SNMP V2c Agent for Microchip TCP/IP Stack

Author: Amit Shirbhate
Microchip Technology Inc.

INTRODUCTION

Simple Network Management Protocol (SNMP) is one of the key components of a Network Management System (NMS). SNMP is an application layer protocol that facilitates the exchange of management information among network devices. It is a part of the TCP/IP protocol suite.

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.

SNMP is used in a variety of applications where remote monitoring and controlling of the network node is desired, such as a network printer, online Uninterrupted Power Supply (UPS), security cameras, home and industrial appliances monitor and control, automatic energy meter readings, etc.

Unlike more familiar human-oriented protocols, like HTTP, SNMP is considered a machine-to-machine protocol.

There are three key components of TCP/IP Network Management:

- MIB – Management Information Base is a collection of the variables (managed objects) the network elements/agents store. The individual variables are identified by Object Identifiers (OIDs).
- SMI – Structure of Management Information is a set of common structures and a way to refer to variables in the database.
- SNMP – Simple Network Management Protocol is the protocol used to communicate between the manager and the network element agent.

Microchip TCP/IP Stack supports two versions of SNMP:

- SNMP Version 1 (SNMP V1)
- SNMP Version 2 Community-Based (SNMP V2c)

Most of the features are common to both versions. SNMP V2c offers additional protocol operations, such as `Get_Bulk`, a rich set of error indications, community-based access, etc.

The SNMP Agent described here is designed to run on Microchip's PIC® microcontrollers and is implemented using services provided by the free Microchip TCP/IP Stack. The following main features are included:

- Portable across all PIC18, PIC24, PIC32 MCUs and dsPIC® DSC families of microcontrollers
- Functions independently of RTOS or application
- SNMP 'C' source code is supported on Microchip's MPLAB® IDE and can be compiled with PIC18, PIC24, PIC32 MCUs and dsPIC DSC compilers 'out of the box'
- Supports SNMP Version V1 and V2c over User Datagram Protocol (UDP)
- Supports `Get`, `Get_Bulk`, `Get_Next`, `Set` and `Trap` Protocol Data Units (PDUs)
- Supports up to 255 dynamic OIDs and unlimited constant OIDs
- Supports sequence variables with a 7-bit index
- Supports enterprise-specific `Trap`(s) with one variable information
- Handles access to constant OIDs automatically
- Utilizes a MIB that can be stored internally or externally on EEPROM
- Includes PC-based Microchip customized MIB script compiler

Note: The Microchip SNMP Agent does not contain built-in TCP/UDP statistic counters. The number of TCP/UDP packets transmitted/received by the node is generally maintained for applications running on switches, routers or where the packet count is crucial to the node application. The user application, if required, should implement the packet counter mechanism and also define and manage the corresponding changes to the application MIB file.

This document describes the SNMP protocol to explain the implementation and design of the SNMP Agent. For more information about the SNMP protocol, refer to:

- RFC 3416 and related documents
- `TCPIP Stack Help.chm` file (Microchip Solutions\Microchip\Help), which comes with the Microchip TCP/IP Stack

The latest Microchip TCP/IP Stack (in installer format) can be downloaded from (www.microchip.com/tcpip).

The TCP/IP Stack and the accompanying software tools, particularly the MPFS2 Microchip File System Generator and mib2bib Microchip MIB script compiler utility, are required utilities for the Microchip SNMP Agent.

SNMP PROTOCOL 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 1.

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. Once a device is SNMP-enabled, any SNMP compatible host system SNMP Manager/Client can monitor and control that device. The SNMP Manager uses the UDP Port Number 161 to send requests to the agent. The agent uses the UDP Port Number 162 to send `Trap`(s) or notification events to the manager. The manager can request data from the agent or set variable values in the agent. The agent can respond and report events.

The SNMP collects Network Management Station (NMS) information in two ways:

- The agents on the network are polled by NMS software.
- Agents send alerts to the SNMP Managers. The alert can be sent to all the SNMP Managers in the community.

FIGURE 1: LOCATION OF SNMP IN THE TCP/IP PROTOCOL STACK

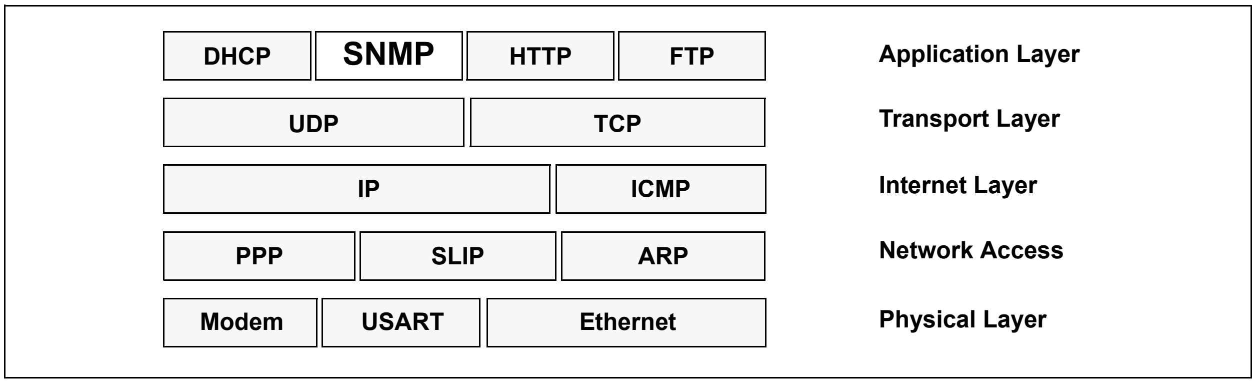
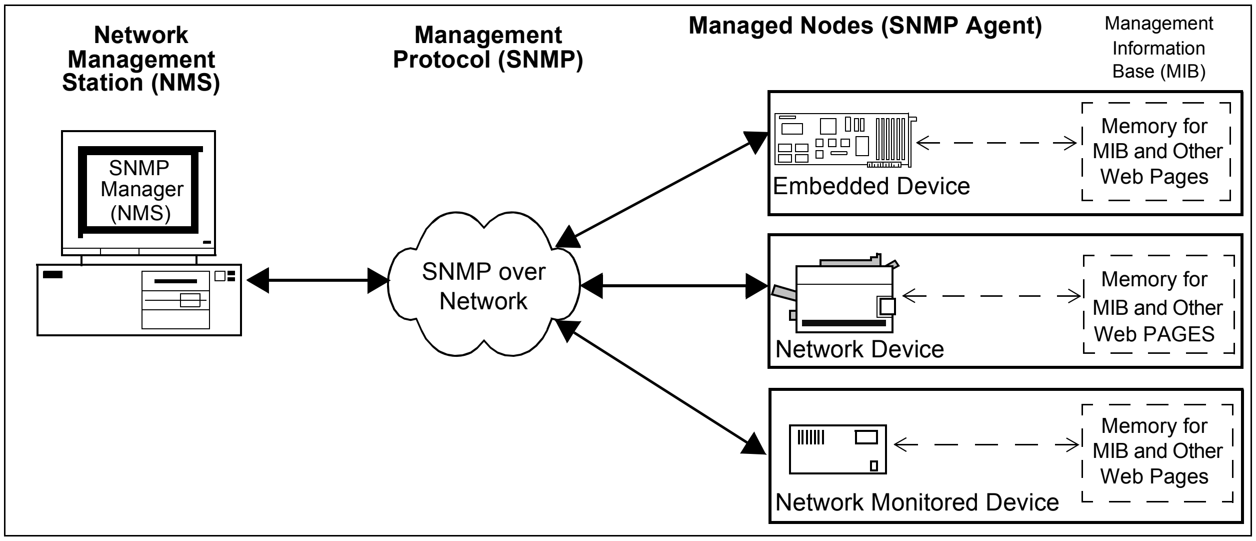


FIGURE 2: OVERVIEW OF THE SNMP MODEL


Network Management Station (NMS)
Management Protocol (SNMP)
Managed Nodes (SNMP Agent)
Management Information Base (MIB)
SNMP Manager (NMS)
SNMP over Network
Embedded Device
Memory for MIB and Other Web Pages
Network Device
Memory for MIB and Other Web PAGES
Network Monitored Device
Memory for MIB and Other Web Pages

SNMP TERMINOLOGY

This application note frequently uses terminology described by the SNMP specification, which we will briefly review here. Figure 2 shows how the associated terminology applies to the typical SNMP model. The SNMP Agent is also referred to as SNMP server and the SNMP Client is also referred to as SNMP Manager.

Network Management Station (NMS)

The *NMS* is one side of the SNMP Client-Server setup; the other side is the SNMP Agent. Because our focus in this document is on the agent, we will only discuss NMS here briefly.

Typically, the NMS is a personal computer running special or customized software that monitors and controls managed devices; it could be any other embedded device too. NMS acts as an SNMP Client, periodically polling an SNMP Agent for data.

