse) at the beginning of a trace line.

```
cDPIpacket: Major=2, Version=2, Release=0, Id=1, Type=SNMP_DPI_REGISTER
cDPIreg: subtree=1.3.6.1.2.3.4.5.6., priority=0, timeout=4
         view_selection=No
         bulk_selection=No
pDPIpacket: Major=2, Version=2, Release=0, Id=1, Type=SNMP_DPI_RESPONSE
pDPIresp: ret_code=0 [0x0] (noError), ret_index=255
pDPIset: subtree=1.3.6.1.2.3.4.5.6, instance=** NONE **
         object=1.3.6.1.2.3.4.5.6
         value_type=NULL ['04'H], value_len=0
         value=** NULL **
```

Next is an example of a ″get″ packet that is received by a subagent. Immediately following that is the response packet that the subagent built (indicated by the letter c) by calling **mkDPIresponse()**.

```
pDPIpacket: Major=2, Version=2, Release=0, Id=2, Type=SNMP_DPI_GET
            Community=** NONE **
pDPIget: subtree=1.3.6.1.2.3.4.5.6., instance=1.0
         object=1.3.6.1.2.3.4.5.6.1.0
cDPIpacket: Major=2, Version=2, Release=0, Id=2, Type=SNMP_DPI_RESPONSE
cDPIresp: ret_code=0 [0x0] (noError), ret_index=0
cDPIset: subtree=1.3.6.1.2.3.4.5.6., instance=1.0
         object=1.3.6.1.2.3.4.5.6.1.0
         value_type=Integer32 ['81'H], value_len=4
         value=1 [0x00000001]
```

Next is an example of the trace with the *level* parameter set to 2. This causes a hexadecimal dump of the DPI packet to be generated when **pDPIpacket()** is called, in addition to the trace level of 1. Next is the same packet as parsed by **pDPIpacket()**, and immediately following that is the response packet that the subagent built by calling **mkDPIresponse()**.

```
Dump of 33 byte incoming DPI packet:
  00 1f 02 02 00 00 03 02 00 00 f1 4b f3 4b f6 4b
  f1 4b f2 4b f3 4b f4 4b f5 4b f6 4b 00 f5 4b f0
  00
pDPIpacket: Major=2, Version=2, Release=0, Id=3, Type=SNMP_DPI_GETNEXT
            Community=** NONE **
pDPInext: subtree=1.3.6.1.2.3.4.5.6., instance=5.0
          object=1.3.6.1.2.3.4.5.6.5.0
cDPIpacket: Major=2, Version=2, Release=0, Id=3, Type=SNMP_DPI_RESPONSE
cDPIresp: ret_code=0 [0x0] (noError), ret_index=0
cDPIset: subtree=1.3.6.1.2.3.4.5.6., instance=6.0
         object=1.3.6.1.2.3.4.5.6.6.0
         value_type=Counter32 ['86'H], value_len=4
         value=6 [0x00000006]
```

API introduced: V3R6

Top | UNIX-Type APIs | APIs by category

disconnectSNMP()—End Connection with SNMP Agent

Syntax

```
#include <qtossapi.h>

int           disconnectSNMP(
          char        *queue_name,
          char        *lib_name,
          long int    timeout );
```

Service Program Name: QTOSSAPI
Default Public Authority: *USE
Threadsafe: No

The **disconnectSNMP()** function ends the logical connection between the SNMP subagent and the i5/OS® SNMP agent.

Authorities

So that the subagent can receive messages from the SNMP agent, the following conditions must be met:

- The library and data queue whose names are passed as a parameter in the **connectSNMP()** call must exist prior to the call.
- The data queue and library names passed as parameters in the **disconnectSNMP()** call must be the same as used in the previous, successful **connectSNMP()** call.

Parameters

queue_name
(Input) The name of the data queue (as a null-terminated string) on which the subagent was receiving Distributed Protocol Interface (DPI®) packets. This should be the same data queue name as previously used in a call to **connectSNMP()**.

lib_name
(Input) The name of the i5/OS library (as a null-terminated string) to which the data queue belongs. This should be the same library name as previously used in a call to **connectSNMP()**.

timeout
(Input) The amount of time in seconds that the subagent is willing to wait for a disconnection. This field may contain any of these values:

0	Immediate disconnect, independent of whether or not the SNMP agent is available or has responded
> 0	The number of seconds to wait (maximum is 2 147 483 647)

Any other values result in an error return code.

Return Value

The indicated return values are defined in the **<qtossapi.h>** file.

0 *snmpsa_RC_ok*

The **disconnectSNMP()** function was successful.

-1 *snmpsa_RC_err*

An exception occurred. Check the subagent job log for the exception information, correct the condition, and resubmit the subagent job. (This return code is only used when a more specific return code is not available.)

-2 *snmpsa_RC_noagent*

The SNMP agent is not available.

-3 *snmpsa_RC_mismatch*

There is a code-level mismatch between the agent and the subagent. If this occurs, report the problem to the appropriate service organization.

-4 *snmpsa_RC_timedout*

The specified timeout value was exceeded.

-7 *snmpsa_RC_parmerr*

A parameter error occurred. This is more likely caused by errors in the value of a parameter (for example, a value was too large or too small) or by a pointer parameter that has a NULL value and should not. For char* parameters, it may also be caused if the length of the string exceeds some limit.

-8 *snmpsa_RC_lengtherr*

During an attempt to communicate with the agent, a length exception occurred. See any messages in the job log and correct any errors that are indicated, then retry the operation.

-9 *snmpsa_RC_buffer*

An internal buffer was not obtained. See any messages in the job log and correct any errors that are indicated, then retry the operation.

-14 *snmpsa_RC_sync*

A synchronization problem occurred between the agent and subagent. If this occurs, report the problem to the appropriate service organization.

For more information, see the Simple Network Management Protocol (SNMP) Support manual.

Usage Notes

The **disconnectSNMP()** function ends the logical connection between the SNMP agent and a subagent. This is normally the last subagent API that a subagent calls.

Related Information

- The <**qtossapi.h**> file (see “Header Files for UNIX-Type Functions” on page 57)
- “connectSNMP()—Establish Connection with SNMP Agent” on page 4—Establish Connection with SNMP Agent

Example

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 68.

```
#include <qtossapi.h>
int      rc;

rc = disconnectSNMP( "QABCDEFG", "LIBABC", 0 );
if (rc) {
   /* Handle exception. */
}
```

API introduced: V3R6

Top | UNIX-Type APIs | APIs by category

DPI_PACKET_LEN()—Get Length of DPI Packet

Syntax

```
#include <qtossapi.h>

int    DPI_PACKET_LEN( unsigned char *packet_p );
```

Service Program Name: QTOSSAPI
Default Public Authority: *USE
Threadsafe: No

The **DPI_PACKET_LEN()** macro returns the length (number of bytes) of a Distributed Protocol Interface (DPI®) packet.

Authorities and Locks

None.

Parameters

packet_p
(Input) A pointer to a (serialized) DPI packet.

Return Value

value An integer value that represents the total DPI packet length.

For more information, see the Simple Network Management Protocol (SNMP) Support manual.

Usage Notes

The **DPI_PACKET_LEN()** macro generates a C expression that returns an integer that represents the total length of a DPI packet. It uses the first 2 bytes (in network byte order) of the packet to calculate the length. The length returned includes these first 2 bytes.

Example

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 68.

```
#include <qtossapi.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 or subagent. */
 }
```

API introduced: V3R6

Top | UNIX-Type APIs | APIs by category

fDPIparse()—Free Storage from DPI Packet Parse

Syntax

```
#include <qtossapi.h>

void   fDPIparse( snmp_dpi_hdr *hdr_p );
```

Service Program Name: QTOSSAPI
Default Public Authority: *USE
Threadsafe: No

The **fDPIparse()** function frees storage that was previously allocated by a call to **pDPIpacket()** to store the DPI® packet.

Authorities and Locks

None.

Parameters

hdr_p (Input) A pointer to an snmp_dpi_hdr structure.

Usage Notes

The **fDPIparse()** function frees dynamic storage that was previously created by a call to **pDPIpacket()**. After calling **fDPIparse()**, no further references should be made to hdr_p, which pointed to the snmp_dpi_hdr structure.

A complete or partial DPI snmp_dpi_hdr structure is also implicitly freed by a call to a DPI function that serializes an snmp_dpi_hdr structure 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()**.

Related Information

- The <**qtossapi.h**> file (see “Header Files for UNIX-Type Functions” on page 57)
- “mkDPIresponse()—Make a DPI Response Packet” on page 20—Make a DPI Response Packet
- “pDPIpacket()—Parse a DPI Packet” on page 27—Parse a DPI Packet

Example

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 68.

```
#include <qtossapi.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);
```

API introduced: V3R6

Top | UNIX-Type APIs | APIs by category

fDPIset()—Free Storage from DPI Set Packet

Syntax

```
#include <qtossapi.h>

void   fDPIset( snmp_dpi_set_packet *packet_p );
```

Service Program Name: QTOSSAPI
Default Public Authority: *USE
Threadsafe: No

The **fDPIset()** function frees storage that was previously allocated for snmp_dpi_set_packet structures.

Authorities and Locks

None.

Parameters

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

Usage Notes

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. Similarly, if you pass the chain of snmp_dpi_set_packet structures to **mkDPItrap()** to make a DPI trap request, the storage will be freed.

Unnecessary free operations may result in an MCH6902 (type 2). If this occurs, remove the call to **fDPIset()**.

Related Information

- The <**qtossapi.h**> file (see “Header Files for UNIX-Type Functions” on page 57)
- “mkDPIresponse()—Make a DPI Response Packet” on page 20—Make a DPI Response Packet
- “mkDPIset()—Make a DPI Set Packet” on page 22—Make a DPI Set Packet

Example

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 68.

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

/* ...                                                   */

/* The subagent was waiting for work from the SNMP agent, and */
/* a message arrives...                                       */

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 successful, then */
   first_p = set_p;                  /* save pointer 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) {                      /*If successful, 2nd one */
      pack_p = mkDPIresponse(hdr_p,      /* makes response.   */
                    SNMP_ERROR_noError,  /* It will also free */
                    0L, first_p);        /* the set_p tree.   */
      /* Send DPI response to agent. */
   } else {                         /* If 2nd mkDPIset fails, */
      fDPIset(first_p);                /* it must free chain. */
   }
}
```

API introduced: V3R6

Top | UNIX-Type APIs | APIs by category

mkDPIAreYouThere()—Make a DPI AreYouThere Packet

Syntax

```
#include <qtossapi.h>

unsigned char     *mkDPIAreYouThere( void );
```

Service Program Name: QTOSSAPI
Default Public Authority: *USE
Threadsafe: No

The **mkDPIAreYouThere()** function makes a DPI® AreYouThere packet and returns a pointer to the packet.

Authorities and Locks

None.

Parameters

None.

Return Value

value The value returned is a pointer to the DPI packet.

If successful, then 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 **DPI_PACKET_LEN()** function can be used to calculate the total length of the DPI packet.

NULL If unsuccessful, then a NULL pointer is returned.

Be aware that the static buffer for the DPI packet is shared by other mkDPI*xxxx*() functions that create a serialized DPI packet.

For more information, see the Simple Network Management Protocol (SNMP) Support manual.

Usage Notes

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

If your connection to the agent is still intact, 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 intact, the agent may send a response with an error indication, or may not send a response at all.

Related Information

- The <**qtossapi.h**> file (see "Header Files for UNIX-Type Functions" on page 57)
- "DPI_PACKET_LEN()—Get Length of DPI Packet" on page 10—Get Length of DPI Packet

Example

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 68.

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

 pack_p = mkDPIAreYouThere();
 if (pack_p) {
    /* Send the packet to the agent. */
 }

 /* Wait for response with waitDPIpacket().               */
 /* Normally the response should come back fairly quickly,  */
 /* but it depends on the load of the agent.               */
```

API introduced: V3R6

Top | UNIX-Type APIs | APIs by category

mkDPIclose()—Make a DPI Close Packet

Syntax

```
#include <qtossapi.h>

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

Service Program Name: QTOSSAPI
Default Public Authority: *USE
Threadsafe: No

The **mkDPIclose()** function makes a DPI® close packet and returns a pointer to the packet.

Authorities and Locks

None.

Parameters

reason_code
(Input) The reason for the close. See the **<qtossapi.h>** file in the QSYSINC library for the list of defined reason codes.

Return Value

value The value returned is a pointer to the DPI packet.

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

NULL If unsuccessful, then a NULL pointer is returned.

Be aware that the static buffer for the DPI packet is shared by other mkDPI*xxxx*() functions that create a serialized DPI packet.

For more information, see the Simple Network Management Protocol (SNMP) Support manual.

Usage Notes

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

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

Related Information

- The <**qtossapi.h**> file (see "Header Files for UNIX-Type Functions" on page 57)
- "DPI_PACKET_LEN()—Get Length of DPI Packet" on page 10—Get Length of DPI Packet

Example

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 68.

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

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

API introduced: V3R6

Top | UNIX-Type APIs | APIs by category

mkDPIopen()—Make a DPI Open Packet

Syntax

