AN1109

An SNMP Agent Using BSD Socket API

Author: *Sean Justice*
Microchip Technology Inc.

INTRODUCTION

Simple Network Management Protocol (SNMP), an Internet protocol, was designed to manage network devices: file servers, hubs, routers, etc. However, it is also useful to manage and control embedded systems that are connected on an IP network. These systems can communicate through SNMP to transfer control and status information, thereby creating a truly distributed system. Unlike more familiar human-oriented protocols like HTTP, SNMP is considered a machine-to-machine protocol.

This Microchip SNMP agent application note and the included FAT16 module, supplemented by the TCP/IP application note AN1108, "*Microchip TCP/IP Stack with BSD Socket API*", provide an SNMP agent that can be integrated with almost any application on a Microchip 32-bit microcontroller product.

The TCP/IP application note and the FAT16 module are required to compile and run the SNMP agent module. All notes and files mentioned in this document are available for download from www.microchip.com.

The software in the installation files includes a sample application that demonstrates all of the features offered by this SNMP agent module.

Questions and answers about the SNMP agent module are provided at the end of this document in **"Answers to Common Questions"** on page 44.

ASSUMPTION

The author assumes that the reader is familiar with the following Microchip development tools: MPLAB® IDE and MPLAB® REAL ICE™ in-circuit emulator. It is also assumed that the reader is familiar with C programming language, as well as TCP/IP stack, FAT16 file system, and Management Information Base (MIB) Script concepts. Terminology from these technologies is used in this document, and only brief overviews of the concepts are provided. Advanced users are encouraged to read the associated specifications.

FEATURES

This application note provides one of the main components of an SNMP management system, the SNMP agent that runs on the managed device.

The simple agent presented here incorporates the following features:

- Provides portability across the 32-bit family of PIC® microcontrollers
- SNMP agent APIs (Application Program Interfaces) are compatible with PIC18/24 SNMP agent APIs
- Functions independently of RTOS or application
- Supports Microchip's MPLAB® C32 C Compiler
- Supports SNMP version 1 over UDP
- Supports `Get`, `Get-Next`, `Set` and `Trap` PDUs
- Automatically handles access to constant OIDs
- Supports up to 255 dynamic OIDs and unlimited constant OIDs
- Supports sequence variables with 7-bit index
- Supports enterprise-specific `Trap` with one variable information
- Uses an MIB that can be stored using FAT16
- Includes a PC-based MIB compiler
- Does not contain built-in TCP/UDP/IP statistics counters (user application must define and manage the required MIB)

This document offers discussion of the SNMP protocol sufficient to explain implementation and design of the SNMP agent. Refer to specification RFC 1157 and related documents for more detailed information about the protocol. The TCP/IP stack and its accompanying software modules, along with the software and hardware for using FAT16, are prerequisites for creating the SNMP agent.

LIMITATIONS

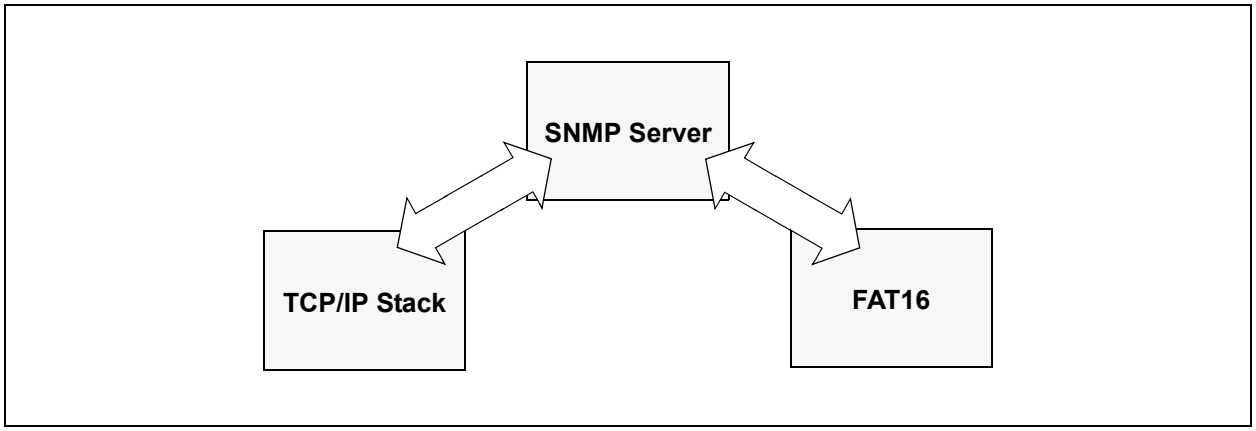
The SNMP agent is developed on an embedded system, so the standard ASCII format for the MIB file is reduced to a binary file (BIB). This increases the speed, because parsing a binary file is accomplished much more quickly than parsing an ASCII file.

This application note implements SNMP agent specifications, version 1. Support for version 2, or higher, would require code changes.

TYPICAL HARDWARE

A typical SNMP agent application requires the use of TCP/IP (AN1108, "*Microchip TCP/IP Stack with BSD Socket API*") and FAT16 hardware and software. The SNMP agent demo runs on an Explorer 16 board that has the appropriate TCP/IP connection (Microchip Part Number AC164123) and a FAT16-type media storage device (Microchip Part Number AC164122). The Explorer 16 board must be populated with two PICtail™ Plus connectors in J5 and J6. Refer to Explorer 16 documentation for more information.

FIGURE 1: SNMP SERVER DEPENDENCIES



RESOURCE REQUIREMENTS

Program memory required by the SNMP agent is stated in the following table.

TABLE 1: MEMORY REQUIREMENTS

Resource	Memory
`SNMP.c` module	13,800 bytes
RAM memory required by the SNMP module	30 bytes

Note: The minimum stack size for this module is 3072 bytes. This requirement can be decreased by defining `_SNMP_USE_DYMANIC_MEMORY` in `SNMP.h` to allocate all UDP packets on the heap, which should be at least 3072 bytes.

The compiler used for the memory requirements was the Microchip MPLAB® C32 C Compiler, version 1.00. The optimization was not used. Note that the use of compilers and optimization settings may increase or decrease the memory requirements.

INSTALLING SOURCE FILES

The complete source code for the Microchip SNMP Agent is available for download from the Microchip web site (see **Appendix A: "Source Code for the SNMP Agent"** on page 46).

The source code is distributed in a single Windows® installation file:
`pic32mx_bsd_tcp_ip_v1_00_00.exe`.

Perform the following steps to complete the installation:

1. Execute the file. A Windows installation wizard will guide you through the installation process.
2. Click **I Accept** to consent to the software license agreement.
3. After the installation process is completed, the **SNMP Agent Using BSD Socket API** item is available under the **Microchip** program group.

   The complete source files are copied to the following directory, in your choice of installation path:

   `\pic32_solutions\microchip\bsd_tcp_ip\source\bsd_snmp_agent`

   The "include" files are copied to the following directory:
   `\pic32_solutions\microchip\include\bsd_tcp_ip\`

   The demonstration application for the BSD SNMP agent is located in the following directory:
   `\pic32_solutions\bsd_snmp_agent_demo`
4. For the latest version-specific features and limitations, refer to the version HTML page, which can be accessed through `index.html`.

SOURCE FILE ORGANIZATION

The SNMP agent server consists of multiple files. These files are organized in multiple directories.

Table 2 shows the directory structure.

Table 3 on page 4 lists the server-related source files.

TABLE 2: SOURCE FILE DIRECTORY STRUCTURE

Directory	Description
\pic32_solutions\microchip\ bsd_tcp_ip\source\bsd_snmp_agent	SNMP agent source code and documentation
\pic32_solutions\microchip\include\bsd_tcp_ip\	SNMP agent include files
\pic32_solutions\bsd_snmp_agent_demo	SNMP agent project and demo related source files
\pic32_solutions\microchip\bsd_tcp_ip\source	TCP/IP source files
\pic32_solutions\microchip\include\bsd_tcp_ip\ templates	SNMP Agent and TCP/IP template header files
\pic32_solutions\microchip\bsd_tcp_ip\templates	SNMP Agent template source files
\pic32_solutions\microchip\fat16\source	File I/O source files
\pic32_solutions\microchip\include\fat16	File I/O header files
\pic32_solutions\microchip\include\fat16\template	File I/O template header files

TABLE 3: SOURCE FILES

File	Directory	Description
bsd_snmp_agent_demo.mcp	\pic32_solutions\bsd_snmp_agent_demo	MPLAB® REAL ICE™ in-circuit emulator SNMP agent demo project file
bsd_snmp_agent_demo.mcw	\pic32_solutions\bsd_snmp_agent_demo	MPLAB® REAL ICE™ in-circuit emulator SNMP agent demo workspace file
main.c	\pic32_solutions\bsd_snmp_agent_demo\source	Main demo file
snmpex.c	\pic32_solutions\bsd_snmp_agent_demo\source	User-modifiable SNMP agent source file
snmpex.h	\pic32_solutions\bsd_snmp_agent_demo\source	User-modifiable SNMP agent include file
snmpex_private.h	\pic32_solutions\bsd_snmp_agent_demo\source	User-modifiable SNMP agent include file
eTCP.def	\pic32_solutions\bsd_snmp_agent_demo\source	User-modifiable FAT16 defines
fat.def	\pic32_solutions\bsd_snmp_agent_demo\source	User-modifiable HTTP defines
mib2bib.exe	\pic32_solutions\microchip\ bsd_tsp_ip\tools\bsd_snmp_agent	Application to convert MIB file to BIB file
snmp.c	\pic32_solutions\microchip\ bsd_tcp_ip\source\bsd_snmp_agent	SNMP agent source file
snmp_private.h	\pic32_solutions\micro-chip\bsd_tcp_ip\source\bsd_snmp_agent	SNMP agent private include file
snmpex.tmpl	\pic32_solutions\microchip\bsd_tcp_ip\ template	User-modifiable SNMP agent source file template
snmp.h	\pic32_solutions\microchip\include\ bsd_tcp_ip\	SNMP agent include file
snmpex.tmpl	\pic32_solutions\microchip\include\ bsd_tcp_ip\templates	User-modifiable SNMP agent include file template
snmpex_private.tmpl	\pic32_solutions\microchip\include\ bsd_tcp_ip\templates	User-modifiable SNMP agent include file template

DEMO APPLICATION

The Microchip SNMP Agent includes a complete working application to demonstrate the SNMP agent running on the Microchip BSD TCP/IP stack. This application is designed to run on Microchip's Explorer 16 demonstration board. However, it can be easily modified to support any board.

Programming the Demo Application

If you need more information about SNMP, a more extensive overview is presented on page 9.

If you are already familiar with SNMP and the Microchip stack, the following information describes the process of incorporating an SNMP agent into an application.

The flowchart in Figure 2 outlines the general steps for developing a Microchip SNMP Agent. There are two main processes involved, developing the MIB, and using the MIB to develop the actual agent. Each process has several steps. Each process is explained later in this document.

The major steps are:

1. Downloading and installing the accompanying source files for the SNMP agent.
2. Using the MIB script (page 30) to define your private MIB, along with other standard MIB that your application may require.
3. Using the included MIB compiler, `mib2bib` (on page 37), to build a binary MIB image (BIB).
4. Placing the BIB file into the FAT16 storage media device (i.e., an SD card).

