DMH Software
15 Arborwood Rd,
Acton, MA 01720 USA
Tel: 978-263-0526
Fax: 801-760-9746
info@dmhsoftware.com
support@dmhsoftware.com
http://www.dmhsoftware.com

21520 30th Drive SE #110
Bothell, WA 98021 USA
Tel: (425) 951-8086
Fax: (425) 951-8095
sales@datalight.com
support@datalight.com
www.datalight.com

SNMP Agent for ROM-DOS

WHITEPAPER SERIES

Overview

The Simple Network Management Protocol is a standard for monitoring and managing devices participating on communication networks. These devices come in various shapes and sizes with a host of hardware and operating system mixes. It is therefore necessary if they are to all being managed from a central computer to have a standard method of communication ported to each device. We can use a standard transmission layer common to virtually every connected device, which is the TCP/IP layer and add to that an SNMP Agent that is capable of communicating across TCP/IP to a central SNMP Manager application and performs the management tasks.

DMH Software offers an extensible SNMP Agent that has been ported to work with ROM-DOS / SOCKETS. The SNMP Agent SDK includes a SMIv2 MIB Compiler, which helps in the development of additional system specific MIBs.

Specific information about this solution may be obtained directly from DMH Software at www.dmhsoftware.com or email to info@dmhsoftware.com.

The remainder of this White Paper is an excerpt from the information available online through DMH Software:

SNMP Agent Architecture

The SNMP portable agent provides the basic software components implementing the SNMP protocol and MIB II. The system also provides tools for easy development of additional MIB's, namely the SNMP SMIv2 MIB compiler, which produces 'C' files from an ASN.1 definition, and the SNMP kernel services that interact with MIB modules.

Main Components

1. *The SNMP kernel*. This is the heart of SNMP. It processes incoming requests, identifies the various parts of a request, check for proper command syntax, and eventually interacts with the MIB modules to retrieve or modify the value of a requested object. The kernel interacts with the underlying datagram services subsystem (usually UDP/IP) for reception and transmission of SNMP datagrams.

2. *MIB-II*: In general, all MIB's interacting with the SNMP kernel are 'C' modules produced by a MIB compiler (part of the SNMP developing environment). The MIB compiler accepts as input an ASN.1 MIB definition, and produces as output a set 'C' and 'H' files with code implementing the given MIB. In the case of the MIB-II module, this is a set of files produced from the MIB-II ASN.1 definition, plus additional code added by *DMH Software* to facilitate the integration of MIB-II with the host-system.

3. *Additional MIB's*: Additional MIB's can easily be added to the system as explained in the section "Adding Additional MIB's". The process involves compiling the ASN.1 files defining the MIB's, inserting user code in the appropriate sections within the produced 'C' and 'H' files, and providing the API services as needed. Note that, as SNMP supports dynamic registration of MIB objects, there is no change to be done to the SNMP kernel in order to support additional MIB's (each MIB implementation file include the code for dynamic registration of the objects implemented in that file).

4. *Datagram Services*: An SNMP agent requires the services of a datagram module for reception and transmission of SNMP datagrams. Usually this module is UDP/IP (the User Datagram Protocol, see [13]), using UDP port 161; there exist implementations over other transport protocols, although these are less popular due to interoperability restrictions. SNMP is designed for integration with any datagram services module, as explained in the section "Integrating with the Underlying Datagram Services".

5. *MIB API (Application Interface)*: This is system specific code required meeting the requirements of MIB-II and other MIB's as those are added to the system. For example, in order to retrieve the object *sysUpTime,* SNMP looks for an API function called *GetTimeTicks* in the MIB API library provided by the host-system. In the section on "Implementing MIB-II objects" we detail how to implement the API module to fulfill the requirements of MIB-II objects. API modules for additional MIB's are specific to the MIB application; however, the section on "Adding Additional MIB's" provides examples of such API modules.