```
#include <qtossapi.h>

unsigned char     *mkDPIopen(
     char             *oid_p,
     char             *description_p,
     unsigned long    timeout,
     unsigned long    max_varbinds,
     char             character_set,
     unsigned long    password_len,
     unsigned char    *password_p );
```

Service Program Name: QTOSSAPI
Default Public Authority: *USE
Threadsafe: No

The **mkDPIopen()** function makes a Distributed Protocol Interface (DPI®) open packet and returns a pointer to the packet.

Authorities and Locks

None.

Parameters

oid_p
(Input) A pointer to a NULL-terminated character string that represents the OBJECT IDENTIFIER, which uniquely identifies the subagent.

description_p
(Input) A pointer to a NULL-terminated character string, which is a descriptive name for the subagent. This can be any DisplayString, which basically is a byte string that contains only characters from the ASCII network virtual terminal (NVT) set.

timeout
(Input) 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 then use its own default timeout value. The upper bound and default timeout values for DPI subagents are maintained by the SNMP agent in the subagent MIB. For details about the subagent MIB, see ″SNMP Subagent MIB″ in the Simple Network Management Protocol (SNMP) Support

manual.

max_varBinds
(Input) The maximum number of varbinds per DPI packet that the subagent is prepared to handle. The agent tries to combine up to this number of varbinds (belonging to the same subtree) in a single DPI packet. If zero is specified, there is no explicit upper bound on the number of varbinds. In all cases, the actual number of varbinds is constrained by buffer sizes.

character_set
(Input) The character set that you want to use for string-based data fields in the DPI packets and structures. In general, the SNMP agent communicates to all SNMP managers in NVT ASCII and stores information in its own MIBs in ASCII. However, the agent will do some translations. Currently, only DPI_NATIVE_CSET is supported. For a System i™ product, this is EBCDIC (coded character set identifier (CCSID) 500).

The specifics are as follows:

- On SET, COMMIT and UNDO requests from the agent, if the OID Structure of Management Information (SMI) type is SNMP_TYPE_OCTET_STRING and the textual convention is DisplayString, the agent will translate from ASCII to EBCDIC. The **<qtossapi.h>** file contains the C-language defines for these SMI types.

  Note: A subagent implementation with DisplayString OIDs that have read/write access should check the value_type in the snmp_dpi_set_packet (see the **<qtossapi.h>** file). If the value_type is not equal to the SNMP_TYPE_DisplayString in the set request, then the agent will not have converted from ASCII to EBCDIC. In this case, the subagent should perform the translation.
- If the textual convention is DisplayString during the processing of a GET or GETNEXT from a subagent, the agent will convert from EBCDIC to ASCII.
- When processing a DPI open packet, the agent will translate the description (see the description_p parameter) from EBCDIC to ASCII for storage in the subagent MIB.

- In the SNMP MIB II system group, there are a number of DisplayString OIDs. These are all stored in ASCII. (The Internet standard *RFC 1213*, ″Management Information Base for Network Management of TCP/IP-based internets: MIB-II″, defines MIB II and the system group as well as other groups.)

password_len
(Input) The length (in bytes) of an optional password. For the agent on a System i platform, subagents do not need to supply a password. If not, then a zero length may be specified.

password_p
(Input) A pointer to an byte string that represents the password for this subagent. This corresponds to an SNMP agent community name. A password may include any character value, including the NULL character. If the password_len is zero, then this can be a NULL pointer.

Return Value

value The value returned is a pointer to the DPI packet.

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

NULL If unsuccessful, then a NULL pointer is returned.

Be aware that the static buffer for the DPI packet is shared by other mkDPI*xxxx*() functions that create a serialized DPI packet.

For more information, see the Simple Network Management Protocol (SNMP) Support manual.

Usage Notes

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

The SNMP agent will send a DPI response packet back to the subagent with a code that can be used to determine if the open request was successful. This will be one of the SNMP_ERROR_DPI_* return codes found in **<qtossapi.h>**. Following receipt of this response packet, the subagent will need to call the **pDPIpacket()** to parse this DPI packet. The error_code should be checked.

If the error_code is SNMP_ERROR_DPI_duplicateSubAgentIdentifier, then another subagent with the same subagent OID has already sent an open DPI packet and the SA MIB OID saAllowDuplicateIDs is 2 (No). Either choose a different OID for this subagent, change saAllowDuplicateIDs to 1 (Yes) or stop the other subagent that has the requested identifier. The **fDPIparse()** function would normally be called after that to free the parsed DPI response packet. For information about saAllowDuplicateIDs, see ″SNMP Subagent MIB″ in the Simple Network Management Protocol (SNMP) Support manual.

Related Information

- The <**qtossapi.h**> file (see “Header Files for UNIX-Type Functions” on page 57)
- “DPI_PACKET_LEN()—Get Length of DPI Packet” on page 10—Get Length of DPI Packet
- “fDPIparse()—Free Storage from DPI Packet Parse” on page 11—Free Storage from DPI Packet Parse
- “pDPIpacket()—Parse a DPI Packet” on page 27—Parse a DPI Packet

Example

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 68.

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

pack_p = mkDPIopen("1.3.6.1.2.3.4.5",
            "Sample DPI sub-agent"
            0L,2L, DPI_NATIVE_CSET,
            0,(char *)0);
if (pack_p) {
   /* Send packet to the agent. */
}
```

API introduced: V3R6

Top | UNIX-Type APIs | APIs by category

mkDPIregister()—Make a DPI Register Packet

Syntax

```
#include <qtossapi.h>

unsigned char     *mkDPIregister(
    unsigned short     timeout,
    long int           priority,
    char               *group_p,
    char               bulk_select);
```

Service Program Name: QTOSSAPI
Default Public Authority: *USE
Threadsafe: No

The **mkDPIregister()** function makes a Distributed Protocol Interface (DPI®) register packet and returns a pointer to the packet.

Authorities and Locks

None.

Parameters

timeout
(Input) 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
(Input) The requested priority, relative to other DPI subagents. This field may contain any of these values:

-1 The best available priority.

0 A better priority than the highest priority currently registered. Use this value to obtain the SNMP DPI version 1 behavior.

nnn Any other positive value. You will receive that priority if available; otherwise, the next best priority that is available.

group_p
(Input) A pointer to a NULL-terminated character string that represents the subtree to be registered. This group ID must have a trailing dot.

bulk_select
(Input) Whether you want the agent to pass GETBULK on to the subagent or to map them into

multiple GETNEXT requests. The possible value follows:

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

Return Value

value The **mkDPIregister()** function was successful. The value returned is a pointer to the DPI packet.

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

NULL **The mkDPIregister()** function was not successful.

If unsuccessful, then a NULL pointer is returned.

Be aware that the static buffer for the DPI packet is shared by other mkDPI*xxxx*() functions that create a serialized DPI packet.

For more information, see the Simple Network Management Protocol (SNMP) Support manual.

Usage Notes

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

The SNMP agent will send a DPI response packet back to the subagent with a code that can be used to determine if the register request was successful. This will be one of the SNMP_ERROR_DPI_* return codes found in **<qtossapi.h>**. Following receipt of this response packet, the subagent will need to call the **pDPIpacket()** to parse the incoming DPI packet and to check the response packet error_code. Then, **fDPIparse()** would normally be called to free the parsed DPI packet.

If the response from the SNMP agent is SNMP_ERROR_DPI_higherPriorityRegistered, then a DPI subagent has already registered the same subtree at a higher priority than requested in this call. If so, this subagent will be contained in the subagent Management Information Base (MIB), and using an appropriate SNMP management application, you can determine its priority. You may want to consider requesting a higher priority or even -1 (best available) for your subagent.

If the response from the SNMP agent is SNMP_ERROR_DPI_alreadyRegistered, then the requested subtree registration was for a portion of the overall MIB that is supported by an SNMP agent directly or by other system-implemented programs. Generally, registration of any subtree root, which would have the effect of masking all or portions of these subtrees (if allowed to occur), is prohibited.

Not all protected subtrees are currently supplied on a System i™ platform, although most are. If a subtree is currently not supplied, then the first subagent that dynamically registers it will be allowed, and later subagents will be disallowed. Refer to the "SNMP Agent Set Processing and Supported SNMP MIBs" in the Simple Network Management Protocol (SNMP) Support manual for information about the MIB groups currently supplied with i5/OS®.

Following are the protected subtrees and the associated MIB name:

1.3.6.1.2.1.1	System
1.3.6.1.2.1.2	Interfaces
1.3.6.1.2.1.3	Address translation

1.3.6.1.2.1.4	Internet Protocol
1.3.6.1.2.1.5	Internet Control Message Protocol
1.3.6.1.2.1.6	Transmission Control Protocol (TCP)
1.3.6.1.2.1.7	User Datagram Protocol (UDP)
1.3.6.1.2.1.10.7	Ethernet
1.3.6.1.2.1.10.9	Token ring
1.3.6.1.2.1.10.15	Fiber distributed data interface (FDDI)
1.3.6.1.2.1.10.32	Frame relay
1.3.6.1.2.1.11	SNMP
1.3.6.1.2.1.25	Host
1.3.6.1.3.6	Interface extensions
1.3.6.1.4.1.2.2.12	Subagent
1.3.6.1.4.1.2.2.1	Distributed Protocol Interface (DPI) (See the Internet standard *RFC 1592*, “Simple Network Management Protocol Distributed Protocol Interface”, Version 1.0.)
1.3.6.1.4.1.2.6.2.13	Advanced Peer-to-Peer Networking® (APPN)
1.3.6.1.4.1.2.6.4.5	NetView/6000 subagent computer system group
1.3.6.1.4.1.2.6.50	Client management
1.3.6.1.4.1.23.2.5	Internetwork Packet Exchange (IPX) protocol
1.3.6.1.4.1.23.2.19	Netware Link Services Protocol (NLSP)
1.3.6.1.4.1.23.2.20	Router Information Protocol (RIP) and Service Advertising Protocol (SAP)

Related Information

- The <**qtossapi.h**> file (see “Header Files for UNIX-Type Functions” on page 57)
- “DPI_PACKET_LEN()—Get Length of DPI Packet” on page 10—Get Length of DPI Packet
- “fDPIparse()—Free Storage from DPI Packet Parse” on page 11—Free Storage from DPI Packet Parse
- “pDPIpacket()—Parse a DPI Packet” on page 27—Parse a DPI Packet

Example

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 68.

```
#include <qtossapi.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. */
}
```

API introduced: V3R6

Top | UNIX-Type APIs | APIs by category

mkDPIresponse()—Make a DPI Response Packet

Syntax

```
#include <qtossapi.h>

unsigned char     *mkDPIresponse(
     snmp_dpi_hdr       *hdr_p,
     long int           error_code,
     long int           error_index,
     snmp_dpi_set_packet *packet_p );
```

Service Program Name: QTOSSAPI
Default Public Authority: *USE
Threadsafe: No

The **mkDPIresponse()** function makes a DPI® response packet and returns a pointer to the packet.

Authorities and Locks

None.

Parameters

hdr_p (Input) A pointer to the snmp_dpi_hdr structure of the DPI request to which this DPI packet will be the response. The function uses this structure to copy the packet_id and the DPI version and release so that the DPI packet is correctly formatted as a response.

error_code
(Input) The error code from the **<qtossapi.h>** file.

error_index
(Input) 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
(Input) A pointer to a chain of snmp_dpi_set_packet structures. This partial structure will be freed by the mkDPIresponse() function. Therefore, on return, you cannot refer to it anymore. Pass a NULL pointer if there are no varbinds to be returned.

Return Value

value The value returned is a pointer to the DPI packet.

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

NULL If unsuccessful, then a NULL pointer is returned.

Be aware that the static buffer for the DPI packet is shared by other mkDPI*xxxx*() functions that create a serialized DPI packet.

For more information, see the Simple Network Management Protocol (SNMP) Support manual.

Usage Notes

The **mkDPIresponse()** function is used by a subagent to prepare a DPI RESPONSE packet to a GET, GETNEXT, SET, COMMIT, or UNDO request. The resulting packet can be sent to the SNMP agent.

Unnecessary free operations may result in an MCH6902 (type 2). If this occurs, remove the call to **fDPIset()**.

Related Information

- The <**qtossapi.h**> file (see “Header Files for UNIX-Type Functions” on page 57)
- “DPI_PACKET_LEN()—Get Length of DPI Packet” on page 10—Get Length of DPI Packet

Example

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 68.

```
#include <qtossapi.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 subagent. */
      }
   }
}
```

API introduced: V3R6

Top | UNIX-Type APIs | APIs by category

mkDPIset()—Make a DPI Set Packet

Syntax

```
#include <qtossapi.h>

snmp_dpi_set_packet *mkDPIset(
     snmp_dpi_set_packet *packet_p,
     char                *group_p,
     char                *instance_p,
     int                 value_type,
     int                 value_len,
     void                *value_p );
```

Service Program Name: QTOSSAPI
Default Public Authority: *USE
Threadsafe: No

The **mkDPIset()** function makes a DPI set structure and adds it to a chained list of set structures if previous calls have been made.

Authorities and Locks

None.

Parameters

packet_p

(Input) A pointer to a chain of snmp_dpi_set_packet structures. Pass a NULL pointer if this is the first structure to be created. Typically, to handle multiple varbinds, this routine will be called repeatedly with this parameter having as its value the result returned from the previous call. Each new snmp_dpi_set_packet will be chained at the end.

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

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

value_type
(Input) The type of the value.

See the **<qtossapi.h>** file for a list of currently defined value types.

value_len
(Input) A signed integer that specifies the length (in bytes) of the value pointed to by the value_p parameter. The length may be zero if the value is of type SNMP_TYPE_NULL.

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

Return Value

value The value returned is a pointer to the DPI packet.

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

NULL If unsuccessful, then a NULL pointer is returned.

For more information, see the Simple Network Management Protocol (SNMP) Support manual.

Usage Notes

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 then later used to create a DPI packet, using a call to **mkDPIresponse()** or **mkDPItrap()**, which can then be sent to an SNMP agent. Each occurrence of an snmp_dpi_set_packet corresponds to a varbind in a protocol data unit (PDU).

This function is unlike the other subagent APIs that have names beginning mkDPI, in that this function does not make a DPI packet that can be sent directly. Hence, it returns a pointer to an snmp_dpi_set_packet rather than a char * (as do the other mkDPI functions).

Note that if the nth (n > 1) call to this function fails for some reason, the pointer to the chain of previously built snmp_dpi_set_packet structures will be lost unless the caller saves it.

Related Information

- The <**qtossapi.h**> file (see “Header Files for UNIX-Type Functions” on page 57)
- “DPI_PACKET_LEN()—Get Length of DPI Packet” on page 10—Get Length of DPI Packet
- “fDPIset()—Free Storage from DPI Set Packet” on page 11—Free Storage from DPI Set Packet

- “mkDPIresponse()—Make a DPI Response Packet” on page 20)—Make a DPI Response Packet
- “mkDPItrap()—Make a DPI Trap Packet”—Make a DPI Trap Packet

Example

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 68.

```
#include <qtossapi.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 subagent. */
      }
   }
```

API introduced: V3R6

Top | UNIX-Type APIs | APIs by category

mkDPItrap()—Make a DPI Trap Packet

Syntax

```
#include <qtossapi.h>

unsigned char     *mkDPItrap(
     long int            generic,
     long int            specific,
     snmp_dpi_set_packet *packet_p,
     char                *enterprise_p );
```

Service Program Name: QTOSSAPI
Default Public Authority: *USE
Threadsafe: No

The **mkDPItrap()** function makes a DPI™ trap packet and returns a pointer to the packet.

Authorities and Locks

None.