Once a device is SNMP enabled, any commercial or customized NMS software can be used to monitor the device. 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. Uploading the new device's ASN.1 MIB file to the NMS software provides the option to manage the device. All of these features give SNMP the functionality that makes it a popular choice for network as well as device management. There are several SNMP NMS Manager/Client or MIB browser software vendors, for instance, SNMPc Manager from CastleRock Computing, LoriotPro from LUTEUS, iReasoning MIB browser and Trap Receiver and so on.

Managed Node (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 is the software running on the Managed Node or the network elements, such as a microcontroller on an embedded device, switches, routers and so on. 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 several Network Management Stations. The agent waits for NMS requests and responds with the appropriate information.

Management Information Base (MIB)

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

Figure 3 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 4 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 (PEN) from Internet Assigned Number Authority (IANA). Once assigned a PEN, the manufacturer may add or remove any number of subtrees beneath it, as required. Applications can be made at the web site: http://pen.iana.org/pen/PenApplication.page.

FIGURE 3: GENERIC STRUCTURE OF MANAGEMENT INFORMATION (SMI)

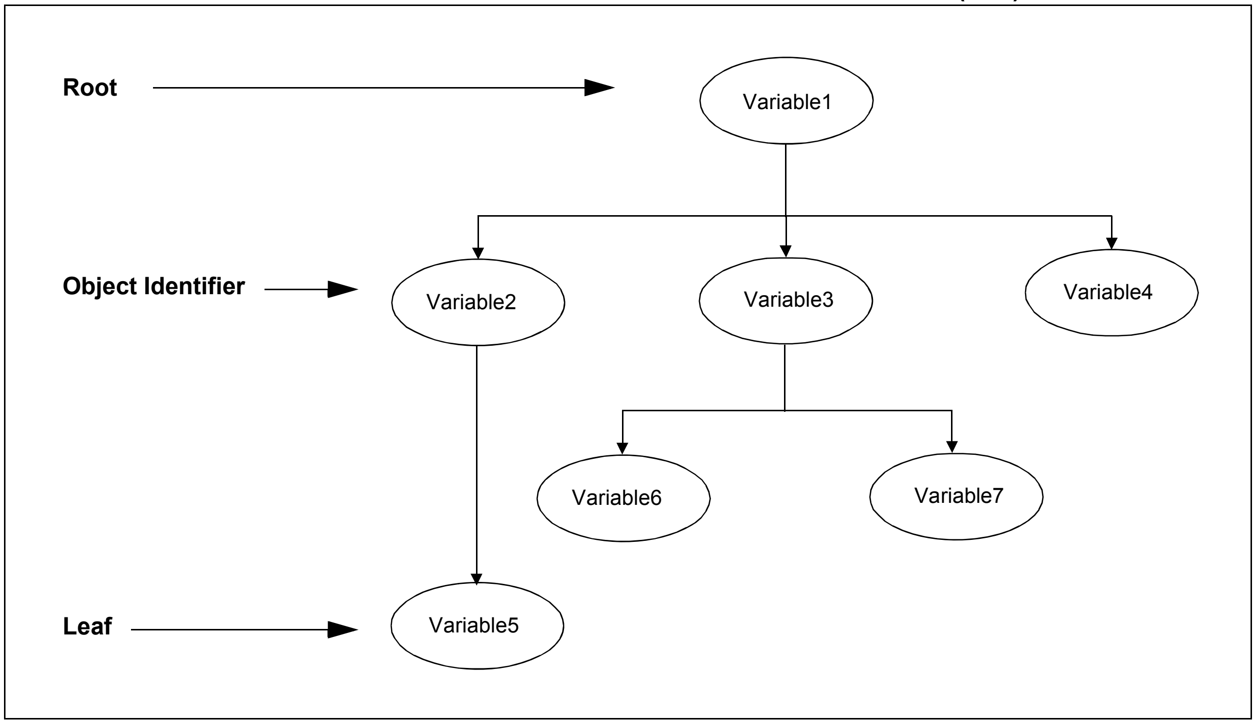
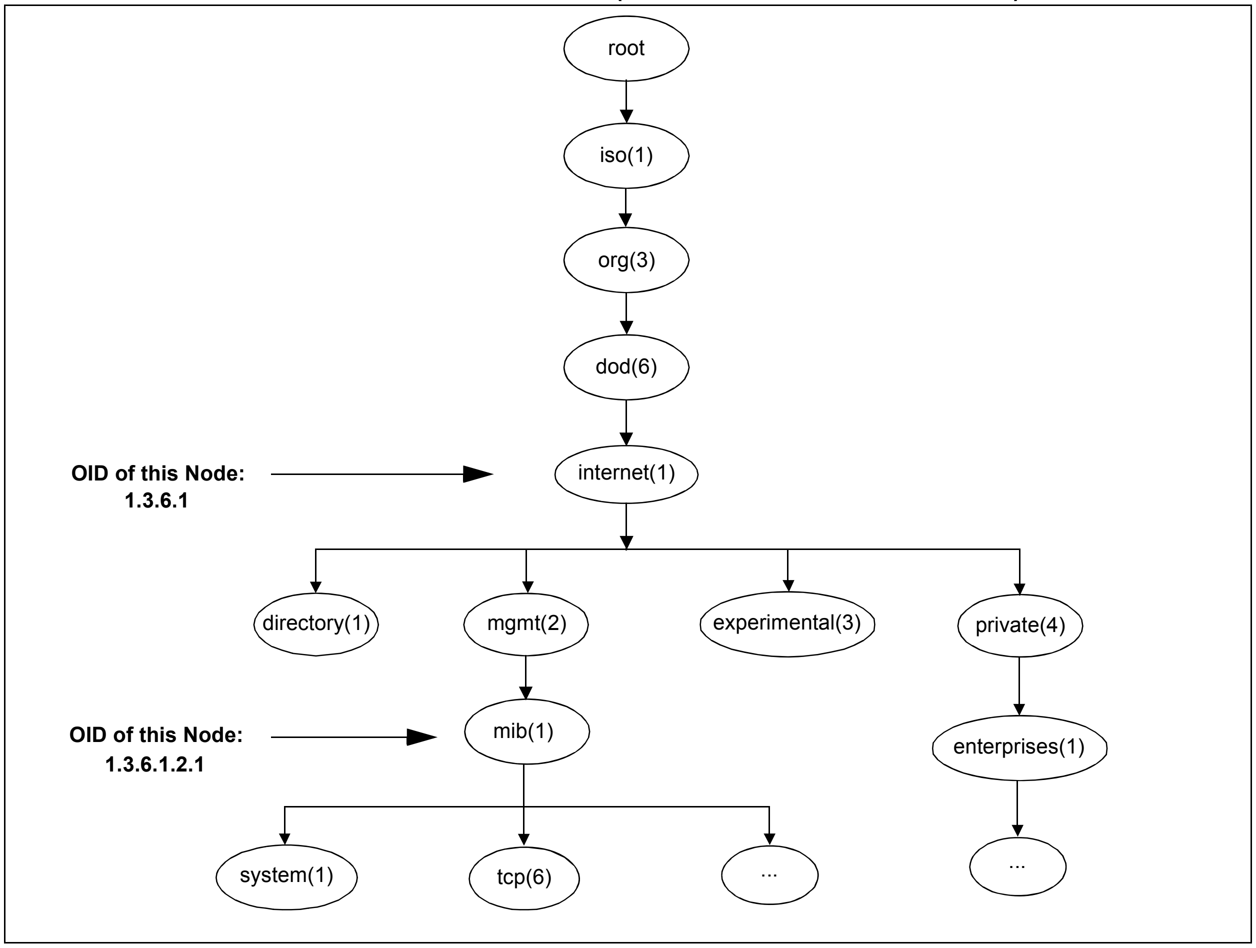


FIGURE 4: EXAMPLE OF AN ACTUAL SMI (PARTIAL INTERNET SUBTREE)



Object Identifier (OID)

Each node in the MIB tree is identified by a sequence of decimal numbers, called Object Identifier (OID). A specific node is uniquely referenced by its own OID and that of its parents' OIDs. Such 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 4 is written as '1.3.6.1.2.1.1'. 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).system(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 '43.6.1.2.1.1', not '1.3.6.1.2.1.1'.

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

SNMP Security

In SNMP, security is administered in two ways:

- Through community-based access architecture
- By sending a `Trap` if a predefined event occurs

There are different types of configuration parameters to be configured to an agent depending on the kind of security required:

1. A firewall should be used to protect the SNMP Agent from the Internet.
2. Authentication failure `Trap`s are sent to the Manager NMS.
3. Only requests from the SNMP community group members should be responded to.
4. Accept request/information PDU from the host belonging to a list of IP addresses. This is an application-specific implementation.

SNMP Community

Pairing of the SNMP Agent with some arbitrary set of SNMP application entities is called an SNMP community. An SNMP community is a group to which an agent and the NMS belongs. Each SNMP community is named by a string of octets, which is known as 'community name' for the particular community. The community name is used to identify the group.