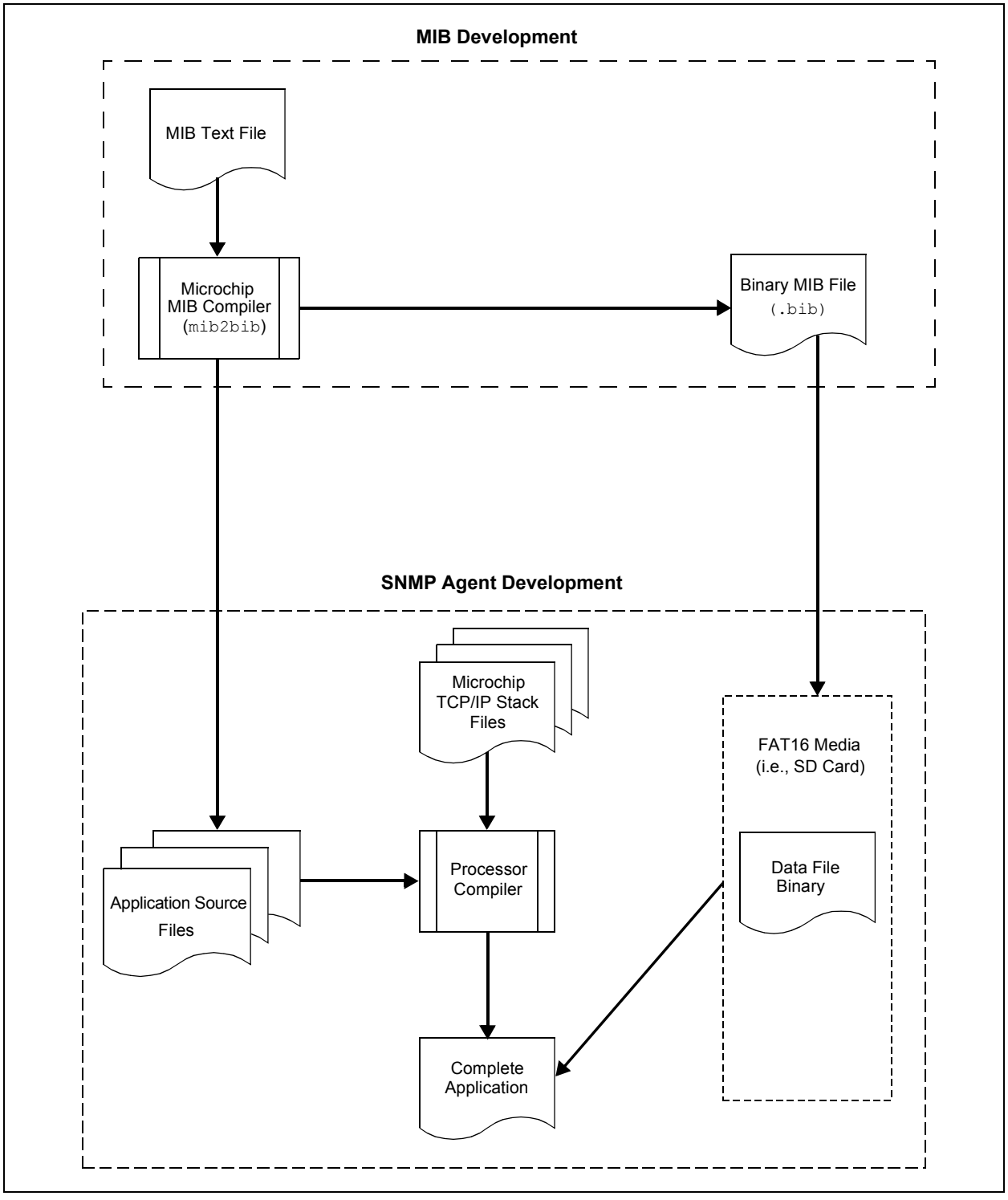
To program a target board with the demo application, you must have access to a PIC microcontroller programmer. The following procedure assumes that you will be using MPLAB REAL ICE in-circuit emulator as a programmer. If not, refer to the instructions for your specific programmer.

1. Connect MPLAB REAL ICE to the Explorer 16 board or to your target board.
2. Apply power to the target board.
3. Launch the MPLAB IDE.
4. Select the PIC device of your choice (this step is required only if you are importing a hex file that was previously built).
5. Enable the MPLAB REAL ICE in-circuit emulator as your programming tool.
6. If you want to use the previously-built hex file, simply import the following hex file:

   `bsd_snmp_agent_demo\release\bsd_snmp_agent_demo.hex`
7. If you are rebuilding the hex file, open the project file:
   `bsd_snmp_agent_demo\bsd_snmp_agent_demo.mcp`, and follow the build procedure to create the application hex file.
8. The demo application contains necessary configuration options required for the Explorer 16 board. If you are programming another type of board, make sure that you select the appropriate oscillator mode from the MPLAB REAL ICE configuration settings menu.
9. Select the Program menu option from the MPLAB REAL ICE in-circuit emulator menu to begin programming the target.
10. After a few seconds, you should see the message "Programming successful". If not, check your board and your MPLAB REAL ICE connection. Click Help on the menu bar for further assistance.
11. Remove power from the board and disconnect the MPLAB REAL ICE cable from the target board.

When successfully built, you can use any standard SNMP management software to access your SNMP agent device.

FIGURE 2: OVERVIEW OF THE SNMP AGENT DEVELOPMENT PROCESS



Setting Demo Application Hardware

In order to run the SNMP demo correctly, you must set up the hardware on the Explorer 16 board to use the TCP/IP stack and FAT16. Refer to AN1108, "*Microchip TCP/IP Stack with BSD Socket API*", for the proper hardware setup.

The demo requires that the TCP/IP connection (Microchip Part Number AC164123) uses SPI 1 and the FAT16-type media storage device (Microchip Part Number AC164122) uses SPI 2.

Executing the Demo Application

When the programmed microcontroller is installed on the Explorer 16 demo board and powered up, the LCD display shows the following information:

PIC32 BSD SNMP

<*Current IP address*>

BUILDING THE DEMO SNMP AGENT

The demo SNMP agent application included in this application note can be built using Microchip's 32-bit MPLAB C32 C Compiler. However, you can port the source to whichever compiler that you routinely use with Microchip microcontroller products.

The demo application also includes the following predefined SNMP agent project file:
`bsd_snmp_agent_demo.mcp`. The file is used with the Microchip MPLAB IDE. The project was created using a PIC32 device. If you are using a different device, you must select the appropriate device by using the MPLAB IDE menu command. In addition, the demo application project uses additional "include" paths as defined in the **Build Options** of MPLAB IDE:

```
.\source
..\microchip\include
```

Table 4 on page 8 lists the source files needed to build the demo SNMP agent application, and their respective locations.

The following instructions describe a high-level procedure for building the demo application. This procedure assumes that you are familiar with MPLAB IDE and will be using MPLAB IDE to build the application. If not, refer to the instructions of the development environment you are using to create and build the project.

1. Make sure that source files for the Microchip SNMP Agent are installed. If not, refer to **"Installing Source Files"**on page 3.
2. Launch MPLAB IDE and open the `bsd_snmp_agent_demo.mcp` project file.
3. Use the appropriate MPLAB IDE menu commands to build the project. Note that the demo project was created to compile properly when the source files are located in the directory structure that is suggested by the installation wizard. If you installed the source files to other locations, you must recreate or modify existing project settings to accomplish the build.
4. The build process should finish successfully. If not, make sure that your MPLAB IDE and compiler are set up correctly.

TABLE 4: DEMO SNMP AGENT APPLICATION PROJECT FILES

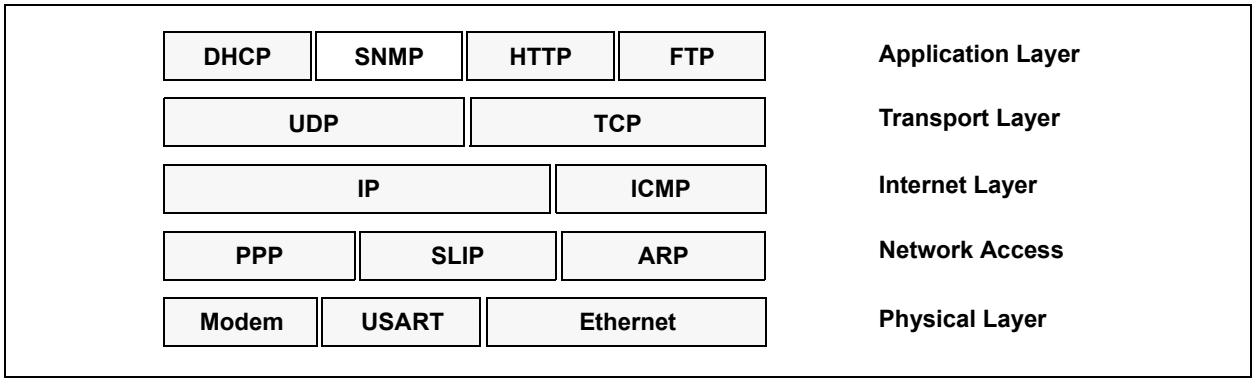
File	Location
main.c	\pic32_solutions\bsd_snmp_agent_demo\source
snmpex.c	\pic32_solutions\bsd_snmp_agent_demo\source
eTCP.def	\pic32_solutions\bsd_snmp_agent_demo\source
fat.def	\pic32_solutions\bsd_snmp_agent_demo\source
snmp.c	\pic32_solutions\microchip\bsd_tcp_ip\source\bsd_snmp_agent
block_mdr.c	\pic32_solutions\microchip\bsd_tcp_ip\source
earp.c	\pic32_solutions\microchip\bsd_tcp_ip\source
eicmp.c	\pic32_solutions\microchip\bsd_tcp_ip\source
eip.c	\pic32_solutions\microchip\bsd_tcp_ip\source
ENC28J60.c	\pic32_solutions\microchip\bsd_tcp_ip\source
etcp.c	\pic32_solutions\microchip\bsd_tcp_ip\source
ether.c	\pic32_solutions\microchip\bsd_tcp_ip\source
eudp.c	\pic32_solutions\microchip\bsd_tcp_ip\source
gpfunc.c	\pic32_solutions\microchip\bsd_tcp_ip\source
pkt_queue.c	\pic32_solutions\microchip\bsd_tcp_ip\source
route.c	\pic32_solutions\microchip\bsd_tcp_ip\source
socket.c	\pic32_solutions\microchip\bsd_tcp_ip\source
tick.c	\pic32_solutions\microchip\bsd_tcp_ip\source
fat.c	\pic32_solutions\microchip\fat16\source
fileio.c	\pic32_solutions\microchip\fat16\source
mediasd.c	\pic32_solutions\microchip\fat16\source
mstimer.c	\pic32_solutions\microchip\common
exlcd.c	\pic32_solutions\microchip\common

SNMP OVERVIEW

SNMP is an application-layer communication protocol that defines a client-server relationship. Its relationship to the TCP/IP protocol stack is shown in Figure 3.

SNMP describes a standard method to access variables residing in a remote device. It also specifies the format in which this data must be transferred and interpreted. When a device is SNMP enabled, any SNMP compatible host system can monitor and control that device.

FIGURE 3: LOCATION OF SNMP IN THE TCP/IP PROTOCOL STACK



SNMP Terminology

This application note frequently uses terminology described by the SNMP specification which we will review here briefly. Figure 4 on page 10 shows the typical SNMP model and the associated terminology.

NETWORK MANAGEMENT STATION

The Network Management Station (NMS) is half of the SNMP client-server setup; the other half is the agent. Because the focus of this document is on the agent, the NMS is mentioned here solely to be thorough.

