KwikNet™
SNMP Agent

User's Guide

First Printing: February 15, 1999
Last Printing: November 1, 2000

Manual Order Number: PN303-9S

KADAK Products Ltd.
206-1847 West Broadway Avenue
Vancouver, B.C., Canada, V6J 1Y5
Phone: (604) 734-2796
Fax: (604) 734-8114

TECHNICAL SUPPORT

KADAK Products Ltd. is committed to technical support for its software products. Our programs are designed to be easily incorporated in your systems and every effort has been made to eliminate errors.

Engineering Change Notices (ECNs) are provided periodically to repair faults or to improve performance. You will automatically receive these updates for a period of one year. After that period, you may purchase additional updates. Please keep us informed of the primary user in your company to whom these update notices and other pertinent information should be directed.

Should you require direct technical assistance in your use of this KADAK software product, engineering support is available by telephone, fax or e-mail without charge. KADAK reserves the right to charge for technical support services which it deems to be beyond the normal scope of technical support.

We would be pleased to receive your comments and suggestions concerning this product and its documentation. Your feedback helps in the continuing product evolution.

KADAK Products Ltd.
#206 - 1847 West Broadway Avenue
Vancouver, B.C., Canada, V6J 1Y5

Phone: (604) 734-2796
Fax: (604) 734-8114
e-mail: amxtech@kadak.com

KwikNet SNMP Agent User's Guide
Table of Contents

KwikNet SNMP Agent User's Guide
Table of Contents (continued)

KwikNet SNMP Agent User's Guide
Table of Figures

1. KwikNet SNMP Agent Overview

1.1 Introduction

The Simple Network Management Protocol (SNMP) is a standard protocol used by network administrators to observe and control interconnected network devices. The SNMP administrator is called the SNMP manager. The network nodes for which the manager is responsible are called managed devices. Each such device includes an SNMP agent responsible for communication with the manager. The UDP protocol is used by the manager and agent for network communication.

The KwikNet™ SNMP Agent implements the SNMP protocol on top of the KwikNet TCP/IP Stack, a compact, reliable, high performance TCP/IP stack, well suited for use in embedded networking applications.

The KwikNet SNMP Agent is best used with a real-time operating system (RTOS) such as KADAK's AMX™ Real-Time Multitasking Kernel. However, the KwikNet SNMP Agent can also be used in a single threaded environment without an RTOS. The KwikNet Porting Kit User's Guide describes the use of KwikNet with your choice of RT/OS. Note that throughout this manual, the term RT/OS is used to refer to any operating system, be it a multitasking RTOS or a single threaded OS.

You can readily tailor the KwikNet stack to accommodate your SNMP needs by using the KwikNet Configuration Builder, a Windows® utility which makes configuring KwikNet a snap. Your KwikNet stack will only include the SNMP features required by your application.

This manual makes no attempt to describe the Simple Network Management Protocol (SNMP), what it is or how it operates. It is assumed that you have a working knowledge of the SNMP protocol as it applies to your needs. Reference materials are provided in Appendix A of the KwikNet TCP/IP Stack User's Guide.

The purpose of this manual is to provide the system designer and applications programmer with the information required to properly configure and implement a networking system using the KwikNet TCP/IP Stack and SNMP. It is assumed that you are familiar with the architecture of the target processor.

KwikNet and its options are available in C source format to ensure that regardless of your development environment, your ability to use and support KwikNet is uninhibited. The source program may also include code fragments programmed in the assembly language of the target processor to improve execution speed.

The C programming language, commonly used in real-time systems, is used throughout this manual to illustrate the features of KwikNet and its SNMP Agent.

1.2 General Operation

The Simple Network Management Protocol (SNMP) is a standard protocol used by network administrators to observe and control interconnected network devices. SNMP version 1 is formally defined by the IETF document RFC-1157. The KwikNet SNMP Agent is compliant with this specification. The RFC should be consulted for any detailed questions concerning the SNMP protocol. The KwikNet SNMP Agent implements the SNMP features typically required for use by a managed device in an embedded application.

SNMP uses the connectionless UDP transport protocol. One machine, the network SNMP manager, sends a request to another machine, the SNMP agent. The agent handles the request and, if necessary, sends a separate response message to the manager. The SNMP agent can also send a special message called a trap, to an SNMP manager, informing the manager of a specific event or error condition which has occurred within the managed device.

The SNMP manager sends each SNMP message in a UDP datagram to the well known SNMP port number 161 within the managed device. The SNMP agent's response, if any, is directed to the port identified by the manager in its request. The SNMP agent directs its trap messages to the well known SNMP trap port number 162.

The KwikNet SNMP Library provides the services needed to implement an SNMP agent capable of servicing requests from multiple SNMP managers.