Parameters

generic

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

specific
(Input) The (enterprise) specific trap type. This can be any value that is valid for the Management Information Base (MIB) subtrees that the subagent implements.

packet_p
(Input) A pointer to a chain of snmp_dpi_set_structures that represents the varbinds to be passed with the trap. This partial structure will be freed by the mkDPItrap() function; therefore, you cannot refer to it anymore on completion of the call. A NULL pointer means that there are no varbinds to be included in the trap.

enterprise_p
(Input) A pointer to a NULL-terminated character string that represents 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.

Note: This OID must not end in a period (.).

Return Value

value The value returned is a pointer to the DPI packet.

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

NULL If unsuccessful, then a NULL pointer is returned.

Be aware that the static buffer for the DPI packet is shared by other mkDPI*xxxx*() functions that create a serialized DPI packet.

For more information, see the Simple Network Management Protocol (SNMP) Support manual.

Usage Notes

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

Unnecessary free operations may result in an MCH6902 (type 2). If this occurs, remove the call to **fDPIset()**.

Related Information

- The <**qtossapi.h**> file (see "Header Files for UNIX-Type Functions" on page 57)
- "DPI_PACKET_LEN()—Get Length of DPI Packet" on page 10—Get Length of DPI Packet

Example

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 68.

```
#include <qtossapi.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 subagent. */
   }
}
```

API introduced: V3R6

Top | UNIX-Type APIs | APIs by category

mkDPIunregister()—Make a DPI Unregister Packet

Syntax

```
#include <qtossapi.h>

unsigned char     *mkDPIunregister(
   char              reason_code,
   char              *group_p);
```

Service Program Name: QTOSSAPI
Default Public Authority: *USE
Threadsafe: No

The **mkDPIunregister()** function makes a DPI® unregister packet and returns a pointer to the packet.

Authorities and Locks

None.

Parameters

reason_code
: (Input) The reason for the unregister operation. See the **<qtossapi.h>** file for a list of defined reason codes.

group_p
: (Input) A pointer to a NULL-terminated character string that represents the subtree to be unregistered. The subtree must have a trailing dot.

Return Value

pointer value
: The **mkDPIunregister()** function was successful. The value returned is a pointer to the DPI packet.

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

NULL
: The **mkDPIunregister()** function was not successful.

  If unsuccessful, then a NULL pointer is returned.

Be aware that the static buffer for the DPI packet is shared by other mkDPI*xxxx*() functions that create a serialized DPI packet.

For more information, see the Simple Network Management Protocol (SNMP) Support manual.

Usage Notes

The **mkDPIunregister()** function creates a serialized DPI UNREGISTER packet that can then be sent to the SNMP agent. Normally, the SNMP peer then sends a DPI RESPONSE packet back, which details if the unregister was successful or not.

Related Information

- The <**qtossapi.h**> file (see "Header Files for UNIX-Type Functions" on page 57)
- "DPI_PACKET_LEN()—Get Length of DPI Packet" on page 10—Get Length of DPI Packet

Example

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 68.

```
#include <qtossapi.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 or subagent and await response. */
}
```

API introduced: V3R6

Top | UNIX-Type APIs | APIs by category

---

pDPIpacket()—Parse a DPI Packet

Syntax

```
#include <qtossapi.h>

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

Service Program Name: QTOSSAPI
Default Public Authority: *USE
Threadsafe: No

The **pDPIpacket()** function parses a serialized Distributed Protocol Interface (DPI) packet to make it available for processing by the subagent.

Authorities and Locks

None.

Parameters

packet_p
(Input) A pointer to a serialized (incoming) DPI packet.

Return Value

pointer value The **pDPIpacket()** function was successful. The value returned is a pointer to the DPI packet.

If successful, a pointer to the snmp_dpi_hdr structure is returned. Storage for the structure has been dynamically allocated, and it is the caller's responsibility to free it when no longer needed. You can use the **fDPIparse()** function to free the structure.

NULL The **pDPIpacket()** function was not successful.

If unsuccessful, a NULL pointer is returned.

For more information, see the Simple Network Management Protocol (SNMP) Support manual.

Usage Notes

The **pDPIpacket()** function parses the buffer that is 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.

Typical follow-on processing will examine the packet_type in the returned snmp_dpi_hdr, and take various actions to process the various types of DPI packets that may arrive. A subagent would normally expect to handle all the possible DPI packet types listed in **<qtossapi.h>**, except SNMPv2 types (SNMP_DPI_GETBULK and SNMP_DPI_TRAPV2), and types sent only to SNMP agents (SNMP_DPI_OPEN, SNMP_DPI_REGISTER, SNMP_DPI_TRAP, and SNMP_DPI_INFORM). Note that a close or unregister packet can be sent from an agent to the subagent. And if the subagent receives an are-you-there packet, it should build and send a response packet with the proper error_code.

Related Information

- The <**qtossapi.h**> file (see “Header Files for UNIX-Type Functions” on page 57)
- “fDPIparse()—Free Storage from DPI Packet Parse” on page 11—Free Storage from DPI Packet Parse
- “pDPIpacket()—Parse a DPI Packet” on page 27—Parse a DPI Packet

Example

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 68.

```
#include <qtossapi.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. */
   switch( hdr_p->packet_type) {
      ...
      }
}
```

API introduced: V3R6

Top | UNIX-Type APIs | APIs by category

receiveDPIpacket()—Receive a DPI Packet from the SNMP Agent

Syntax

```
#include <qtossapi.h>

int   receiveDPIpacket(
          sa_dataq_msg      *dataq_msg_p,
          void              *dpi_msg_p,
          unsigned long int *length_p );
```

Service Program Name: QTOSSAPI
Default Public Authority: *USE
Threadsafe: No

The **receiveDPIpacket()** function obtains a copy of a DPI® packet sent by the SNMP agent to the subagent, and returns the DPI packet to the caller.

Authorities

Unlike the **waitDPIpacket()** function, this function does not actually refer to the subagent‚s data queue.

Parameters

dataq_msg_p
(Input) A pointer to the data queue message that was sent to the subagent to tell it that a DPI packet is pending. Note that the message must have already been received from the data queue by the subagent and placed in a buffer. This is a pointer to that message in the buffer. Use of this function assumes that the data queue messages are handled directly by the subagent‚s own code. See **waitDPIpacket()** for an alternative.

The sa_dataq_msg structure is defined in the **<qtossapi.h>** file.

dpi_msg_p
(I/O) A pointer to a buffer set up by the subagent that will contain the DPI serialized packet on successful return from this routine.

length_p
(Output) A pointer to an integer that will contain the length of the DPI packet contained in the subagent‚s buffer on successful return.

Return Value

The return values are defined in the **<qtossapi.h>** file.

0 *snmpsa_RC_ok*

The call was successful.

-1 *snmpsa_RC_err*

An exception occurred. Check the subagent job log for the exception information, correct the condition, and resubmit the subagent job. (This return code is only used when a more specific return code is not available.)

-2 *snmpsa_RC_noagent*

The SNMP agent is not available.

-3 *snmpsa_RC_mismatch*

A previous DPI packet was found. The subagent may want to process this packet or call **receiveDPIpacket()** again to get the next packet. See any messages in the job log and correct any errors that are indicated, then retry the operation.

-5 *snmpsa_RC_nonagentmsg*

The data queue message is not from the SNMP agent. (There is no DPI packet pending.)

-7 *snmpsa_RC_parmerr*

A parameter error occurred, probably a null pointer.

-8 *snmpsa_RC_lengtherr*

A parameter was an incorrect length.

-9 *snmpsa_RC_buffer*

Check the job log of the subagent for MCH3802. If found, the problem was likely due to agent workload, and the subagent can retry the request. If a different exception is found, see any messages in the job log, correct any errors that are indicated, and then retry the operation.

-12 *snmpsa_RC_connectfirst*

The subagent must connect to the SNMP agent before making this call.

For more information, see the Simple Network Management Protocol (SNMP) Support 📖 manual.

Usage Notes

The **receiveDPIpacket()** function obtains a copy of a DPI packet sent to the subagent. The copy is placed in a buffer owned by the subagent.

Use of this function by a subagent requires that the subagent programmer must wait for and receive the prompting message on the subagent,s data queue. An alternative is to use the **waitDPIpacket()** function, which handles the data queue for the subagent and also receives the DPI packet.

Related Information

- The <**qtossapi.h**> file (see "Header Files for UNIX-Type Functions" on page 57)
- "waitDPIpacket()—Wait for a DPI Packet" on page 32—Wait for a DPI Packet

Example

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 68.

```
#include  <qtossapi.h>
#define   MAX_MSG_SIZE       4096
char      dpibuff[MAX_MSG_SIZE],
          dataqbuff[80];
int       rc, len;

/* Wait for message on data queue.  When it arrives...        */
QRCVDTAQ( ... )
/* Handle exceptions. */

rc = receiveDPIpacket( &dataqbuff[0],
                       &dpibuff[0], &len );
if (rc) {
   /* Handle exceptions. */
}
```

API introduced: V3R6

Top | UNIX-Type APIs | APIs by category

sendDPIpacket()—Send a DPI Packet to the SNMP Agent

Syntax

```
#include <qtossapi.h>

int  sendDPIpacket( void *dpimsg_p, int length );
```

Service Program Name: QTOSSAPI
Default Public Authority: *USE
Threadsafe: No

The **sendDPIpacket()** function sends a copy of a Distributed Protocol Interface (DPI®) packet to the SNMP agent (on the same system as the subagent).

Authorities and Locks

None.

Parameters

dpimsg_p
(Input) A pointer to the serialized DPI packet.

length (Input) The length in bytes of the DPI packet to be sent.

Return Value

The return values are defined in the **<qtossapi.h>** file.

0 *snmpsa_RC_ok*

The routine was successful.

-1 *snmpsa_RC_err*

An exception occurred. Check the subagent job log for the exception information, correct the condition, and resubmit the subagent job. (This return code is only used when a more specific return code is not available.)

-2 *snmpsa_RC_noagent*

The SNMP agent is not available.

-4 *snmpsa_RC_timedout*

An internal time-out occurred. See the job log for further information about the exception.

-7 *snmpsa_RC_parmerr*

A parameter error occurred, probably a null pointer.

-8 *snmpsa_RC_lengtherr*

The length parameter may be incorrect, or the DPI packet to be sent is longer than the maximum length supported, or the length specified is not a positive number. See any messages in the job log and correct any errors that are indicated, then retry the operation.

-9 *snmpsa_RC_buffer*

If the subagent was trying to send a response to an SNMP agent request (for example, using get packets), it cannot be sent. The subagent may continue. (The SNMP manager may retry the original request.) If the subagent was trying to send a subagent-initiated packet (for example, using open or register packets), then a dynamic buffer was unavailable, probably due to agent workload. The subagent may try to send the packet again.

-11 *snmpsa_RC_canttrap*

A trap cannot be sent to the SNMP agent at this time, probably due to pending agent workload. The subagent may retry.

-12 *snmpsa_RC_connectfirst*

The subagent must connect to the SNMP agent before making this call.

For more information, see the Simple Network Management Protocol (SNMP) Support manual.

Usage Notes

The **sendDPIpacket()** function sends a copy of a DPI packet that was sent to the SNMP agent.

Related Information

- The <**qtossapi.h**> file (see "Header Files for UNIX-Type Functions" on page 57)

Example

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 68.

```
#include  <qtossapi.h>
unsigned  char *pack_p;
int       rc;

pack_p = mkDPIopen("1.3.6.1.2.3.4.5",
              "Sample DPI sub-agent"
              0L,2L, DPI_NATIVE_CSET,
              0,(char *)0);
if (pack_p) {

   /* Send packet to the agent. */
   rc = sendDPIpacket( pack_p, DPI_PACKET_LEN(pack_p) );

}
```

API introduced: V3R6

Top | UNIX-Type APIs | APIs by category

waitDPIpacket()—Wait for a DPI Packet

Syntax

```
#include <qtossapi.h>

int   waitDPIpacket(
        long int            timeout,
        void                *dpimsgbuff_p,
        unsigned long int   *length );
```

Service Program Name: QTOSSAPI
Default Public Authority: *USE
Threadsafe: No

The **waitDPIpacket()** function waits for a message on the data queue with which the subagent has previously connected (see "connectSNMP()—Establish Connection with SNMP Agent" on page 4). When a Distributed Protocol Interface (DPI®) packet arrives, this function receives the packet and copies it to a subagent buffer.

Authorities

So that the subagent can receive messages from the SNMP agent, the following conditions must be met:

- The SNMP agent job must have write access to the data queue. If you suspect a problem with the data queue, check the job log of the SNMP agent job (QTMSNMP in subsystem QSYSWRK) for TCP4041 messages with reason code 001.

Parameters

timeout

(Input) The number of seconds that the subagent is willing to wait for a message (a call to this function will block the subagent until a message is received or until this timeout is reached).

Possible values have the indicated meaning;

< 0 Unlimited wait

0 No wait. This causes an immediate return if a data queue message is not present.

> 0 The number of seconds to wait (maximum is 99999).

dpimsgbuff_p

(I/O) A pointer to a buffer that is owned by the subagent. This will contain the serialized packet from the SNMP agent when *snmpsa_RC_ok* is returned. The maximum length of a DPI packet is SNMP_DPI_BUFSIZE, defined in the **<qtossapi.h>** file. The buffer will contain the data queue message itself if that message is not from the SNMP agent, and **waitDPIpacket()** will return *snmpsa_RC_nonagentmsg*.

length (Output) When *snmpsa_RC_ok* is returned, the length (in bytes) of the DPI packet received. When *snmpsa_RC_nonagentmsg* is returned, the length of the data queue message. Otherwise, this value is 0.

Return Value

The return values are defined in the **<qtossapi.h>** file.

0 *snmpsa_RC_ok*

The routine was successful.

-1 *snmpsa_RC_err*

An exception occurred. Check the subagent job log for the exception information, correct the condition, and resubmit the subagent job. (This return code is only used when a more specific return code is not available.)

-2 *snmpsa_RC_noagent*

The SNMP agent is not available.

-3 *snmpsa_RC_mismatch*

A previous DPI packet was found. The subagent may want to process this packet or call the **receiveDPIpacket()** function again to get the next packet.

-4 *snmpsa_RC_timedout*

No message was received within the specified timeout.

-5 *snmpsa_RC_nonagentmsg*

A data queue message arrived that is not from the SNMP agent.

-6 *snmpsa_RC_dqinvalid*

The subagent data queue or library is invalid. This refers to the data queue and library used in the **connectSNMP()** call.

-7 *snmpsa_RC_parmerr*

A parameter error occurred, probably a null pointer.

-8 *snmpsa_RC_lengtherr*

A parameter was an incorrect length.

-9 *snmpsa_RC_buffer*

Check the job log of the subagent for MCH3802. If found, the problem was likely due to agent workload, and the subagent can retry the request. If a different exception is found, see any messages in the job log, correct any errors that are indicated, and then retry the operation.

-12 *snmpsa_RC_connectfirst*

The subagent must connect to the SNMP agent before making this call.

For more information, see the Simple Network Management Protocol (SNMP) Support manual.

Usage Notes

The **waitDPIpacket()** function waits for a message on the data queue that the subagent specified on the **connectSNMP()** call. When a data queue message is received, the corresponding DPI packet is copied to the specified subagent buffer.

If a data queue message arrives that is not from the SNMP agent, then it is returned in the buffer and the code *snmpsa_RC_nonagentmsg* is returned.

Related Information

- The <**qtossapi.h**> file (see “Header Files for UNIX-Type Functions” on page 57)
- “connectSNMP()—Establish Connection with SNMP Agent” on page 4—Establish Connection with SNMP Agent
- “pDPIpacket()—Parse a DPI Packet” on page 27—Parse a DPI Packet

Example

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 68.