An SNMP Agent can be a member of more than one SNMP community. An agent does not respond to the request from management stations which do not belong to one of its communities. The pairing of SNMP Access mode with the SNMP MIB view is called an SNMP community profile. A shared printer in a LAN can be used as an example to understand the SNMP V2c community concept. Every user in the LAN has access to some of the limited variables of the printer, such as the `printer_name`, `location`, `uptime`, `contact` and so on. These are standard MIB2 system variables.

These variables provide general printer information to users (SNMP Clients). These variables are the members of the 'public' community of the printer, and every user in LAN is also a default member of the 'public' community.

Some of the MIB variables of the printer, like queuing priorities, order print job, suspend, printing formats and so on, can be accessed by limited users. These types of variables are known as private variables. The private variables are members of the 'private' community and can be accessed only by a member of the same 'private' community. These variables will be located as child to 'enterprise' node, identified by PEN as OID in the MIB.

SNMP PROTOCOL DATA UNIT (SNMP PDU)

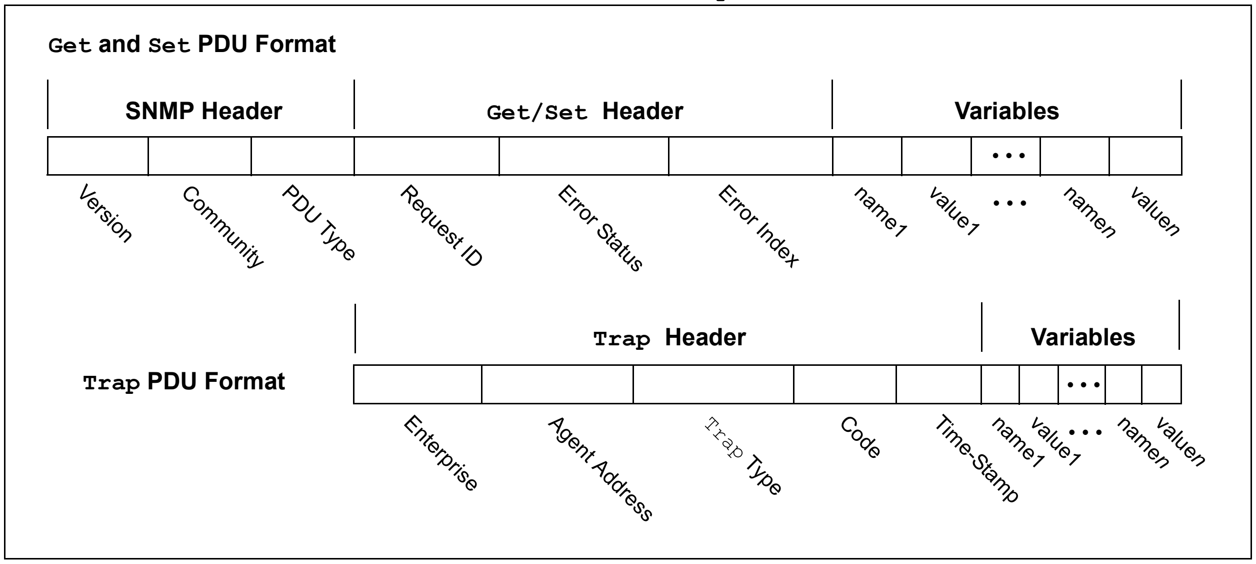
Data packets exchanged between two SNMP nodes are called Protocol Data Units (PDUs). SNMP V1 and V2c define six main types of data packets, which are exchanged between the SNMP Agent and Client.

- `Get_Request` – Get one or more variables' information from the agent (manager to agent).
- `Get_Next_Request` – Get next variable information in the MIB after one or more variables are specified in the request (manager to agent).
- `Get_Bulk_Request` – Get bulk operation is normally used to retrieve a large amount of data, particularly from large data tables residing in agent memory (manager to agent).
- `Set_Request` – Set is to configure one or more variables in the agent (manager to agent).
- `Get_Response` – Return variable information for the requested OID (agent to manager).
- `Trap` – Notification from the agent to the manager of a predefined event occurrence (agent to manager).

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 5.

FIGURE 5: PDU FORMATS FOR `Get/Set` AND `Trap` PACKETS



Note:	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. For more information on the individual PDU fields, refer to RFC 1157 and RFC 3416.

SNMP PDU TYPES

It is mandatory for all SNMP Agents to be able to generate `Get_Response` and `Trap` PDUs, and to receive and process `Get_Request`, `Get_Next_Request`, `Get_Bulk_Request` and `Set_Request` PDUs.

`Get_Request`

- (PDU Type = 0x00)

  This PDU is generated and transmitted as a request to get information or value of the requested OID.

  Upon receipt of the `Get_Request` PDU, the receiving entity (the agent) processes each variable binding in the variable binding list to produce a `Get_Response` PDU. The response PDU has the same values as the corresponding fields of the received request, except in cases stated below:

  - If the requested OID is found in the MIB database, and the request is authenticated for the access privileges, the value field is set to the value of the requested variable. Access privileges refer to community name and the read/ write access to the variable.
  - If the requested variable binding name does not have an OID prefix that exactly matches the OID prefix of any potential variable accessible by this request, then the value field is set to **noSuchObject**.
  - Otherwise, the value field is set to **noSuchInstance**, and the **error-status** and the **error-index** fields would be correspondingly set.

`Get_Next_Request`

- (PDU Type = 0x01)

  This PDU is generated and transmitted as a request to get information or the value of the lexicographical successor of the requested OID. Upon receipt of this request PDU, the receiving entity (the agent) processes each variable binding in the variable binding list to produce a `Get_Response` PDU. The response PDU has the same values as the corresponding fields of the received request, except in cases stated below:

  - The variable is located, which is in the lexicographically (in alphabetical order) ordered list of the names of all variables accessible by this request, and whose name is the first lexicographical successor of the variable binding's name in the incoming `Get_Next_Request` PDU. The corresponding variable binding's name and value fields in the response PDU are set to the name and value of the located variable.
  - If there is no lexicographical successor to the OID in the requested PDU, the value field in the response PDU is set to **endOfMibView** and the name field is set to the requested OID.
  - Otherwise, the **error-status** field and the **error-index** would be correspondingly set.

`Get_Bulk_Request`

- (PDU Type = 0x05)

  `Get_Bulk_Request` PDU is to request and transfer a large amount of information from the agent to the client (NMS) within the single PDU (particularly data from large tables). This will significantly reduce the PDU network traffic. Upon receipt of this request PDU, the receiving entity (the agent) processes each variable binding in the variable binding list to produce a `Get_Response` PDU, with the same request ID field.

  `Get_Bulk_Request` is made by giving an OID list along with a **Max-Repetitions** value and **Non-Repeater** value.

  The `Get_Bulk` operation is a continuous `Get_Next` operation depending on the **Max-Repetitions** value. The **Non-Repeater** value determines the number of the first 'N' variables in the variable list of the request PDU for which a simple `Get` operation must be performed.

  The `Get_Bulk` operation is a result of a combination of `Get` operations for the first 'N' (**Non-Repeater**) variables in the request PDU and `Get_Next` continuous operation for each of the remaining 'R' variables in the variable list. The `Get_Next` operation must be repeated for 'M' (**Max-Repetitions**) number of times for each of the 'R' variables.

`Set_Request`

- (PDU Type = 0x03)

  `Set_Request` PDU is generated and transmitted from an SNMP Client or the management station. The PDU contains the variable and the corresponding value to be set to the variable in the agent. The `Set_Request` will be processed at the agent if it is originated from the source, which is part of the group/community an agent belongs to. Otherwise, the agent can generate an `AuthenticationFailure Trap`, if configured so. The variable to be set has to be of the read/write type.

  Depending upon the data type, variable type and access mode, the agent can generate the **notWritable, wrongType, wrongLength, wrongEncoding, wrongValue, noCreation, inconsistentName, inconsistentValue, resourceUnavailable, genErr, NoError, commitFailed** and **undoFailed** errors in the response PDU.

`Trap` PDU

- (PDU Type = 0x04)

  A `Trap` PDU is generated and transmitted by an agent if an exception occurs. The exception has to be defined with the agent and the `Trap` destination IPs to be configured with. There are different types of `Trap`(s):

  - **`ColdStart`** `Trap` (Type = 0): If the agent is reinitializing itself with either the agent's configuration or the protocol entity implementation is altered.
  - **`WarmStart`** `Trap` (Type = 1): If the agent is reinitializing itself with neither the agent's configuration, nor the protocol entity, the implementation is altered.
  - **`LinkDown`** `Trap` (Type = 2): If one of the communication links of the agent fails, the `Trap` PDU will have the name and the instance of the interface instance that failed.
  - **`LinkUp`** `Trap` (Type = 3): Indicates that one of the communication links of the agent has come up.
  - **`AuthenticationFailure`** `Trap` (Type = 4): Signifies that an unauthenticated request has come for private variable access from a non-member of the community. This `Trap` adds the security to the agent. In this case, the agent can be configured to hibernate to protect from any unauthorized access or snooping.
  - **`EgpNeighborLoss`** `Trap` (Type = 5): Signifies that an Exterior Gateway Protocol (EGP) neighbor, with whom the agent had an EGP link, is no longer in the link and the peer relationship is no longer obtained.
  - **`EnterpriseSpecific`** `Trap` (Type = 6): Signifies that the sending agent has encountered an enterprise-specific event. The `Trap` field will have the code of the event that occurred.