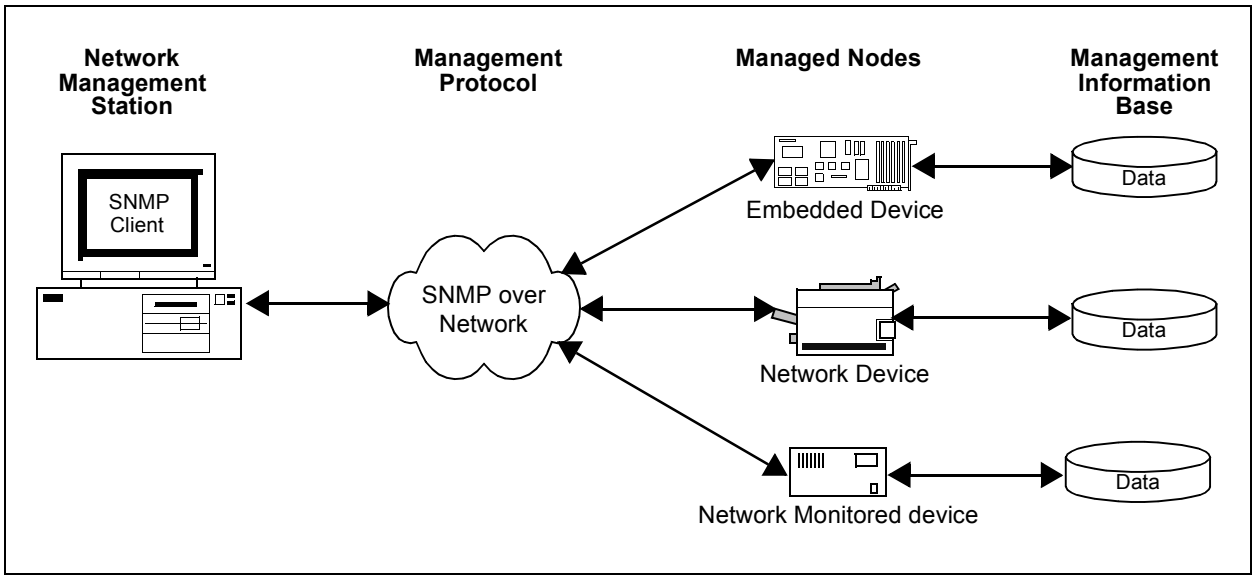
Typically, the NMS is on a personal computer running special software, although it could very well be any embedded device. NMS acts as an SNMP client, periodically polling the SNMP agent for data. NMS can be used to monitor a collection of similar or dissimilar devices.

When a device is SNMP-enabled, any NMS software available commercially or otherwise can be used. One of the advantages of the PC-based NMS systems that are available commercially is that many of them provide graphical representations of the managed devices. Also, these systems may allow the addition of devices to a network without requiring changes in the NMS software; they dynamically load information about devices that are added, as well as providing the option of manage those devices. These features give SNMP the functionality that makes it a popular choice for network and device management.

MANAGED NODE OR SNMP AGENT

A managed node (SNMP agent) is the device that is being managed by the NMS. SNMP agent implements the server portion of the SNMP protocol, acting as the agent between the device application and the NMS software. The relationship is not necessarily one-to-one, as a single agent can simultaneously serve data to many NMSs. The agent waits for NMS requests and responds with appropriate information.

FIGURE 4: OVERVIEW OF THE SNMP MODEL



MANAGEMENT INFORMATION BASE (MIB)

Each SNMP agent manages its own special collection of variables, called a Management Information Base (MIB). To organize the MIB, SNMP defines a schema known as the Structure of Management Information (SMI).

Figure 5 shows a generic SMI. The MIB is structured in a tree-like fashion, with one root at the top of the tree and one or more children below the root. Each child may contain one or more children of its own, thus creating an entire tree. The bottom-most nodes that do not have any children are called leaf nodes. These nodes contain the actual data.

SNMP and other RFC documents for the Internet define several MIBs. Figure 6 shows a subtree of the actual MIB for the Internet. Subtrees, such as “system”, “UDP”, and “TCP”, are standard MIBs that are defined by specific RFC documents. These and other standard MIBs should not be modified if the SNMP agent needs to be compatible with other NMS software.

A special subtree, called “enterprise”, is defined for private enterprises. Any SNMP agent device manufacturer may obtain its own private enterprise number. When assigned, the manufacturer may add or remove any number of subtrees beneath it as they may require. Private enterprise numbers may be obtained by applying to IANA (Internet Assigned Number Authority). Applications can be made at their web site, www.iana.org/cgi-bin/enterprise.pl.

OBJECT IDENTIFIER (OID)

Each node in the MIB tree is identified by a sequence of decimal numbers called an *Object Identifier* (OID). A specific node is uniquely referenced by its own OID and that of its parents’ OIDs. An OID is written in “dotted-decimal” notation, similar to those used by IP addresses (but not limited to four levels). For example, the OID for the *system* node in Figure 6 is written as ‘1.3.6.1.2.1’. For the convenience of readers, an OID is frequently written with each node name and its OID in parenthesis. Using this convention, the OID for the system node can be rewritten as “iso(1).org(3).dod(6).internet(1).mgmt(2).mib(1)”.

By virtue of OID assignments, the first number is always either ‘1’ or ‘2’, and the second number is less than 40. The first two numbers, *a* and *b*, are encoded as one byte having the value 40*a* + *b*. For the Internet, this number is 43. As a result, the system OID is transmitted as ‘43.6.1.2.1’, *not* ‘1.3.6.1.2.1’.

Note:	The Microchip SNMP MIB script that is discussed later in this document requires that all SNMP OIDs start with ‘43’.

FIGURE 5: GENERIC STRUCTURE OF MANAGEMENT INFORMATION (SMI)

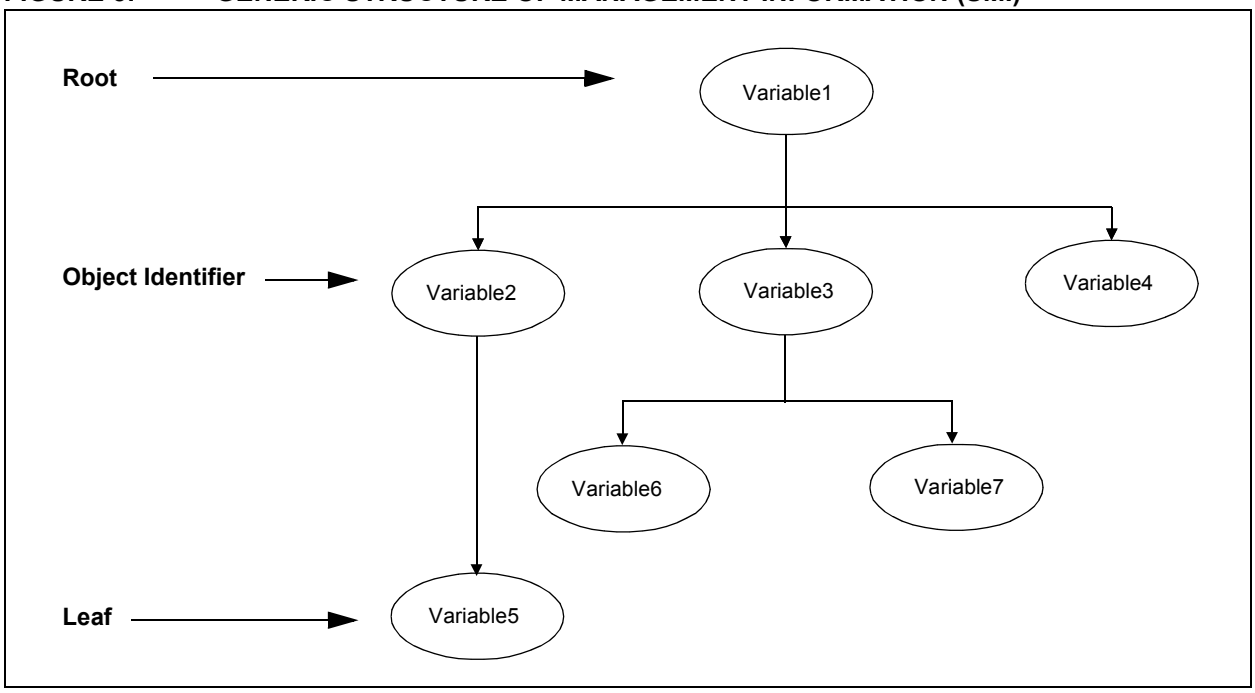
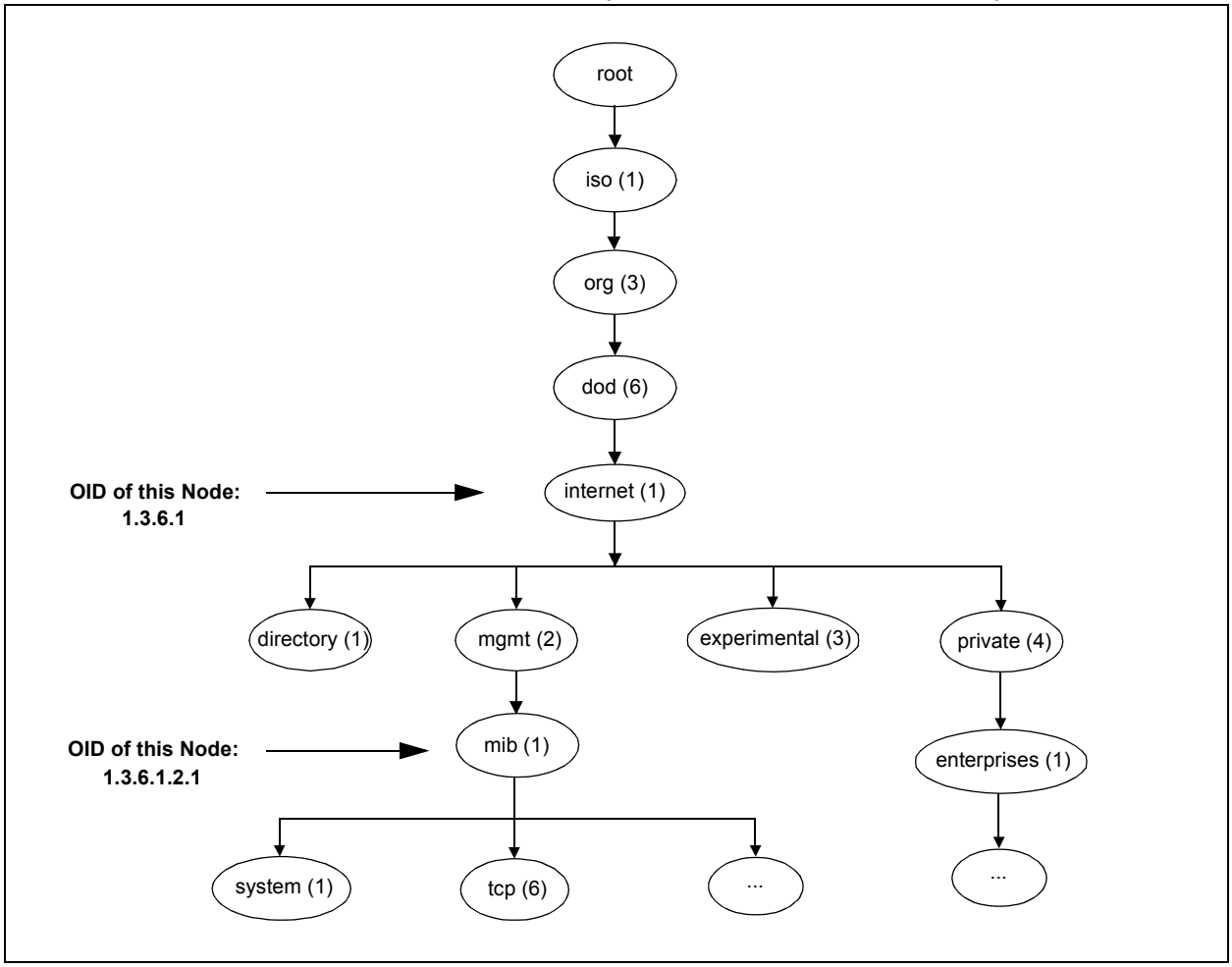


FIGURE 6: EXAMPLE OF AN ACTUAL SMI (PARTIAL INTERNET SUBTREE)



Abstract Syntax Notation (ASN) Language

Each MIB variable contains several attributes, such as data type, access type, and object identifier. SNMP uses special language called Abstract Syntax Notation version 1 (ASN.1) to describe detail about variables. ASN.1 is also used to describe SNMP and other protocol data exchange formats. ASN.1 is written as a text file and compiled using an ASN syntax compiler. Most NMS and SNMP agent software are designed to read ASN files and build MIB accordingly. An example of a variable description in ASN.1 syntax is shown in Example 1.

