
An SNMP Agent for the Microchip TCP/IP Stack

<i>Author: Nilesh Rajbharti Microchip Technology Inc.</i>

INTRODUCTION

Simple Network Management Protocol (SNMP) is an Internet protocol that was originally designed to manage different network devices, such as file servers, hubs, routers and so on. It can also be used to manage and control an ever increasing number of small embedded systems connected to one another over any IP network. Systems can communicate with each other using SNMP to transfer control and status information, creating a truly distributed system. Unlike more familiar human-oriented protocols like HTTP, SNMP is considered a machine-to-machine protocol.

This application note provides one of the key components of the SNMP management system: the *SNMP Agent* that runs on the managed device. The simple Agent presented here is designed to run on Microchip's PICmicro[®] microcontrollers, and is implemented using services provided by the free Microchip TCP/IP Stack. Its main features include:

- Based on the free Microchip TCP/IP Stack
- Portable across all PIC18 family of microcontrollers
- Functions independently of RTOS or application
- Supports both Microchip's MPLAB[®] C18 and Hitech PICC 18[™] C compilers 'out of the box'
- Supports SNMP Version 1 over UDP
- Supports *Get*, *Get-Next*, *Set* and *Trap* PDUs
- 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
- Handles access to constant OIDs automatically
- Utilizes a Management Information Base (MIB) that can be stored internally or on external EEPROM
- Includes its own PC-based MIB compiler
- Does not contain built-in TCP/UDP/IP statistics counters. User application must define and manage required MIB.

This document briefly describes the SNMP protocol just enough to explain the implementation and design of the SNMP Agent. Interested readers are encouraged to refer to RFC 1157 and related documents for more detailed information about the protocol. Users are also strongly encouraged to download and review Microchip

application note, AN833 "The Microchip TCP/IP Stack". The Stack and its accompanying software tools, particularly the MPFS builder, are prerequisites for creating the SNMP Agent.

PREVIEW: HOW TO BUILD THE SNMP AGENT

For those who are already familiar with SNMP and the Microchip Stack, we will start by outlining the process for incorporating the SNMP Agent into an application. If you need to familiarize yourself a little more with SNMP first, refer to the overview that starts on page 3.

The flow chart in Figure 1 outlines the general steps for developing a Microchip SNMP Agent. There are two main processes involved: developing the MIB, and using that to develop the actual agent. Each process, in turn, has several steps. All of these are covered later in this document.

The major steps are:

1. Download and install the accompanying source files for the SNMP Agent.
2. Using the MIB script (page 22), define your private MIB along with other standard MIB that your application may require.
3. Use the included MIB compiler ("mib2bib", page 29) to build a binary MIB image ("BIB").
4. Include the generated BIB file into an MPFS image, and either download or link the MPFS image data file.
5. Create an application project that contains all of your required files, plus the following Microchip TCP/IP Stack and SNMP Agent files:
 - MAC.c
 - ARP.c
 - ARPTsk.c
 - IP.c
 - UDP.c
 - SNMP.c
 - StackTsk.c
 - MPFS.c
 - Keeprom.c or MPFSImg.c
 - Helpers.c
 - Delay.c

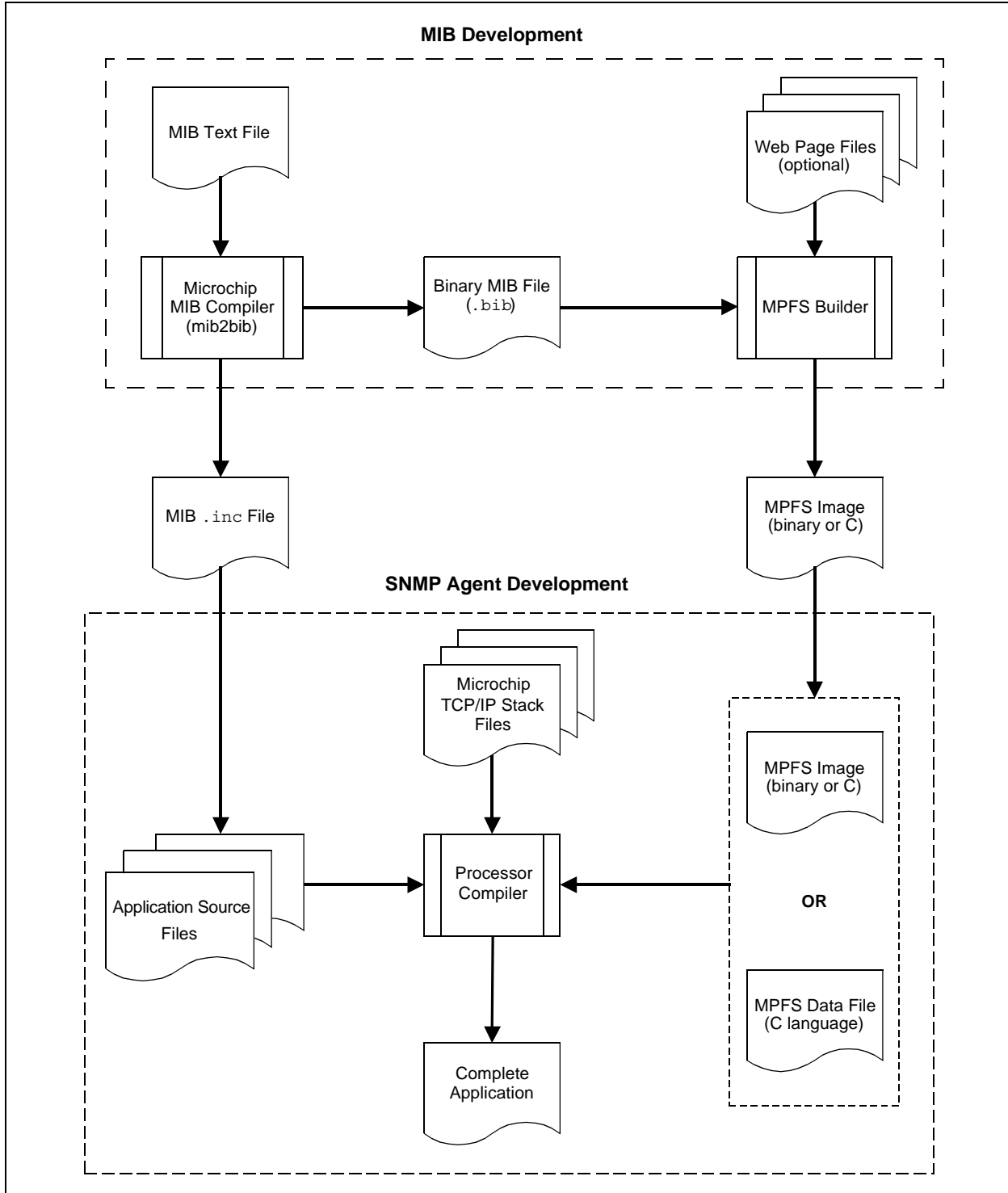
Keep in mind that you may have to include other Microchip files depending on the other modules that you select.