`Get_Response`

- (PDU Type = 0x02)

  The response PDU is generated by the SNMP Agent once it receives a `Get_Request`, `Get_Next_Request`, `Get_Bulk_Request` or `Set_Request` PDU.

  If the error status field of the response PDU is not zero, the value fields of the variable bindings in the variable bindings list are ignored on the manager's side.

  If the **error-status** and **error-index** fields are both non-zero, then the **error-index** value is the index of the variable in the variable binding for which the request processing failed. The SNMP Manager/Client NMS should be able to properly handle errors, such as **noSuchName**, **badValue**, **readOnly**, etc.

SNMP PDU PROCESSING

The actual SNMP Agent is implemented by several files working together with the Microchip TCP/IP Stack. The SNMP Agent and the corresponding APIs are implemented in `SNMP.c` and `CustomSNMPApp.c`. Apart from this, a few callback functions must also be implemented to provide communication among the SNMP module, the host application and the rest of the TCP/IP Stack.

The SNMP Application Program Interfaces (APIs) are well-defined methods for communicating between applications and the SNMP Agent, and are also designed to make application design easier for the user. Example 1 illustrates the PDU processing by the agent. This pseudocode enables the users to understand the execution flow at the Microchip SNMP Agent for the received request PDUs from the manager/client.

EXAMPLE 1: SNMP PDU PROCESSING FLOW – PSEUDOCODE

```
main()
{
    while(1)
    StackApplications();
}

StackApplications()
{
    #if defined(STACK_USE_SNMP_SERVER)
        SNMPTask();
    #endif
}

SNMPTask()
{
    //Read the request PDU and populate the PDU Info Data Base
    ProcessHeader(&pduInfoDB,community, &communityLen)
    {
      // This function populates response as it processes community string.
        IsValidCommunity(community, len);

      // Fetch and validate PDU type.
        IsValidPDU(&pdu);

      // Ask main application to verify community name against requested PDU type.
        SNMPValidateCommunity(community);
    }
}
```

EXAMPLE 1: SNMP PDU PROCESSING FLOW – PSEUDOCODE (CONTINUED)

```
//Process each of the variables in varbind list
ProcessVariables(&pduInfoDB,community, communityLen);
{
    while(1)
    {
        switch(StateMachineState)
        {
        // Before each variables are processed, prepare necessary header.
        case SM_PKT_STRUCT_LEN_OFFSET:
        case SM_RESPONSE_PDU_LEN_OFFSET:
        case SM_ERROR_STATUS_OFFSET :
        case SM_ERROR_INDEX_OFFSET :
        case SM_FIND_NO_OF_REQUESTED_VARBINDS:
            //Find number of OIDs/varbinds's data requested in received PDU.
            FindOIDsInRequest(varBindingLen.Val);
        case SM_FIND_NO_OF_RESPONSE_VARBINDS:
            //Calulate number of variables to be responded for the received request
            //Refer RFC 3416
            noOfVarToBeInResponse = Getbulk_N + (Getbulk_M * Getbulk_R);
        case SM_VARBIND_STRUCT_OFFSET:
        case SM_VARSTRUCT_LEN_OFFSET:
        case SM_POPULATE_REQ_OID:
            //Populate received pdu for the requested OIDs.
            IsValidOID(OIDValue, &OIDLen);
            //Verify if trying to access the private object
            if(SNMPCheckIfPvtMibObjRequested(OIDValue))
                if(SNMPValidateCommunity(rxedCommunityName)==READ_COMMUNITY ||
                WRITE_COMMUNITY
        case SM_FIND_OID_IN_MIB:
            //Search for the requested OID in the MIB database with the agent.
            OIDLookup(pduDbPtr,OIDValue, OIDLen, &OIDInfo);
            if(oidLookUpRet != TRUE )
                //Set the error index and error code accrodingly
            else
                //proceed to next state in the statte machine
        case SM_NON_REPETITIONS:
            /*Variables in get,get_next,set and get_bulk request(non-repetition
            variables)are processed in this part of the state machine.*/
            if(pduType == SNMP_SET)
                ProcessSetVar(pduDbPtr,&OIDInfo, &errorStatus);
            else if(pduType == SNMP_GET || pduType == SNMP_V2C_GET_BULK)
                ProcessGetVar(&OIDInfo,FALSE);
            else if(pduDbPtr->pduType == SNMP_GET_NEXT)
                ProcessGetNextVar(&OIDInfo);
            /*If the request command processing is failed, update the error status,
            error index accordingly and response pdu.*/
        case SM_MAX_REPETITIONS:
            /*Process each variable in request as Get_Next for Getbulk_M
            (Max_repetition) times */
            for(repeatCntr=0;repeatCntr<Getbulk_M;repeatCntr++)
            {
                //Process every veriable in the request.
                for(varBindCntr=0;varBindCntr<Getbulk_R;varBindCntr++)
                {
                    OIDLookup(pduDbPtr,OIDValue, OIDLen, &OIDInfo))
                    ProcessGetBulkVar(&OIDInfo, &OIDValue[0],&OIDLen,&successor);
                }
            }
        }
    }
}
```

ABSTRACT SYNTAX NOTATION (ASN) LANGUAGE

Each MIB variable contains several attributes, such as data type, access type and Object Identifier. SNMP uses a special language, called Abstract Syntax Notation Version 1 (ASN.1), to describe details 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 of the NMS and SNMP Agent software is designed to read ASN files and build MIBs accordingly. An example of a variable description in ASN.1 syntax is shown in Example 2.

There are commercially available ASN.1MIB builders that allow users to build ASN.1 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 14.

Example 2 provides the Microchip demo ASN.1 MIB file.

EXAMPLE 2: MICROCHIP SNMP ASN.1 MIB FILE

```
Microchip DEFINITIONS ::= BEGIN

IMPORTS
   enterprises, IpAddress, Gauge, TimeTicks     FROM RFC1155-SMI
   DisplayString                                FROM RFC1213-MIB
   OBJECT-TYPE                                  FROM RFC-1212
   TRAP-TYPE                                    FROM RFC-1215;

microchip                           OBJECT IDENTIFIER ::=  { enterprises 17095 }

product                             OBJECT IDENTIFIER ::=  { microchip 1 }
setup                               OBJECT IDENTIFIER ::=  { microchip 2 }
control                             OBJECT IDENTIFIER ::=  { microchip 3 }

   ON-OFF          ::=   INTEGER { ON(1), OFF(0) }

name    OBJECT-TYPE
   SYNTAX DisplayString
   ACCESS read-only
   STATUS mandatory
   DESCRIPTION
      "Name of product. e.g. PICDEM.net etc."
   ::= { product 1 }

version    OBJECT-TYPE
   SYNTAX DisplayString
   ACCESS read-only
   STATUS mandatory
   DESCRIPTION
      "Version string. e.g. 1.0"
   ::= { product 2 }

date    OBJECT-TYPE
   SYNTAX DisplayString
   ACCESS read-only
   STATUS mandatory
   DESCRIPTION
      "Date of version"
   ::= { product 3 }

ledD5 OBJECT-TYPE
    SYNTAX INTEGER { OFF(0), ON(1) }
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
        "D5 LED connected LATA2"
    ::= { control 1 }

ledD6 OBJECT-TYPE
    SYNTAX INTEGER { OFF(0), ON(1) }
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
        "D6 LED connected to LATA3"
    ::= { control 2 }
```

EXAMPLE 2: MICROCHIP SNMP ASN.1 MIB FILE (CONTINUED)

```
pushButton    OBJECT-TYPE
   SYNTAX INTEGER { OPEN(1), CLOSED(0) }
   ACCESS read-only
   STATUS mandatory
   DESCRIPTION
      "8-bit A/D value"
   ::= { control 3 }

analogPot0    OBJECT-TYPE
   SYNTAX INTEGER
   ACCESS read-only
   STATUS mandatory
   DESCRIPTION
      "10-bit A/D value"
   ::= { control 4 }

analogPot1    OBJECT-TYPE
   SYNTAX INTEGER
   ACCESS read-only
   STATUS mandatory
   DESCRIPTION
      "10-bit A/D value"
   ::= { control 5 }

lcdDisplay  OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..15))
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
        "Second line of LCD on PICDEM.net board"
    ::= { control 6 }

traps OBJECT-TYPE
    SYNTAX SEQUENCE OF TrapEntry
    ACCESS not-accessible
    STATUS mandatory
    DESCRIPTION
        "Trap table"
    ::= { setup 1 }
trapEntry OBJECT-TYPE
    SYNTAX TrapEntry
    ACCESS not-accessible
    STATUS mandatory
    DESCRIPTION
        "Single trap entry containing trap receiver info."
    INDEX { trapReceiverNumber }
     ::= { traps 1 }
```