There are commercially available MIB builders that allow users to build MIBs graphically without needing to learn ASN syntax. The Microchip SNMP Agent uses its own special script to describe its agent OIDs. It also uses its own script compiler to create compact binary representations of the MIB. The custom script also allows the assignment of constant data to OIDs. The Microchip MIB script and its compiler are described in greater detail on page 30.

EXAMPLE 1: TYPICAL ASN.1 DESCRIPTION OF A VARIABLE

```
org       OBJECT IDENTIFIER ::= { iso 3 }
dod       OBJECT IDENTIFIER ::= { org 6 }
internet  OBJECT IDENTIFIER ::= { dod 1 }
.
.
.
update OBJECT-TYPE
    SYNTAX SEQUENCE OF UdpEntry
    ACCESS not-accessible
    STATUSmandatory
    DESCRIPTION
        "A table containing..."
    ::= { udp 5 }
```

Binary Encoding Rules (BER)

SNMP uses ASN.1 syntax to describe its packet and variable contents. ASN is an abstract syntax; that is, it does not specify how the actual data is encoded and transmitted between two nodes. A special set of rules, called Binary Encoding Rules (BER), is used to encode what is described by the ASN.1 syntax. BER is self-contained and platform independent. Each data item encoded with BER contains its data type, data length, and its actual value; this is in contrast to regular data, in which only the data content is given.

A data variable encoded by BER consists of a “tag byte”, one or more “length bytes” and one or more “value bytes”. The tag byte describes the data type associated with the current data variable. The length byte(s) gives the number of bytes used to describe data content. The value bytes are the actual data content. Figure 7 shows the breakdown of typical BER values and an example of encoding. An example of typical BER encoding is provided in Figure 8.

It is not necessary for you to learn the encoding rules. The SNMP agent automatically handles encoding and decoding of all supported data types.

FIGURE 7: GENERIC BER FORMAT

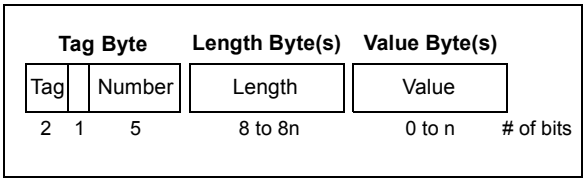
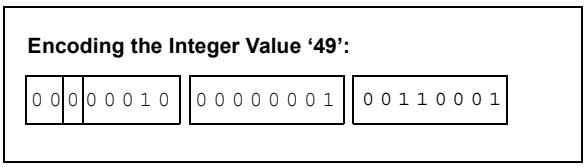


FIGURE 8: EXAMPLE OF BER ENCODING



Protocol Data Unit (PDU)

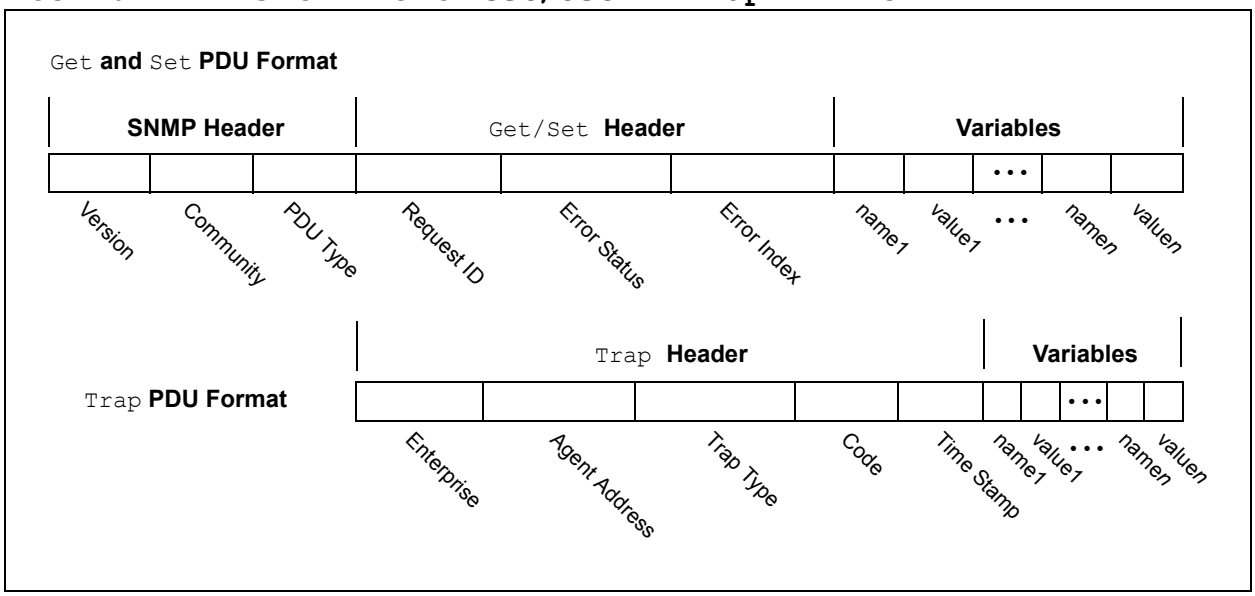
Data packets exchanged between two SNMP nodes are called Protocol Data Units (PDU). SNMP version 1 defines a total of five main types of PDUs:

- Get-request
- Get-Next-response
- Get-response
- Set-request
- Trap

All Get and Set PDUs share a common message format, while the format of Trap PDUs is somewhat different. The two formats are compared in Figure 9.

You do not need to know the details of the PDU format or its encoding to use the Microchip SNMP Agent. The SNMP agent module automatically handles all of the low-level protocol details, including the encoding and decoding of data variables. If you are interested in learning details, refer to specification RFC 1157 for information about individual PDU fields.

FIGURE 9: PDU FORMATS FOR Get/Set AND Trap PACKETS



MICROCHIP SNMP AGENT APIs

The SNMP agent is implemented by several files working together with the Microchip BSD TCP/IP Stack. Like the other components of the stack, the core of the SNMP agent is implemented by a single file, `snmp.c`. In addition, at least five other callback functions must be implemented to provide communication between the SNMP module, the host application, and the rest of the TCP/IP stack.

The SNMP agent also makes use of APIs. These are well-defined methods for communicating between applications and the SNMP agent, and are also designed to make application design easier for the user.

There are a total of 10 functions associated with the SNMP agent. A complete description of the APIs follows through page 29.

SNMPInit

`SNMPInit` is used to initialize the SNMP agent module.

Syntax

```
void SNMPInit(void)
```

Parameters

None

Return Values

None

Precondition

There must be at least one free UDP socket available, and the TCP/IP has been initialized.

Side Effects

One UDP socket will be used.

Remarks

None

Example

```
// Do Stack manager Init.
TCPIPSetDefaultAddr();
InitializeBoard();
InitStackMgr();
TickInit();

// Initialize SNMP module
SNMPInit();

// Initialize other modules...
...
```

SNMPTask

SNMPTask is the main state machine task. It handles all incoming SNMP packets, processes them for correct operation and calls back the main application.

Syntax

```
BOOL SNMPTask(void)
```

Parameters

None

Return Values

TRUE, if SNMP state machine has completed its task; the stack state machine can be changed.

FALSE, if otherwise.

Precondition

SNMPInit() has been called.

Side Effects

An incoming SNMP packet is processed and acted on. Packets are discarded after being processed.

Remarks

None

Example

```
// Do Stack manager Init.
TCPIPSetDefaultAddr();
InitializeBoard();
InitStackMgr();
TickInit();

// Initialize SNMP module
SNMPInit();

// Initialize other modules...
...

// Enter into main loop
while( 1 )
{
    // Main Microchip TCP/IP Stack task
    StackMgrProcess();

    // Call SNMP Task
    SNMPTask();

    // Call another Stack tasks...
    ...
}
```

SNMPGetVar

SNMPGetVar is a callback used by the SNMP agent module to request a variable value from the main application. If the current OID is a simple variable, index will always be '0'. If the current OID is a sequence variable, index may be any value from '0' through '127'.

Syntax

```
BOOL SNMPGetVar(SNMP_ID var, SNMP_INDEX index, BYTE *ref, SNMP_VAL *val)
```

Parameters

var [in]

OID variable ID whose value is requested.

index [in]

Index of OID variable. index is useful when the OID variable is of type sequence and NMS can query any of the available values.

ref [in/out]

Reference for multi-byte Get. ref is set to SNMP_START_OF_VAR (value of 0x00) to mark the beginning of a data transfer. The application may read and set this parameter to keep track of a multi-byte transfer. When the multi-byte data transfer is complete, the application must set ref to SNMP_END_OF_VAR.

val [out]

Pointer to a buffer of up to 4 bytes, depending on the data type of var:

If data type is BYTE, the application should copy the value in val->byte

If data type is WORD, the application should copy the value in val->word

If data type is DWORD, the application should copy the value in val->dword

If data type is IP_ADDRESS, the application may copy the value in either val->dword or val->v[], with the LSB being the MSB of the IP address

If data type is COUNTER32, TIME_TICKS or GAUGE32, the application should copy the value in val->dword

If data type is ASCII_STRING or OCTET_STRING, the application should copy the value in val->byte, one byte at a time. In this case, ref may be used to keep track of the multi-byte transfer.

Return Values

TRUE, if a value exists for a given var at given index; data is copied in val.

FALSE, if otherwise.

Precondition

None

Side Effects

None

Remarks

For a definition of the data types associated with val, refer to the DeclareVar description on page 31.

SNMPGetVar (Continued)

Example

```
BOOL SNMPGetVar(SNMP_ID var, SNMP_INDEX index, BYTE *ref, SNMP_VAL* val)
{
    BYTE myRef;
    myRef = *ref;

    switch(var)
    {
    case LED_D5:     // LED D5 control variable.
        val->byte = LED_D5_CONTROL;// Return LED D5 value
        return TRUE;

    case ANALOG_POT0:// 10-bit value of ADC
        val->word = AN0Value.Val;
        return TRUE;

    case TRAP_COMMUNITY:// ASCII_STRING variables
        // Make sure that given index is within our range.
        // TRAP_COMMUNITY is part of larger table trapInfo
        if ( index < trapInfo.Size )
        {
            // If it is empty string, this is the end.
            if ( trapInfo.table[index].communityLen == 0 )
                *ref = SNMP_END_OF_VAR;
            else
            {
                val->byte = trapInfo.table[index].community[myRef];

                // Prepare for next byte transfer
                    myRef++;
                // If we transferred all bytes, mark it as an end
                if ( myRef == trapInfo.table[index].communityLen )
                    *ref = SNMP_END_OF_VAR;
                else
                    // Or else, set ref to track it.
                    *ref = myRef;
            }
        }
        return TRUE;
    }...

    // All unknown variables are cannot be retrieved.


    return FALSE;
}
```

SNMPGetNextIndex

SNMPGetNextIndex is a callback used by the SNMP agent module to request next index after given index (if there is one).

Syntax

```
BOOL SNMPGetNextIndex(SNMP_ID var, SNMP_INDEX *index)
```

Parameters

var [in]

OID variable ID whose next index value is requested. Only var of type sequence is called with this function.

index [in/out]

Pointer to index of OID variable. The application should read the value pointed to by this pointer and update its content with the next available index, if there is one. If there is none, there is no need to modify its content.

INDEX_INVALID if no index is given. In that case, the next index is the very first available index.

Return Values

TRUE, if next index exists after given index.

FALSE, if otherwise.

Precondition

None

Side Effects

None

Remarks

SNMPGetNextIndex is only called for sequence index variables. The application needs to handle only index type variables in this callback.