Managed Data

Network devices are managed using a very simple strategy. An artificial element called a **variable** is assigned to each feature of a device which is subject to observation or control. Every such variable has an associated **value**. The SNMP manager monitors and controls the managed device by reading and writing these management variables.

The management variables are maintained within the managed device in a Management Information Base (MIB). Each variable is defined using a descriptive language referred to as the Abstract Syntax Notation (ASN.1). The definition gives the variable a name, specifies the type of value associated with the variable and identifies the operations which can be performed on the variable.

The MIB is organized in a tree-like structure with each variable sitting as a leaf on a branch of the tree. Chapter 1.6 describes the structure of the MIB in more detail.

The managed data supported by the KwikNet SNMP Agent conforms to the standards described in RFC-1155, RFC-1212 and RFC-1213. In particular, KwikNet provides built-in support for the MIB variables defined by RFC-1213 and used to monitor the TCP/IP stack and its related protocols.

The specification of a Management Information Base (MIB) is provided in a text file using ASN.1 notation. Unfortunately, the MIB definition itself is frequently referred to as a MIB. Within this manual, the term MIB will be reserved for the actual database accessed by the SNMP agent. The text file used to describe the elements of that MIB will be called a MIB Definition File.

Object Identifiers

A MIB variable is identified by an **object identifier** which uniquely specifies the location of that MIB variable in the Management Information Base. The object identifier is an ordered list of integer numbers separated from each other by the period (`'.'`) character. Each number in the object identifier is called a **sub-identifier**.

For convenience, each MIB variable is also given a human readable name in which the sub-identifiers in the object identifier are replaced by text strings. For example, the MIB-II variable *`icmpInErrors`* with object identifier *`1.3.6.1.2.1.5.2`* has been given the full name of *`iso.org.dod.internet.mgmt.mib-2.icmp.icmpInErrors`*. The text name for the MIB variable is used only in the MIB definition; it is NOT present in the actual MIB database. Hence the SNMP agent has no access to this MIB variable name.

Even the object identifier in its dot separated numeric form is not used by the SNMP agent. Instead, an object identifier is implemented as an array of sub-identifier values called a **subid array**. Since KwikNet can be configured to use either 16-bit or 32-bit sub-identifiers, the variable type *`knsa_subid`* is introduced in KwikNet header file *`KN_SNMP.H`*. Using this definition, a subid array is as an array of *`n`* sub-identifiers of type *`knsa_subid`*.

The KwikNet SNMP Agent also uses a private, compact form of the object identifier called a **compressed object identifier**. This compressed object identifier can be extracted into a subid array for manipulation by your application.

SNMP Communities

The term **community** is used to reference a group of SNMP managers and agents which collectively serve a common management purpose. A community can include a single manager and agent, one manager and several agents or several managers and agents. Managers and agents can each belong to more than one community.

For example, a pipeline managed by a single SNMP manager might have two communities. The pump community would include the manager and all pumping stations along the pipeline. The meter community would include the manager and all stations along the pipeline at which pipeline throughput could be measured. Most pumping stations would belong to both communities. However, many metering stations with no pumping capabilities would only belong to the meter community.

SNMP Messages (Requests, Responses and Traps)

The SNMP manager and agent communicate by sending SNMP messages to each other. Each SNMP message is delivered within a single UDP datagram. Each message consists of a version identifier, an SNMP community name and a **protocol data unit** (PDU).

The version identifier specifies the SNMP version which must be supported in order to decode the message. The KwikNet SNMP Agent only supports SNMP version 1.

The community name identifies the SNMP community to which the message applies. The message will be ignored if the recipient is not configured as a member of the community referenced in the message.

Finally, the PDU identifies the particular management variable to which the message applies and the operation, if any, to be performed by the message recipient. The PDU is used by the SNMP manager to read and write variables. The PDU is used by the SNMP agent to send the value associated with a variable to the SNMP manager which requested the value.

The PDU is also used by the SNMP agent to send a trap message to a particular SNMP manager. The recipient of the SNMP trap message is referred to as a **trap target**.

The protocol data units supported by KwikNet are summarized in Figure 1.2-1.

Protocol Data Unit	**Purpose**
`get-request`	Get the value of a MIB variable from the managed device
`get-next-request`	Get the value of the next MIB variable from the managed device
`set-request`	Set the value of a MIB variable in the managed device
`get-response`	Message from the managed device in response to a *`get-request`*, *`get-next-request`* or *`set-request`* PDU
`trap`	Report an event or error which occurred in the managed device

Figure 1.2-1 KwikNet SNMP Messages (PDUs)

SNMP Security

A limited security scheme is provided with SNMPv1. The definition of each MIB variable specifies whether or not that variable can be read and/or written. Each community to which the managed device belongs is also qualified as to whether or not that community can read and/or write any of the MIB variables present in the managed device. It is the KwikNet SNMP Agent which enforces the access rules summarized in Figure 1.2-2.