AN870

6. Modify your main application source file to include the SNMP header files and the MIB definition file, and implement the SNMP callback functions. Use one of the included demo SNMP application files (page 33) as a reference for making any necessary modifications.

Once successfully built, you can use any standard SNMP Management Software to access your SNMP Agent device.

FIGURE 1: OVERVIEW OF THE SNMP AGENT DEVELOPMENT PROCESS

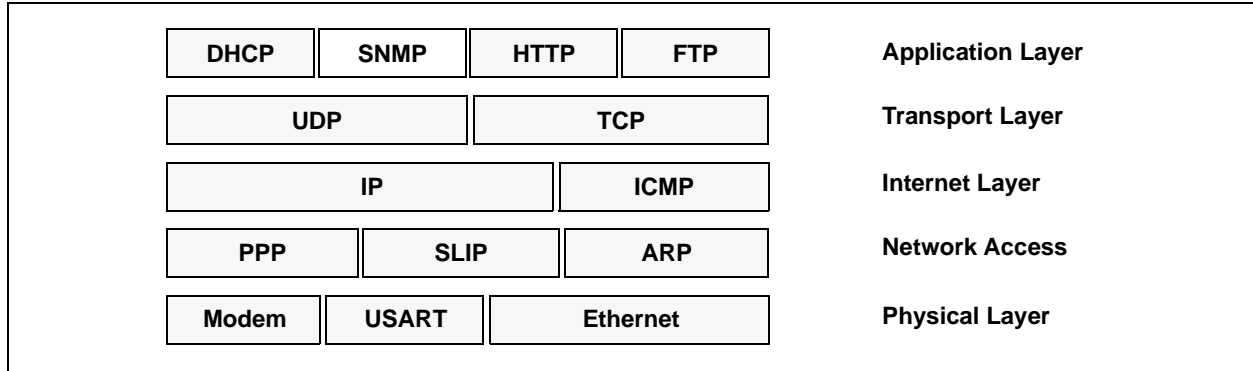


SNMP OVERVIEW

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 2.

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

FIGURE 2: 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 3 shows the typical SNMP model and the associated terminology.

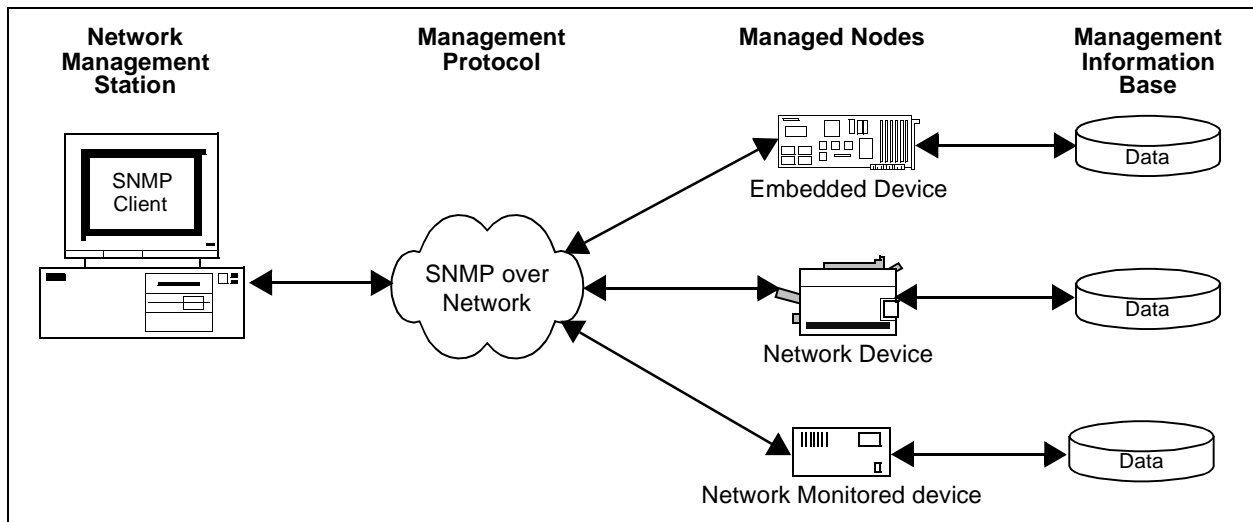
NETWORK MANAGEMENT STATION (NMS)

The NMS is one half of the SNMP client-server setup; the other half being the agent. Because our focus in this document is on the agent, we mention NMS here briefly for the sake of completeness.

Typically, the NMS is a personal computer running special software, although it could very well be any other embedded device. NMS acts as an SNMP client, periodically polling the SNMP Agent for data.

Once a device is SNMP enabled, any commercially or non-commercially available NMS software can be used. NMS can be used to monitor the collection of similar or dissimilar devices. Many of the commercially available PC-based NMS systems provide a graphical representation of managed devices. Also, the addition of devices to a network does not require change in NMS software; it can dynamically load information about a new device and can provide the option to manage that device. All of these features give SNMP the functionality that makes it a popular choice for network and device management.