```
#include <qtossapi.h>
#define             MAX_LEN       4096
#define             waitTIMEOUT   300
unsigned char       *pack_p,
                     dpimsgbuff[MAX_LEN];
snmp_dpi_hdr        *hdr_p;
snmp_dpi_set_packet *set_p;
long int             num, length;

for(;;) {

   rc = waitDPIpacket( waitTIMEOUT,
                       &dpimsgbuff[0], length );

   if (rc<0) {
       /* Handle exceptions. */
       }

   else {
      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 subagent. */

            } /*end if*/
         } /*end if*/
      } /*end if*/
   } /*end else*/
} /*end for*/
```

API introduced: V3R6

Top | UNIX-Type APIs | APIs by category

Simple Network Management Protocol (SNMP) Manager APIs

The SNMP manager APIs are:

- "snmpGet()—Retrieve MIB Objects" (Retrieve MIB objects) is used to get one or more management information base (MIB) objects from an SNMP agent or subagent on a local or remote system.
- "snmpGetnext()—Retrieve Next MIB Object" on page 39 (Retrieve next MIB object) is used to get the value of one or more management information base (MIB) objects from an SNMP agent or subagent on a local or remote system.
- "snmpSet()—Set MIB Objects" on page 43 (Set MIB objects) is used to set one or more management information base (MIB) objects in an SNMP agent or subagent on a local or remote system.

Note: These functions use header (include) files from the library QSYSINC, which is optionally installable. Make sure QSYSINC is installed on your system before using any of the functions. All of the SNMP manager APIs use header file **qtomeapi.h**. You can see this source in source file H, member name QTOMEAPI, in the QSYSINC library.

For examples that use the SNMP manager APIs, see "Using SNMP Manager APIs—Example" on page 53.

For information about trap support, see "SNMP Trap Support" on page 49.

SNMP managing applications typically use APIs to establish communication with local or remote SNMP agents, and then call other APIs to retrieve or modify MIB objects managed by those agents. The i5/OS® SNMP manager APIs accomplish both of these tasks within the same API. Three manager APIs are provided to perform the SNMP GET, GETNEXT, and SET operations. In general, all three APIs are blocked. That is, when the application calls these APIs, the API constructs a proper SNMP message, delivers it to the proper SNMP agent, waits, decodes the response from the agent, and delivers the information to the application. No processing occurs in the application until the API delivers this information or times out. The communications mechanism between the manager APIs and agents uses sockets. Therefore, both systems need to support sockets.

Application programmers who are writing network management applications can use the snmpGet(), snmpGetnext(), and snmpSet() SNMP APIs to retrieve and set management information base (MIB) data so that they can manage their systems and networks. Programmers should have a strong understanding of network management, SNMP, and some transport protocol such as Transmission Control Protocol/Internet Protocol (TCP/IP).

Top | UNIX-Type APIs | APIs by category

snmpGet()—Retrieve MIB Objects

Syntax

```
#include <qtomeapi.h>

int snmpGet(snmppdu  *          pdu_ptr,
            char  *             host_ptr,
            unsigned long int   time_out,
            char  *             comm_ptr,
            unsigned long int   comm_len);
```

Service Program Name: QTOMEAPI
Default Public Authority: *USE
Threadsafe: No

An SNMP managing application uses the **snmpGet()** function to get one or more management information base (MIB) objects from an SNMP agent or subagent on a local or remote system.

Parameters

pdu_ptr

(Input) A pointer to a structure of the protocol data unit (PDU) type as defined in the <**qtomeapi.h**> file.

This structure contains the PDU type (GET in this instance), the error status, the error index, and the pointer to the varbind structure.

The varbind structure (found in the **qtomeapi.h** file) consists of the following:

```
struct _varBind{
   struct _varBind * next;
   char   *oid;                         /* Null Terminated */
   unsigned char asn_type;
   int    val_len;
   union {
      int  * int_val;
      char * str_val;
   } val;
};
```

The fields for this structure are described as follows:

**next* The pointer to the next varbind. This has to be NULL if it is the last varbind in the list.

**oid* The pointer to the OID being set or retrieved (depending on the operation).

asn_type The ASN type of the OID. This field must be set by the user only for the snmpSet function. On the snmpGet or snmpGetnext function, it is returned by the API.

val_len For the snmpSet function, the user must set this to reflect the exact amount of data to be written to the OID. On an snmpGet or snmpGetnext, the user must use this field to indicate how much space to allocate for the value being retrieved. If the value coming back is greater than the amount of space allocated, a return code of 1 is received.

val A union of either a pointer to the string data or a pointer to the integer data. This space is allocated by the user.

host_ptr

(Input) A pointer to the character string that contains the Internet Protocol (IP) address.

This parameter can be stored in dotted decimal notation, that is, 9.130.38.217, or in host address format, that is, oursystem.endicott.ibm.com. This parameter must contain printable characters only.

time_out

(Input) The time-out value.

This parameter is the amount of time in seconds that the management application is willing to wait for the response PDU. The minimum value is 1, and the maximum is 100.

comm_ptr
(Input) A pointer to the character string that contains the community name.

This parameter contains a variable-length field that contains printable and nonprintable values. Therefore, the user must supply the exact length of this value in another parameter. EBCDIC-to-ASCII translation will not be done, and it is the responsibility of the managing application to specify the community name in the correct notation for the SNMP agent system.

comm_len
(Input) The length of the community name.

This parameter is the exact length of the community name. The minimum value is 1, and the maximum is 255.

Authorities

Service Program Authority
*USE

Return Value

The following are the possible return codes posted by the **snmpGet()** function:

0 API_RC_OK

snmpGet() was successful.

-4 API_RC_OUT_OF_MEMORY

There was not enough storage to complete this operation.

-5 API_RC_OUT_OF_BUFFERS

There were not enough internal buffers to continue.

-6 API_RC_OUT_OF_VARBINDS

The maximum number of allowable varbinds was exceeded.

-7 API_RC_SNMP_OUT_OF_VARBINDS

The maximum number of allowable varbinds was exceeded. This return code is equivalent to the -6 return code.

-9 API_RC_SNMP_INVALID_OID

The OID specified in the varbind list is not valid. This return code is equivalent to the -112 return code.

-10 API_RC_INVALID_VALUE

The specified value in the varbind is not valid.

-11 API_RC_INVALID_VALUE_REP

The specified value in the varbind is incorrectly represented.

-12 API_RC_DECODE_ERROR

The SNMP APIs were unable to decode the incoming PDU.

-13 API_RC_DECODE_ERROR

The SNMP APIs were unable to encode the PDU data structure.

-18 API_RC_TIMEOUT

A response to this request was not received within the allotted time-out value.

-21 API_RC_INVALID_PDU_TYPE

The PDU type was not recognized as one of the seven common PDU types.

-103 API_RC_INVALID_IP_ADDRESS

The IP address that was specified is not valid.

-104 `API_RC_INVALID_COMMUNITY_NAME_LENGTH`

The community name length must be greater than 0 and less than 256.

-108 API_RC_INVALID_TIMEOUT_PARM

The time-out value must be greater than 0 and less than or equal to 100.

-110 API_RC_UNKNOWN_HOST

The host name or IP address that is specified is not known on the network.

-112 API_RC_INVALID_OID

The OID that is specified in the varbind list is not valid.

-113 API_RC_INVALID_PDU_POINTER

The pointer value to the PDU structure must be non-NULL.

-114 API_RC_INVALID_HOST_POINTER

The pointer value to the host address must be non-NULL.

-115 API_RC_INVALID_HOST_POINTER

The pointer value to the community name must be non-NULL.

-201 API_RC_SOCKET_ERROR

The APIs have detected a socket error and cannot continue.

-202 API_RC_NOT_OK

The APIs have detected an unknown error and cannot continue. The val_len field of the varbind structure contains a value that is not valid.

1 `API_RC_VAL_LEN_LESS_THAN_RETURNED_VAL_LEN`

The value being returned by the API is greater than the space allocated by the user.

241 API_RC_DOMAIN_ERROR

This is equivalent to an MCH6801 error—stating object domain error.

242 API_RC_INVALID_POINTER

This is equivalent to an MCH3601 error—referenced location in a space does not contain a pointer.

243 API_RC_INVALID_PTR_TYPE

This is equivalent to an MCH3602 error-pointer type not valid for requested operation.

For more information, see the Simple Network Management Protocol (SNMP) Support manual.

Error Conditions

Following are the possible error statuses returned in the error status field of the PDU structure. These values are returned by the SNMP agents.

0 API_SNMP_ERROR_noError

The function was successful.

1 API_SNMP_ERROR_tooBig

The agent could not fit the results of an operation into a single SNMP message.

2 API_SNMP_ERROR_noSuchName

The requested operation identified an unknown variable name.

3 API_SNMP_ERROR_badValue

The requested operation specified an incorrect syntax or value when the management application tried to modify a variable.

5 API_SNMP_ERROR_genErr

A nonspecific error occurred while running this operation on the SNMP agent.

Usage Notes

The area where the data is returned is the responsibility of the user, not the API. To allocate storage, the user may use the AddVarbind routine (see "AddVarbind Routine" on page 53). To deallocate storage, the user may use the FreePdu routine (see "FreePdu Routine" on page 55).

You must use the correct PDU type on AddVarbind. It must match the operation on which you call. For example, if you build a PDU wherein AddVarbind passes a PDU type of Set and then you call the snmpGet operation using the PDU that you just created with Set, you will receive an error on the snmpGet call.

All character strings that are passed to the APIs must be null-terminated unless you explicitly provide the length, if a length field is available.

If you are building a PDU to go to a remote agent, you must remember to do correct translation of strings. The System i™ product is an EBCDIC system, whereas an SNMP agent on an RISC System/6000® (RS/6000®) computer is an ASCII system. Therefore, you must provide string values as you would see them on that system. For example, if you are sending a PDU to an RS/6000 system and the community name is `public`, you would enter the community name string in hexadecimal, X'7075626C6963'. See the data conversion APIs to convert data from EBCDIC to ASCII and vice versa.

These APIs are blocked, which means that on a call to the API a PDU is sent across a communications protocol to an SNMP agent on a local or remote system. The call returns when a response has been received from the agent or when the command times out. On the return, all returned data is placed in the appropriate locations. You need do no further action to retrieve such data.

Related Information

- The <**qtomeapi.h**> file (see "Header Files for UNIX-Type Functions" on page 57)
- "snmpGetnext()—Retrieve Next MIB Object"—Retrieve Next MIB Object
- "snmpSet()—Set MIB Objects" on page 43—Set MIB Objects

Example

For examples that pertain to the SNMP manager APIs, see "Using SNMP Manager APIs—Example" on page 53.

API introduced: V3R6

Top | UNIX-Type APIs | APIs by category

snmpGetnext()—Retrieve Next MIB Object

Syntax

```
#include <qtomeapi.h>

int snmpGetnext(snmppdu  *        pdu_ptr,
             char  *           host_ptr,
             unsigned long int  time_out,
             char  *           comm_ptr,
             unsigned long int  comm_len);
```

Service Program Name: QTOMEAPI
Default Public Authority: *USE
Threadsafe: No

An SNMP managing application uses the **snmpGetnext()** function to get the value of one or more management information base (MIB) objects from an SNMP agent or subagent on a local or remote system. The **snmpGetnext()** function gets the value of the object instance that is next in lexicographic order.

Parameters

pdu_ptr

(Input) A pointer to a structure of the protocol data unit (PDU) type as defined in the <**qtomeapi.h**> file.

This structure contains the PDU type (GET NEXT in this instance), the error status, the error index, and the pointer to the varbind structure.

The varbind structure (found in the **qtomeapi.h** file) consists of the following:

```
struct _varBind{
   struct _varBind * next;
   char   *oid;                       /* Null Terminated */
   unsigned char asn_type;
   int    val_len;
   union {
      int  * int_val;
      char * str_val;
   } val;
};
```

The fields for this structure are described as follows:

*next The pointer to the next varbind. This has to be NULL if it is the last varbind in the list.

*oid The pointer to the OID being set or retrieved (depending on the operation).

_asn_type_ The ASN type of the OID. This field must be set by the user only for the snmpSet function. On the snmpGet or snmpGetnext function, it is returned by the API.

_val_len_ For the snmpSet function, the user must set this to reflect the exact amount of data to be written to the OID. On an snmpGet or snmpGetnext, the user must use this field to indicate how much space to allocate for the value being retrieved. If the value coming back is greater than the amount of space allocated, a return code of 1 is received.

val A union of either a pointer to the string data or a pointer to the integer data. This space is allocated by the user.

host_ptr

(Input) A pointer to the character string that contains the Internet Protocol (IP) address.

This parameter can be stored in dotted decimal notation, that is, 9.130.38.217, or in host address format, that is, oursystem.endicott.ibm.com. This parameter must contain printable characters only.

time_out

(Input) The time-out value.

This parameter is the amount of time in seconds that the management application is willing to wait for the response PDU. The minimum value is 1, and the maximum is 100.

comm_ptr
(Input) A pointer to the character string that contains the community name.

This parameter contains a variable-length field that contains printable and nonprintable values. Therefore, the user must supply the exact length of this value in another parameter. EBCDIC-to-ASCII translation will not be done, and it is the responsibility of the managing application to specify the community name in the correct notation for the SNMP agent system.

comm_len
(Input) The length of the community name.

This parameter is the exact length of the community name. The minimum value is 1, and the maximum is 255.

Authorities

Service Program Authority
*USE

Return Value

The following are the possible return codes posted by the **snmpGetnext()** function:

0 API_RC_OK

snmpGetnext() was successful.

-4 API_RC_OUT_OF_MEMORY

There was not enough storage to complete this operation.

-5 API_RC_OUT_OF_BUFFERS

There were not enough internal buffers to continue.

-6 API_RC_OUT_OF_VARBINDS

The maximum number of allowable varbinds was exceeded.

-7 API_RC_SNMP_OUT_OF_VARBINDS

The maximum number of allowable varbinds was exceeded. This return code is equivalent to the -6 return code.

-9 API_RC_SNMP_INVALID_OID

The OID specified in the varbind list is not valid. This return code is equivalent to the -112 return code.

-10 API_RC_INVALID_VALUE

The specified value in the varbind is not valid.

-11 API_RC_INVALID_VALUE_REP

The specified value in the varbind is incorrectly represented.

-12 API_RC_DECODE_ERROR

The SNMP APIs were unable to decode the incoming PDU.

-13 API_RC_DECODE_ERROR

The SNMP APIs were unable to encode the PDU data structure.

-18 API_RC_TIMEOUT

A response to this request was not received within the allotted time-out value.

-21 API_RC_INVALID_PDU_TYPE

The PDU type was not recognized as one of the seven common PDU types.

-103 API_RC_INVALID_IP_ADDRESS

The IP address that was specified is not valid.

-104 API_RC_INVALID_COMMUNITY_NAME_LENGTH

The community name length must be greater than 0 and less than 256.

-108 API_RC_INVALID_TIMEOUT_PARM

The time-out value must be greater than 0 and less than or equal to 100.

-110 API_RC_UNKNOWN_HOST

The host name or IP address that is specified is not known on the network.

-112 API_RC_INVALID_OID

The OID that is specified in the varbind list is not valid.

-113 API_RC_INVALID_PDU_POINTER

The pointer value to the PDU structure must be non-NULL.

-114 API_RC_INVALID_HOST_POINTER

The pointer value to the host address must be non-NULL.

-115 API_RC_INVALID_HOST_POINTER

The pointer value to the community name must be non-NULL.

-201 API_RC_SOCKET_ERROR

The APIs have detected a socket error and cannot continue.

-202 API_RC_NOT_OK

The APIs have detected an unknown error and cannot continue. The val_len field of the varbind structure contains a value that is not valid.

1 API_RC_VAL_LEN_LESS_THAN_RETURNED_VAL_LEN

The value being returned by the API is greater than the space allocated by the user.

241 API_RC_DOMAIN_ERROR

This is equivalent to an MCH6801 error—stating object domain error.

242 API_RC_INVALID_POINTER

This is equivalent to an MCH3601 error—referenced location in a space does not contain a pointer.

243 API_RC_INVALID_PTR_TYPE

This is equivalent to an MCH3602 error-pointer type not valid for requested operation.

For more information, see the Simple Network Management Protocol (SNMP) Support manual.

Error Conditions

Following are the possible error statuses returned in the error status field of the PDU structure. These values are returned by the SNMP agents.

0 API_SNMP_ERROR_noError

The function was successful.

1 API_SNMP_ERROR_tooBig

The agent could not fit the results of an operation into a single SNMP message.

2 API_SNMP_ERROR_noSuchName

The requested operation identified an unknown variable name.

3 API_SNMP_ERROR_badValue

The requested operation specified an incorrect syntax or value when the management application tried to modify a variable.

5 API_SNMP_ERROR_genErr

A nonspecific error occurred while running this operation on the SNMP agent.

Usage Notes

The area where the data is returned is the responsibility of the user, not the API. To allocate storage, the user may use the AddVarbind routine (see “AddVarbind Routine” on page 53). To deallocate storage, the user may use the FreePdu routine (see “FreePdu Routine” on page 55).

You must use the correct PDU type on AddVarbind. It must match the operation on which you call. For example, if you build a PDU wherein AddVarbind passes a PDU type of Set and then you call the snmpGet operation using the PDU that you just created with Set, you will receive an error on the snmpGet call.

All character strings that are passed to the APIs must be null-terminated unless you explicitly provide the length, if a length field is available.

If you are building a PDU to go to a remote agent, you must remember to do correct translation of strings. The System i™ product is an EBCDIC system, whereas an SNMP agent on an RISC System/6000® (RS/6000®) computer is an ASCII system. Therefore, you must provide string values as you would see them on that system. For example, if you are sending a PDU to an RS/6000 system and the community name is `public`, you would enter the community name string in hexadecimal, X’7075626C6963’. See the data conversion APIs to convert data from EBCDIC to ASCII and vice versa.

These APIs are blocked, which means that on a call to the API a PDU is sent across a communications protocol to an SNMP agent on a local or remote system. The call returns when a response has been received from the agent or when the command times out. On the return, all returned data is placed in the appropriate locations. You need do no further action to retrieve such data.

Related Information

- The <**qtomeapi.h**> file (see “Header Files for UNIX-Type Functions” on page 57)
- “snmpGet()—Retrieve MIB Objects” on page 35—Retrieve MIB Objects
- “snmpSet()—Set MIB Objects”—Set MIB Objects

Example

For examples that pertain to the SNMP manager APIs, see “Using SNMP Manager APIs—Example” on page 53.

API introduced: V3R6

Top | UNIX-Type APIs | APIs by category

snmpSet()—Set MIB Objects

Syntax

```
#include <qtomeapi.h>

int snmpSet(snmppdu  *          pdu_ptr,
            char  *             host_ptr,
            unsigned long int   time_out,
            char  *             comm_ptr,
            unsigned long int   comm_len);
```

Service Program Name: QTOMEAPI
Default Public Authority: *USE
Threadsafe: No

An SNMP managing application uses the **snmpSet()** function to set one or more management information base (MIB) objects in an SNMP agent or subagent on a local or remote system.

Parameters

pdu_ptr

(Input) A pointer to a structure of the protocol data unit (PDU) type as defined in the <**qtomeapi.h**> file.

This structure contains the PDU type (SET in this instance), the error status, the error index, and the pointer to the varbind structure.

The varbind structure (found in the **qtomeapi.h** file) consists of the following:

```
struct _varBind{
   struct _varBind * next;
   char   *oid;                          /* Null Terminated */
   unsigned char asn_type;
   int    val_len;
   union {
      int  * int_val;
      char * str_val;
   } val;
};
```

The fields for this structure are described as follows:

*next The pointer to the next varbind. This has to be NULL if it is the last varbind in the list.

*oid The pointer to the OID being set or retrieved (depending on the operation).

asn_type The ASN type of the OID. This field must be set by the user only for the snmpSet function. On the snmpGet or snmpGetnext function, it is returned by the API.

val_len For the snmpSet function, the user must set this to reflect the exact amount of data to be written to the OID. On an snmpGet or snmpGetnext, the user must use this field to indicate how much space to allocate for the value being retrieved. If the value coming back is greater than the amount of space allocated, a return code of 1 is received.

val A union of either a pointer to the string data or a pointer to the integer data. This space is allocated by the user.

host_ptr

(Input) A pointer to the character string that contains the Internet Protocol (IP) address.

This parameter can be stored in dotted decimal notation, that is, 9.130.38.217, or in host address format, that is, oursystem.endicott.ibm.com. This parameter must contain printable characters only.

time_out

(Input) The time-out value.

This parameter is the amount of time in seconds that the management application is willing to wait for the response PDU. The minimum value is 1, and the maximum is 100.

comm_ptr

(Input) A pointer to the character string that contains the community name.

This parameter contains a variable-length field that contains printable and nonprintable values. Therefore, the user must supply the exact length of this value in another parameter. EBCDIC-to-ASCII translation will not be done, and it is the responsibility of the managing application to specify the community name in the correct notation for the SNMP agent system.

comm_len

(Input) The length of community name.

This parameter is the exact length of the community name. The minimum value is 1, and the maximum is 255.

Authorities

Service Program Authority

*USE

Return Value

The following are the possible return codes posted by the **snmpSet()** function:

0 API_RC_OK

snmpSet() was successful.

-4 API_RC_OUT_OF_MEMORY

There was not enough storage to complete this operation.

-5 API_RC_OUT_OF_BUFFERS

There were not enough internal buffers to continue.

-6 API_RC_OUT_OF_VARBINDS

The maximum number of allowable varbinds was exceeded.

-7 API_RC_SNMP_OUT_OF_VARBINDS

The maximum number of allowable varbinds was exceeded. This return code is equivalent to the -6 return code.

-9 API_RC_SNMP_INVALID_OID

The OID specified in the varbind list is not valid. This return code is equivalent to the -112 return code.

-10 API_RC_INVALID_VALUE

The specified value in the varbind is not valid.

-11 API_RC_INVALID_VALUE_REP

The specified value in the varbind is incorrectly represented.

-12 API_RC_DECODE_ERROR

The SNMP APIs were unable to decode the incoming PDU.

-13 API_RC_DECODE_ERROR

The SNMP APIs were unable to encode the PDU data structure.

-18 API_RC_TIMEOUT

A response to this request was not received within the allotted time-out value.

-21 API_RC_INVALID_PDU_TYPE

The PDU type was not recognized as one of the seven common PDU types.

-103 API_RC_INVALID_IP_ADDRESS

The IP address that was specified is not valid.

-104 API_RC_INVALID_COMMUNITY_NAME_LENGTH

The community name length must be greater than 0 and less than 256.

-108 API_RC_INVALID_TIMEOUT_PARM

The time-out value must be greater than 0 and less than or equal to 100.

-110 API_RC_UNKNOWN_HOST

The host name or IP address that is specified is not known on the network.

-112 API_RC_INVALID_OID

The OID that is specified in the varbind list is not valid.

-113 API_RC_INVALID_PDU_POINTER

The pointer value to the PDU structure must be non-NULL.

-114 API_RC_INVALID_HOST_POINTER

The pointer value to the host address must be non-NULL.

-115 API_RC_INVALID_HOST_POINTER

The pointer value to the community name must be non-NULL.

-201 API_RC_SOCKET_ERROR

The APIs have detected a socket error and cannot continue.

-202 API_RC_NOT_OK

The APIs have detected an unknown error and cannot continue. The val_len field of the varbind structure contains a value that is not valid.

1 API_RC_VAL_LEN_LESS_THAN_RETURNED_VAL_LEN

The value being returned by the API is greater than the space allocated by the user.

241 API_RC_DOMAIN_ERROR

This is equivalent to an MCH6801 error—stating object domain error.

242 API_RC_INVALID_POINTER

This is equivalent to an MCH3601 error—referenced location in a space does not contain a pointer.

243 API_RC_INVALID_PTR_TYPE

This is equivalent to an MCH3602 error—pointer type not valid for requested operation.

For more information, see the Simple Network Management Protocol (SNMP) Support manual.

Error Conditions

Following are the possible error statuses returned in the error status field of the PDU structure. These values are returned by the SNMP agents.

0 API_SNMP_ERROR_noError

The function was successful.

1 API_SNMP_ERROR_tooBig

The agent could not fit the results of an operation into a single SNMP message.

2 API_SNMP_ERROR_noSuchName

The requested operation identified an unknown variable name.

3 API_SNMP_ERROR_badValue

The requested operation specified an incorrect syntax or value when the management application tried to modify a variable.

5 API_SNMP_ERROR_genErr

A nonspecific error occurred while running this operation on the SNMP agent.

Usage Notes

The area where the data is returned is the responsibility of the user, not the API. To allocate storage, the user may use the AddVarbind routine (see "AddVarbind Routine" on page 53). To deallocate storage, the user may use the FreePdu routine (see "FreePdu Routine" on page 55).

You must use the correct PDU type on AddVarbind. It must match the operation on which you call. For example, if you build a PDU wherein AddVarbind passes a PDU type of Set and then you call the snmpGet operation using the PDU that you just created with Set, you will receive an error on the snmpGet call.

All character strings that are passed to the APIs must be null-terminated unless you explicitly provide the length, if a length field is available.

If you are building a PDU to go to a remote agent, you must remember to do correct translation of strings. The System i™ product is an EBCDIC system, whereas an SNMP agent on an RISC System/6000® (RS/6000®) computer is an ASCII system. Therefore, you must provide string values as you would see them on that system. For example, if you are sending a PDU to an RS/6000 system and the community name is `public`, you would enter the community name string in hexadecimal, X'7075626C6963'. See the data conversion APIs to convert data from EBCDIC to ASCII and vice versa.

These APIs are blocked, which means that on a call to the API a PDU is sent across a communications protocol to an SNMP agent on a local or remote system. The call returns when a response has been received from the agent or when the command times out. On the return, all returned data is placed in the appropriate locations. You need do no further action to retrieve such data.

Related Information

- The <**qtomeapi.h**> file (see "Header Files for UNIX-Type Functions" on page 57)
- "snmpGet()—Retrieve MIB Objects" on page 35—Retrieve MIB Objects
- "snmpGetnext()—Retrieve Next MIB Object" on page 39—Retrieve Next MIB Object

Example

For examples that pertain to the SNMP manager APIs, see "Using SNMP Manager APIs—Example" on page 53.

API introduced: V3R6

Top | UNIX-Type APIs | APIs by category

Concepts

These are the concepts for this category.

Debugging IP over SNA Configurations

Two commands can be helpful in debugging IP over SNA configurations:

- The Start Mode (STRMOD) CL command can help you determine if your SNA configuration is correct. As input to the STRMOD command, you need the remote location name. You can determine the remote location name from the destination IP address by using the Convert IP over SNA Interface (CVTIPSIFC) command. The message you receive when STRMOD completes tells you whether it was successful.
- The TCP/IP FTP command can help you determine if your AnyNet® configuration is correct. If you get the *User* prompt, the AnyNet configuration is correct.

  Note: When FTP fails, it does not give a detailed reason for the failure. To get a detailed reason, you should run a sockets program that reports the value for *errno* when the failure occurs.

Common IP over SNA Configuration Errors

Sockets Error (value of *errno*)	**Possible Causes**
EHOSTUNREACH	1. Missing ADDIPSLOC command on client system. 2. Missing ADDIPSIFC command on client system. 3. Type of service points to a non-existent mode description on client system. 4. ADDIPSLOC command on client system resulted in a location name that is not found. 5. ADDIPSLOC command on client system resulted in a location name that is on a non-APPC device description.
EADDRNOTAVAIL	1. AnyNet not active on client system (ALWANYNET attribute set to *NO), but TCP is started. 2. Mode could not be added to device on client system.
EUNATCH	1. AnyNet not active on client system (ALWANYNET attribute set to *NO), and TCP is not started.
ECONNREFUSED	1. AnyNet not active on client system (ALWANYNET attribute set to *NO). 2. *listen()* not active on server system.
ECONNABORTED	1. Line error 2. Device/controller/line varied off on client or server system while in use. 3. User not authorized to APPC device description object on server system.
ETIMEDOUT	1. ADDIPSLOC command on client system points to a location name that does not exist or is on a system that is not responding in the APPN network. 2. Messages (especially inquiry messages) on message queue QSYSOPR are waiting for a reply.
EACCES	1. User not authorized to port on client system. 2. User not authorized to APPC device description object on client system.

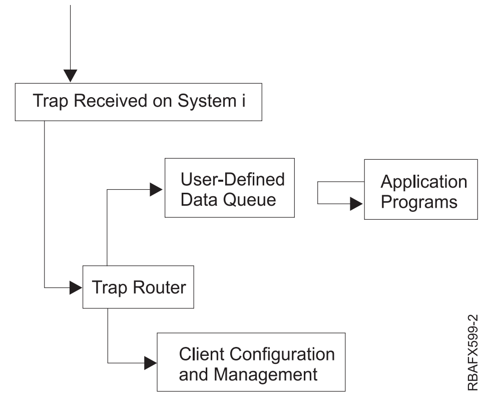
Top | UNIX-Type APIs | APIs by category

SNMP Trap Support

You can monitor for unsolicited SNMP trap messages by using the SNMP trap support. These trap messages may contain helpful data for managing a network.

By using the i5/OS® SNMP manager, it is possible to deliver SNMP traps to data queues. All traps that are received on a System i™ platform can be routed to user-defined data queues as shown in Figure: SNMP Trap Support (page 49). Your applications should monitor the data queue to receive trap information.

SNMP Trap Support



Configuring Trap Support

SNMP trap support uses the exit point QIBM_QZCA_SNMPTRAP and a data queue that you define. To use SNMP trap support, do the following:

1. Use the Work with Registration Information (WRKREGINF) command to determine if the QIBM_QZCA_SNMPTRAP exit point exists on your system (see Figure: Work with Registration Information (WRKREGINF) Display (page 50)). If the exit point does not exist, create and register the exit point by using this command:

   ```
   CALL PGM(QUSRGPT)
     PARM('QIBM_QZCA_SNMPTRAP  '
     'ZCAT0100' X'00000000' X'00000000')
   ```

   Note: The first parameter must be 20 characters long.
2. Define a data queue of 32780 bytes. For example, to define a data queue that is called MYQUEUE in library QGPL, enter:

   ```
   CRTDTAQ DTAQ(QGPL/MYQUEUE) MAXLEN(32780)
   ```
3. Register the exit program and exit program data with the QIBM_QZCA_SNMPTRAP exit point by using the Work with Registration Information (WRKREGINF) command. For example, see Figure: Work with Registration Information (WRKREGINF) Display (page 50) through Figure: Add Exit Program - Display 2 of 2 (page 51).

This configuration only registers the data queue name. You may want to add the program name and library that will use this data queue even though this information is not used by the system.

Figure: Work with Registration Information (WRKREGINF) Display