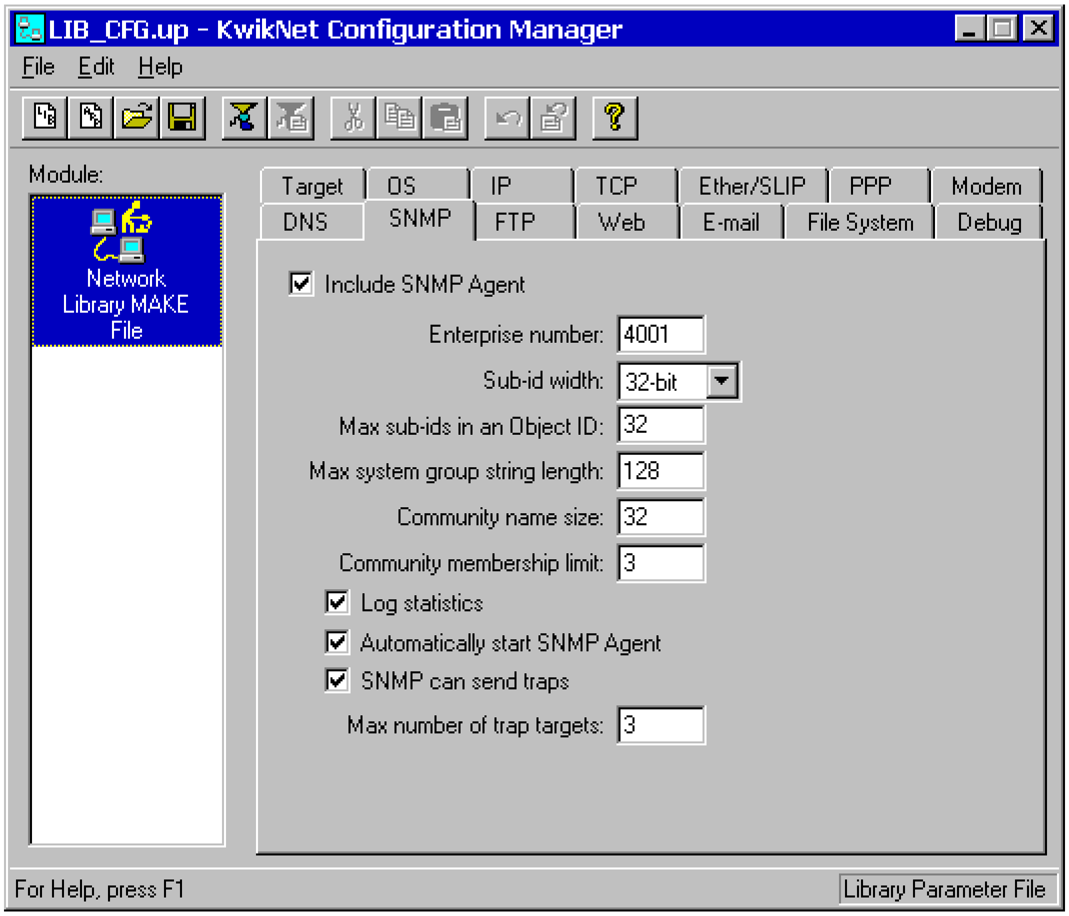
When the SNMP agent receives a request to fetch or modify the value of a MIB variable, it checks the access rights of the community and MIB variable specified in the request. If the community does not have access rights appropriate to the request, then the SNMP agent does not respond to the request. Instead, it sends an `authenticationFailure` trap message to all SNMP managers identified as trap targets to warn them of the unauthorized access attempt.

MIB Variable Access Rights	**Community Access Rights**			
	read	read/write	write	none
read	get	get	—	—
read/write	get	get, set	set	—
write	—	set	set	—
none	—	—	—	—

Figure 1.2-2 MIB Access Rights and Allowed Operations

1.3 KwikNet SNMP Library Configuration

You can readily tailor the KwikNet stack to accommodate your SNMP needs by using the KwikNet Configuration Builder to edit your KwikNet Library Parameter File. The KwikNet SNMP Library parameters are edited on the SNMP property page. The layout of the window is shown below.



☑ Include SNMP Agent

Enterprise number: 4001

Sub-id width: 32-bit

Max sub-ids in an Object ID: 32

Max system group string length: 128

Community name size: 32

Community membership limit: 3

☑ Log statistics

☑ Automatically start SNMP Agent

☑ SNMP can send traps

Max number of trap targets: 3

SNMP Library Parameters (continued)

Include SNMP Agent

Check this box to permit your application to act as an SNMP managed device. Otherwise, leave this box unchecked.