FIGURE 3: OVERVIEW OF THE SNMP MODEL



MANAGED NODE OR SNMP AGENT

A *Managed Node* (or SNMP Agent, as it is very often called) is the device that is being managed by NMS. The 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 the appropriate information.

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 4 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 5 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. Once 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.

FIGURE 4: GENERIC STRUCTURE OF MANAGEMENT INFORMATION (SMI)

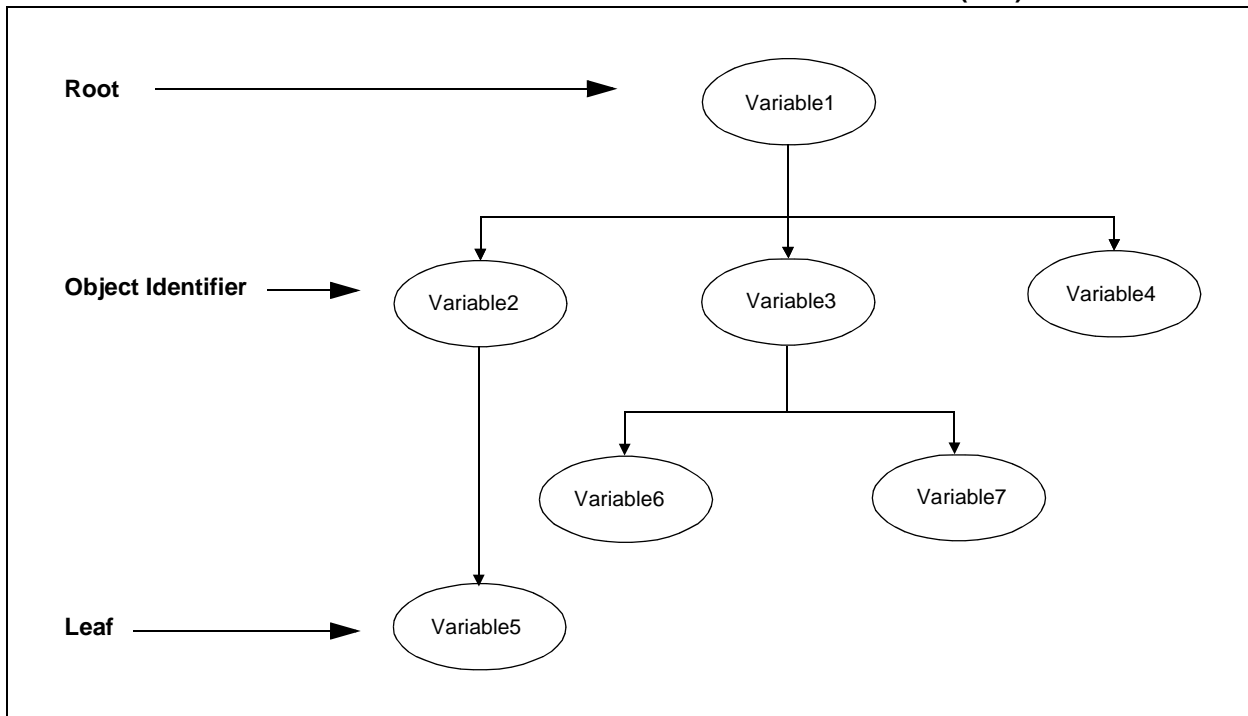
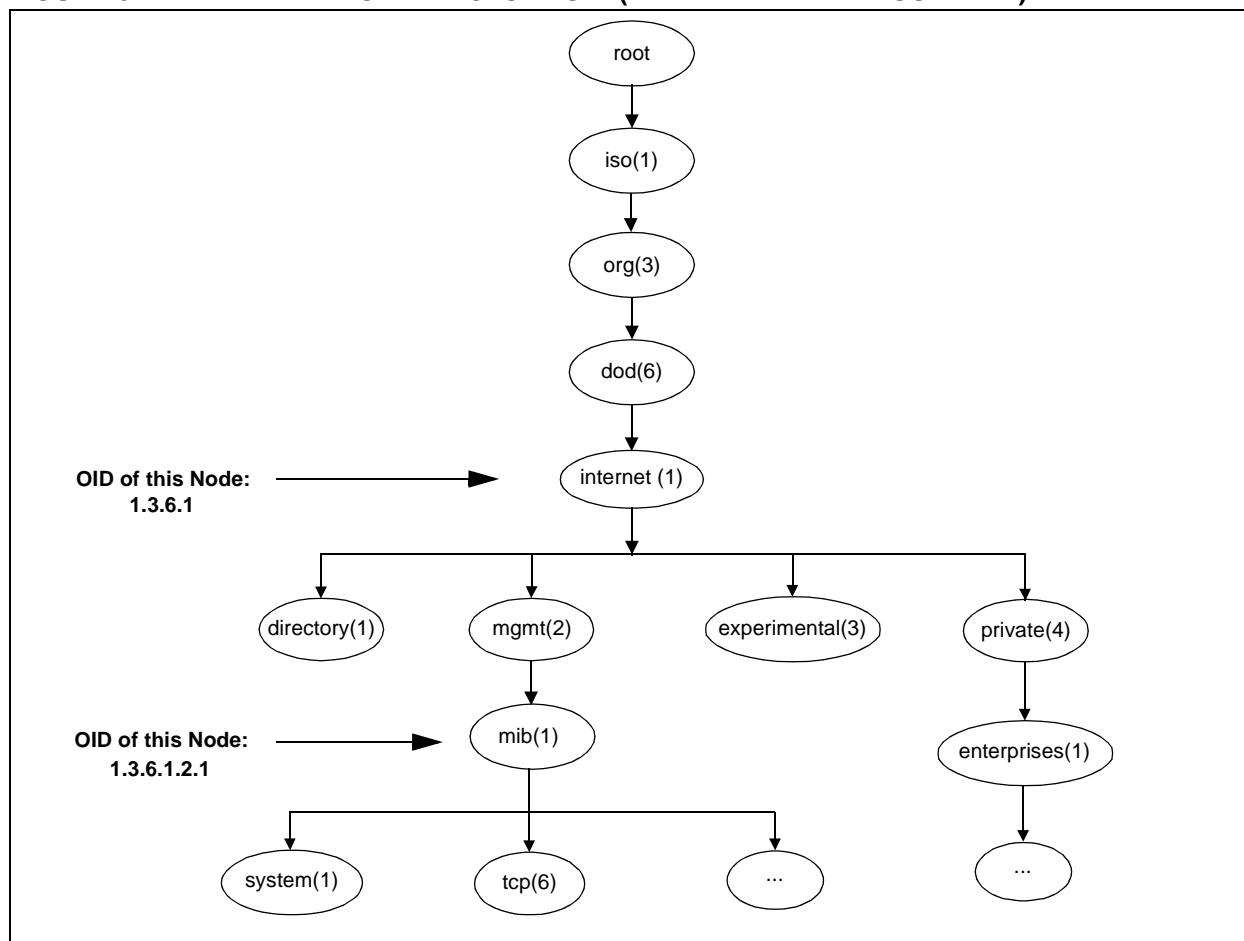


