IBM Systems - iSeries
UNIX-Type -- Simple Network Management Protocol (SNMP) APIs

Version 5 Release 4
IBM Systems - iSeries

UNIX-Type -- Simple Network Management Protocol (SNMP) APIs

Version 5 Release 4
Note

Before using this information and the product it supports, be sure to read the information in “Notices,” on page 49.
Simple Network Management Protocol (SNMP) 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 34

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

You can get more information about SNMP and Distributed Protocol Interface (DPI(R)) from Requests for Comment (RFC) on the Internet. A file called ways_to_get_rfc$s 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_rfc$s

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

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 iSeries) 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 iSeries SNMP agent.
- "DPI_PACKET_LEN()—Get Length of DPI Packet” on page 9 (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 10 (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 iSeries 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 iSeries 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 term **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

A
- `connectSNMP()`
  - `mkDPlopen()`
    - (D)

B
- `mkDPlregister()`
  - (D)

C
- (D')
  - (D)
  - `// other`
  - `// MIB implementation`
  - `mkDPlget()`
  - `mkDPlresponse()`

D
- `sendDPlpacket()`
  - `waitDPlpacket()`
  - `pDPlpacket()`
  - `// check packet return code`

E
- `mkDPlset()`

F
- `mkDPlclose()`
  - `sendDPlpacket()`
  - `disconnectSNMP()`
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 iSeries server) 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  \texttt{snmpsa\_RC\_ok}

The call was successful.

-1  \texttt{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  \texttt{snmpsa\_RC\_noagent}

The SNMP agent is not available.

-3  \texttt{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  \texttt{snmpsa\_RC\_timeout}

The specified timeout value was exceeded.

-7  \texttt{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  \texttt{snmpsa\_RC\_lengtherr}

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

-9  \texttt{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  \texttt{snmpsa\_RC\_duplicate}

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

-13  \texttt{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 \texttt{disconnectSNMP()} function, then the \texttt{connectSNMP()} function.

For more information, see “SNMP Subagent Problem Determination” in the \textbf{Simple Network Management Protocol (SNMP) Support} book.

**Usage Notes**

The \texttt{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 \texttt{debug DPI0, DPI PACKET LEN0}, and \texttt{mkDPI AreYouThere()}.  

**Related Information**

- The \texttt{qtossapi.h} file (see \textbf{Header Files for UNIX-Type Functions})
- “disconnectSNMP()—End Connection with SNMP Agent” on page 8—End Connection with SNMP Agent
Example
See Code disclaimer information for information pertaining to code examples.

```c
#include <qtossapi.h>

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

API introduced: V3R6

### debugDPI()—Set DPI Packet Trace

**Syntax**
```c
#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(R)) 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 `mkDPIxxx()` 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)
- “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

See Code disclaimer information for information pertaining to code examples.

```c
#include <qtossapi.h>

dbgDPI(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 line. Immediately following 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.

```c
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().

```c
pDPIpacket: Major=2, Version=2, Release=0, Id=1, Type=SNMP_DPI_GET
   cDPIget: 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
pDPIpacket: Major=2, Version=2, Release=0, Id=2, Type=SNMP_DPI_RESPONSE
pDPIresp: ret_code=0 [0x0] (noError), ret_index=0
pDPIset: 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().

```c
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
   cDPInext: 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
pDPIpacket: Major=2, Version=2, Release=0, Id=3, Type=SNMP_DPI_RESPONSE
pDPIresp: ret_code=0 [0x0] (noError), ret_index=0
pDPIset: 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
disconnectSNMP()—End Connection with SNMP Agent

Syntax
#include <qtossapi.h>

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

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

The specified timeout value was exceeded.

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.

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.

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

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

For more information, see “SNMP Subagent Problem Determination” in the Simple Network Management Protocol (SNMP) Support book.

**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](#)
- [connectSNMP()—Establish Connection with SNMP Agent](#)

**Example**

See [Code disclaimer information](#) for information pertaining to code examples.

```c
#include <qtossapi.h>

int rc;

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

API introduced: V3R6
#include <qtossapi.h>

int DPI_PACKET_LEN( unsigned char *packet_p );

Service Program Name: QTOSSAPI
Default Public Authority: *USE
Threadsake: 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 “SNMP Subagent Problem Determination” in the Simple Network Management Protocol (SNMP) Support book.

**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**

See Code disclaimer information for information pertaining to code examples.

```c
#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

---

**fDPIparse()—Free Storage from DPI Packet Parse**

**Syntax**

```c
#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)
• “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
See Code disclaimer information for information pertaining to code examples.
```
#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

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](#))
- "`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**

See [Code disclaimer information](#) for information pertaining to code examples.

```c
#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;

/* ... */

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 */
```
API introduced: V3R6

```
def fDPIset(first_p):
    /*
     * Send DPI response to agent. */
    /* If 2nd mkDPIset fails, */
    fDPIset(first_p);  /* it must free chain. */
```

The `mkDPIAreYouThere()` function makes a DPI ARE_YOU_THERE 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 `mkDPIxxxx()` functions that create a serialized DPI packet.