Enterprise Number

Specify the enterprise number used by your organization to reference your managed device. Enterprise numbers are integer values allocated by the Internet Assigned Numbers Authority (IANA). To apply for an enterprise number, contact the IANA at www.iana.org.

Sub-id Width

Specify the number of bits to be used for the sub-identifier numbers in a MIB object identifier. Pick either 16-bit or 32-bit from the pull down list. If you choose 16-bit sub-ids, the maximum sub-identifier value allowed in any MIB object identifer will be 65535. If you require sub-ids greater than 65535, you must use 32-bit sub-ids.

Although the use of 16-bit sub-ids will minimize the memory occupied by your MIB, you may pay a performance penalty if your memory system does not operate efficiently with 16-bit access. As a general rule, if you do not require sub-ids greater than 65535, use 16-bit sub-ids with 16-bit memory systems.

Maximum Sub-ids in an Object ID

Specify the maximum number of sub-identifiers required in any MIB object identifier which will be used to reference a MIB variable in your managed device. For example, 20 sub-identifiers exist in the object identifiers used to reference some of the TCP MIB variables specified by RFC-1213. A typical value is 32. Few managed devices will require a larger value.

Maximum System Group String Length

Specify the maximum number of bytes allowed in any of the string variables present in the MIB-II system group. The count must include the terminating `'\0'` character even though this character is never delivered within the PDU of an SNMP message.

SNMP Library Parameters (continued)

Community Name Size

Specify the maximum number of bytes in the string which provides the name of a community to which the managed device belongs. The count must include the terminating `'\0'` character even though this character is never included within an SNMP message.

Community Membership Limit

Specify the maximum number of communities to which your managed device can belong. The managed device must belong to at least one community in order to be accessible.

Log Statistics

Check this box if your application will use the KwikNet function *`kn_netstats()`* to generate a summary of the SNMP events monitored by the SNMP agent. The summary will be directed to the KwikNet logging device, if one has been provided. Otherwise, leave this box unchecked. Leaving this box unchecked reduces the code size by eliminating the service from the KwikNet SNMP Library.

Automatically Start SNMP Agent

Check this box if the KwikNet SNMP Agent is to be activated automatically by KwikNet when KwikNet is started. If this box is unchecked, the KwikNet SNMP Agent will remain idle until your application calls KwikNet procedure *`knsa_control()`* to start the agent.

SNMP Can Send Traps

Check this box if the KwikNet SNMP Agent is to be allowed to generate SNMP trap messages. Otherwise, leave this box unchecked. Leaving this box unchecked reduces the code size by eliminating all trap message generation logic from the SNMP procedures in the KwikNet SNMP Library.

Maximum Number of Trap Targets

If you enable SNMP trap generation, you must specify the maximum number of trap targets (SNMP managers) to which SNMP trap messages will be directed by the KwikNet SNMP Agent. You must allow at least one trap target.

1.4 SNMP Agent Operation

The KwikNet SNMP Library includes the SNMP agent and a collection of services for use by your application as it interacts with the agent. Services are provided to allow you to dynamically configure the SNMP agent and control its operation. Other services are available for use by your MIB access functions as described in Chapter 2.

When used with a real-time operating system such as KADAK's AMX Real-Time Multitasking Kernel or in a single threaded environment, the SNMP agent operates within the context of the KwikNet Task. The SNMP agent is initialized by the KwikNet Task as soon as KwikNet is operational. During this process, the SNMP agent calls function *`snmp_agentinfo()`* in source module *`SNMPUSER.C`* to fetch your custom definition of the managed device. The agent is then ready for use but remains idle until started by KwikNet or by your application.

If your KwikNet SNMP Library has been built with the auto-start feature enabled, the KwikNet Task will automatically start the SNMP agent as soon as it has been made ready for use.

If your KwikNet SNMP Library has been built with the auto-start feature disabled, your application must start the SNMP agent with a call to KwikNet procedure *`knsa_control()`*. By default, the SNMP agent will use your custom definition of the managed device from source module *`SNMPUSER.C`*. However, prior to starting the SNMP agent, you are free to inject an alternate definition or to adapt the default definition to your particular needs. Service procedures *`knsa_config()`*, *`knsa_community()`* and *`knsa_traptarget()`* in the SNMP Library are provided for this purpose.

Once started, the SNMP agent will service all SNMP requests directed to SNMP port number 161 at any of the IP addresses assigned to the KwikNet network interfaces.

SNMP traps, if enabled within your KwikNet SNMP Library, are handled by the SNMP agent as described in Chapter 1.5.

Your application can use KwikNet procedure *`knsa_control()`* to control the operation of the SNMP agent. You can disable and enable the servicing of SNMP requests by the SNMP agent, thereby easily adjusting the visibility of your managed device to all SNMP managers. You can also enable or disable the SNMP agent's ability to generate SNMP trap messages.