Example

```
BOOL SNMPGetNextIndex(SNMP_ID var, SNMP_INDEX *index)
{
    SNMP_INDEX tempIndex;
    tempIndex = *index;

    switch(var)
    {
    case TRAP_RECEIVER_ID:
        // There is no next possible index if table itself is empty.
        if ( trapInfo.Size == 0 )
            return FALSE;
        // INDEX_INVALID means start with first index.
        if ( tempIndex == SNMP_INDEX_INVALID )
        {
            *index = 0;
            return TRUE;
        }
        // Next index is one more than current one but less than size of table.
        else if ( tempIndex < (trapInfo.Size-1) )
        {
            *index = tempIndex+1;
            return TRUE;
        }
        break;
    }
    return FALSE;
}
```

SNMPIsValidSetLen

SNMPIsValidSetLen is a callback used by the SNMP agent module to determine if a variable can be written with a specific length of value. When NMS performs a Set-request operation, it supplies the new value. The SNMP agent passes the length of this value to the application and confirms that the current variable can hold the given length of data. If data length is too long for the variable to handle, application returns FALSE and the SNMP agent fails the current request.

Syntax

```
BOOL SNMPIsValidSetLen(SNMP_ID var, BYTE len)
```

Parameters

var [in]

OID variable ID whose Set capability is to be checked.

len [in]

Length of Set-request data as issued by NMS.

Return Values

TRUE, if given variable var is designed to handle given length len of data.

FALSE, if otherwise.

Precondition

None

Side Effects

None

Remarks

SNMPIsValidSetLen is called for a dynamic OID with a READWRITE access attribute and ASCII_STRING or OCTET_STRING data types only. For a definition of the READWRITE access type, refer to the DeclareVar description on page 31.

Example

```
BOOL SNMPIsValidSetLen(SNMP_ID var, BYTE len)
{
    switch(var)
    {
    case TRAP_COMMUNITY:
        // Length must be less than our allocated memory.
        if ( len < MAX_COMMUNITY_LEN+1 )
            return TRUE;
        break;

    case LCD_DISPLAY:
        // Similarly LCD length must be less than LCD capability.
        if ( len < LCD_DISPLAY_LEN+1 )
            return TRUE;
        break;
    }
    return FALSE;
    }
```

SNMPSetVar

SNMPSetVar is a callback used by the SNMP agent module to modify a dynamic OID variable whose access type is READWRITE.

Syntax

```
BOOL SNMPSetVar(SNMP_ID var, SNMP_INDEX index, BYTE ref, SNMP_VAL val)
```

Parameters

var [in]

OID variable ID whose value needs to be modified.

index [in]

Index of OID variable var. If this is a simple variable, index will always be '0'. In other cases, application must validate given index before using it.

ref [in]

Reference to track multi-byte Set.

The very first Set callback will contain SNMP_START_OF_VAR (0x00) and subsequent callbacks will contain ascending ref values to indicate the index of the byte being transferred. After transfer is complete, the value of SNMP_END_OF_VAR will be passed to mark the end of transfer. The application should use this indication to update local flags and values.

val [in]

Pointer to data value of up to 4 bytes, depending on the data type of var:

If data type is BYTE, the variable value is in val.byte.

If data type is WORD, the variable value is in val.word.

If data type is DWORD, the variable value is in val.dword.

If data type is IP_ADDRESS, the variable value is in val.v[ ] or val.dword.

If data type is GAUGE32, TIME_TICKS or COUNTER32, the variable value is in val.dword.

If data type is ASCII_STRING or OCTET_STRING, one byte of variable value is in val.byte. A multi-byte transfer will be performed to transfer the entire data string.

Return Values

TRUE, if val is successfully written to the variable var.

FALSE, if otherwise.

Precondition

None

Side Effects

None

Remarks

SNMPSetVar is called for a dynamic OID with the READWRITE access attribute. In the case of ASCII_STRING and OCTET_STRING with more than one byte to Set, this function will be called multiple times to transfer up to 127 bytes of data.

If given variable is of type simple, index will always be '0'.

For a definition of the data types associated with val, refer to the DeclareVar description on page 31.

SNMPSetVar (Continued)

Example

```
BOOL SNMPSetVar(SNMP_ID var, SNMP_INDEX index, BYTE ref, SNMP_VAL val)
{
    switch(var)
    {
    case LED_D5:// D5 is 8-bit control variable.
        LED_D5_CONTROL = val->byte;
        return TRUE;

    case TRAP_RECEIVER_IP:// This is Sequence variable
        // Make sure that index is within our range.
         if ( index < trapInfo.Size )
         {
           // This is just an update to an existing entry.
           trapInfo.table[index].IPAddress.Val = val.dword;
           return TRUE;
         }
         else if ( index < TRAP_TABLE_SIZE )
         {
           // This is an addition to table.
           trapInfo.table[index].IPAddress.Val = val.dword;
           // Create other empty entries.
           trapInfo.table[index].communityLen = 0;

           // Update table size.
           trapInfo.Size++;
           return TRUE;
        }
        break;

    case LCD_DISPLAY:
        // Copy all bytes until all bytes are transferred
         if ( ref != SNMP_END_OF_VAR )
         {
           LCDDisplayString[ref] = val.byte;
           LCDDisplayStringLen++;
        }
         else
         {
           // Display it on the first line of the LCD
           XLCDGoto(0, 0);
           XLCDPutString(LCDDisplayString);
         }
    return TRUE;
    }

// All unknown variables cannot be Set.
return FALSE;
}
```

SNMPValidate

SNMPValidate is a callback used by the SNMP agent module to ask the application if the given community is a valid string for the given operation.

Syntax

```
BOOL SNMPValidate(SNMP_ACTION SNMPAction, char *community)
```

Parameters

SNMPAction [in]

SNMP action type. Possible values for this parameter are:

Value	Meaning
SNMP_GET	Get-request is being performed to fetch one or more variables
SNMP_SET	Set-request is being performed to set one or more variables

community [in]

Community string that was passed along with given action.

Return Values

TRUE, if the community is a allowed to perform a given operation.

FALSE, if otherwise.

Precondition

None

Side Effects

None

Remarks

None

Example

```
char PUBLIC_COMMUNITY[] = "public";
#define PUBLIC_COMMUNITY_LEN(sizeof(PUBLIC_COMMUNITY)-1)

char PRIVATE_COMMUNITY[] = "private";
#define PRIVATE_COMMUNITY_LEN(sizeof(PRIVATE_COMMUNITY)-1)

BOOL SNMPValidate(SNMP_ACTION SNMPAction, char* community)
{
    if ( memcmp(community, (ROM void*)PUBLIC_COMMUNITY,
       PUBLIC_COMMUNITY_LEN) )
    {
        if ( SNMPAction == SNMP_GET )
            return TRUE;
    }
    else if ( memcmp(community, (ROM void*)PRIVATE_COMMUNITY,
        PRIVATE_COMMUNITY_LEN) )
    {
        if ( SNMPAction == SNMP_SET )
            return TRUE;
    }
    return FALSE;
}
```

SNMPNotifyPrepare

SNMPNotifyPrepare is used by the application to prepare to send SNMP Trap to the remote host.

Syntax

```
void SNMPNotifyPrepare(IP_ADDR *remoteHost,
                       char *community,
                       BYTE communityLen,
                       SNMP_ID agentIDVar,
                       BYTE notificationCode,
                       DWORD timestamp);
```

Parameters

remoteHost [in]

Remote host IP address that needs to notified.

community [in]

Community string to use for this notification.

communityLen [in]

Length of community string.

agentIDVar [in]

OID ID that is already defined as Agent ID in Microchip MIB script.

notificationCode [in]

Notification code that is to be used in this notification, this is the "Trap Type".

timestamp [in]

Time stamp (10 ms resolution) at which this notification event occurred.

Return Values

None

Precondition

None

Side Effects

None

Remarks

SNMPNotifyPrepare is called at the beginning of a notification. With this function call, the application transfers notification information to the SNMP agent module. To complete notification, the application must also call SNMPNotifyIsRead() and SNMPNotify().

SNMPNotifyPrepare (Continued)

Example

```
// This function is wrapper to send a notification to remote NMS
// as stored in local trap table.

static BOOL SendNotification(BYTE receiverIndex,
               SNMP_ID var,
               SNMP_VAL val)
{
    static enum { SM_PREPARE, SM_NOTIFY_WAIT } smState = SM_PREPARE;
    IP_ADDR IPAddress;

    // Copy interested trap receiver IP address into local
    // variable - in network order.
    IPAddress.v[0] = trapInfo.table[receiverIndex].IPAddress.v[3];
    IPAddress.v[1] = trapInfo.table[receiverIndex].IPAddress.v[2];
    IPAddress.v[2] = trapInfo.table[receiverIndex].IPAddress.v[1];
    IPAddress.v[3] = trapInfo.table[receiverIndex].IPAddress.v[0];

    // Process to send notification must be written in co-operative
    // multi-tasking fashion.
    // Initial state prepares SNMP agent module by supplying
    // necessary information.
    switch(smState)
    {
    case SM_PREPARE:
         SNMPNotifyPrepare(&IPAddress,
                trapInfo.table[receiverIndex].community,
                trapInfo.table[receiverIndex].communityLen,
                MICROCHIP,          // Agent ID Var
                6,                  // Notification code
                TickGet() );        // Timestamp
        smState = SM_NOTIFY_WAIT;
        break;

    case SM_NOTIFY_WAIT:
        // When notify prepare is done,
        // wait for SNMP agent to be ready.
        if ( SNMPIsNotifyReady(&IPAddress) )
        {
            // When it is ready, supply interested variable.
            // In this version, only one variable
            // can be sent per notification.
            SNMPNotify(var, val, 0);
            return TRUE;
        }
    }
    return FALSE;
}
```

SNMPNotifyIsReady

SNMPNotifyIsReady is used by the application to check whether the SNMP agent is ready for a SNMPNotify() call.

Syntax

```
BOOL SNMPNotifyIsReady(IP_ADDR *remoteHost)
```

Parameters

remoteHost [in]

Remote host IP address that needs to notified.

Return Values

TRUE, if SNMP agent is ready for SNMPNotify().

FALSE, if otherwise. The application should maintain a time-out counter and abort calling this function if it does not return TRUE within the time-out value.

Precondition

SNMPNotifyPrepare() is already called.

Side Effects

None

Remarks

SNMPNotifyIsReady performs ARP resolution and obtains the MAC address for a given IP address. When ARP resolution is complete, it returns TRUE and the application is free to call SNMPNotify()to actually notify the host.

SNMPNotifyIsReady (Continued)

Example

```
// This function is wrapper to send a notification to remote NMS
// as stored in local trap table.
static BOOL SendNotification(BYTE receiverIndex,
               SNMP_ID var,
               SNMP_VAL val)
{
   static enum { SM_PREPARE, SM_NOTIFY_WAIT } smState = SM_PREPARE;
   IP_ADDR IPAddress;

   // Copy interested trap receiver IP address into local
   // variable - in network order.
   IPAddress.v[0] = trapInfo.table[receiverIndex].IPAddress.v[3];
   IPAddress.v[1] = trapInfo.table[receiverIndex].IPAddress.v[2];
   IPAddress.v[2] = trapInfo.table[receiverIndex].IPAddress.v[1];
   IPAddress.v[3] = trapInfo.table[receiverIndex].IPAddress.v[0];

   // Process to send notification must be written in co-operative
   // multi-tasking fashion.
   // Initial state prepares SNMP agent module by supplying
   // necessary information.
   switch(smState)
   {
   case SM_PREPARE:
      SNMPNotifyPrepare(&IPAddress,
               trapInfo.table[receiverIndex].community,
               trapInfo.table[receiverIndex].communityLen,
               MICROCHIP,         // Agent ID Var
               6,                 // Notification code
               TickGet());        // Timestamp
      smState = SM_NOTIFY_WAIT;
      break;

   case SM_NOTIFY_WAIT:
      // When notify prepare is done, wait for SNMP agent to be ready.
      if ( SNMPIsNotifyReady(&IPAddress) )
      {
         // When it is ready, supply interested variable.
         // In this version, only one variable
         // can be sent per notification.
         SNMPNotify(var, val, 0);
         return TRUE;
      }
   }
   return FALSE;
}
```

SNMPNotify

SNMPNotify is used by the application to transfer the variable that caused notification.