6. *Basic Services*: These are services needed by the SNMP kernel and the MIB modules regarding memory allocation, string formatting, memory copying, and console I/O for error reporting. Most systems already include services such as *malloc* and *sprintf*.

Model of Operation

Following is a procedural description of the handling of an SNMP command from the moment it is received by the Agent, to the production and sending of a response to the requester.

1. **A Manager sends SNMP request**. An SNMP request PDU is made by a manager and sent to the IP address of the Agent network entity. By definition, the destination port of the UDP datagram encapsulating the SNMP PDU is *SNMP_AGENT_PORT* (161).

2. **Agent receives UDP datagram**. The Agent receives the UDP datagram and checks the UDP destination port. If the destination port is *SNMP_AGENT_PORT* (161) the entry point function that processes the inbound SNMP request, "*SnRcvDatagram*()" is called. This function will process the SNMP request, build an SNMP reply and send it back to the requester. (Other UDP ports represent other applications such as TFTP, BOOTP, etc.).

3. **ASN.1/BER Parsing**. The ASN.1/BER parser parsers the encoded request and creates an SNMP PDU with a Variables link list in actual representation.

4. **Handle Authentication and Privacy**. Depending on whether an SNMPv1, SNMPv2c or SNMPv3 Message is being processed, several degrees of security are available, including privacy protocols (encryption) and authentication protocols. In SNMPv1 and SNMPv2c there is a trivial authentication mechanism called *community names*.

5. **Search the MIB tree.** SNMP maintains a MIB database with information on every SNMP object maintained by the system (e.g., registered to SNMP). Such information includes object name, type, associated access function, optional instance function, and the like. The SNMP kernel now searches the MIB tree to find an entry for the requested object (if this is a get-next request, the 'next' object is retrieved, or, for tables, a similar object with a 'next' instance value is retrieved). Note that only a subset of the whole MIB is “visible” to a given client application, based on the so-called “MIB view”.

6. **Evaluate the Instance.** This is performed for tabular objects only: tabular objects are conceptual collections of rows in a table, which are addressed by a part of the SNMP variable name called instance. For each such object, SNMP calls an "instance evaluator" routine associated with that object, which searches application tables for a row matching a given input instance (or the next one in case of Get-Next commands). The information is then maintained in a static global variable to be then used by the object access routines (see below). Note that if several objects are requested within the same row of a given table, the instance function for that row is called only once, saving the system unnecessary work.

7. **Object Access**. Depending on the command a read or write operation is performed:

    a. For Get and Get-Next commands (and for Get-Bulk in SNMPv2c), the access routine returns the data representing the object value at this time, possibly using instance information evaluated by the instance function. The retrieved data is encoded into a response PDU.

    b. For Set commands, the access routine is called twice, first to test if a given input value is acceptable, and then to commit (e.g., to actually set an internal object to a given input value). In both cases instance information evaluated by the instance function might be used.

8. **ASN.1/BER Build**. The actual representation of the SNMP PDU consisting of the header and the Variable linked list is now encoded back to ASN.1/BER machine independent representation.

    a. In case of a Get PDU, the response contains the retrieved data with the original object names.

    b. In case of a Get-Next PDU, the response contains the retrieved data with the updated object names reflecting the 'next' object relative to the original input name.

    c. In case of a Set PDU, the response is contains the original names and data.

    d. In case of a processing error (if a response is at all generated), the response contains the original names and data, with an error code indicating what the error was (e.g., no such object...).

9. **Add Authentication and Privacy**. In SNMPv1 and SNMPv2c add the trivial community name. In SNMPv3, provide optional encryption for privacy, and evaluate an authentication digest to ensure the destination can validate the response.

10. **Send the reply**. Finally the encoded, possibly secure ASN.1/BER data is encapsulated in a UDP datagram. The destination port is set to the UDP source port of the request. The UDP datagram is sent to the IP address of the requester.