SNMP Agent Definition

Your SNMP managed device must appear as a unique, identifiable SNMP entity on the network. The SNMP agent must be able to provide a description of the device and to identify the communities to which the device belongs. If SNMP traps are to be supported, the SNMP agent must be able to identify all of the trap targets (SNMP managers) to whom the trap messages are to be directed.

When the SNMP agent is first initialized, it calls function *snmp_agentinfo()* in source module *SNMPUSER.C* to fetch your custom definition of the managed device. This function returns a pointer to an SNMP agent definition structure which provides the following information for the benefit of the SNMP agent.

MIB-II system group information
List of community definitions
List of trap targets

The MIB-II system group is a collection of MIB variables defined by RFC-1213. These variables provide an object identifier for the managed device, an indication of the services which the device offers and text strings which describe the managed device and its vendor. The values for these variables are provided in the default device description provided in source module *SNMPUSER.C*.

The list of communities defines the communities to which the managed device belongs. Each definition also specifies the MIB access rights granted to that community.

The list of trap targets defines the SNMP managers to which all SNMP generated trap messages will be sent. Each trap target definition specifies the IP address of an SNMP manager to whom the trap message is to be sent. The definition also specifies the community name to be presented in the trap message when it is sent to that particular trap target. If SNMP traps are not enabled within your KwikNet SNMP Library, the list of trap targets will be ignored.

You are free to edit the SNMP agent definition module *SNMPUSER.C* to meet the needs of your application. After editing the file, you must build your KwikNet SNMP Library to incorporate the revised module.

Once KwikNet has been started, your application may reconfigure the SNMP agent at any time using the KwikNet SNMP Library service procedures *knsa_config()*, *knsa_community()* and *knsa_traptarget()*.

1.5 SNMP Traps

An SNMP trap is an unsolicited signal from an SNMP managed device to an SNMP manager indicating that an event or error condition of possible interest has occurred. The signal is an SNMP trap message which is sent by the managed device to one or more SNMP managers called trap targets. SNMP trap messages are always directed to the well known SNMP trap port number 162 at the trap target.

Your managed device can only generate SNMP traps if your KwikNet SNMP Library has been configured with SNMP traps enabled. Edit your KwikNet Library Parameter File and check the box labelled "SNMP can send traps" on the SNMP property page. Be sure to generate an updated set of KwikNet Libraries after editing the Library Parameter File.

There are three kinds of SNMP traps generated by the KwikNet SNMP Agent during its normal course of operation. The SNMP agent generates a cold start trap when it is first started. If the SNMP agent is stopped and restarted by your application, the agent generates a warm start trap. If the SNMP agent receives an unauthorized SNMP request for access to a MIB variable, it generates an authentication failure trap.

Your application can generate an SNMP trap with a call to KwikNet procedure *knsa_trap()*. In a multitasking system, only tasks of lower priority than the KwikNet Task can generate SNMP traps. Of course, any application function which executes in the context of the KwikNet Task can also generate an SNMP trap.

Once the KwikNet SNMP Agent has been initialized by KwikNet, your application can call KwikNet procedure *knsa_control()* to globally disable the generation of traps. Thereafter, any attempt by the SNMP agent or by your application to generate a trap will be refused until such time as you again call procedure *knsa_control()* to enable trap generation.

Trap Messages

An SNMP trap message is an SNMP message which contains a trap protocol data unit (trap PDU). The trap PDU specifies the trap type and identifies the managed device which is generating the trap. Application specific trap PDUs can also include the object identifiers and values for one or more SNMP variables if so desired.

The following SNMP trap types are defined by SNMP.

coldStart	Cold start trap generated by SNMP agent	
warmStart	Warm start trap generated by SNMP agent	
linkUp	Network available	(not generated)
linkDown	Network unavailable	(not generated)
authenticationFailure	Request for MIB access denied by SNMP agent	
egpNeighborLoss	EGP peer has been lost	(not generated)
enterpriseSpecific	Enterprise specific trap generated by your application	

Trap Targets

A trap target is the SNMP manager to whom an SNMP trap message is sent. The KwikNet SNMP Agent maintains a list of all known trap targets to whom SNMP traps are to be sent. Each SNMP trap message generated by the SNMP Agent or by your application is sent to every trap target in that list.

You must specify the maximum number of trap targets required by your application. Edit your KwikNet Library Parameter File and adjust parameter labelled "Max number of trap targets" on the SNMP property page. Be sure to generate an updated set of KwikNet Libraries after editing the Library Parameter File.

Each trap target is identified by its IP address. Associated with each trap target is a text string which provides the name of the SNMP community which is to be inserted into each SNMP trap message sent to that particular trap target. The default set of trap targets is provided in your custom definition of the managed device in source module *SNMPUSER.C*. You must edit that module to specify your particular set of trap targets.