For more information, see “SNMP Subagent Problem Determination” in the Simple Network Management Protocol (SNMP) Support book.

### 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)
- "DPI_PACKET_LEN()—Get Length of DPI Packet" on page 9—Get Length of DPI Packet

Example

See Code disclaimer information for information pertaining to code examples.

```
#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

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 `mkDPIxxxx()` functions that create a serialized DPI packet.
For more information, see "SNMP Subagent Problem Determination" in the Simple Network Management Protocol (SNMP) Support book.

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)
- "DPI_PACKET_LEN()—Get Length of DPI Packet“ on page 9—Get Length of DPI Packet

Example

See Code disclaimer information for information pertaining to code examples.

```c
#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

mkDPIopen()—Make a DPI Open Packet

Syntax

```c
#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

**oids**  (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 book.

**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 the iSeries server, 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 iSeries server agent, 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 mkDPIxxxx() functions that create a serialized DPI packet.

For more information, see "SNMP Subagent Problem Determination" in the Simple Network Management Protocol book.

**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 (Yes). Either choose a different OID for this subagent, change saAllowDuplicateIDs to 1 (No) 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 book.

**Related Information**
- The <qtossapi.h> file (see Header Files for UNIX-Type Functions)
- "DPI_PACKET_LEN()—Get Length of DPI Packet" on page 9—Get Length of DPI Packet
- "fDPIparse()—Free Storage from DPI Packet Parse" on page 10—Free Storage from DPI Packet Parse
- "pDPIpacket()—Parse a DPI Packet" on page 27—Parse a DPI Packet

**Example**

See Code disclaimer information for information pertaining to code examples.

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

pack_p = mkDPIopen("1.3.6.1.2.3.4.5",
    "Sample DPI sub-agent",
    0L2L, DPI_NATIVE_CSET,
    0,(char *)0);
if (pack_p) {
    /* Send packet to the agent. */
}
```
API introduced: V3R6

### mkDPIregister()—Make a DPI Register Packet

**Syntax**

```c
#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 mkDPIxxxx() functions that create a serialized DPI packet.

For more information, see “SNMP Subagent Problem Determination” in the Simple Network Management Protocol book.

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 <qtosapi.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 the iSeries server, 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 “i5/OS SNMP Agent Set Processing and Supported SNMP MIBs” in the Simple Network Management Protocol book for information on the MIB groups currently supplied with i5/OS.

Following are the protected subtrees and the associated MIB name:

<table>
<thead>
<tr>
<th>OID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.3.6.1.2.1.1</td>
<td>System</td>
</tr>
<tr>
<td>1.3.6.1.2.1.2</td>
<td>Interfaces</td>
</tr>
<tr>
<td>1.3.6.1.2.1.3</td>
<td>Address translation</td>
</tr>
<tr>
<td>1.3.6.1.2.1.4</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>1.3.6.1.2.1.5</td>
<td>Internet Control Message Protocol</td>
</tr>
<tr>
<td>1.3.6.1.2.1.6</td>
<td>Transmission Control Protocol (TCP)</td>
</tr>
<tr>
<td>1.3.6.1.2.1.7</td>
<td>User Datagram Protocol (UDP)</td>
</tr>
<tr>
<td>1.3.6.1.2.1.10.7</td>
<td>Ethernet</td>
</tr>
</tbody>
</table>
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
1.3.6.1.2.1.10.11  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
1.3.6.1.4.1.2.2.2.1  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(R) (APPN(R))
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)
- “DPI_PACKET_LEN()—Get Length of DPI Packet” on page 9—Get Length of DPI Packet
- “fDPIparse()—Free Storage from DPI Packet Parse” on page 10—Free Storage from DPI Packet Parse
- “pDPIpacket()—Parse a DPI Packet” on page 27—Parse a DPI Packet

Example

See Code disclaimer information for information pertaining to code examples.

```c
#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

**mkDPIresponse()—Make a DPI Response Packet**

**Syntax**

```c
#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 mkDPIxxx() functions that create a serialized DPI packet.


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")
- "DPI_PACKET_LEN()—Get Length of DPI Packet" on page 9—Get Length of DPI Packet

Example

See "Code disclaimer information" for information pertaining to code examples.

```c
#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

---

**mkDPIset()—Make a DPI Set Packet**

**Syntax**

```c
#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
(In) The type of the value.
See the <qtossapi.h> file for a list of currently defined value types.

value_len
(In) 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
(In) 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 “SNMP Subagent Problem Determination” in the Simple Network Management Protocol book.

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)
- “DPI_PACKET_LEN()—Get Length of DPI Packet” on page 9—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” on page 24—Make a DPI Trap Packet

Example

See Code disclaimer information for information pertaining to code examples.
API introduced: V3R6

**mkDPItrap()**—Make a DPI Trap Packet

**Syntax**

```c
#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 mkDPIxxxx() functions that create a serialized DPI packet.

For more information, see “SNMP Subagent Problem Determination” in the Simple Network Management Protocol book.