```
+--------------------------------------------------------------------------------+
|                       Work with Registration Information                       |
|                                                                                |
| Type options, press Enter.                                                     |
|   5=Display exit point   8=Work with exit programs                             |
|                                                                                |
|                            Exit                                                |
|      Exit                  Point                                               |
| Opt  Point                 Format    Registered  Text                          |
|      QIBM_QTA_TAPE_TMS     TMS00200     *YES                                   |
|      QIBM_QTF_TRANSFER     TRAN0100     *YES     Original File Transfer Functi |
|      QIBM_QVP_PRINTERS     PRNT0100     *YES     Original Virtual Print Server |
|      QIBM_QZCA_ADDC        ZCAA0100     *YES     Add Client exit point         |
|      QIBM_QZCA_REFC        ZCAF0100     *YES     Refresh Client Information ex |
|      QIBM_QZCA_RMVC        ZCAR0100     *YES     Remove Client exit point      |
|  8   QIBM_QZCA_SNMPTRAP    ZCAT0100     *YES                                   |
|      QIBM_QZCA_UPDC        ZCAU0100     *YES     Update Client Information exi |
|      QIBM_QZDA_INIT        ZDAI0100     *YES     Database Server - entry       |
|      QIBM_QZDA_NDB1        ZDAD0100     *YES     Database Server - database a  |
|      QIBM_QZDA_NDB1        ZDAD0200     *YES     Database Server - database a  |
|                                                                        More... |
| Command                                                                        |
| ===>                                                                           |
| F3=Exit   F4=Prompt   F9=Retrieve   F12=Cancel                                 |
+--------------------------------------------------------------------------------+
```

Figure: Work with Exit Programs Display (page 50) is reached using Option 8 from the display shown in the Figure above (page 50).

Figure: Work with Exit Programs Display

```
+--------------------------------------------------------------------------------+
|                            Work with Exit Programs                             |
|                                                                                |
| Exit point:   QIBM_QZCA_SNMPTRAP       Format:   ZCAT0100                      |
|                                                                                |
| Type options, press Enter.                                                     |
|   1=Add   4=Remove   5=Display   10=Replace                                    |
|                                                                                |
|               Exit                                                             |
|             Program     Exit                                                  |
| Opt          Number     Program        Library                                 |
|  1                      TRAPCHECK      QGPL                                    |
|                                                                                |
|   (No exit programs found.)                                                    |
|                                                                                |
|                                                                                |
|                                                                                |
|                                                                                |
|                                                                                |
|                                                                                |
| Command                                                                        |
| ===>                                                                           |
| F3=Exit   F4=Prompt   F5=Refresh   F9=Retrieve   F12=Cancel                    |
+--------------------------------------------------------------------------------+
```

Figure: Add Exit Program (page 50) is reached using Option 1 from the Figure above (page 50) and pressing F10 for additional parameters.

Figure: Add Exit Program - Display 1 of 2

```
+--------------------------------------------------------------------------------+
|                          Add Exit Program (ADDEXITPGM)                         |
|                                                                                |
| Type choices, press Enter.                                                     |
|                                                                                |
| Exit point . . . . . . . . . . . > QIBM_QZCA_SNMPTRAP                           |
| Exit point format  . . . . . . . > ZCAT0100      Name                           |
| Program number . . . . . . . . . > 1             1-2147483647, *LOW, *HIGH      |
| Program  . . . . . . . . . . . . > TRAPCHECK     Name                           |
|   Library  . . . . . . . . . . . >   QGPL        Name, *CURLIB                  |
| Text 'description' . . . . . . .   Reroute traps                                |
|                                                                                |
|                                                                                |
|                            Additional Parameters                               |
|                                                                                |
| Replace existing entry . . . . . > *NO           *YES, *NO                      |
| Create exit point  . . . . . . .   *NO           *YES, *NO                      |
|                                                                                |
|                                                                                |
|                                                                                |
|                                                                                |
|                                                                       More...  |
| F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display    |
| F24=More keys                                                                  |
+--------------------------------------------------------------------------------+
```

Figure: Add Exit Program - Display 2 of 2

```
+--------------------------------------------------------------------------------+
|                          Add Exit Program (ADDEXITPGM)                         |
|                                                                                |
| Type choices, press Enter.                                                     |
|                                                                                |
| Exit program data:                                                             |
|   Coded character set ID . . . .   *NONE         Number, *NONE, *JOB            |
|   Length of data . . . . . . . .   *CALC         0-2048, *CALC                  |
|   Program data . . . . . . . . .   QGPL/MYQUEUE                                 |
|                                                                                |
|                                                                                |
|                                                                                |
|                                                                                |
|                                                                                |
|                                                                                |
|                                                                                |
|                                                                                |
|                                                                                |
|                                                                                |
|                                                                                |
|                                                                                |
|                                                                        Bottom  |
| F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display    |
| F24=More keys                                                                  |
+--------------------------------------------------------------------------------+
```

Notes:

1. The program data field on the Add Exit Program (ADDEXITPGM) display contains the library name and the data queue name that will be used by the trap manager. This data must not exceed 21 bytes.
2. The exit program and library do not have to exist when they are added to the exit point.
3. The format for the ZCAT0100 trap data queue entry follows. For details about the SNMP trap, refer to the Internet standard for the trap message described in RFC 1155 and RFC 1157. The values for the object type field can be found in i5/OS library QSYSINC, file H, member QTOMEAPI.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Entry type (always *SNMPTRAP)
10	A	CHAR(2)	Entry ID (currently 01)
12	C	BINARY(4)	Version (This is the start of the trap header. All displacements are from the start of the trap header.)
16	10	BINARY(4)	Length of community name
20	14	BINARY(4)	Displacement to community name
24	18	BINARY(4)	Length of enterprise object ID
28	1C	BINARY(4)	Displacement to enterprise object ID
32	20	BINARY(4)	Length of agent address
36	24	BINARY(4)	Displacement to agent address
40	28	BINARY(4)	Generic trap type
44	2C	BINARY(4)	Specific trap code
48	30	BINARY(4)	Time stamp
52	34	BINARY(4)	Number of variable bindings
56	38	BINARY(4)	Displacement to first variable binding
Note: An array of variable bindings follows.			
These fields repeat for each variable binding		BINARY(4)	Length of object name
		BINARY(4)	Displacement to object name
		BINARY(4)	Length of value
		BINARY(4)	Displacement to value
		BINARY(4)	Value type (Values for this field can be found in i5/OS library QSYSINC, file H, member QTOMEAPI.)
Note: All object names and values follow.			
		CHAR(*)	Object names and values for all variable bindings

4. The library name and data queue must be specified in uppercase on the exit point.
5. Multiple exit programs are supported on the QIBM_QZCA_SNMPTRAP exit point. Each exit program must contain only one data queue.
6. A maximum of 100 data queues can be defined.
7. The data queue names are retrieved from the exit point only when the trap manager is started. To activate any changes to the data queues, you must end the trap manager with the End Trap Manager (ENDTRPMGR) command and restart the trap manager with the Start Trap Manager (STRTRPMGR) command.
8. In the preceding scenario, all traps are added to the data queue. If the queue is locked, damaged, destroyed, or named incorrectly, the traps are lost. It is the responsibility of the user application to remove traps from the queue. No messages are sent if the queue is full or traps not removed.

Top | UNIX-Type APIs | APIs by category

Using SNMP Manager APIs—Example

The examples in this topic provide two small routines that may aid in the use of several SNMP manager APIs. In addition, a sample snmpGet loop is provided to show the use of the two sample programs and its relation to an snmpGet call.

These examples are for the SNMP manager APIs snmpGet, snmpSet, and snmpGetnext.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 68.

AddVarbind Routine

This routine is used to create an initial protocol data unit (PDU), and subsequent calls will add varbinds to that PDU.

The value of this routine is that you will be able to create PDUs and add varbinds to those PDUs. The burden of storage allocation for each varbind and its values is removed from you, as is pointer maintenance on the varbinds. Remember that you need to deallocate any dynamic storage when you are done with it. The FreePdu routine (see page "FreePdu Routine" on page 55) is an easy way to do this. The AddVarbind sample code follows:

```
#include <qtomeapi.h>

int AddVarbind(snmppdu **pdu, char * oid, value v, unsigned
char pdu_type, unsigned char asn_type)
{

   varBind * t;                           /* Varbind pointer.       */
   int str_len,i;

   switch ( pdu_type ) {                  /* Check to make sure that*/
      case GET_PDU_TYPE:                  /* the PDU type is a known*/
      case SET_PDU_TYPE:                  /* value.  If not, the    */
      case GETNEXT_PDU_TYPE: break;       /* you may want to set a  */
      defaults: return(-2005);            /* return code value of   */
                                          /* your liking (for       */
   }                                      /* example, -2005).       */


   if (pdu[0] == NULL ||
       pdu[0] == 0 ||
       pdu[0] == '\0')
       {                  /* Check if PDU is null (meaning new PDU).*/
          pdu[0] = ( snmppdu *) malloc(sizeof(snmppdu));
                          /* Allocate storage for the PDU.          */
          memset( (snmppdu *) pdu[0],0,sizeof(pdu[0]));
                          /* Initialize the PDU to zeros.           */
          pdu[0]->pdu_type = pdu_type;
                          /* Initialize the PDU type.               */
          pdu[0]->varbind = ( varBind * ) malloc(sizeof(varBind));
                          /* Allocate storage for the varbind.      */
          str_len = strlen(oid);
                          /* Set the length of the OID.             */

         if (str_len > API_MAX_OID_SIZE ) return(-2000);
                          /* If OID length is not valid return.     */
         pdu[0]->varbind->oid =

           (char *) malloc(API_MAX_OID_SIZE+1);
         strcpy(pdu[0]->varbind->oid,oid);          /* Copy the
OID.*/
         pdu[0]->varbind->oid[str_len] = '\0';
                                               /*Null terminate OID.*/
         pdu[0]->varbind->next = NULL;      /* Nullify next
pointer.*/
                                     /* This signifies last varbind.*/
         t = pdu[0]->varbind;   /* Set temporary pointer to
varbind.*/
         t->val.str_val =
           (char *) malloc(API_MAX_VALUE_SIZE+1); /*Allocate storage */
                                         /* for the value of the OID.*/
/*********************************************************************/
/* Note:  This sample code shows a malloc of the maximum value size  */
/* plus 1 for null termination.  It would be in your best interest   */
/* to allocate only the amount of the actual value, plus 1.  This    */
/* reduces the amount of space allocated on each PDU.               */
/*********************************************************************/
   }
   else
   {
      if ( pdu[0]->pdu_type != pdu_type ) \keyword{return(-2001);
                           /* If this is not the initial call to      */
                           /* add a varbind, then check to make       */
                           /* sure the PDU type of this call          */
                           /* matches the original.                   */
      t = pdu[0]->varbind;
                           /* Store temporary pointer to this varbind.*/
      i = 0;               /* Initialize loop variable.               */

      while ( t->next != NULL ) /* Loop until you locate last varbind.*/
      {
         t = t->next;

         i++;