Once the KwikNet SNMP Agent has been initialized by KwikNet, your application can call KwikNet procedure *knsa_traptarget()* to dynamically revise the trap target list maintained by the agent. Trap targets can be added to or deleted from the list. Hence, your application can dynamically initialize its list of trap targets and then adapt the list to changing requirements as time goes on.

Authentication Traps

The KwikNet SNMP agent sends the *authenticationFailure* trap whenever it detects an attempt by an unauthorized SNMP manager to access or modify a MIB variable. The SNMP request causing the violation is ignored and an SNMP trap is generated instead. Such access violations can occur in two ways.

A violation occurs if the community specified in an SNMP request does not have the access rights necessary to service the request. For example, if the community has read-only access rights, an SNMP request from that community to modify any MIB variable will cause an access violation.

A violation also occurs if an SNMP request specifies an unknown community.

It is also important to know when an *authenticationFailure* trap will not be sent. If the SNMP request specifies a known community with the access rights necessary to perform the requested operation but the MIB variable of interest does not exist or has access rights which preclude the requested operation, no trap is generated. In this case, an error indication is provided in the SNMP message which is sent as a reply to the SNMP manager making the request.

1.6 MIB Data Organization

The data in an SNMP Management Information Base (MIB) is organized in a tree-like structure. Each branch in the tree is given a number and a human readable name. The name is only descriptive; it is not used by the SNMP protocol. Each branch can contain zero or more managed variables as well as zero or more branches leading further down the tree. Figure 1.6-1 illustrates the standard MIB tree defined by RFC-1155 and extended by RFC-1213.