Syntax

```
BOOL SNMPNotify(SNMP_ID var, SNMP_VAL val, SNMP_INDEX index)
```

Parameters

var [in]

OID ID that is to be included in this notification.

val [in]

Value of var that is to be included in this notification.

index [in]

Index of OID ID that is to be included in this notification.

Return Values

TRUE, if remote host was successfully notified.

FALSE, if otherwise.

Precondition

```
SNMPIsNotifyReady() = TRUE
```

Side Effects

None

Remarks

SNMPNotify builds the SNMP Trap PDU and sends it to the previously-specified remote host.

Only variables of the data types BYTE, WORD, DWORD, IP_ADDRESS, COUNTER32, and GAUGE32 can be used in SNMPNotify; in other words, only variables of these data types can generate notification. In addition, these variables must be declared as dynamic.

SNMPNotify (Continued)

Example

```
// This function is wrapper to send a notification to remote NMS
// as stored in local trap table.
static BOOL SendNotification(BYTE receiverIndex,
               SNMP_ID var,
               SNMP_VAL val)
{
   static enum { SM_PREPARE, SM_NOTIFY_WAIT } smState = SM_PREPARE;
   IP_ADDR IPAddress;

   // Copy interested trap receiver IP address into local
   // variable - in network order
   IPAddress.v[0] = trapInfo.table[receiverIndex].IPAddress.v[3];
   IPAddress.v[1] = trapInfo.table[receiverIndex].IPAddress.v[2];
   IPAddress.v[2] = trapInfo.table[receiverIndex].IPAddress.v[1];
   IPAddress.v[3] = trapInfo.table[receiverIndex].IPAddress.v[0];

   // Process to send notification must be written in co-operative
   // multi-tasking fashion.
   // Initial state prepares SNMP agent module by supplying
   // necessary information.
   switch(smState)
   {
   case SM_PREPARE:
      SNMPNotifyPrepare(&IPAddress,
               trapInfo.table[receiverIndex].community,
               trapInfo.table[receiverIndex].communityLen,
               MICROCHIP,         // Agent ID Var
               6,                 // Notification code
               TickGet() );       // Timestamp
      smState = SM_NOTIFY_WAIT;
      break;

   case SM_NOTIFY_WAIT:
      // When notify prepare is done, wait for SNMP agent to be ready.
      if ( SNMPIsNotifyReady(&IPAddress) )
      {
         // When it is ready, supply interested variable. - In this
         // version, only one variable can be sent per notification.
         SNMPNotify(var, val, 0);
         return TRUE;
      }
   }
   return FALSE;
}
```

DESCRIBING THE MIB WITH MICROCHIP MIB SCRIPT

Microchip's SNMP Agent uses a custom script to describe the MIB. This script is designed to simplify the MIB definition and its integration with the main application. The actual MIB used by the SNMP agent is a binary image created by the Microchip MIB to BIB compiler (page 37).

Microchip MIB Script Commands

A Microchip MIB file is an ASCII text file consisting of multiple command lines. Each command line consists of a single command that begins with the dollar sign character ($), and one or more command parameters that are delimited with commas and enclosed in parentheses. Lines that do not start with a dollar sign are interpreted as comments and are not processed by the compiler. Commands must be written in a single line, they cannot span multiple lines.

The MIB script language includes a total of five commands, each with a specific syntax. Only one command, `DeclareVar`, is mandatory; the others are optional, depending on the application and the types of information to be defined. In practice, at least one other command will be used in defining an MIB. The syntax of the script commands is explained on pages 31 through 36.

Example 2 shows part of a typical Microchip MIB file. In this example, three separate items are being defined. In the first script "paragraph", a read-only node is being established at the OID of 43.6.1.2.1.1.5. It contains the identifier string "Microchip SNMP Agent" as static information.

In the second paragraph, a node with dynamic temperature information is being established at the OID of 43.6.1.4.1.17095.3.1. The variable called `TempAlarm` is assigned an identifier of '`1`'.

In the final paragraph, a two-column data array is being created with the variables `DigInputs` and `DigChannel`. The variables, themselves, are located in two separate nodes with neighboring OIDs. In addition, `DigChannel` is being used as the index for the array.

EXAMPLE 2: PARTIAL LISTING OF A MICROCHIP MIB (TEXT) FILE

```
$DeclareVar(sysName, ASCII_STRING, SINGLE, READONLY, 43.6.1.2.1.1.5)
$StaticVar(sysName, Microchip SNMP Agent)


$DeclareVar(TempAlarm, BYTE, SINGLE, READWRITE, 43.6.1.4.1.17095.3.1)
$DynamicVar(TempAlarm, 1)


$DeclareVar(DigInputs, BYTE, SEQUENCE, 43.6.1.4.1.17095.16.1.1)
$DeclareVar(DigChannel, BYTE, SEQUENCE, 43.6.1.4.1.17095.16.1.2)
$SequenceVar(DigInputs, DigChannel)
$SequenceVar(DigChannel, DigChannel)
```

DeclareVar

DeclareVar declares a single variable and all of its mandatory attributes.

Status

Mandatory

Syntax

```
$DeclareVar(oidName, dataType, oidType, accessType, oidString)
```

Parameters

oidName

Name of this OID variable. This name must be unique and must follow the ANSI 'C' naming convention; i.e., it must not start with a number and must not contain special characters ('&', '+', etc.). If this variable is declared to be dynamic, the MIB compiler will define a 'C' define symbol using the variable name in the header file mib.h. The main application includes this header file and refers to this OID using oidName.

dataType

Data type of this OID variable. Valid keywords are:

Keyword	Description
BYTE	8-bit data.
WORD	16-bit (2-byte) data.
DWORD	32-bit (4-byte) data.
IP_ADDRESS	4-byte IP address data.
COUNTER32	4-byte COUNTER32 data as defined by SNMP specification.
GAUGE32	4-byte GAUGE32 data as defined by SNMP specification.
OCTET_STRING	Up to 127 bytes of binary data bytes.
ASCII_STRING	Up to 127 bytes of ASCII data string.
OID	Up to 127 bytes of dotted-decimal OID string value. If any of the individual OID values are greater than 127, the total number of allowable OID bytes will be less than 127.

oidType

OID variable type. Valid keywords are:

Keyword	Description
SINGLE	If this variable contains single value.
SEQUENCE	If this variable contains array of values. All variables with an oidType of SEQUENCE must be assigned an "index" OID variable using the SequenceVar command.

AccessType

OID access type: Valid keywords are:

Keyword	Description
READONLY	If this variable can only be read.
READWRITE	If this variable can be read and written.

oidString

Full "dotted-decimal" string describing this variable. If this OID is part of the Internet MIB subtree, the first two OIDs, iso(1).org(3), must be written as decimal '43' (i.e., system OID will be written as '43.6.1.2.1.1').

The OID string for all OID variables must contain the same root, i.e., if the first OID variable is declared with 43 as a root node, all following variables must also contain 43 as a root node.

DeclareVar (Continued)

Result

If compiled successfully, this command will create a new OID variable. This variable can be used as an OID parameter in other commands, such as `StaticVar`, `DynamicVar`, or `SequenceVar`.

Precondition

None

Examples

This command declares an OID variable named `sysName` as defined in the standard MIB subtree system:

```
$DeclareVar(sysName, ASCII_STRING, SINGLE, READONLY, 43.6.1.2.1.1.5)
```

This command declares an OID variable of type `BYTE`:

```
$DeclareVar(LED_D5, BYTE, SINGLE, READWRITE, 43.6.1.4.1.17095.3.1)
```

StaticVar

StaticVar declares a previously defined OID variable as static (i.e., OID containing constant data) and assigns constant data to it.

Status

Optional; required only if the application needs to define static OID variables.

Syntax

```
$StaticVar(oidName, data, …)
```

Parameters

oidName

Name of OID variable that is being declared as a static. This oidName must have been declared by a previous DeclareVar command.

data

Actual constant data for oidName. This data will be interpreted using the data type defined in the DeclareVar command:

Data Type	Format Requirement
BYTE, WORD, or DWORD	Must be written in decimal notation.
IP_ADDRESS and OID	Must be written in appropriate dotted-decimal notation for data type.
ASCII_STRING	Must be free-form ASCII string with no quotes. Commas, parentheses and back-slashes must be preceded by the back-slash (\) as an escape character.
OCTET_STRING	Must be written in multiple individual bytes separated by commas.

Result

If compiled successfully, this command will declare given oidName as a static OID. A static OID contains constant data that is stored in the BIB. Static OIDs are automatically managed by the SNMP agent module; the application does not have to implement callback logic to provide data for this OID variable.

Precondition

The given oidName must have been declared using previous DeclarVar command.

Examples

StaticVar declares an OID variable named sysName as defined in the standard MIB subtree system:

```
$StaticVar(sysName, Explorer 16 running Microchip SNMP Agent)
```

These commands declare an OID variable named sysID:

```
$DeclareVar(sysID, OID, SINGLE, READONLY, 43.6.1.2.1.1.2)
$StaticVar(sysID, 43.6.1.4.1.17095)
```

These commands declare an OID variable of type MAC address:

```
$DeclareVar(macID, OCTET_STRING, SINGLE, READONLY, 44.6.1.4.1.17095.10)
$StaticVar(macID, 0, 1, 2, 3, 4, 5)
```

DynamicVar

DynamicVar declares a previously defined OID variable as dynamic. A dynamic OID variable is managed by the main application. The main application is responsible for providing or updating the value associated with this variable.

Status

Optional; required only if application requires dynamic OID variables.

Syntax

```
$DynamicVar(oidName, id)
```

Parameters

oidName

Name of OID variable that is being declared as a dynamic. It must have been declared by a previous DeclareVar command.

id

Any 8-bit identifier value from 0 to 255. It must be unique among all dynamic OID variables. The main application uses this value to refer to actual OID string defined by oidName.

Note: An OID variable of data type OID cannot be declared as dynamic.

Result

If compiled successfully, this command will declare given oidName as a dynamic variable. An entry will be created in the header file mib.h file of the form:

```
#define oidName  id
```

An application can refer to this dynamic OID by including the header "mib.h" in the source file that needs to refer to this OID.

Precondition

The given oidName must have been declared using previous DeclareVar command.

Example

These commands declare an OID variable named LED_D5 as a dynamic variable:

```
$DeclareVar(LED_D5, BYTE, SINGLE, READWRITE, 43.6.1.4.1.17095.3.1)
$DynamicVar(LED_D5, 5)
```

SequenceVar

SequenceVar declares a previously defined OID variable as a sequence variable and assigns an index to it. A sequence variable can consist of an array of values and any instance of its values can be referenced by index. More than one sequence variable may share a single index creating multi-dimensional arrays. The current version limits the size of the index to 7 bits wide, meaning that such arrays can contain up to 127 entries.

Status

Optional; required only if application needs to define sequence variables.

Syntax

$SequenceVar(oidName, indexName)

Parameters

oidName

Name of OID variable that is being declared as a sequence. This oidName must have been declared by a previous DeclareVar command with oidType of SEQUENCE.

indexName

Name of OID variable that will form an index to this sequence. It must have been declared by a previous DeclareVar command with dataType of BYTE.

Note: The dataType of indexName must be BYTE. All sequence variables must also be declared as dynamic.

Result

If compiled successfully, this command will declare given oidName as a dynamic variable.

Precondition

A given oidName must have been declared using previous DeclareVar command with oidType of SEQUENCE.