EXAMPLE 2: MICROCHIP SNMP ASN.1 MIB FILE (CONTINUED)

```
trapEntry ::=
    SEQUENCE {
        trapReceiverNumber
            INTEGER,
        trapEnabled
            INTEGER,
        trapReceiverIPAddress
            IpAddress,
        trapCommunity
            DisplayString
    }
trapReceiverNumber  OBJECT-TYPE
    SYNTAX INTEGER (0.. 4)
    ACCESS not-accessible
    STATUS mandatory
    DESCRIPTION
        "Index of trap receiver"
    ::= { trapEntry 1 }

trapEnabled OBJECT-TYPE
    SYNTAX INTEGER { Yes(1), No(0) }
    ACCESS read-write
    STATUS mandatory
    DESCRIPTION
        "Indicates if this trap entry is enabled or not."
    ::= { trapEntry 2 }

trapReceiverIPAddress OBJECT-TYPE
    SYNTAX  IpAddress
    ACCESS  read-write
    STATUS mandatory
    DESCRIPTION
        "Trap receiver IP address"
    ::= { trapEntry 3 }

trapCommunity OBJECT-TYPE
    SYNTAX  DisplayString (SIZE (0..7))
    ACCESS  read-write
    STATUS mandatory
    DESCRIPTION
        "Trap community to be used by agent to send trap"
    ::= { trapEntry 4 }

END
```

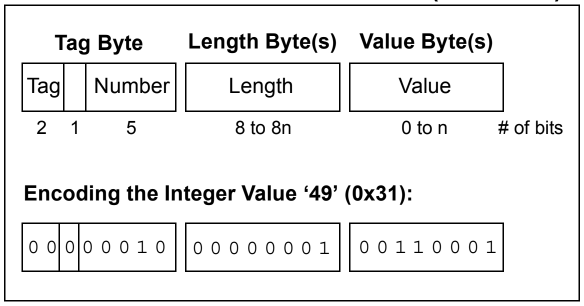
Binary Encoding Rules (BER)

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 the 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)



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 mib2bib (MIB to BIB) compiler (page 23).

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 page 14 through 26.

Example 3 shows a Microchip MIB file. In this example, seven separate items are being defined. In the script, "Microchip MIB Compiler (mib2bib)", a read-only node is being established at the OID of 43.6.1.2.1.1.5; it contains the identifier string, "`PICDEM.net 2`", as static information.

In the fourth section of the Microchip MIB script, "`microchip.control`", a node with dynamic LED5 status information is being established at the OID of 43.6.1.4.1.1.17095.3.1. The variable, called "`LED_D5`", is assigned an identifier of 1.

In section, "`microchip.setup`", a two-column, four-row data array is being created with the following variables:

- `TRAP_RECEIVER_ID`
- `TRAP_RECEIVER_ENABLED`
- `TRAP_RECEIVER_IP`
- `TRAP_COMMUNITY`

Note:
- Both the ASN.1 MIB file and the Microchip MIB script use the same `.mib` file extension, but the files have distinctly different purposes. The ASN.1 MIB file is used by the MIB browser (NMS) to properly display context for your application.
- The Microchip MIB script is compiled using mib2bib to create a BIB file. The BIB file is later converted using MPFS2 to store the MIB data for your application in internal Flash or EEPROM.

Example 3 provides the Microchip script demo, SNMP MIB file.

`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 `mib.h` header file. 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 the SNMP specification.
`GAUGE32`	4-byte `GAUGE32` data as defined by the 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 a single value.
`SEQUENCE`	If this variable contains an 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).

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`

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 the 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 need not implement callback logic to provide data for this OID variable.

Precondition

The given *`oidName`* must have been declared using the 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 a MAC type address:

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

`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 the application requires dynamic OID variables.

Syntax

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

Parameters

`oidName`

Name of OID variable that is being declared as 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 the 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 the given *`oidName`* as a dynamic variable. An entry will be created in the `mib.h` header file of the form:

```
#define oidName id
```

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

Precondition

The given *`oidName`* must have been declared using the 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 contain an array of values and any instance of its values can be referenced by an 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 the 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 an *`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 a *`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 the given *`oidName`* as a dynamic variable.

Precondition

A given *`oidName`* must have been declared using a previous `DeclareVar` command with an *`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

This command assigns a previously declared OID variable of type OID as an Agent ID for this SMNP Agent. The OID variable defined to be an Agent ID must be supplied in the `SNMPNotify` function to generate a `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 an *`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 the given *`oidName`* as a dynamic variable.

Precondition

The given *`oidName`* must have been declared using a previous `DeclareVar` command with an *`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)
```

EXAMPLE 3: MICROCHIP SNMP MIB SCRIPT

```
****************************************************************************
*    MIB-2 SYSTEM MIB
****************************************************************************
$DeclareVar(SYS_NAME, ASCII_STRING, SINGLE, READONLY, 43.6.1.2.1.1.5)
$StaticVar(SYS_NAME, PICDEM.net 2)

$DeclareVar(SYS_DESCR, ASCII_STRING, SINGLE, READONLY, 43.6.1.2.1.1.1)
$StaticVar(SYS_DESCR, Microchip TCP/IP stack running SNMP Agent)

$DeclareVar(SYS_CONTACT, ASCII_STRING, SINGLE, READONLY, 43.6.1.2.1.1.4)
$StaticVar(SYS_CONTACT, techsupport@microchip.com)

$DeclareVar(SYS_LOCATION, ASCII_STRING, SINGLE, READONLY, 43.6.1.2.1.1.6)
$StaticVar(SYS_LOCATION, Near Your Desk)

$DeclareVar(SYS_SERVICES, BYTE, SINGLE, READONLY, 43.6.1.2.1.1.7)
$StaticVar(SYS_SERVICES, 7)

$DeclareVar(SYS_UP_TIME, TIME_TICKS, SINGLE, READONLY, 43.6.1.2.1.1.3)
$DynamicVar(SYS_UP_TIME, 250)

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

This declaration is must if this agent is going to send traps out.
Application must pass this OID name as one of the parameter when generating
trap.  Without a valid AgentID definition, SNMPNotify would fail.
$AgentID(MICROCHIP, 255)
****************************************************************************
*    END OF MIB-2 SYSTEM MIB
****************************************************************************

****************************************************************************
*    MICROCHIP - PICDEM.net MIB
****************************************************************************

----------------------------------------------------------------------------
-    microchip.product
----------------------------------------------------------------------------
$DeclareVar(PRODUCT_NAME, ASCII_STRING, SINGLE, READONLY, 43.6.1.4.1.17095.1.1)
$StaticVar(PRODUCT_NAME, Microchip SNMP Agent)

$DeclareVar(PRODUCT_VERSION, ASCII_STRING, SINGLE, READONLY, 43.6.1.4.1.17095.1.2)
$StaticVar(PRODUCT_VERSION, v1.0)

$DeclareVar(VERSION_DATE, ASCII_STRING, SINGLE, READONLY, 43.6.1.4.1.17095.1.3)
$StaticVar(VERSION_DATE, May 2003)
```

EXAMPLE 3: MICROCHIP SNMP MIB SCRIPT (CONTINUED)

```
-----------------------------------------------------------------------------
-    microchip.setup
-----------------------------------------------------------------------------
TRAP RECEIVER is table with following format:
    TRAP_RECEIVER_ID is index.

 -----------------------------------------------------------------------------
 TRAP_RECEIVER_ID  |    TRAP_ENABLED  |  TRAP_RECEIVER_IP  |      TRAP_COMMUNITY
 -----------------------------------------------------------------------------
    0..3           |     OFF(0)/ON(0) |  X.X.X.X           |     ASCII_STRING(0..7)
 ------------------------------------------------------------------------------

$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)


-----------------------------------------------------------------------------
-    microchip.control
-----------------------------------------------------------------------------
microchip.control.ledD5
$DeclareVar(LED_D5, BYTE, SINGLE, READWRITE, 43.6.1.4.1.17095.3.1)
$DynamicVar(LED_D5, 5)

microchip.control.ledD6
$DeclareVar(LED_D6, BYTE, SINGLE, READWRITE, 43.6.1.4.1.17095.3.2)
$DynamicVar(LED_D6, 6)

microchip.control.pushButton
$DeclareVar(PUSH_BUTTON, BYTE, SINGLE, READONLY, 43.6.1.4.1.17095.3.3)
$DynamicVar(PUSH_BUTTON, 7)

microchip.control.analogPot0
$DeclareVar(ANALOG_POT0, WORD, SINGLE, READONLY, 43.6.1.4.1.17095.3.4)
$DynamicVar(ANALOG_POT0, 8)

microchip.control.lcdDisplay
$DeclareVar(LCD_DISPLAY, ASCII_STRING, SINGLE, READWRITE, 43.6.1.4.1.17095.3.6)
$DynamicVar(LCD_DISPLAY, 10)

*****************************************************************************
*    END OF MICROCHIP - Demo MIB
*****************************************************************************
```

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 the Microsoft® Windows® operating system. 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 (MPFS2) 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 `snmp.bib` and `mib.h` output files in the same directory.

On the contrary, the command:

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

compiles the `MySNMP.mib` script file and overwrites the existing output files. It also specifies that the `snmp.mib` file is located in the subdirectory, `WebPages`. Because it is not 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, and the output file size. A typical display following a successful run is shown in Example 4.

The MIB compiler is a simple rule script compiler. While it can detect and report many types of parsing errors, it has 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 the error. A list of errors, with their explanations, is provided in Table 1.

EXAMPLE 4: TYPICAL OUTPUT DISPLAY FOR A mib2bib COMPILATION

```
C:\Microchip Solutions\Microchip\TCPIP Stack\Utilities>mib2bib.exe snmp.mib
mib2bib  v1.0.1 (Oct 14 2003)
Copyright (c) 2003 Microchip Technology Inc.