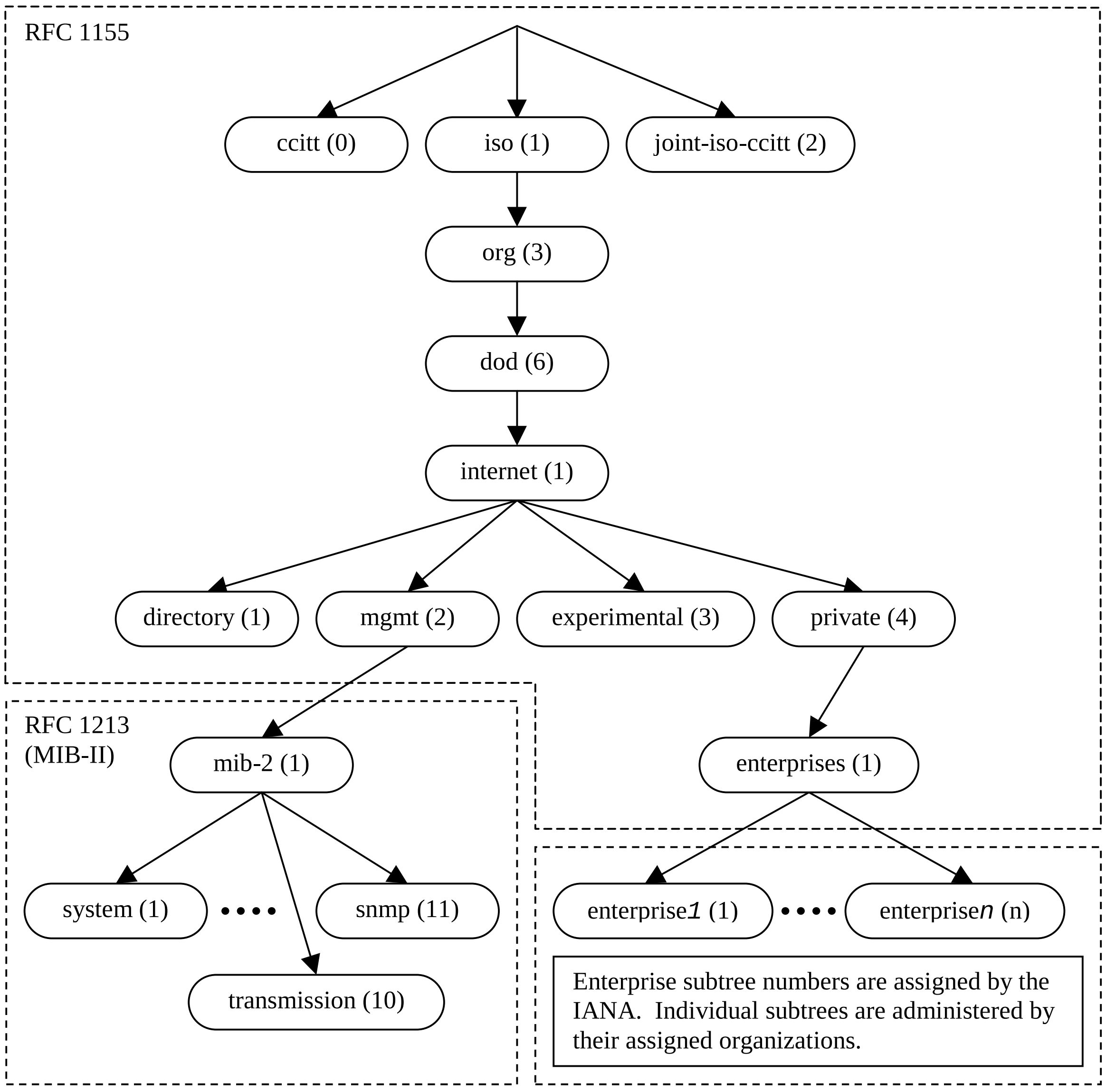


Figure 1.6-1 Standard MIB Tree Structure

Each MIB variable has a unique object identifier which specifies the exact location of the MIB variable within the MIB tree. The object identifier is a sequence of numbers which specify the tree branches to be followed to find the variable.

For example, from Figure 1.6-1, all of the MIB-II variables defined in RFC-1213 are contained in the subtree beginning at branch *`mib-2`* which is identified as follows:

string form:	*`iso.org.dod.internet.mgmt.mib-2`*
numeric form:	*`1.3.6.1.2.1`*

Since the MIB-II variable *`snmpOutPkts (2)`* is located in the *`snmp`* branch of the *`mib-2`* subtree, its object identifier is:

string form:	*`iso.org.dod.internet.mgmt.mib-2.snmp.snmpOutPkts`*
numeric form:	*`1.3.6.1.2.1.11.2`*

Figure 1.6-1 also shows the *`enterprises`* branch which contains a number of *`enterprise`* subtrees. These subtrees are assigned to organizations by the Internet Assigned Numbers Authority (IANA). The structure and data managed by these private subtrees is completely controlled by their assigned organizations. It is this branch of the MIB tree which will contain the custom MIB variables, if any, required to support your managed device.

The MIB used by your managed device must be constructed using the KwikNet MIB Compiler as described in Chapter 2. In addition to your own custom MIB definitions, you can also add support for MIBs defined by other RFCs.

MIB-II Support

The MIB-II Management Information Base defined in RFC-1213 is the standard MIB used to manage TCP/IP based devices. The MIB-II variables are usually included as part of the MIB for each such managed device. For this reason, the KwikNet SNMP Option includes built-in support for MIB-II variables.

The KwikNet SNMP Library provides support for the MIB-II Management Information Base defined in RFC-1213. This is the standard MIB used to manage TCP/IP based devices.

The MIB-II groups and tables are listed below.

System group	descriptions of the managed device and its capabilities
Interfaces group	information about all network interfaces
Interfaces table	information and statistics for individual network interfaces
Address translation table	map of network (IP) addresses to sub-network (physical) addresses for all networks interfaces
IP group	information and statistics gathered by the IP layer
IP address table	IP addressing information for each network interface
IP routing table	IP routing information for the managed device
IP address translation table	map of network (IP) addresses to sub-network (physical) addresses for individual network interfaces
ICMP group	statistics gathered by the ICMP layer
TCP group	information and statistics gathered by the TCP layer
TCP connection table	descriptions of individual TCP connections
UDP group	information and statistics gathered by the UDP layer
UDP connection table	descriptions of individual UDP listeners
EGP group	information and statistics gathered by the EGP layer (Exterior Gateway Protocol)
EGP neighbour table	descriptions of individual EGP neighbours
Transmission group	information and statistics gathered by various transmission media
SNMP group	information and statistics gathered by the SNMP agent

The KwikNet SNMP Option includes support for all of the MIB-II groups and tables listed above with the following exceptions:

- The EGP group and the EGP neighbour table are not supported.
- The transmission group is declared by the MIB-II specification but the data managed by this group is defined in other RFCs which are not supported.
- Changing the status of a network interface by writing to its *`ifAdminStatus`* MIB variable in the interfaces table is supported for SLIP and PPP networks but not for Ethernet networks.
- Deleting a TCP connection by writing to its *`tcpConnState`* MIB variable in the TCP connections table is not supported.

1.7 SNMP Error and Statistics Logging

KwikNet provides a data logging service which can be used to advantage to observe the progress of stack activity. This service is described in Chapter 1.6 of the KwikNet TCP/IP Stack User's Guide.

The KwikNet SNMP Agent can be configured to record error information on the KwikNet logging device. To enable this feature, edit your KwikNet Library Parameter File and check the box labeled SNMP logic in the Debug Tracing column on the Debug property page.

The SNMP agent does not log the normal receipt of SNMP requests or the sending of SNMP response or trap messages. Since these operations involve normal UDP transactions, they can be observed using the standard KwikNet debug logging and trace facilities.