FIGURE 5: EXAMPLE OF AN ACTUAL SMI (PARTIAL INTERNET SUBTREE)



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. Such 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 5 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 $40a + b$. 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 discussed later in this document requires that all SNMP OIDs start with '43'.

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 format. ASN.1 is written as a text file and compiled using an ASN syntax compiler. Most of the 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 the need to learn ASN syntax first. The Microchip SNMP Agent uses its own special script to describe its agent OIDs, as well as 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, starting on page 22.

AN870

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
    STATUS mandatory
    DESCRIPTION
        "A table containing..."
    ::= { udp 5 }
    
```

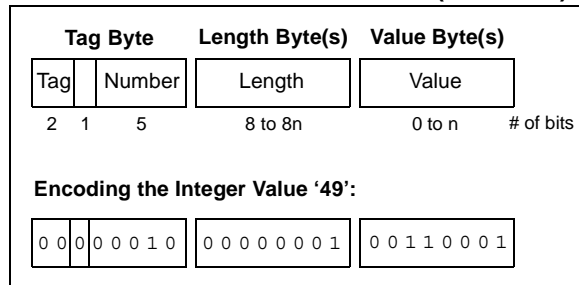
Binary Encoding Rules (BER)

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, where 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 6 shows the breakdown of typical BER values and an example of encoding.

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

FIGURE 6: GENERIC BER FORMAT (TOP) AND AN EXAMPLE OF BER ENCODING (BOTTOM)



Protocol Data Unit (PDU)

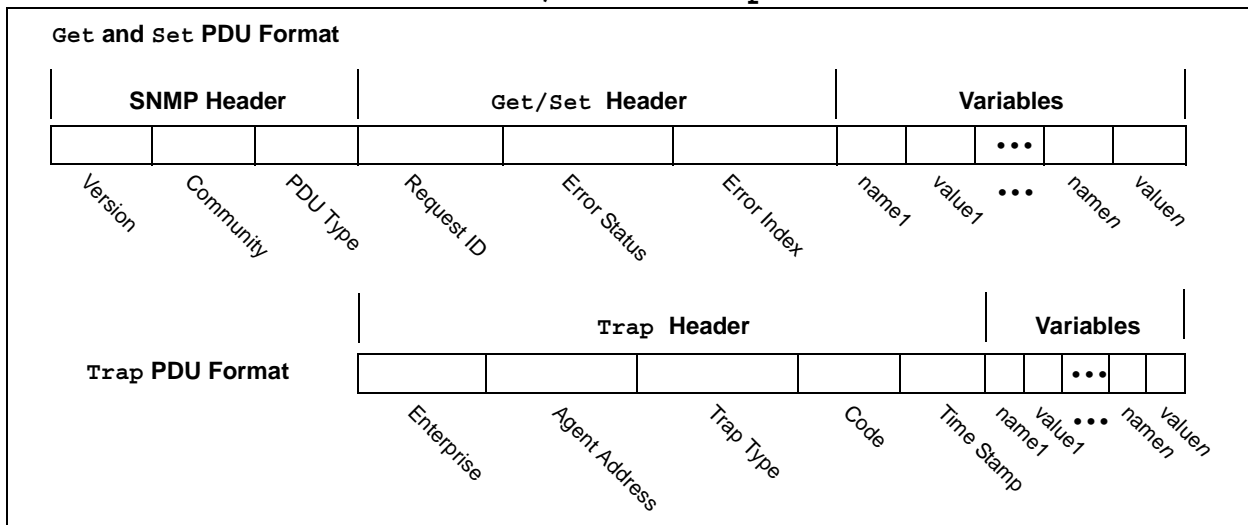
Data packets exchanged between two SNMP nodes are called *Protocol Data Units (PDU)*. SNMP Version 1 defines a total of five main types of PDUs:

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

All *Get* and *Set* PDUs share a common message format; the format for *Trap* PDUs is somewhat different. The two formats are compared in Figure 7.

Users of the Microchip SNMP Agent do not need to know the details of the PDU format or its encoding; the SNMP Agent module automatically handles all of the low level protocol details, including the encoding and decoding of data variables. Those who are interested in the details are encouraged to refer to RFC 1157 for more information about the individual PDU fields.

FIGURE 7: PDU FORMATS FOR Get/Set AND Trap PACKETS



MICROCHIP SNMP AGENT APIs

The actual SNMP Agent is implemented by several files working together with the Microchip 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 also 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 Application Program Interfaces (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 their APIs follows from here through page 21.

SNMPInit

This function is used to initialize the SNMP Agent module.

Syntax

```
void SNMPInit(void)
```

Parameters

None

Return Values

None

Pre-Condition

There must be at least one free UDP socket available and `UDPInit` is already called.

Side Effects

One UDP socket will be used.

Remarks

None

Example

```
// Do Stack manager Init. This will initialize UDPInit too.
StackInit();

// Initialize SNMP module
SNMPInit();

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

AN870

SNMPTask

This function 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.