Example

These commands declare a Trap table called TRAP_RECEIVER consisting of four columns:

TRAP_RECEIVER_ID

TRAP_ENABLED

TRAP_RECEIVER_IP

TRAP_COMMUNITY

Any row in this table can be accessed using TRAP_RECEIVER_ID as an index.

```
$DeclareVar(TRAP_RECEIVER_ID, BYTE, SEQUENCE, READWRITE, 43.6.1.4.1.17095.2.1.1.1)
$DynamicVar(TRAP_RECEIVER_ID, 1)
$SequenceVar(TRAP_RECEIVER_ID, TRAP_RECEIVER_ID)

$DeclareVar(TRAP_RECEIVER_ENABLED, BYTE, SEQUENCE, READWRITE, 43.6.1.4.1.17095.2.1.1.2)
$DynamicVar(TRAP_RECEIVER_ENABLED, 2)
$SequenceVar(TRAP_RECEIVER_ENABLED, TRAP_RECEIVER_ID)

$DeclareVar(TRAP_RECEIVER_IP, IP_ADDRESS, SEQUENCE, READWRITE,
43.6.1.4.1.17095.2.1.1.3)
$DynamicVar(TRAP_RECEIVER_IP, 3)
$SequenceVar(TRAP_RECEIVER_IP, TRAP_RECEIVER_ID)

$DeclareVar(TRAP_COMMUNITY, ASCII_STRING, SEQUENCE, READWRITE,
43.6.1.4.1.17095.2.1.1.4)
$DynamicVar(TRAP_COMMUNITY, 4)
$SequenceVar(TRAP_COMMUNITY, TRAP_RECEIVER_ID)
```

AgentID

AgentID assigns a previously declared OID variable of type OID as an Agent ID for this SMNP Agent. OID variable defined to be Agent ID must be supplied in SNMPNotify function to generate Trap.

Status

Optional; required only if application needs to generate Trap(s).

Syntax

```
$AgentID(oidName, id)
```

Parameters

oidName

Name of OID variable that is being declared as a sequence. This oidName must have been declared by a previous DeclareVar command with oidType of OID.

id

An 8-bit identifier value to identify this Agent ID variable.

Note: The data type of oidName must be OID. oidName must be declared static.

Result

If compiled successfully, AgentID will declare given oidName as a dynamic variable.

Precondition

The given oidName must have been declared using a previous DeclareVar command with oidType of OID. It must also have been declared static using a previous StaticVar command.

Example

The following command sequence declares the Agent ID for this SNMP agent:

```
$DeclareVar(MICROCHIP, OID, SINGLE, READONLY,   43.6.1.2.1.1.2)
$StaticVar(MICROCHIP, 43.6.1.4.1.17095)
$AgentID(MICROCHIP, 255)
```

MICROCHIP MIB COMPILER mib2bib

In addition to the source code for the SNMP agent, the companion file archive for this application note includes a simple command-line compiler for 32-bit versions of Microsoft Windows®. The compiler, named "mib2bib" (management information base to binary information base), converts the Microchip MIB script into a binary format compatible with the Microchip SNMP Agent. It accepts Microchip MIB script in ASCII format and generates two output files: the binary information file snmp.bib, and the C header file mib.h. The binary file can be placed on an SD media card to be read by the FAT16 code.

The complete command line syntax for mib2bib is:

```
mib2bib [/?] [/h] [/q] <MIBFile>
[/b=<OutputBIBDir>] [/I=<OutputIncDir]
```

where:

/? displays command line help

/h displays detail help for all script commands

/q overwrites existing "snmp.bib" and "mib.h" files

<MIBFile> is the input MIB script file.

<OutputBIBDir> is the output BIB directory where snmp.bib should be copied. If a directory is not specified, the current directory will be used.

<OutputIncDir> is the output Inc directory where mib.h should be copied. If a directory is not specified, the current directory will be used.

For example, the command:

```
mib2bib MySNMP.mib
```

compiles the script MySNMP.mib and generates the output files snmp.bib and mib.h in the same directory.

In contrast, the command:

```
mib2bib /q MySNMP.mib /b=WebPages
```

compiles the script file MySNMP.mib and overwrites the existing output files. Additionally, it specifies that the file snmp.mib is located in the subdirectory WebPages. Because it isn't specified, mib.h is assumed to be in the current directory.

If compilation is successful, mib2bib displays the statistics on the binary file, including the number of OIDs and the Agent ID, as well as the output file size. A typical display following a successful run is shown in Example 3.

The MIB compiler is a simple rule script compiler. While it can detect and report many types of parsing errors, it does have these known limitations:

- All command lines must be written in single line.
- All command parameters must immediately end with either a comma ',' or right parenthesis. For example, $DeclareVar(myOID, ASCII_STRING , …) will fail because the ASCII_STRING keyword is not immediately followed by a comma.
- All numerical data must be written in decimal notation.

The mib2bib compiler reports all errors with a script name, line number, error code, and actual description of error. A list of errors, along with their explanations, is provided in Table 5 on page 38.

EXAMPLE 3: TYPICAL OUTPUT DISPLAY FOR AN `mib2bib` COMPILATION

```
C:\pic32_solutions\bsd_snmp_agent_demo\Source>mib2bib /q snmp.mib /b=WebPages
mib2bib  v1.0 (May 27 2003)
Copyright (c) 2003 Microchip Technology Inc.

Input MIB File : C:\pic32_solutions\bsd_snmp_agent_demo\Source\snmp.mib
Output BIB File: C:\pic32_solutions\bsd_snmp_agent_demo\Source\WebPages\snmp.bib
Output Inc File: C:\pic32_solutions\bsd_snmp_agent_demo\Source\mib.h

BIB File Statistics:

    Total Static OIDs         : 9
       Total Static data bytes: 129
    Total Dynamic OIDs        : 11
    (mib.h entries)
       Total Read-Only OIDs   : 4
       Total Read-Write OIDs  : 7
----------------------------------------
    Total OIDs                : 20

    Total Sequence OIDs       : 4
    Total AgentIDs            : 1
========================================
    Total MIB bytes           : 302
    (snmp.bib size)
```

TABLE 5: `mib2bib` RUN-TIME ERROR

Error	Description	Reason
1000	Unexpected end-of-file found	End-of-file was reached before end-of-command
1001	Unexpected end-of-line found	End-of-line was reached before end-of-command
1002	Invalid escape sequence detected; only ‘,’, ‘\’, ‘(‘, or’)’ may follow ‘\’	All occurrences of ‘,’, ‘(‘, ’)’, ‘\’ must be preceded by ‘\’
1003	Unexpected empty command string received	Command does not contain any parameter
1004	Unexpected right parenthesis found	Right parenthesis was found in place of a parameter
1005	Invalid or empty command received	Command does not contain sufficient parameters
1006	Unexpected escape character received	A ‘\’ character was detected before or after parameters were expected
1007	Unknown command received	
1008	Invalid parameters: expected `$DeclareVar (oidName, dataType, oidType, access, oid)`	
1009	Duplicate OID name found	Specified OID name is already in use
1010	Unknown data type received	Data type keyword does not match one of the allowed keywords
1011	Unknown OID type received	OID type keyword does not match one of the allowed keywords
1012	Empty OID string received	
1013	Invalid parameters: expected `$DynamicVar (oidName, id)`	
1014	OID name is not defined	
1015	Invalid OID ID received, must be between 0-255 inclusive	

TABLE 5: `mib2bib` RUN-TIME ERROR (CONTINUED)

Error	Description	Reason
1016	Invalid parameters: expected `$StaticVar (oidName, value)`	
1017	Invalid parameters: expected `$SequenceVar (oidName, index)`	
1018	Current OID already contains a static value	This OID has already been declared static
1019	Invalid number of index parameters received	All `SequenceVar` must include only one index
1020	OID of `sequence` type cannot contain static data	All sequence OID variables must be dynamic
1021	This is a duplicate OID, the root of this OID is not the same as previous OID(s), or this OID is a child of a previously defined OID	All OID string must contain same root OID
1022	Invalid index received, must be `BYTE` data value	All sequence index OID must be of data type `BYTE`
1023	Invalid OID access type received: must be `READONLY` or `READWRITE`	
1024	Current OID is already assigned an ID value	Current OID is already declared as dynamic
1025	Duplicate dynamic ID found	Current OID is already declared as dynamic with duplicate ID
1026	No static value found for this OID	Current OID was declared static, but does not contain any data
1027	No index value found for this OID	Current OID was declared as sequence but does not contain any index
1028	OID data scope (dynamic/static) is not defined	Current OID was declared, but was not defined to be static or dynamic
1029	Invalid data value found	Data value for current OID does not match with its data type
1030	Invalid parameters: expected `$AgentID (oidName, id)`	
1031	Only OID data type is allowed for this command	`AgentID` command must use OID name of OID data type
1032	This OID must contain static OID data	`AgentID` command must use OID name of static data
1033	This OID is already declared as an Agent ID	Only one `AgentID` command is allowed
1034	An Agent ID is already assigned	Only one `AgentID` command is allowed
1035	OID with `READWRITE` access cannot be static	An OID was declared `READWRITE` and made static
1036	OID of OID data type cannot be dynamic	Current version does not support OID variable of data type OID
1037	This OID is already declared as dynamic	
1038	This OID is already declared as static	
1039	This OID does not contain Internet root, Internet root of '43' must be used if this is Internet MIB	All internet OIDs must start with ‘43’, this is a warning only and will not stop script generation
1040	Given value was truncated to fit in specified data type	An OID was declared as `BYTE` or `WORD` but the value given in `StaticVar` exceeded the data range
1041	Given string exceeds maximum length of 127	All `OCTET_STRING` and `ASCII_STRING` must be less than 128
1042	Invalid OID name detected. OID name must follow standard 'C' variable naming convention	All OID names must follow ‘C’ naming convention as these names are used to create ‘define’ statements in the `mib.h` file
1043	Total number of dynamic OIDs exceeds 256	This version supports total dynamic OIDs of 256 only. All dynamic OID IDs must range from 0-255

BIB Format

The binary image of the MIB generated by the compiler is an optimized form of a modified binary tree. The core SNMP module reads this information from the binary file on the FAT16 media and uses it to respond to remote NMS requests.

A BIB image consists of one or more node or OID records. A parent node is stored first, followed by its left-most child. This structure is repeated until the leaf nodes of this tree are reached. The second left-most child of the original parent is then stored in the same manner, and the process is repeated until the entire tree is stored.

Each record consists of several fields defined below. The format of a single BIB record takes the form:

EXAMPLE 4: FORMAT OF A SINGLE BIB RECORD

```
<oid>, <nodeInfo>, [id], [siblingOffset], [distantSiblingOffset], [dataType],
[dataLen], [data], [{IndexCount, <IndexNodeInfo>, <indexDataType>}]…
```

Some fields indicated by angle brackets (< >) are always present; other fields in square brackets ([ ]) are optional depending on characteristics of the current node. The `IndexCount`, `IndexNodeInfo` and `indexDataType` fields, delimited with braces ({ }), are optional but always occur together. The `siblingOffset` and `distantSiblingOffset` are 16 bits wide; all other fields are 8 bits wide.

The `oid` field is the 8-bit OID value.

The `nodeInfo` field is an 8-bit data structure with each bit serving as a flag for a different node feature.

TABLE 6: `nodeinfo` BITS

Bit	Name	When Set (= `1`)
0	`blsDistantSibling`	Node has distant sibling
1	`blsConstant`	Node has constant data
2	`blsSequence`	Node is sequence
3	`blsSibling`	Node has a sibling
4	`blsParent`	Node is a parent
5	`blsEditable`	Node is writable
6	`blsAgentID`	Node is an Agent ID variable
7	`blsIDPresent`	Node contains ID

The `id` field is the 8-bit variable ID for the node as defined by the MIB script command `DynamicVar`. This field is only defined for leaf nodes where `bIsIDPresent` = `1`. A leaf node is one that does not have any child, i.e., `bIsParent` = `0`.

The `siblingOffset` field contains the offset (with respect to beginning of the BIB image) to the sibling node immediately to its right. Here we define a sibling as a node that shares the same parent node; a parent is the linked node immediately above it. This is defined only if `bIsSibling` is '`1`'.