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)
• “DPI_PACKET_LEN()—Get Length of DPI Packet” on page 9—Get Length of DPI Packet

Example

See Code disclaimer information for information pertaining to code examples.

```c
#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

---

### mkDPIunregister()—Make a DPI Unregister Packet

**Syntax**

```c
#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 **mkDPIxxxx()** functions that create a serialized DPI packet.

For more information, see “SNMP Subagent Problem Determination” in the [Simple Network Management Protocol](https://www.ibm.com) book.

#### 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](#))
- “DPI_PACKET_LEN()—Get Length of DPI Packet” on page 9—Get Length of DPI Packet

Example

See [Code disclaimer information](#) for information pertaining to code examples.

```c
#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

pDPIpacket()—Parse a DPI Packet

Syntax

```c
#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

- 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 "SNMP Subagent Problem Determination" in the Simple Network Management Protocol (SNMP) Support book.

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)
- "fDPIparse()—Free Storage from DPI Packet Parse” on page 10—Free Storage from DPI Packet Parse
- "pDPIpacket()—Parse a DPI Packet” on page 27—Parse a DPI Packet

Example
See Code disclaimer information for information pertaining to code examples.

```c
#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

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 “SNMP Subagent Problem Determination” in the Simple Network Management Protocol book.

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)
• “waitDPIpacket()—Wait for a DPI Packet” on page 32—Wait for a DPI Packet

Example
See Code disclaimer information for information pertaining to code examples.

```
#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

---

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 iSeries server 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 "SNMP Subagent Problem Determination" in the Simple Network Management Protocol.

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)

Example
See Code disclaimer information for information pertaining to code examples.
#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

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 QSYSW RK) 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 “SNMP Subagent Problem Determination” in the Simple Network Management Protocol book.

### 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](#))
- "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
See [Code disclaimer information](#) for information pertaining to code examples.

```c
#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 else*/
} /*end for*/
```

API introduced: V3R6

### 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.

For information about trap support, see SNMP Trap Support.

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(TM) 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).

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:

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

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

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

The APIs have detected a socket error and cannot continue.

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.

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

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

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

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

For more information, see “Problem Determination for SNMP Manager APIs” in the Simple Network Management Protocol book.

**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`). 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 iSeries server is an EBCDIC system, whereas an SNMP agent on an RISC System/6000(R) (RS/6000(R)) 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 `pub11c`, 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`)
- “snmpGetnext()—Retrieve Next MIB Object”—Retrieve Next MIB Object
- “snmpSet()—Set MIB Objects” on page 43—Set MIB Objects

Example

See `Code disclaimer information` for information pertaining to code examples.

For examples that pertain to the SNMP manager APIs, see `Using SNMP Manager APIs—Example`.

API introduced: V3R6

```
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

---

Simple Network Management Protocol (SNMP) APIs 39
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:

```c
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

*USE

Service Program Authority

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 “Problem Determination for SNMP Manager APIs” in the Simple Network Management Protocol book.

**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). 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 iSeries server 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 pub1ic, 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)
- "snmpGet()—Retrieve MIB Objects” on page 35—Retrieve MIB Objects
- "snmpSet()—Set MIB Objects”—Set MIB Objects

Example

See Code disclaimer information for information pertaining to code examples.

For examples that pertain to the SNMP manager APIs, see Using SNMP Manager APIs—Example

API introduced: V3R6

snmpSet()—Set MIB Objects

Syntax

```c
#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
Threads safe: 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:

```c
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 “Problem Determination for SNMP Manager APIs” in the Simple Network Management Protocol book.

**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). 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 iSeries server 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 pub11c, 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)
- "snmpGet()—Retrieve MIB Objects” on page 35—Retrieve MIB Objects
- "snmpGetnext()—Retrieve Next MIB Object” on page 39—Retrieve Next MIB Object

Example

See Code disclaimer information for information pertaining to code examples.

For examples that pertain to the SNMP manager APIs, see Using SNMP Manager APIs—Example

API introduced: V3R6

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

<table>
<thead>
<tr>
<th>Sockets Error (value of <em>errno</em>)</th>
<th>Possible Causes</th>
</tr>
</thead>
</table>
| 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]
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:
Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, 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.

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:

(C) IBM 2006. Portions of this code are derived from IBM Corp. Sample Programs. (C) Copyright IBM Corp. 1998, 2006. All rights reserved.

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

### Programming Interface Information

This Application Programming Interfaces (API) 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 Printing
Advanced Peer-to-Peer Networking
AFP
AIX
AS/400
COBOL/400
CUA
DB2
DB2 Universal Database
Distributed Relational Database Architecture
Domino
DPI
DRDA
eServer
GDDM
IBM
Integrated Language Environment
Intelligent Printer Data Stream
IPDS
i5/OS
iSeries
Lotus Notes
MVS
Netfinity
Net.Data
NetView
Notes
OfficeVision
Operating System/2
Operating System/400
OS/2
OS/400
PartnerWorld
PowerPC
PrintManager
Print Services Facility
RISC System/6000
RPG/400
RS/6000
SAA
SecureWay
System/36
System/370
System/38
System/390
VisualAge
WebSphere
xSeries

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, and 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.