Pre-Condition

SNMPInit() is already called.

Side Effects

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

Remarks

None

Example

```
// Do Stack manager Init. This will initialize UDPInit too.
StackInit();

// Initialize SNMP module
SNMPInit();

// Initialize other modules...
...

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

    // Call SNMP Task
    SNMPTask();

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


SNMPGetVar

This function 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 OID variable is of type sequence and NMS can query any of 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 value in `val->byte`.

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

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

If data type is `IP_ADDRESS`, the application may copy 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 value in `val->dword`.

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

Pre-Condition

None

Side Effects

None

Remarks

For a definition of the data types associated with *val*, refer to the `DeclareVar` description on page 23.

AN870

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

This function is a callback used by the SNMP Agent module to request next *index* after given *index* (if there is any).

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.

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 any. 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.

Pre-Condition

None

Side Effects

None

Remarks

This function is called for only 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;
}
```

AN870

SNMPIsValidSetLen

This function 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.

Pre-Condition

None

Side Effects

None

Remarks

This function 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 23.

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

This function 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 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.

Pre-Condition

None

Side Effects

None

Remarks

This function 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 23.

AN870

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

This function 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 allowed to perform a given operation.

FALSE, if otherwise.

Pre-Condition

None

Side Effects

None

Remarks

None

Example

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

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

BOOL SNMPValidate(SNMP_ACTION SNMPAction, char* community)
{
    if ( !memcmppgm2ram(community, (ROM void*)PUBLIC_COMMUNITY,
        PUBLIC_COMMUNITY_LEN) )
    {
        if ( SNMPAction == SNMP_GET )
            return TRUE;
    }
    else if ( !memcmppgm2ram(community, (ROM void*)PRIVATE_COMMUNITY,
        PRIVATE_COMMUNITY_LEN) )
    {
        if ( SNMPAction == SNMP_SET )
            return TRUE;
    }
    return FALSE;
}
```

AN870

SNMPNotifyPrepare

This function is used by the application to prepare to send SNMP Trap to 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

Pre-Condition

None

Side Effects

None

Remarks

This function is called at the beginning of 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:
        // Once notify prepare is done,
        // wait for SNMP Agent to be ready.
        if ( SNMP_IsNotifyReady(&IPAddress) )
        {
            // Once 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;
}

```

AN870

SNMPNotifyIsReady

This function 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.

Pre-Condition

`SNMPNotifyPrepare()` is already called.

Side Effects

None

Remarks

This function performs ARP resolution and obtains the MAC address for a given IP address. Once 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:
        // Once notify prepare is done, wait for SNMP Agent to be ready.
        if ( SNMPIsNotifyReady(&IPAddress) )
        {
            // Once 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;
}

```

AN870

SNMPNotify

This function 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.

Pre-Condition

```
SNMP_IsNotifyReady() = TRUE
```

Side Effects

None

Remarks

This function 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 this function; 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:
        // Once notify prepare is done, wait for SNMP Agent to be ready.
        if ( SNMPIsNotifyReady(&IPAddress) )
        {
            // Once 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 29).

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, starting with the dollar sign character (“\$”), and one or more command parameters 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 having 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 23 through 28.

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 43.6.1.4.1.1.17095.3.1. 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

This command 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 <code>oidType</code> of SEQUENCE must be assigned an "index" OID variable using the <code>SequenceVar</code> 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).

AN870

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`.

Pre-Condition

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

This command 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 backslashes must be preceded by the backslash ("\") 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.

Pre-Condition

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

Examples

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

```
$StaticVar(sysName, PICDEM.net 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)
```

AN870

DynamicVar

This command 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.

Pre-Condition

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

This command 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.

Pre-Condition

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)
```

AN870

AgentID

This command assigns a previously declared OID variable of type OID as an Agent ID for this SNMP 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, this command will declare given *oidName* as a dynamic variable.

Pre-Condition

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 included in a Microchip File System (MPFS) image.

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.

mib2bib 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 1.

EXAMPLE 3: TYPICAL OUTPUT DISPLAY FOR AN mib2bib COMPILATION

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

Input MIB File : C:\MCHPStack\Source\snmp.mib
Output BIB File: C:\MCHPStack\Source\WebPages\snmp.bib
Output Inc File: C:\MCHPStack\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)
```

AN870

TABLE 1: mib2bib RUN-TIME ERROR CODES

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 allowed keywords.
1011	Unknown OID type received	OID type keyword does not match one of 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	
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 or 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.

TABLE 1: mib2bib RUN-TIME ERROR CODES (CONTINUED)

Error	Description	Reason
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 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 MPFS image 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:

```
<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.

Bit	Name	When Set (= 1)
0	<i>blsDistantSibling</i>	Node has distant sibling
1	<i>blsConstant</i>	Node has constant data
2	<i>blsSequence</i>	Node is sequence
3	<i>blsSibling</i>	Node has a sibling
4	<i>blsParent</i>	Node is a parent
5	<i>blsEditable</i>	Node is writable
6	<i>blsAgentID</i>	Node is an Agent ID variable
7	<i>blsIDPresent</i>	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 *blsIDPresent* = 1. A leaf node is one that does not have any child (i.e., *blsParent* = 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 *blsSibling* is '1'.

The *distantSiblingOffset* field contains the offset to a distant sibling. This is present only if *blsDistantSibling* 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 (*blsParent* = 0). The supported data types are shown in the following table.

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 *blsConstant* = 1 (i.e., a static node).

The *data* field contains the actual data bytes. As above, only leaf nodes with *blsConstant* = 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 (*blsSequence* = 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 (*blsSequence* = 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 (*blsSequence* = 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 Microchip's PICDEM.net™ 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

Programming the PICDEM.net Board for the Demo SNMP Application

To run the demo application, it is necessary to have a HEX file. One option is to use one of the supplied demo files: either `DemoSNMPAgent.hex` or `HtDemoSNMPAgent.hex`. For evaluation purposes, these two files are essentially the same. Note, however, that `DemoSNMPAgent.hex` was built using the Microchip C18 compiler, while `HtDemoSNMPAgent.hex` was built using the Hitech PICC 18 C Compiler. If you need to rebuild the project, simply open the appropriate demo project and rebuild it using MPLAB® 6.x and an appropriate compiler.