Input MIB File : C:\Microchip Solutions\Microchip\TCPIP Stack\Utilities\snmp.mib
Output BIB File: C:\Microchip Solutions\Microchip\TCPIP Stack\Utilities\snmp.bib
Output Inc File: C:\Microchip Solutions\Microchip\TCPIP Stack\Utilities\mib.h

BIB File Statistics:

    Total Static OIDs         : 9
       Total Static data bytes: 57
    Total Dynamic OIDs        : 10
    (mib.h entries)
       Total Read-Only OIDs   : 3
       Total Read-Write OIDs  : 7
-------------------------------------------
    Total OIDs                : 19

    Total Sequence OIDs       : 4
    Total AgentIDs            : 1
===========================================
    Total MIB bytes           : 224
    (snmp.bib size)
```

TABLE 1: mib2bib RUN-TIME ERROR CODES

Error	Description	Reason
1000	Unexpected End-Of-File (EOF) found	End-Of-File was reached before the end of command.
1001	Unexpected End-Of-Line (EOL) found	End-Of-Line was reached before the 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 `$DeclareVar(`*`oidName, dataType, oidType, accessType, oidType`*`)`	—
1009	Duplicate OID name found	Specified OID name is already in use.
1010	Unknown data type received	Data type keyword does not match one of the allowed keywords.
1011	Unknown OID type received	OID type keyword does not match one of the allowed keywords.
1012	Empty OID string received	—
1013	Invalid parameters: expected `$DynamicVar(`*`oidName, id`*`)`	—
1014	OID name is not defined	—
1015	Invalid OID ID received – must be between 0-255 inclusive	—
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 strings must contain the 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.

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

Error	Description	Reason
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 the Internet root. The Internet root of '43' must be used if this is an Internet MIB	All internet OIDs must start with '43'. This is a warning only and will not stop script generation.
1040	The given value was truncated to fit in a specified data type	An OID was declared as `BYTE` or `WORD` but the value given in `StaticVar` exceeded the data range.
1041	The given string exceeds a 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 MPFS2 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	*`bIsDistantSibling`*	Node has distant sibling
1	*`bIsConstant`*	Node has constant data
2	*`bIsSequence`*	Node is a sequence
3	*`bIsSibling`*	Node has a sibling
4	*`bIsParent`*	Node is a parent
5	*`bIsEditable`*	Node is writable
6	*`bIsAgentID`*	Node is an Agent ID variable
7	*`bIsIDPresent`*	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 the 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 provided 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 *`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`).

COMPILING 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 **"SNMP Protocol Overview"** section on page 2.

The flowchart in Figure 7 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 Microchip custom MIB script (page 14), define the enterprise-specific private MIB. The ASN.1 standard MIB file should also be defined, which maps to the custom script.
3. Use the included MIB compiler (mib2bib, page 23) to build a binary MIB image ("BIB").
4. Include the generated BIB file into an MPFS2 image and either download or link the MPFS2 image data file. For more information on how to use the MPFS2 tool, refer to the "**Getting Started>>Uploading Web Pages**" section of Microchip `TCPIP Stack Help.chm`.
5. Create an application project that contains all of your required files, plus the following Microchip TCP/IP Stack and SNMP Agent files:
   - `ETH97J60.c/h` (MAC driver layer for Microchip PIC18F97J60 family TCPIP Stack), `ENC28J60.c/h` (MAC driver layer for Microchip 10Base-T stand-alone Ethernet controller), `ENCX24J600.c/h` (MAC driver layer for Microchip 10/100Base-T stand-alone Ethernet controller) or ZeroG wireless PHY and MAC related files if using a wireless WiFi solution.
   - `ARP.c`
   - `IP.c`
   - `UDP.c`
   - `SNMP.c`
   - `CustomSNMPApp.c`
   - `StackTsk.c`
   - `MPFS2.c`
   - `MPFSImg2.c` (to save the MIB file image, `MPFSImg2`, in the code memory)
   - `Helpers.c`
   - `Delay.c`

   Use `TCPIP Demo App-CXX` according to the selected host controller (C18, C30 or C32).

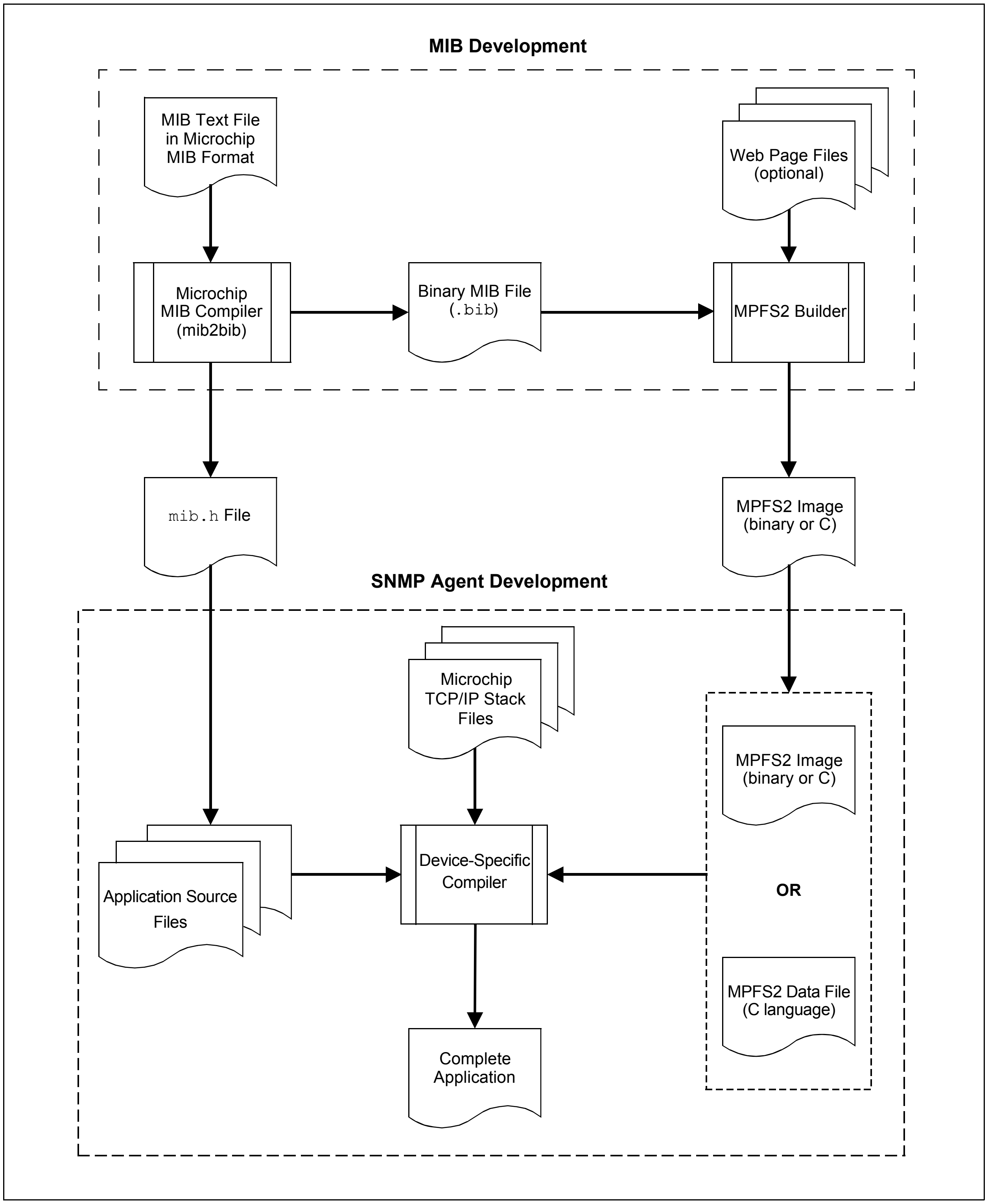
   It contains all of the required files and also the Microchip TCPIP Stack and SNMP Agent files.

Note:	The other Microchip files may have to be included depending on the other modules that are selected.

6. Modify your main application source file to include the SNMP header files and the MIB definition file, and implement the SNMP callback functions.

Once successfully built, you can use any standard SNMP management software or NMS software to access your SNMP Agent device. Users can develop their own customized NMS software to monitor and control their SNMP Agent.

FIGURE 7: OVERVIEW OF THE SNMP AGENT DEVELOPMENT PROCESS


MIB Development
MIB Text File in Microchip MIB Format
Web Page Files (optional)
Microchip MIB Compiler (mib2bib)
Binary MIB File (.bib)
MPFS2 Builder
mib.h File
MPFS2 Image (binary or C)
SNMP Agent Development
Microchip TCP/IP Stack Files
MPFS2 Image (binary or C)
OR
MPFS2 Data File (C language)
Application Source Files
Device-Specific Compiler
Complete Application

DEMO SNMP AGENT APPLICATION