      }

      if ( i > 100 /* MAX_NUM... */ ) \keyword{return(-2002);
                           /* Return if you exceed maximum varbinds.  */
      t->next = ( varBind * ) malloc(sizeof(varBind));
                           /* Allocate storage for this varbind.      */
      t = t->next;         /* Set new temporary varbind pointer.      */
      str_len = strlen(oid);   /* Set length of OID.                  */

      if (str_len > API_MAX_OID_SIZE ) return(-2000);
                             /* If OID length exceed maximum, return. */
      t->oid = (char *) malloc(API_MAX_OID_SIZE+1);
                               /* Allocate storage for the OID.       */
      strcpy(t->oid,oid);
                               /* Copy OID to storage.                */
      t->oid[str_len] = '\0';
                               /* Null terminate the OID.             */
      t->val.str_val = (char *) malloc(API_MAX_VALUE_SIZE+1);
                               /* Allocate storage to hold value.     */
      t->val_len = API_MAX_VALUE_SIZE+1;
/**********************************************************************/
/* Note:  This sample code shows a malloc of the maximum value size   */
/* plus 1 for null termination.  It would be in your best interest    */
/* to allocate only the amount of the actual value, plus 1.  This     */
/* reduces the amount of space allocated on each PDU.                */
/**********************************************************************/
      t->next = NULL;
                               /* Nullify next varbind pointer        */
   }                           /* signifying the last varbind.        */

   if ( pdu_type == SET_PDU_TYPE )           /* For sets only         */
    {
      t->asn_type = asn_type;                /* Save ASN type         */

      switch (asn_type) {
         case API_ASN_OCTET_STRING:          /* All string types      */
         case API_ASN_OBJECT_IDENTIFIER:
         case API_ASN_IpAddress:
         case API_ASN_Opaque:
            str_len = strlen(v.str_val);        /* Store length       */
            strcpy(t->val.str_val,v.str_val);   /* Copy string        */
            t->val.str_val[str_len] = '\0';
                                                /* Null terminate     */
            t->val_len = str_len;               /* Save length        */
            break;
         case API_ASN_INTEGER:
         case API_ASN_Counter:
         case API_ASN_Gauge:
         case API_ASN_TimeTicks:
            *t->val.int_val = *v.int_val;       /* Save integer value */
            t->val_len = sizeof(int);           /* Save length of     */
            break;                              /* an integer.        */
         default: return(-2003);
      }
   }
   return(API_RC_OK);
}
```

FreePdu Routine

This routine is used to free all the dynamically allocated storage from AddVarbind.

The value of this routine is that you can free all the dynamically allocated (user domain) storage with one call. The FreePdu sample code follows:

```
#include <qtomeapi.h>

void FreePdu(snmppdu * pdu)   /* Pass in pointer to PDU.              */
{
   varBind * vb, *t;          /* Define pointers to varbinds.         */

   vb = pdu->varbind;         /* Set first varbind pointer.           */
   while (vb != NULL){        /* Loop as long as varbinds exist.      */
      t = vb;                 /* Save current varbind pointer.        */
      vb = vb->next;          /* Pointer to next varbind.             */
      free(t->oid);           /* Free storage allocated for OID.      */
      free(t->val.str_val);   /* Free storage allocated for value.    */
      free(t);       /* Free storage allocated for temporary varbind. */
   }
   free(pdu);                 /* Free storage allocated for PDU.      */
}
```

snmpGet Call Example

When you use the following example to call the snmpGet, snmpSet, or snmpGetnext API, it is important to note the following:

- The area where the data is returned is the responsibility of the user, not the API. To allocate storage, the user may use the AddVarbind routine (see "AddVarbind Routine" on page 53). To deallocate storage, the user may use the FreePdu routine (see "FreePdu Routine").
- You must use the correct PDU type on AddVarbind. It must match the operation on which you call. For example, if you build a PDU wherein AddVarbind passes a PDU type of Set and then you call the snmpGet operation using the PDU that you just created with Set, you will receive an error on the snmpGet call.
- All character strings that are passed to the APIs must be null-terminated unless you explicitly provide the length, if a length field is available.

- If you are building a PDU to go to a remote agent, you must remember to do correct translation of strings. The System i™ product is an EBCDIC system, whereas an SNMP agent on an RISC System/6000® (RS/6000®) computer is an ASCII system. Therefore, you must provide string values as you would see them on that system. For example, if you are sending a PDU to an RS/6000 system and the community name is public, you would enter the community name string in hexadecimal, X'7075626C6963'.
- These APIs are blocked, which means that on a call to the API a PDU is sent across a communications protocol to an SNMP agent on a local or remote system. The call returns when a response has been received from the agent or when the command times out. On the return, all returned data is placed in the appropriate locations. You need do no further action to retrieve such data.

The snmpGet sample code follows:

```
#include <qtomeapi.h>

void main() {

typedef union
 {
   int  * int_val;
   char * str_val;
 } value;                                   /* Value typedef.         */

   snmppdu *pdu;                            /* PDU pointer.           */
   value   v;                               /* Value container.       */
   int     rc;                              /* Return code.           */
   char    community_name[120];           /* Community container.   */

   pdu = NULL;                              /* Nullify PDU pointer.   */
   rc = AddVarbind(&pdu,                    /* Add varbind with       */
                      "1.3.6.1.2.1.1.1.0",  /* OID, value, type of    */
                      v,                    /* PDU this is for, ASN   */
                      GET_PDU_TYPE,         /* type.  PDU pointer     */
                      0);                   /* is set to non-null.    */
   if ( rc < 0 ) {                          /* Check error code user  */
      printf("Error: %d\n",rc);             /* defined here.  Sample  */
      exit(1);                              /* is print return code.  */
   }

   rc = AddVarbind(&pdu,                    /* Add second varbind.    */
                      "1.3.6.1.2.1.1.1.1",  /* PDU pointer is now     */
                      v,                    /* non-null after 1st     */
                      GET_PDU_TYPE,         /* invocation of Add-     */
                      0);                   /* Varbind.               */
   if ( rc < 0 ) {
      printf("Error: %d\n",rc);           /* Again, check return code.*/
      exit(1);
   }
      strcpy(community_name,"public");      /* Set community name.    */