If you need to recreate the demo project from the ground up, make sure that following files and options are included:

- `DemoSNMPAgent.c`
- `Delay.c`
- `SNMP.c`
- `MAC.c`
- `ARP.c`
- `ARPTsk.c`
- `IP.c`
- `UDP.c`
- `ICMP.c`
- `DHCP.c`
- `MPFS.c`
- `Keeprom.c`
- `Helpers.c`
- `Tick.c`
- `Xlcd.c`
- `C18Cfg.c` (if using Microchip C18 compiler)
- `18f452.lkr` (or other appropriate linker script file if using Microchip C18 compiler)

The demo SNMP application requires that the following four symbols be defined. You may define them either on the compiler command line, or in the `StackTsk.h` header file:

- `MPFS_USE_EEPROM`
- `STACK_USE_DHCP`
- `STACK_USE_ICMP`
- `STACK_USE_SNMP_SERVER`

Once a HEX file is built or selected, follow the standard procedure for your device programmer when programming the microcontroller. Make sure that the following configuration options are set:

- Oscillator: HS
- Watchdog Timer: Disabled
- Low Voltage Programming: Disabled
- Background Debug: Disabled

When the programmed microcontroller is installed on the PICDEM.net demo board and powered up, the system LED should blink to indicate that the application is running. The LCD display will show:

DemoSNMP v1.0

on the first line (the version number may differ depending on the release level of the application), and either a configuration message or an IP address on the second line.

Once programmed, the demo application may still need to be configured properly before it is put on a real network. The instructions below are specific to Microsoft Windows and the HyperTerminal terminal emulator; your procedure may vary if you use a different operating system or terminal software.

1. Program a PIC18 microcontroller as noted above, and install it on the PICDEM.net board.
2. Connect the PICDEM.net board to an available serial port on the computer using a standard RS-232 cable.
3. Launch HyperTerminal (Start > Programs > Accessories).
4. At the "Connection Description" dialog box, enter any convenient name for the connection. Click "OK".
5. At the "Connect To" dialog box, select the COM port that the PICDEM.net board is connected to. Click "OK".
6. Configure the serial port connected to the PICDEM.net board:
 - 19200 bps,
 - 8 data bits, 1 STOP bit and no parity
 - no flow controlClick "OK" to initiate the connection.
7. Apply power to the board while holding the S3 switch, or press and hold both the RESET and S3 switches; then, release the RESET switch. The LCD display shows the message:

DemoSNMP v1.0

Board Setup...

(The version number may differ depending on the release level of the application). Release S3.

The Configuration menu appears in the terminal window:

```
MCHPStack SNMP Agent
Demo Application v1.0
(Microchip TCP/IP Stack 2.20, <DATE>
```

1. Change board serial number.
2. Change default IP address.
3. Change default gateway address.
4. Change default subnet mask.
5. Enable DHCP and IP Gleaning.
6. Disable DHCP and IP Gleaning.
7. Download MPFS image.
8. Save & Quit.

Enter a menu choice (1-8):

8. Select each of the items that need to be configured and enter the new values. Select item 8 to save the changes and exit configuration; the new addresses are saved to the data EEPROM. The application exits Configuration mode and runs the SNMP Agent.

Connecting to an Ethernet Network

When running the SNMP demo application, the PICDEM.net board can be directly connected to an Ethernet network with no other modifications. Of course, the IP configuration must be compatible with that of the network. By default, the demo application uses these values for configuration:

- IP Address: 10.10.5.15
- Gateway Address: 10.10.5.15
- Subnet Mask: 255.255.255.0

Even if the IP address is compatible, the gateway and mask may not be. If changes are required, there are several ways to go about it.

AUTOMATIC CONFIGURATION WITH DHCP

If the network uses DHCP configuration, no additional work is needed. When the board is connected to the network and powered up, it will be assigned an IP configuration by the DHCP server. During this process, the LCD display shows the message:

DCHP/Gleaning...

After several seconds, the display shows the assigned IP address, for example:

```
100.100.100.1 1
```

The actual IP address displayed is the assigned address of the board. The number on the far right indicates the number of times the DHCP lease has been renewed. This is shown for informational purposes only.

Depending on how the network has been configured, the PICDEM.net board's IP address may change after being powered down for an extended period (i.e., the board's DHCP lease has expired and the old address has been taken by another device). Always use the IP address currently displayed to communicate with the board.

PRE-DEFINED NETWORK CONFIGURATIONS

Some networks may be "hard configured"; that is, each device has an address that has been manually assigned by the network administrator. In these cases, the PICDEM.net board should be configured manually before attaching it to the network with the IP configuration provided by the administrator. Refer back to "Programming the PICDEM.net Board for the Demo SNMP Application" (page 33) for details.

SETTING THE IP ADDRESS WITH IP GLEANING

If the board is connected to the network and only requires a change of IP address, IP gleaning can be used. This method is best suited to configure the IP address but not the gateway or subnet mask.

To use IP gleaning, the MAC address of the device must be known. This is always a 6-byte hexadecimal number of the format "xx-xx-xx-xx-xx-xx". For PICDEM.net boards, the MAC is always 00-04-A3-00-nn-nn, where "nn-nn" is the serial number of the board in hexadecimal format. Thus, a board with serial number 1234 (or 04D2h) has a MAC address 00-04-A3-00-04-D2.

Once the MAC address and new IP address of the device are determined, the address is determined by resetting the device, then issuing from a remote terminal the `arp` and `ping` commands. Continuing with the example above, if we wanted to assign the previously mentioned board the new IP address of 10.10.5.50, we would send the commands:

```
> arp -s 10.10.5.50 00-04-a3-00-04-d2
> ping 10.10.5.50
```

A successful `ping` response indicates that the IP address has been changed.