To better demonstrate the abilities of the SNMP Agent, the TCP/IP stack is provided with a demo application. You can find the *"SNMPv2c Getting Started Guide"* at `C:\Microchip Solutions\Microchip\Help`. Using Microchip's PICDEM.net™ 2 or Explorer 16 Development Board as a hardware platform, it allows the user to control the board parameters 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
- Facilitates retrieval of a large amount of information from the agent by supporting `Get_Bulk` operation
- Implements DHCP to obtain automatic IP address
- Other configuration parameters
- MIB storage location selection

  The following macros decide the storage location for the MIB image (MPFSImg2):

  - `MPFS_USE_EEPROM` (MPFSImg2 gets stored in external EEPROM)
  - `MPFS_USE_SPI_FLASH` (MPFSImg2 gets stored in external SPI Flash)

    The demo SNMP application requires that one of the above macros be defined or both commented in `TCPIPConfig.h`. If both the macros are commented, then the internal Flash is selected as the storage media for the MPFS2 image.

- `STACK_USE_SNMP_SERVER` (enables SNMP server)

Once a HEX file is built or selected, follow the standard procedure for your device programmer when programming the microcontroller.

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

TCPStack Vx.xx <IP address>

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

Once programmed, the application may still need to be configured properly before it is put on a real network.

Refer to the **"Getting Started>>Connecting to the Network"** section in the Microchip `TCPIP Stack Help.chm` file.

Downloading the MPFS2 Image

The Microchip File System (MPFS2) allows users to store binary image information for Stack related components in memory. The software utility, `mib2bib.exe`, for creating MPFS2 binary images comes with the Stack.

Users can store their MIB information (in BIB format) in memory using MPFS2. The SNMP demo application includes an MPFS2 binary image named, `mpfsimg2.bin`, which contains the MIB in binary format.

Refer to the "**Getting Started>>Uploading Web Pages"** section in the Microchip `TCPIP Stack Help.chm` file.

Using NMS Software with the SNMP Agent and Microchip MIB

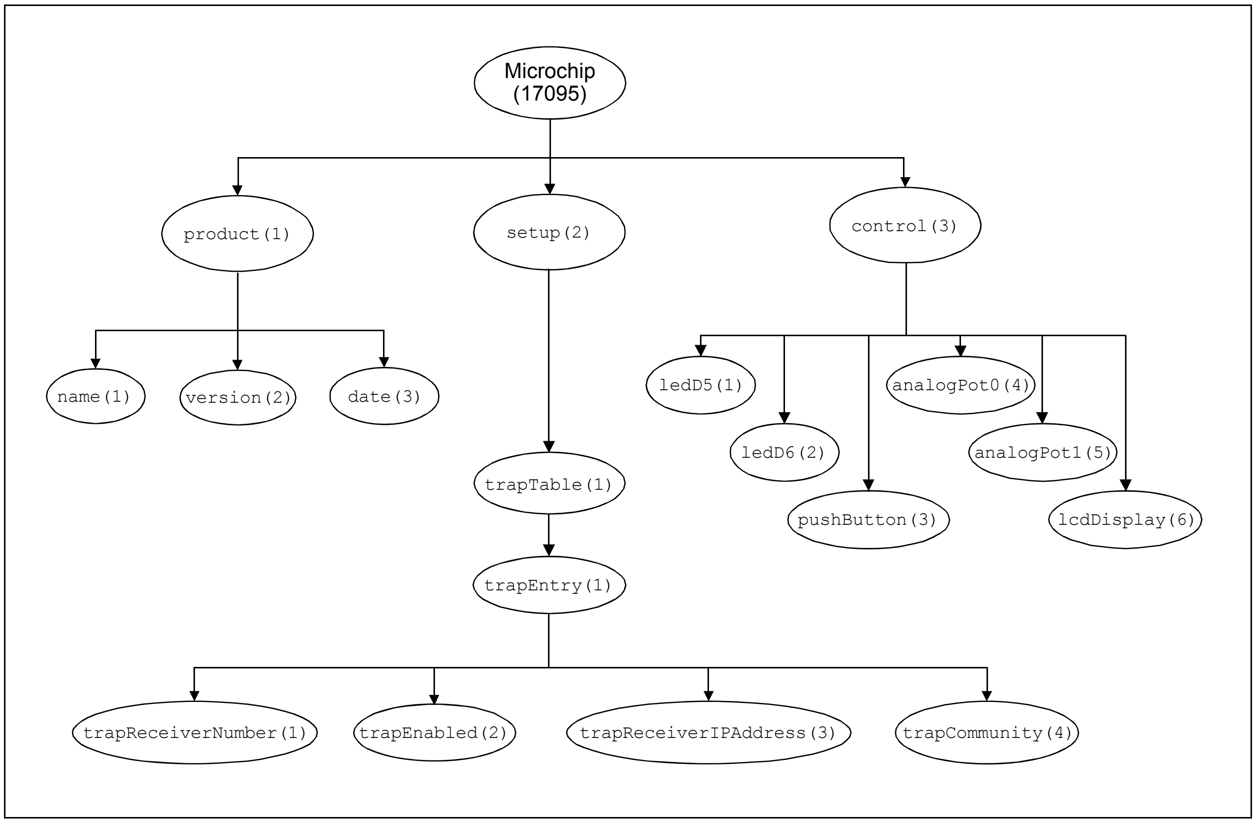
The demo application includes a MIB definition file written in ASN.1 syntax. This file, `mchip.mib`, defines the SMI for the development board's private Microchip MIB; it is also the basis for the MIB in the MPFS2 image. Figure 8 shows the full tree view of the MIB.

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 SNMP Agent's demo application. Refer to the "*SNMPV2c Getting Started Guide*", distributed with the Microchip TCPIP Stack, to know how to upload a MIB file to a particular MIB browser, how to access the MIB variables, experience a `Trap` demo and to understand how the SNMP Agent responds to third-party NMS.

There are several SNMP NMS Manager/Client software vendors, such as SNMPc Manager from CastleRock Computing, LoriotPro from LUTEUS, iReasoning MIB Browser and Trap Receiver, and so on.

Note: The list of NMS software vendors provided here is for reference only. There are many other commercial NMS software available from different vendors. Users can also develop their own customized NMS software. Which NMS software to be used is solely at the users discretion and pertaining to application requirements.

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 development 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 header file, `SNMP.h`. Once a `Trap` table entry is created with `TrapEnabled` set (= `1`), the development 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 development 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 date (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: `1` = On `0` = Off
`LedD6`	Read/Write, Integer	Switch on/off LED, D6: `1` = On `0` = 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
`LcdDisplay`	Read/Write, 16 Characters Long String	Reads/writes first line of on-board LCD

Experimenting with the SNMP Agent Demo Application

ADDING STATIC VARIABLE OIDs TO THE MIB

Any number of static OIDs can be added to the MIB without making any changes to the SNMP Agent source files.

- Define the new static variable to both the ASN.1 MIB and Microchip MIB scripts.
- Create a new BIB file with the mib2bib compiler.
- Include this file in the MPFS2 image and download the new image into the EEPROM.

ADDING DYNAMIC VARIABLE OIDs TO THE MIB

- Define the dynamic OID to both the ASN.1 MIB and Microchip MIB scripts.
- Compile the Microchip MIB script using the mib2bib tool. The new header file, `MIB.h`, is generated. Note the new dynamic or sequence variable ID defined in the `MIB.h` file by the mib2bib tool.
- Now, change the `CustomSNMPApp.c` file.
- Make corresponding changes in the `SNMPGetVar()`, `SNMPGetNextIndex()` and `SNMPSetVar()` callback functions to accommodate these new dynamic or sequence variable IDs. Map them to the respective variable's RAM value.
- Build the project and program the microcontroller.

Those already familiar with the Microchip TCP/IP Stack and its accompanying HTTP2 server should be aware of incorporating the web server pages and the MIB for the SNMP Agent into a single MPFS2 image (ensure that there is enough room in the EEPROM for everything). The process assumes that the files for the Stack are already installed and the files for the web pages are in the `WebPages2` directory.

First, generate the SNMP MIB's BIB image as mentioned on page 23; use the command prompt:

```
mib2bib /q snmp.mib /b=WebPages2
```

This writes `snmp.bib` to the directory, `WebPages2` (the `mib.h` header file will be written to its default directory).

CONFIGURING A NEW `Trap`

`Trap` is an event notification from the SNMP Agent to the Manager/Client NMS when a predefined event occurs at the agent.

The event should already be defined by the user in the application. The event can be an analog potentiometer value, thermostat reading, counter value crossing the threshold value, an `AuthenticationFailure Trap` in case of access of MIB variables with a wrong community name or a push of a button on the target board, etc.

To enable the agent to send the `Trap` to the manager:

1. Comment the following macro in the `TCPIPConfig.h` file,

   ```
   #define SNMP_TRAP_DISABLED
   ```

2. Configure the destination addresses for the `Trap` PDU (`Trap` capture utility's/manager's IP address); use the `SET` feature from the manager in the agent's `Trap` information table (MIB variable: `trapReceiverIPAddress`).
3. Enable sending the `Trap` for the particular destination (MIB variable: `trapEnabled`).
4. Configure the `Trap` community name for the individual destination IP addresses. The community name in the `Trap` message is the community name for the manager to receive the `Trap` with. Manager's `Trap` capture utility will not process the received `Trap` PDU destined to its address if the community name does not match. Depending on the community name, the received PDU is forwarded from the manager to the application, which processes the `Trap`(s) for the particular community.

   (MIB variable: `trapCommunity`)

5. Enable the `Trap` capture feature for the manager. For a few SNMP Managers, the `Trap` capture is enabled by default and the captured `Trap` information can be seen as the `Trap` is received. For some managers, it has to be enabled or a separate `Trap` capture utility has to be installed.

To demonstrate the Microchip `Trap` demo applications, refer to "*SNMPv2c Getting Started Guide*" in Microchip TCP/IP Stack Version 5.00 and later (**C:\Microchip Solutions\Microchip\Help**).

ADDING A Trap HANDLER

This section provides the information about what functions must be used to add a Trap handler. It also explains the Trap processing at the Microchip SNMP Agent.

EXAMPLE 5: ADDING A Trap HANDLER

```
main()
{
    #if defined(STACK_USE_SNMP_SERVER) && !defined(SNMP_TRAP_DISABLED)
        SNMPTrapDemo();     /*Trap Demo for (Analog Pot0 value > 512, or if BUTTON3 on the dev
                                 board is Pushed*/

        /*"gSendTrapFlag" GLobal flag is SET if the predefined event occurs. It is getting SET
            for MIB variable access with wrong community name and 'AuthenticationFailure' trap
            is generated.*/
        if(gSendTrapFlag)
        {
            SNMPSendTrap(); //Prepares Trap PDU for the event
        }
    #endif
}
```

In the above code snippet:

- The SNMPTrapDemo(); generates the Trap PDU if the demo event occurs. Users can add their Trap-specific events in this function definition.
- The SNMPSendTrap(); generates the Trap PDU depending on the below global variable values.
- The gSendTrapFlag is the global flag to send the Trap, SET, if the event occurs.

  BYTE gSendTrapFlag=FALSE;

- gOIDCorrespondingSnmpMibID is the MIB variable ID from MIB.h, generated by mib2bib.exe. This is used to search the variable OID in the MIB to be sent in the Trap PDU. If the event is not for a particular MIB variable, then the Enterprise OID can be sent: BYTEgOIDCorrespondingSnmpMibID= MICROCHIP;
- For the user-defined events, this gGenericTrapNotification type should always be ENTERPRISE_SPECIFIC BYTE gGenericTrapNotification= ENTERPRISE_SPECIFIC;
- gSpecificTrapNotification is a vendor-specific Trap code for the particular event or the event Trap ID assigned by vendor. This is the ID of the variable by which manager implementation can understand the event type.

  BYTE gSpecificTrapNotification= VENDOR_TRAP_DEFAULT;

  Define the event-specific ID in the 'enum' in the SNMP.h file as shown below:

```
typedef enum
{
  VENDOR_TRAP_DEFAULT =0x0,
  BUTTON_PUSH_EVENT   =0x1,
  POT_READING_MORE_512=0x2
}VENDOR_SPECIFIC_TRAP_NOTIFICATION_TYPE;
```

Update the above global variables in the application code where it is required to generate a Trap. The SNMPSendTrap() will make sure that the Trap is sent to the preconfigured IP addresses in the Trap information table.

The following pseudo code depicts a typical implementation example to add a `Trap` handler:

EXAMPLE 6: ADDING A `Trap` HANDLER EXAMPLE

```
main()
{while(1){
    SNMPTrapDemo();//Use any one of these two functions to define your Trap Event.
    SNMPSendTrap();}
}

/*This function handles the Trap Event definition, Trap PDU generation and transmission
  together.*/
SNMPTrapDemo()
{
    if(AnalogPot0 >512) //Analog Potentiometer Event
    {
        gSendTrapFlag=TRUE;
        gSpecificTrapNotification=POT_READING_MORE_512;
        gGenericTrapNotification=ENTERPRISE_SPECIFIC;

        //Make and send Trap PDU
        SendNotification(anaPotNotfyCntr, ANALOG_POT0, analogPotVal);
    }
    if(BUTTON3_IO == 0) //Push Button Event
    {
        gSendTrapFlag=TRUE;
        gSpecificTrapNotification=BUTTON_PUSH_EVENT;
        gGenericTrapNotification=ENTERPRISE_SPECIFIC;
        SendNotification(buttonPushNotfyCntr, PUSH_BUTTON, buttonPushval);
    }
    if(User_Defined_Event_Happened)//User event definition can be added here
    {
        gSendTrapFlag=TRUE;
        gSpecificTrapNotification=User_define_event;
        gGenericTrapNotification=ENTERPRISE_SPECIFIC;
        sendNotification();
    }
}
```

Users can also use the following function to send `Trap` notification; the event is defined in a separate function. The global variables, as discussed above, are updated with the event information.

If the configured destination manager IP address is not reachable, this notification function will retry for a maximum of 5 seconds before discarding the transmission of the `Trap` PDU.

EXAMPLE 7: `Trap` NOTIFICATION

```
/*This function handles only trap generation and transmission. The global variables for the trap
notification and transmission are updated in the application code. This will send the trap only
in the next iteration on the super while(1) loop in the main() function.*/

void SNMPSendTrap(void)
{

    switch(smState)//State Machine
    {

        case SM_PREPARE:
            //Prepare the TRAP PDU
            SNMPNotifyPrepare(remHostIpAddrPtr,trapInfo.table[receiverIndex].community,
                                trapInfo.table[receiverIndex].communityLen,
                                MICROCHIP,                       // Agent ID Var
                                gSpecificTrapNotification,   // Notification code.
                                SNMPGetTimeStamp());
            smState++;
            break;

        case SM_NOTIFY_WAIT:
            /*Check if the destination is available*/

            if(SNMPIsNotifyReady(remHostIpAddrPtr))
            {
                smState = SM_PREPARE;
                val.byte = 0;
                receiverIndex++;

            //application has to decide on which SNMP var OID to send. Ex. PUSH_BUTTON
                SNMPNotify(gOIDCorrespondingSnmpMibID, val, 0);//Send TRAP PDU
                smState = SM_PREPARE;
                UDPDiscard();
                break;
            }
    }

    //Try for max 5 seconds to send TRAP, do not get block in while()
    if((TickGet()-TimerRead)>(5*TICK_SECOND)|| (receiverIndex == TRAP_TABLE_SIZE))
    {
        UDPDiscard();
        smState = SM_PREPARE;
        gSendTrapFlag=FALSE;
        return;
    }

}
```

GENERATING AN MPFS2 IMAGE

Microchip provides the MPFS2 image generator tool, `MPFS2.exe`, included with the TCPIP Stack. This tool is available in the **Utilities** directory.

1. Set the input directory path of all files to be converted to the MPFS2 format in **Source Settings**.
2. Set **Processing Options** for the image to be generated.
3. Set the destination path for the generated `MPFSImg2.bin` in **Output Files**.

   This tool also uploads the `mpfsupload` bin file to the target board using the HEX image upload capability of the Stack.
4. The corresponding setting should also be made in **Upload Settings** of the tool.

   This image is copied to the external EEPROM.

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 that can run on any of the 8/16/32-bit PIC® microcontrollers. Its ability to run independently of an RTOS or application makes it versatile, while its ability to handle up to 256 dynamic variable OIDs and an unlimited number of static OIDs makes it flexible.

REFERENCES

- Internet Engineering Task Force (IETF)
  - RFC 1157 (for SNMP V1)
  - RFC 3416 (for SNMP V2c)
- 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.
- 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 is available with all the necessary source and support files in installer format for the following:

- Microchip SNMP Agent
- Microchip MIB Script Compiler (mib2bib)
- MPFS2 Image Builder

Refer to the Microchip `TCPIP Stack Help.chm` file for the descriptions of all the APIs of the modules. These files are required for development with the Microchip SNMP Agent. These archive files may be downloaded from the Microchip corporate web site at: www.microchip.com/tcpip.

NOTES:

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

Cleveland
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643

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: 91-80-3090-4444
Fax: 91-80-3090-4080

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: 886-3-6578-300
Fax: 886-3-6578-370

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

03/26/09