 rc = snmpGet(pdu,                          /* Invoke operation.      */
                "system_name_of_snmp_agent_system",  /* Hostname.     */
                10,                         /* Time-out value.        */
                community_name,           /* Pointer to community name. */
                6);                         /* Correct length of      */
 }                                          /* community name.        */
```

Top | UNIX-Type APIs | APIs by category

Header Files for UNIX-Type Functions

Programs using the UNIX®-type functions must include one or more header files that contain information needed by the functions, such as:

- Macro definitions
- Data type definitions
- Structure definitions
- Function prototypes

The header files are provided in the QSYSINC library, which is optionally installable. Make sure QSYSINC is on your system before compiling programs that use these header files. For information about installing the QSYSINC library, see Include files and the QSYSINC library.

The table below shows the file and member name in the QSYSINC library for each header file used by the UNIX-type APIs in this publication.

Name of Header File	Name of File in QSYSINC	Name of Member
arpa/inet.h	ARPA	INET
arpa/nameser.h	ARPA	NAMESER
bse.h	H	BSE
bsedos.h	H	BSEDOS
bseerr.h	H	BSEERR
dirent.h	H	DIRENT
errno.h	H	ERRNO
fcntl.h	H	FCNTL
grp.h	H	GRP
inttypes.h	H	INTTYPES
limits.h	H	LIMITS
netdbh.h	H	NETDB
netinet/icmp6.h	NETINET	ICMP6
net/if.h	NET	IF
netinet/in.h	NETINET	IN
netinet/ip_icmp.h	NETINET	IP_ICMP
netinet/ip.h	NETINET	IP
netinet/ip6.h	NETINET	IP6
netinet/tcp.h	NETINET	TCP
netinet/udp.h	NETINET	UDP
netns/idp.h	NETNS	IDP
netns/ipx.h	NETNS	IPX
netns/ns.h	NETNS	NS
netns/sp.h	NETNS	SP
net/route.h	NET	ROUTE
nettel/tel.h	NETTEL	TEL
os2.h	H	OS2
os2def.h	H	OS2DEF

Name of Header File	Name of File in QSYSINC	Name of Member
pwd.h	H	PWD
Qlg.h	H	QLG
qp0lchsg.h	H	QP0LCHSG
qp0lflop.h	H	QP0LFLOP
qp0ljrnl.h	H	QP0LJRNL
qp0lror.h	H	QP0LROR
qp0lrro.h	H	QP0LRRO
qp0lrtsg.h	H	QP0LRTSG
qp0lscan.h	H	QP0LSCAN
Qp0lstdi.h	H	QP0LSTDI
qp0wpid.h	H	QP0WPID
qp0zdipc.h	H	QP0ZDIPC
qp0zipc.h	H	QP0ZIPC
qp0zolip.h	H	QP0ZOLIP
qp0zolsm.h	H	QP0ZOLSM
qp0zripc.h	H	QP0ZRIPC
qp0ztrc.h	H	QP0ZTRC
qp0ztrml.h	H	QP0ZTRML
qp0z1170.h	H	QP0Z1170
qsoasync.h	H	QSOASYNC
qtnxaapi.h	H	QTNXAAPI
qtnxadtp.h	H	QTNXADTP
qtomeapi.h	H	QTOMEAPI
qtossapi.h	H	QTOSSAPI
resolv.h	H	» RESOLV «
semaphore.h	H	SEMAPHORE
signal.h	H	SIGNAL
spawn.h	H	SPAWN
ssl.h	H	SSL
sys/errno.h	H	ERRNO
sys/ioctl.h	SYS	IOCTL
sys/ipc.h	SYS	IPC
sys/layout.h	» SYS «	LAYOUT
sys/limits.h	H	LIMITS
» sys/mman.h	SYS «	MMAN
sys/msg.h	SYS	MSG
sys/param.h	SYS	PARAM
sys/resource.h	SYS	RESOURCE
sys/sem.h	SYS	SEM
sys/setjmp.h	SYS	SETJMP
sys/shm.h	SYS	SHM

Name of Header File	Name of File in QSYSINC	Name of Member
sys/signal.h	SYS	SIGNAL
sys/socket.h	SYS	SOCKET
sys/stat.h	SYS	STAT
sys/statvfs.h	SYS	STATVFS
sys/time.h	SYS	TIME
sys/types.h	SYS	TYPES
sys/uio.h	SYS	UIO
sys/un.h	SYS	UN
sys/wait.h	SYS	WAIT
ulimit.h	H	ULIMIT
unistd.h	H	UNISTD
utime.h	H	UTIME

You can display a header file in QSYSINC by using one of the following methods:

- Using your editor. For example, to display the **unistd.h** header file using the Source Entry Utility editor, enter the following command:
```
STRSEU SRCFILE(QSYSINC/H) SRCMBR(UNISTD) OPTION(5)
```
- Using the Display Physical File Member command. For example, to display the **sys/stat.h** header file, enter the following command:
```
DSPPFM FILE(QSYSINC/SYS) MBR(STAT)
```

You can print a header file in QSYSINC by using one of the following methods:

- Using your editor. For example, to print the **unistd.h** header file using the Source Entry Utility editor, enter the following command:
```
STRSEU SRCFILE(QSYSINC/H) SRCMBR(UNISTD) OPTION(6)
```
- Using the Copy File command. For example, to print the **sys/stat.h** header file, enter the following command:
```
CPYF FROMFILE(QSYSINC/SYS) TOFILE(*PRINT) FROMMBR(STAT)
```

Symbolic links to these header files are also provided in directory /QIBM/include.

Top | UNIX-Type APIs | APIs by category

Errno Values for UNIX-Type Functions

Programs using the UNIX® -type functions may receive error information as *errno* values. The possible values returned are listed here in ascending *errno* value sequence.

Name	Value	Text	Details
EDOM	3001	A domain error occurred in a math function.	
ERANGE	3002	A range error occurred.	
ETRUNC	3003	Data was truncated on an input, output, or update operation.	

Name	Value	Text	Details
ENOTOPEN	3004	File is not open.	You attempted to do an operation that required the file to be open.
ENOTREAD	3005	File is not opened for read operations.	You tried to read a file that is not open for read operations.
EIO	3006	Input/output error.	A physical I/O error occurred or a referenced object was damaged.
ENODEV	3007	No such device.	
ERECIO	3008	Cannot get single character for files opened for record I/O.	The file that was specified is open for record I/O and you attempted to read it as a stream file.
ENOTWRITE	3009	File is not opened for write operations.	You tried to update a file that has not been opened for write operations.
ESTDIN	3010	The stdin stream cannot be opened.	
ESTDOUT	3011	The stdout stream cannot be opened.	
ESTDERR	3012	The stderr stream cannot be opened.	
EBADSEEK	3013	The positioning parameter in fseek is not correct.	
EBADNAME	3014	The object name specified is not correct.	
EBADMODE	3015	The type variable specified on the open function is not correct.	The mode that you attempted to open the file in is not correct.
EBADPOS	3017	The position specifier is not correct.	
ENOPOS	3018	There is no record at the specified position.	You attempted to position to a record that does not exist in the file.
ENUMMBRS	3019	Attempted to use ftell on multiple members.	Remove all but one member from the file.
ENUMRECS	3020	The current record position is too long for ftell.	
EINVAL	3021	The value specified for the argument is not correct.	A function was passed incorrect argument values, or an operation was attempted on an object and the operation specified is not supported for that type of object.
EBADFUNC	3022	Function parameter in the signal function is not set.	
ENOENT	3025	No such path or directory.	The directory or a component of the path name specified does not exist.
ENOREC	3026	Record is not found.	
EPERM	3027	The operation is not permitted.	You must have appropriate privileges or be the owner of the object or other resource to do the requested operation.
EBADDATA	3028	Message data is not valid.	The message data that was specified for the error text is not correct.
EBUSY	3029	Resource busy.	An attempt was made to use a system resource that is not available at this time.
EBADOPT	3040	Option specified is not valid.	
ENOTUPD	3041	File is not opened for update operations.	

Name	Value	Text	Details
ENOTDLT	3042	File is not opened for delete operations.	
EPAD	3043	The number of characters written is shorter than the expected record length.	The length of the record is longer than the buffer size that was specified. The data written was padded to the length of the record.
EBADKEYLN	3044	A length that was not valid was specified for the key.	You attempted a record I/O against a keyed file. The key length that was specified is not correct.
EPUTANDGET	3080	» A write operation should not immediately follow a read operation. «	
EGETANDPUT	3081	» A read operation should not immediately follow a write operation. «	
EIOERROR	3101	A nonrecoverable I/O error occurred.	
EIORECERR	3102	A recoverable I/O error occurred.	
EACCES	3401	Permission denied.	An attempt was made to access an object in a way forbidden by its object access permissions.
ENOTDIR	3403	Not a directory.	A component of the specified path name existed, but it was not a directory when a directory was expected.
ENOSPC	3404	No space is available.	The requested operations required additional space on the device and there is no space left. This could also be caused by exceeding the user profile storage limit when creating or transferring ownership of an object.
EXDEV	3405	Improper link.	A link to a file on another file system was attempted.
EAGAIN	3406	Operation would have caused the process to be suspended.	
EWOULDBLOCK	3406	Operation would have caused the process to be suspended.	
EINTR	3407	Interrupted function call.	
EFAULT	3408	The address used for an argument was not correct.	In attempting to use an argument in a call, the system detected an address that is not valid.
ETIME	3409	Operation timed out.	
ENXIO	3415	No such device or address.	
ECLOSED	3417	Socket closed.	
EAPAR	3418	Possible APAR condition or hardware failure.	
ERECURSE	3419	Recursive attempt rejected.	
EADDRINUSE	3420	Address already in use.	
EADDRNOTAVAIL	3421	Address is not available.	
EAFNOSUPPORT	3422	The type of socket is not supported in this protocol family.	

Name	Value	Text	Details
EALREADY	3423	Operation is already in progress.	
ECONNABORTED	3424	Connection ended abnormally.	
ECONNREFUSED	3425	A remote host refused an attempted connect operation.	
ECONNRESET	3426	A connection with a remote socket was reset by that socket.	
EDESTADDRREQ	3427	Operation requires destination address.	
EHOSTDOWN	3428	A remote host is not available.	
EHOSTUNREACH	3429	A route to the remote host is not available.	
EINPROGRESS	3430	Operation in progress.	
EISCONN	3431	A connection has already been established.	
EMSGSIZE	3432	Message size is out of range.	
ENETDOWN	3433	The network is currently not available.	
ENETRESET	3434	A socket is connected to a host that is no longer available.	
ENETUNREACH	3435	Cannot reach the destination network.	
ENOBUFS	3436	There is not enough buffer space for the requested operation.	
ENOPROTOOPT	3437	The protocol does not support the specified option.	
ENOTCONN	3438	Requested operation requires a connection.	
ENOTSOCK	3439	The specified descriptor does not reference a socket.	
ENOTSUP	3440	Operation is not supported.	The operation, though supported in general, is not supported for the requested object or the requested arguments.
EOPNOTSUPP	3440	Operation is not supported.	The operation, though supported in general, is not supported for the requested object or the requested arguments.
EPFNOSUPPORT	3441	The socket protocol family is not supported.	
EPROTONOSUPPORT	3442	No protocol of the specified type and domain exists.	
EPROTOTYPE	3443	The socket type or protocols are not compatible.	
ERCVDERR	3444	An error indication was sent by the peer program.	
ESHUTDOWN	3445	Cannot send data after a shutdown.	
ESOCKTNOSUPPORT	3446	The specified socket type is not supported.	

Name	Value	Text	Details
ETIMEDOUT	3447	A remote host did not respond within the timeout period.	
EUNATCH	3448	The protocol required to support the specified address family is not available at this time.	
EBADF	3450	Descriptor is not valid.	A file descriptor argument was out of range, referred to a file that was not open, or a read or write request was made to a file that is not open for that operation.
EMFILE	3452	Too many open files for this process.	An attempt was made to open more files than allowed by the value of OPEN_MAX. The value of OPEN_MAX can be retrieved using the sysconf() function.
ENFILE	3453	Too many open files in the system.	A system limit has been reached for the number of files that are allowed to be concurrently open in the system.
EPIPE	3455	Broken pipe.	
ECANCEL	3456	Operation cancelled.	
EEXIST	3457	Object exists.	The object specified already exists and the specified operation requires that it not exist.
EDEADLK	3459	Resource deadlock avoided.	An attempt was made to lock a system resource that would have resulted in a deadlock situation. The lock was not obtained.
ENOMEM	3460	Storage allocation request failed.	A function needed to allocate storage, but no storage is available.
EOWNERTERM	3462	The synchronization object no longer exists because the owner is no longer running.	The process that had locked the mutex is no longer running, so the mutex was deleted.
EDESTROYED	3463	The synchronization object was destroyed, or the object no longer exists.	
ETERM	3464	Operation was terminated.	
ENOENT1	3465	No such file or directory.	A component of a specified path name did not exist, or the path name was an empty string.
ENOEQFLOG	3466	Object is already linked to a dead directory.	The link as a dead option was specified, but the object is already marked as dead. Only one dead link is allowed for an object.
EEMPTYDIR	3467	Directory is empty.	A directory with entries of only dot and dot-dot was supplied when a nonempty directory was expected.
EMLINK	3468	Maximum link count for a file was exceeded.	An attempt was made to have the link count of a single file exceed LINK_MAX. The value of LINK_MAX can be determined using the pathconf() or the fpathconf() function.

Name	Value	Text	Details
ESPIPE	3469	Seek request is not supported for object.	A seek request was specified for an object that does not support seeking.
ENOSYS	3470	Function not implemented.	An attempt was made to use a function that is not available in this implementation for any object or any arguments.
EISDIR	3471	Specified target is a directory.	The path specified named a directory where a file or object name was expected.
EROFS	3472	Read-only file system.	You have attempted an update operation in a file system that only supports read operations.
EC2	3473	C2 pointer validation error.	
EUNKNOWN	3474	Unknown system state.	The operation failed because of an unknown system state. See any messages in the job log and correct any errors that are indicated, then retry the operation.
EITERBAD	3475	Iterator is not valid.	
EITERSTE	3476	Iterator is in wrong state for operation.	
EHRICLSBAD	3477	HRI class is not valid.	
EHRICLBAD	3478	HRI subclass is not valid.	
EHRITYPBAD	3479	HRI type is not valid.	
ENOTAPPL	3480	Data requested is not applicable.	
EHRIREQTYP	3481	HRI request type is not valid.	
EHRINAMEBAD	3482	HRI resource name is not valid.	
EDAMAGE	3484	A damaged object was encountered.	
ELOOP	3485	A loop exists in the symbolic links.	This error is issued if the number of symbolic links encountered is more than POSIX_SYMLOOP (defined in the limits.h header file). Symbolic links are encountered during resolution of the directory or path name.
ENAMETOOLONG	3486	A path name is too long.	A path name is longer than PATH_MAX characters or some component of the name is longer than NAME_MAX characters while _POSIX_NO_TRUNC is in effect. For symbolic links, the length of the name string substituted for a symbolic link exceeds PATH_MAX. The PATH_MAX and NAME_MAX values can be determined using the **pathconf()** function.
ENOLCK	3487	No locks are available.	A system-imposed limit on the number of simultaneous file and record locks was reached, and no more were available at that time.
ENOTEMPTY	3488	Directory is not empty.	You tried to remove a directory that is not empty. A directory cannot contain objects when it is being removed.

Name	Value	Text	Details
ENOSYSRSC	3489	System resources are not available.	
ECONVERT	3490	Conversion error.	One or more characters could not be converted from the source CCSID to the target CCSID.
E2BIG	3491	Argument list is too long.	
EILSEQ	3492	Conversion stopped due to input character that does not belong to the input codeset.	
ETYPE	3493	Object type mismatch.	The type of the object referenced by a descriptor does not match the type specified on the interface.
EBADDIR	3494	Attempted to reference a directory that was not found or was destroyed.	
EBADOBJ	3495	Attempted to reference an object that was not found, was destroyed, or was damaged.	
EIDXINVAL	3496	Data space index used as a directory is not valid.	
ESOFTDAMAGE	3497	Object has soft damage.	
ENOTENROLL	3498	User is not enrolled in system distribution directory.	You attempted to use a function that requires you to be enrolled in the system distribution directory and you are not.
EOFFLINE	3499	Object is suspended.	You have attempted to use an object that has had its data saved and the storage associated with it freed. An attempt to retrieve the object's data failed. The object's data cannot be used until it is successfully restored. The object's data was saved and freed either by saving the object with the STG(*FREE) parameter, or by calling an API.
EROOBJ	3500	Object is read-only.	You have attempted to update an object that can be read only.
EEAHDDSI	3501	Hard damage on extended attribute data space index.	
EEASDDSI	3502	Soft damage on extended attribute data space index.	
EEAHDDS	3503	Hard damage on extended attribute data space.	
EEASDDS	3504	Soft damage on extended attribute data space.	
EEADUPRC	3505	Duplicate extended attribute record.	
ELOCKED	3506	Area being read from or written to is locked.	The read or write of an area conflicts with a lock held by another process.
EFBIG	3507	Object too large.	The size of the object would exceed the system allowed maximum size.
EIDRM	3509	The semaphore, shared memory, or message queue identifier is removed from the system.	

Name	Value	Text	Details
ENOMSG	3510	The queue does not contain a message of the desired type and (msgflg logically ANDed with IPC_NOWAIT).	
EFILECVT	3511	File ID conversion of a directory failed.	To recover from this error, run the Reclaim Storage (RCLSTG) command as soon as possible.
EBADFID	3512	A file ID could not be assigned when linking an object to a directory.	The file ID table is missing or damaged. To recover from this error, run the Reclaim Storage (RCLSTG) command as soon as possible.
ESTALE	3513	File or object handle rejected by server.	
ESRCH	3515	No such process.	
ENOTSIGINIT	3516	Process is not enabled for signals.	An attempt was made to call a signal function under one of the following conditions: • The signal function is being called for a process that is not enabled for asynchronous signals. • The signal function is being called when the system signal controls have not been initialized.
ECHILD	3517	No child process.	
EBADH	3520	Handle is not valid.	
ETOOMANYREFS	3523	The operation would have exceeded the maximum number of references allowed for a descriptor.	
ENOTSAFE	3524	Function is not allowed.	Function is not allowed in a job that is running with multiple threads.
EOVERFLOW	3525	Object is too large to process.	The object's data size exceeds the limit allowed by this function.
EJRNDAMAGE	3526	Journal is damaged.	A journal or all of the journal's attached journal receivers are damaged, or the journal sequence number has exceeded the maximum value allowed. This error occurs during operations that were attempting to send an entry to the journal.
EJRNINACTIVE	3527	Journal is inactive.	The journaling state for the journal is *INACTIVE. This error occurs during operations that were attempting to send an entry to the journal.
EJRNRCVSPC	3528	Journal space or system storage error.	The attached journal receiver does not have space for the entry because the storage limit has been exceeded for the system, the object, the user profile, or the group profile. This error occurs during operations that were attempting to send an entry to the journal.

Name	Value	Text	Details
EJRNRMT	3529	Journal is remote.	The journal is a remote journal. Journal entries cannot be sent to a remote journal. This error occurs during operations that were attempting to send an entry to the journal.
ENEWJRNRCV	3530	New journal receiver is needed.	A new journal receiver must be attached to the journal before entries can be journaled. This error occurs during operations that were attempting to send an entry to the journal.
ENEWJRN	3531	New journal is needed.	The journal was not completely created, or an attempt to delete it did not complete successfully. This error occurs during operations that were attempting to start or end journaling, or were attempting to send an entry to the journal.
EJOURNALED	3532	Object already journaled.	A start journaling operation was attempted on an object that is already being journaled.
EJRNENTTOOLONG	3533	Entry is too large to send.	The journal entry generated by this operation is too large to send to the journal.
EDATALINK	3534	Object is a datalink object.	
ENOTAVAIL	3535	Independent Auxiliary Storage Pool (ASP) is not available.	The independent ASP is in Vary Configuration (VRYCFG) or Reclaim Storage (RCLSTG) processing. To recover from this error, wait until processing has completed for the independent ASP.
ENOTTY	3536	I/O control operation is not appropriate.	
EFBIG2	3540	Attempt to write or truncate file past its sort file size limit.	
ETXTBSY	3543	Text file busy.	An attempt was made to execute an i5/OS® PASE program that is currently open for writing, or an attempt has been made to open for writing an i5/OS PASE program that is being executed.
EASPGRPNOTSET	3544	ASP group not set for thread.	
ERESTART	3545	A system call was interrupted and may be restarted.	
ESCANFAILURE	3546	Object had scan failure.	An object has been marked as a scan failure due to processing by an exit program associated with the scan-related integrated file system exit points.

Top | UNIX-Type APIs | APIs by category

Code license and disclaimer information

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, IBM, ITS PROGRAM DEVELOPERS AND SUPPLIERS MAKE NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY.

UNDER NO CIRCUMSTANCES IS IBM, ITS PROGRAM DEVELOPERS OR SUPPLIERS LIABLE FOR ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

1. LOSS OF, OR DAMAGE TO, DATA;
2. DIRECT, SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC CONSEQUENTIAL DAMAGES; OR
3. LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF DIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

Appendix. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

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

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

```
IBM Corporation
Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.
```

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, IBM License Agreement for Machine Code, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

This API descriptions publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM i5/OS.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

Advanced 36
Advanced Function Presentation
Advanced Peer-to-Peer Networking
AFP
AIX
AnyNet
AS/400
BCOCA
C/400
COBOL/400
Common User Access
CUA
DB2
DB2 Universal Database
Distributed Relational Database Architecture
Domino
DPI
DRDA
Enterprise Storage Server
eServer
FlashCopy
GDDM
i5/OS
IBM
IBM (logo)
InfoColor
Infoprint
Integrated Language Environment
Intelligent Printer Data Stream
IPDS
Lotus
Lotus Notes
MO:DCA
MVS
Net.Data
NetServer
Notes
OfficeVision
Operating System/2
Operating System/400
OS/2
OS/400
PartnerWorld
POWER5+
PowerPC
Print Services Facility
PrintManager
PROFS
RISC System/6000
RPG/400
RS/6000

SAA
SecureWay
SOM
System i
System i5
System Object Model
System/36
System/38
System/390
TotalStorage
VisualAge
WebSphere
xSeries
z/OS

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Terms and conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal Use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of these publications, or any portion thereof, without the express consent of IBM.

Commercial Use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED ″AS-IS″ AND WITHOUT WARRANTY OF ANY KIND, EITHER

EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Printed in USA