The distantSiblingOffset field contains the offset to a distant sibling. This is present only if bIsDistantSibling is '1'. A distant sibling is defined as a leaf node that shares an ancestor (more than one level up) with another leaf node. In other words, for any given node either siblingOffset or distantSiblingOffset will be defined, but not both at once.

The dataType field specifies the data type for this node. This is defined only for leaf nodes (bIsParent = 0). The supported data types are shown in Table 7.

TABLE 7: SUPPORTED DATA TYPES

Hex Value	Data Type
00	BYTE
01	WORD
02	DWORD
03	OCTET_STRING
04	ASCII_STRING
05	IP_ADDRESS
06	COUNTER32
07	TIME_TICKS
08	GAUGE32
09	OID

The dataLen field defines the length of constant data. It is defined only for a leaf node with bIsConstant = 1, i.e., a static node.

The data field contains the actual data bytes. As above, only leaf nodes with bIsConstant = 1 (static nodes) will have this field.

The IndexCount field contains the index number for this node. This is defined only if this node is of the sequence type (bIsSequence = 1). Since only one index is allowed in this version, this value (when defined) will always be '1'.

The IndexNodeInfo field is an 8-bit data structure that works like the nodeInfo field; individual bit definitions are the same. This is defined only if this node is of the sequence type (bIsSequence = 1).

The indexDataType field defines the data type of the index node; it works identically to the dataType field and uses the same definitions. This is defined only if this node is of the sequence type (bIsSequence = 1).

DEMO SNMP AGENT APPLICATION

To better demonstrate the abilities of the SNMP agent, the companion archive file for this application note includes a complete demo application. Using the Microchip Explorer 16 demonstration board as a hardware platform, it allows the user to control the board in real-time. Key features of the demo include:

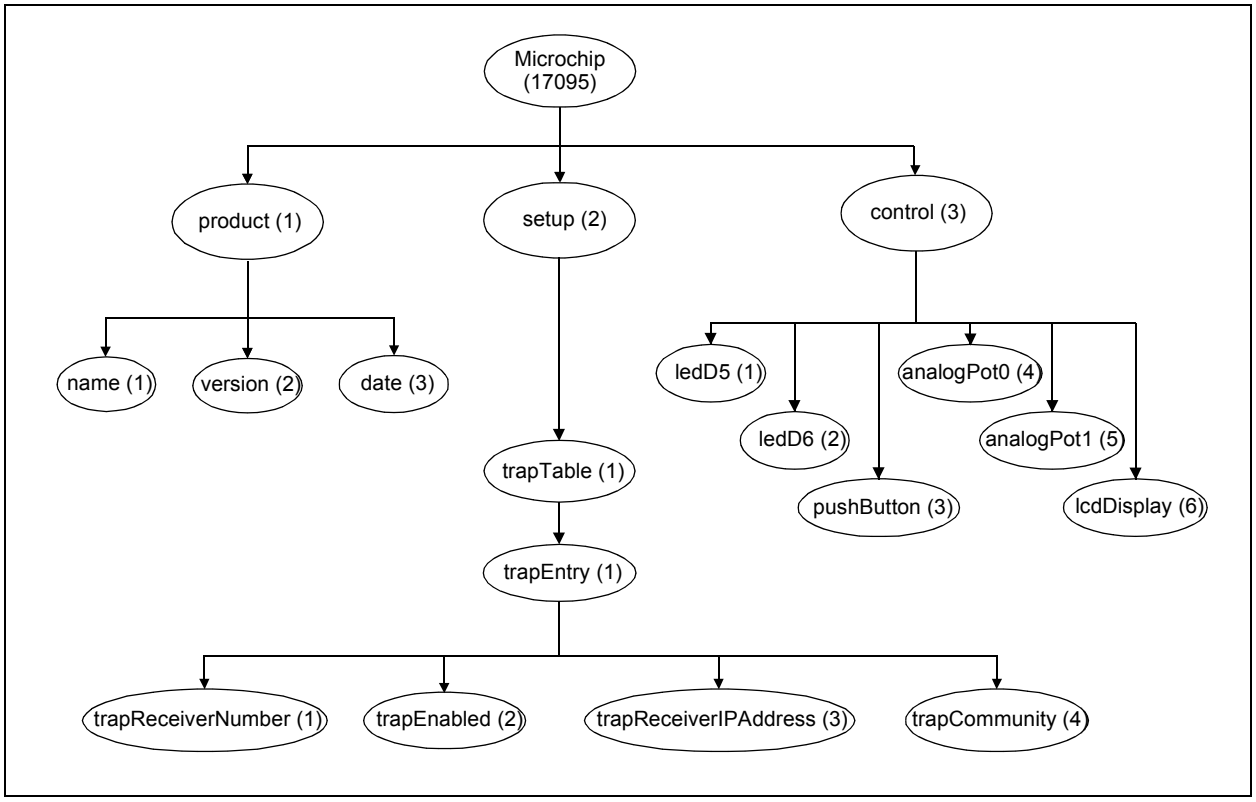
- Implements a complete MIB defined in ASN.1 syntax for use with NMS software
- Provides access to simple variables, such as LEDs and push button switches
- Illustrates read/write access to a multi-byte ASCII_STRING variable
- Implements run-time configurable Trap table
- Illustrates read/write access to a four-column Trap table
- Implements DHCP to obtain automatic IP address and other configuration parameters

Using NMS Software with the SNMP Agent and Microchip MIB

The demo application includes an MIB definition file written in ASN.1 syntax. This file, `mchp.mib`, defines the SMI for the Explorer 16 board's private Microchip MIB; it is also the basis for the MIB in the binary image created by `mim2bib.exe`. Figure 10 shows the full tree view of the MIB.

Any commercial or non-commercial NMS software that is ASN.1 compatible should be able to read and compile it. When it is loaded, you can use the NMS software to display the Microchip MIB and communicate with the demo application.

FIGURE 10: STRUCTURE OF THE PRIVATE MICROCHIP MIB IN THE DEMO APPLICATION



The MIB definition in the demo application allows real-time I/O and management of these features on the Explorer 16 board:

- `Trap` receiver information
- Switch LEDs D5 and D6 on and off
- Read the status of push button S3
- Read two analog potentiometer values
- Write a message of up to 16 characters to the first line of the on-board LCD display

PRODUCT SUBTREE

This subtree provides product related information, such as name, version and date. Its OIDs are listed in Table 8.

`Trap` TABLE SUBTREE

This subtree is an example of how an Agent would remember and accept a `Trap` configuration as set by remote NMS. This is a table consisting of four columns. The size of this table is limited to 2 entries, as defined by `TRAP_TABLE_SIZE` in the source file `main.c`. When a `Trap` table entry is created with `TrapEnabled` set (= `1`), the Explorer 16 board will generate a `Trap` whenever a push button switch is pushed.

The OIDs for this subtree are listed in Table 9.

CONTROL SUBTREE

This subtree provides real-time I/O control of the Explorer 16 board. The OIDs are listed in Table 10.

TABLE 8: PRODUCT SUBTREE AND ASSOCIATED OIDs

OID Name	Access/Data Type	Purpose
`Name`	Read only, String	Board name
`Version`	Read only, String	Version number string
`Date`	Read only, String	Release data (month, year)

TABLE 9: `Trap` TABLE SUBTREE AND ASSOCIATED OIDs

OID Name	Access/Data Type	Purpose
`TrapReceiverNumber`	Read only, Integer	Index to this table
`TrapEnabled`	Read-Write, Integer	Enables this entry to receive `Trap` `1` = Enabled `0` = Disabled
`TrapReceiverIPAddress`	Read-Write, IP Address	IP address of NMS that is interested in receiving `Trap`
`TrapCommunity`	Read-Write, String with length of 8 characters	Community name to be used when sending `Trap` to this receiver

TABLE 10: CONTROL SUBTREE AND ASSOCIATED OIDs

OID Name	Access Type	Purpose
`LedD5`	Read-Write, Integer	Switch on/off LED D5: `0` = On `1` = Off
`LedD6`	Read-Write, Integer	Switch on/off LED D6: `0` = On `1` = Off
`PushButton`	Read only, Integer	Read status of push button switch S3: `1` = Open `0` = Closed
`AnalogPot0`	Read only, Integer	Read 10-bit value of potentiometer AN0
`AnalogPot1`	Read only, Integer	Read 10-bit value of potentiometer AN1
`LcdDisplay`	Read-Write, 16 char. long String	Writes first line of on-board LCD

Experimenting with the Demo Agent Application

You may add any number of static OIDs to the MIB without making any changes to the demo application's source file (`main.c`). After adding the new OIDs to the script file, create a new BIB file with the `mib2bib` compiler.

If you want to add a dynamic OID to the demo, you must change the `snmpex.c` source file. Corresponding changes will also need to be made to the logic in the `SNMPGetVar`, `SNMPGetNextIndex` and `SNMPSetVar` callback functions. Also, you will need to recompile the MIB script file; the new header file, `mib.h`, will contain the new dynamic OIDs. When this is all done, you can build the new project and reprogram the microcontroller.

ANSWERS TO COMMON QUESTIONS

Q: Why is my SNMP Manager program unable to detect my SNMP agent?

A: Make sure that you have the SNMP version set to 1. Also, make sure that your IP address is correct.

Q: Why am I unable to perform a "walk"?

A: The reason could be that the IP address of the SNMP agent is incorrect and/or the SNMP version is not set to 1.

Q: Why isn't my SNMP agent notifying correctly?

A: Make sure that you have set up the trap parameters correctly. You should be able to view them when you perform a walk.

Q: Why am I unable to set any parameters?

A: Make sure that you have the manager configured properly. Also, make sure that your BIB file has the "children" defined correctly.

Q: Why isn't my SNMP agent servicing any requests?

A: It is possible that the BIB file is not correctly installed in the FAT16 storage device.

CONCLUSION

The SNMP agent presented here provides another protocol option for the Microchip BSD TCP/IP Stack. Together with the stack and your application, it provides a compact and efficient over-the-network management agent than can run on any of the PIC32MX 32-bit microcontrollers. Its ability to run independently of an RTOS or application makes it versatile; while its ability to handle up to 256 OIDs and an unlimited number of static OIDs makes it flexible.

REFERENCES

J. Case, M. Fedor, M. Schoffstall and J. Davin, *"A Simple Network Management Protocol (SNMP)"*, RFC 1157. SNMP Research, Performance Systems International and MIT Laboratory for Computer Science, May 1990.

N. Rajbharti, AN833*, "The Microchip TCP/IP Stack"* (DS00833). Microchip Technology Inc., 2002.

A. S. Tanenbaum, *Computer Networks (Third Edition)*. Upper Saddle River NJ: Prentice-Hall PTR, 1996.

W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*. Reading MA: Addison-Wesley, 1994.

APPENDIX A: SOURCE CODE FOR THE SNMP AGENT

Because of their size and complexity, complete source code listings for the software discussed in this application note are not provided here. A complete archive file with all the necessary source and support files is available, in `.zip` format, for the following applications:

- Microchip SNMP Agent
- Microchip MIB Script Compiler (`mib2bib.exe`)
- Demo Application for SNMP Agent and the Explorer 16 Demonstration Board

The complete source file archive that accompanies application note AN1108 "*Microchip TCP/IP Stack with BSD Socket API*" is also available, and includes all of the necessary source and support files for the stack. These files are required to develop the Microchip SNMP agent.

Both of these archive files may be downloaded from the Microchip corporate web site at:

www.microchip.com

After downloading the archive, always check the `version.log` file for the current revision level and a history of changes to the software.

REVISION HISTORY

Revision A (10/2007)

This is the initial released version of this document.

Revision B (03/2008)

Revised "Installing Source Files" section; Revised Tables 2, 3 and 4.

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Trademarks

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2002

Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://support.microchip.com
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-4182-8400
Fax: 91-80-4182-8422

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-572-9526
Fax: 886-3-572-6459

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

01/02/08