Downloading the MPFS Demo Image

The Microchip File System (MPFS) allows users to store binary image information for Stack related components in memory. MPFS is discussed in more detail in AN833, *"The Microchip TCP/IP Stack"*. The software utility for creating MPFS binary images is included in the companion files for both that application note as well as this one.

Users can store their MIB information (in BIB format) in memory using MPFS. The SNMP demo application includes an MPFS binary image named `mpfsimg.bin` which contains the MIB in binary format.

If an MPFS image is to be stored in an external serial EEPROM, it must either be preprogrammed with the MPFS image (via a device programmer) or downloaded from another application. The Web Server demo implements a simple MPFS download routine which accepts an MPFS binary file from a terminal emulator using the Xmodem protocol.

To download the binary MPFS file:

1. If not already done, set up the PICDEM.net board for configuration (see "Programming the PICDEM.net Board for the Demo SNMP Application", steps 1 through 7).
2. At the Configuration menu, type '7' to start the MPFS download. You should see the "Ready to download..." message and the left User LED (D6) should be blinking approximately twice per second.
3. From the HyperTerminal "Transfer" menu, select "Send File...". In the "Send File" dialog box, browse to the directory containing the file "mpfsimg.bin" and select it. Select "Xmodem" as the protocol.
4. Click "Send". Data transfer should start automatically. The User LED will blink as fast as data is received from the computer.
5. When the file is completely transferred, press '8' to exit the Configuration mode.

The SNMP Agent is now ready to run with the Microchip MIB.

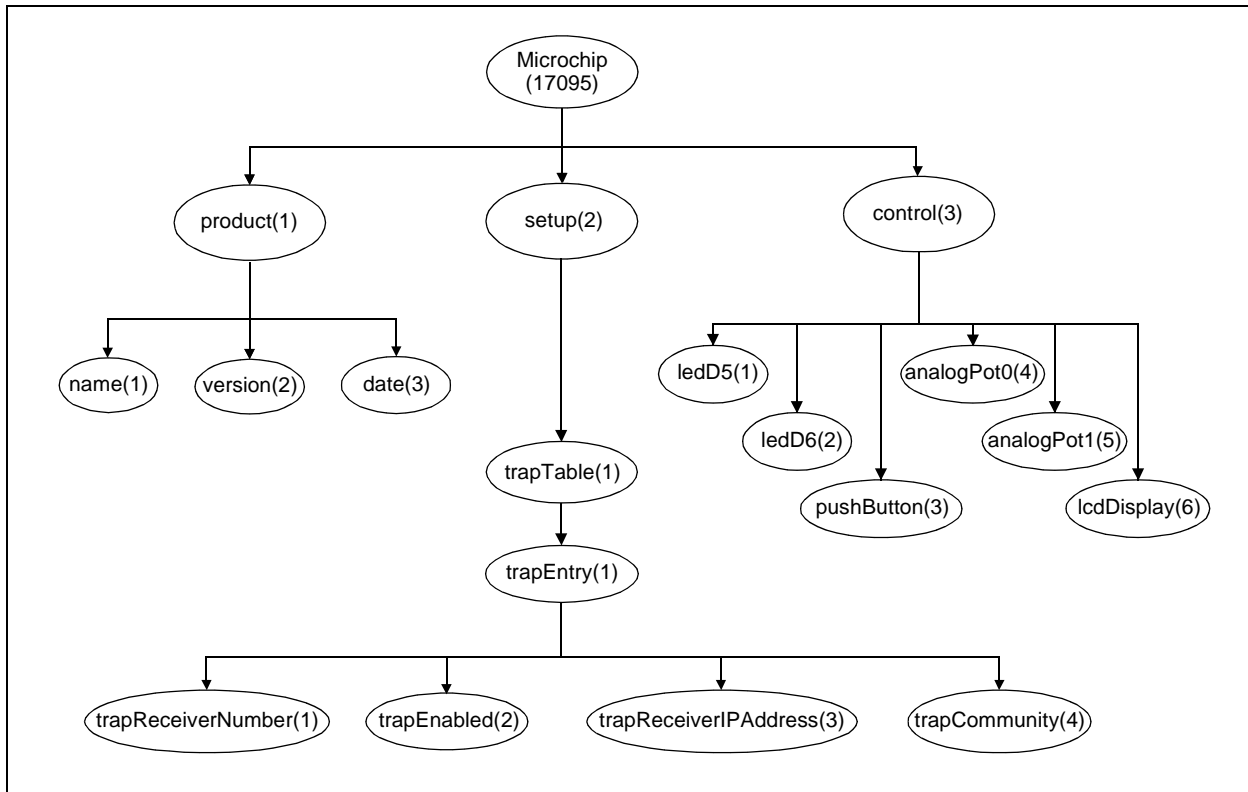
AN870

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 PICDEM.net board's private Microchip MIB; it is also the basis for the MIB in the MPFS image. Figure 8 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. Once it is loaded, you can use the NMS software to display the Microchip MIB and communicate with the demo application.

FIGURE 8: 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 PICDEM.net 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 2.

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 DemoSNMPAgent.c. Once a Trap table entry is created with TrapEnabled set (= 1), the PICDEM.net board will generate a Trap whenever a push button switch is pushed.

The OIDs for this subtree are listed in Table 3.

CONTROL SUBTREE

This subtree provides real-time I/O control of the PICDEM.net board. The OIDs are listed in Table 4.

TABLE 2: 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 3: 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 4: 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 (`DemoSNMPAgent.c`). After adding the new OIDs to the script file, create a new BIB file with the `mib2bib` compiler. Include this file in the MPFS image and download the new image into the EEPROM.

If you want to add a dynamic OID to the demo, you must change the `DemoSNMPAgent.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. Once this is all done, you can build the new project and reprogram the microcontroller along with the EEPROM.

Users who are already familiar with the Microchip TCP/IP Stack and its accompanying HTTP server can incorporate the web server pages and the MIB for the SNMP Agent into a single MPFS image. (You will need to ensure that you have enough room in the EEPROM for everything, of course.) The process assumes that you have already installed the files for the Stack, and the files for the web pages are in the "WebPages" directory.

First, generate your BIB image as before (page 29) but use the command line:

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

This writes `snmp.bib` to the directory "WebPages" (the header file, `mib.h`, will be written to its default directory).

Now, generate the MPFS image with the command:

```
mpfs WebPages mpfsimg.bin
```

This includes all files in "WebPages" into a single MPFS image, including the BIB file you just created. Note that the existing version of `mpfsimg.bin` will be overwritten in the process.

MEMORY USAGE

The total amount of memory used for the SNMP Agent depends on the compiler and optimization level selected. At the time of this publication (July 2003), the fully optimized size for the SNMP module using Microchip's C18 compiler is 2819 words (5638 bytes) of program memory and global RAM of 28 bytes. Data EEPROM is not required.

Note that the SNMP module may require the selection of certain modules in order to successfully build the complete SNMP Agent. Inclusion of these modules will increase overall memory requirements.

CONCLUSION

The SNMP Agent presented here provides another protocol option for the Microchip TCP/IP Stack. Together with the Stack and the user's application, it provides a compact and efficient over-the-network management agent than can run on any of the PIC18 8-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 in .zip format is available with all the necessary source and support files for the following:

- Microchip SNMP Agent
- Microchip MIB Script Compiler (mib2bib)
- Demo Application for SNMP Agent and the PICDEM.net Demonstration Board
- MPFS Image Builder

Also available is the complete source file archive that accompanies AN833, *"The Microchip TCP/IP Stack"*. This includes all necessary source and support files for the Stack itself, as well as the MPFS Image Builder and the demo Web Page Server. These files are a requirement for any development with the Microchip SNMP Agent.

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

www.microchip.com

AN870

NOTES:

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.

Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, dsPIC, KEELoQ, MPLAB, PIC, PICmicro, PICSTART, PRO MATE and PowerSmart are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.


FilterLab, microID, MXDEV, MXLAB, PICMASTER, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

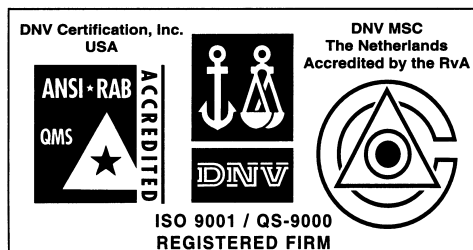
Accuron, Application Maestro, dsPICDEM, dsPICDEM.net, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, microPort, Migratable Memory, MPASM, MPLIB, MPLINK, MPSIM, PICC, PICkit, PICDEM, PICDEM.net, PowerCal, PowerInfo, PowerMate, PowerTool, rLAB, rPIC, Select Mode, SmartSensor, SmartShunt, SmartTel and Total Endurance are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

Serialized Quick Turn Programming (SQTP) is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2003, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.



Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999 and Mountain View, California in March 2002. The Company's quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELoQ® code hopping devices, Serial EEPROMs, microperipherals, non-volatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001 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: 480-792-7627
Web Address: <http://www.microchip.com>

Atlanta

3780 Mansell Road, Suite 130
Alpharetta, GA 30022
Tel: 770-640-0034
Fax: 770-640-0307

Boston

2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848
Fax: 978-692-3821

Chicago

333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071
Fax: 630-285-0075

Dallas

4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423
Fax: 972-818-2924

Detroit

Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo

2767 S. Albright Road
Kokomo, IN 46902
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles

18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888
Fax: 949-263-1338

Phoenix

2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7966
Fax: 480-792-4338

San Jose

2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950
Fax: 408-436-7955

Toronto

6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Australia

Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing

Unit 915
Bei Hai Wan Tai Bldg.
No. 6 Chaoyangmen Beidajie
Beijing, 100027, No. China
Tel: 86-10-85282100
Fax: 86-10-85282104

China - Chengdu

Rm. 2401-2402, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-86766200
Fax: 86-28-86766599

China - Fuzhou

Unit 28F, World Trade Plaza
No. 71 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7503506
Fax: 86-591-7503521

China - Hong Kong SAR

Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Shanghai

Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700
Fax: 86-21-6275-5060

China - Shenzhen

Rm. 1812, 18/F, Building A, United Plaza
No. 5022 Binhe Road, Futian District
Shenzhen 518033, China
Tel: 86-755-82901380
Fax: 86-755-8295-1393

China - Shunde

Room 401, Hongjian Building
No. 2 Fengxiangnan Road, Ronggui Town
Shunde City, Guangdong 528303, China
Tel: 86-765-8395507 Fax: 86-765-8395571

China - Qingdao

Rm. B505A, Fullhope Plaza,
No. 12 Hong Kong Central Rd.
Qingdao 266071, China
Tel: 86-532-5027355 Fax: 86-532-5027205

India

Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaughnessy Road
Bangalore, 560 025, India
Tel: 91-80-2290061 Fax: 91-80-2290062

Japan

Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471-6166 Fax: 81-45-471-6122

Korea

168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-882
Tel: 82-2-554-7200 Fax: 82-2-558-5932 or
82-2-558-5934

Singapore

200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-6334-8870 Fax: 65-6334-8850

Taiwan

Kaohsiung Branch
30F - 1 No. 8
Min Chuan 2nd Road
Kaohsiung 806, Taiwan
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan

Taiwan Branch
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Austria

Durisolstrasse 2
A-4600 Wels
Austria
Tel: 43-7242-2244-399
Fax: 43-7242-2244-393

Denmark

Regus Business Centre
Lautrup høj 1-3
Ballerup DK-2750 Denmark
Tel: 45-4420-9895 Fax: 45-4420-9910

France

Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - 1er Etage
91300 Massy, France
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany

Steinheilstrasse 10
D-85737 Ismaning, Germany
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy

Via Quasimodo, 12
20025 Legnano (MI)
Milan, Italy
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands

P. A. De Biesbosch 14
NL-5152 SC Drunen, Netherlands
Tel: 31-416-690399
Fax: 31-416-690340

United Kingdom

505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44-118-921-5869
Fax: 44-118-921-5820

07/28/03