The KwikNet SNMP Agent maintains a statistical record of the events which occur during its operation. Your application can call KwikNet procedure *`kn_netstats()`* to force KwikNet to dump this SNMP information to the KwikNet logging device. To enable this feature, edit your KwikNet Library Parameter File and check the box labeled Log statistics on the SNMP property page.

Note

To include KwikNet data logging support, edit your Network Parameter File and enable data logging on the Application property page. Then generate and compile an updated copy of your KwikNet Network Configuration Module.

1.8 Adding the SNMP Agent to Your Application

Before you can add the SNMP agent to your application, there are a number of prerequisites which your application must include. You must have a working KwikNet UDP and IP stack. It is imperative that you start with a tested stack with functioning device drivers before you add SNMP. If these components are not operational, the KwikNet SNMP Agent cannot operate correctly.

KwikNet SNMP Library

Begin by deciding which SNMP features must be supported. Review the SNMP property page described in Chapter 1.3. In particular, omit statistics logging and SNMP trap support unless you actually have a need for these services.

Armed with your SNMP feature list, use the KwikNet Configuration Manager to edit your application's KwikNet Library Parameter File to include the SNMP protocol. You might also enable SNMP logic tracing on the Debug property page. Then rebuild your KwikNet Libraries. A new SNMP Library, *KNnnnSNM.A*, will be produced along with your IP Library. The library extension may be *.A* or *.LIB* or some other extension dictated by the toolset which you are using.

KwikNet Task Considerations

Since the KwikNet SNMP Agent executes in the context of the KwikNet Task, you may need to increase the stack size for this task. A stack size of 4096 bytes is considered adequate for use with the SNMP agent. The stack size can be trimmed after the SNMP agent has been tested and actual stack usage observed using your debugger.

In a multitasking system, the KwikNet Task must be of higher priority than any application task which uses any of the KwikNet services listed in Chapter 3 to interact with the KwikNet SNMP Agent.

Reconstructing Your KwikNet Application

Since you are adding the SNMP Agent to an existing KwikNet application, there is little to be done.

If you have enabled SNMP statistics logging or SNMP debug tracing you may have to update your Network Configuration Module to include data logging support.

Your application link and/or locate specification files must be updated to add the KwikNet SNMP Library file *KNnnnSNM.A* prior to the KwikNet IP Library. The object modules which collectively form your SNMP MIB (see Chapter 2), and any support modules which they might require, must also be included in your link specification together with your other application object modules.

With these changes in place, you can link and create an updated KwikNet application with SNMP support included.

AMX Considerations

When reconstructing a KwikNet application which uses the AMX Real-Time Multitasking Kernel, no changes to your AMX User Parameter File or AMX Target Configuration Module are required to support the SNMP agent.

Performance Considerations

A meaningful discussion of all of the issues which affect the performance of an SNMP agent or manager are beyond the scope of this document. Factors affecting the performance of the KwikNet SNMP Agent include the following:

processor speed
memory access speed and caching effects
network type (Ethernet, SLIP, PPP)
network device driver implementation (buffering, polling, DMA support, etc.)
IP packet fragmentation
network hops required for connection
operation of the remote (foreign) connected SNMP manager
KwikNet TCP/IP Stack configuration (clock, number of packets, queue sizes, etc.)

Of all these factors, only the last one can be easily adjusted. Increasing the fundamental clock rate for the KwikNet TCP/IP Stack beyond 50Hz will have little effect and will adversely affect systems with slow processors or memory. Increasing the number of packets available for device use will help if high speed Ethernet devices are in use and the processor is fast enough to keep up.

2. KwikNet MIB Construction

2.1 Introduction

For your network device to be managed using the Simple Network Management Prototcol (SNMP), you must provide a Management Information Base (MIB) which defines your device and its capabilities. It is the purpose of this chapter to describe how such a MIB is constructed for use with the KwikNet SNMP Agent.

Figure 2.1-1 illustrates the MIB construction process. A custom MIB is created using the KwikNet MIB Compiler, a utility program which executes on your development system. You create a text file which defines the structure of the data which makes up your custom MIB. The MIB Compiler translates this description into a set of C source files which, when compiled and linked with your application, form your custom MIB. The MIB Compiler can also generate documentation files which summarize the content of your custom MIB.

As shown in Figure 2.1-1, the KwikNet MIB Compiler needs access to its private MIB Template File *KN_MIB.CT* from which the text files which it generates are derived.

MIB Definition Files

The MIB is defined by a set of one or more MIB Definition Files. A MIB Definition File is a text file which describes a collection of MIB variables, their attributes and the manner in which they are organized into a MIB tree. The MIB definition must follow the organizational rules established by RFC-1155. The MIB definition is expressed using the Abstract Syntax Notation (ASN.1).

Your MIB can be completely defined by a single MIB Definition File. Alternatively, your MIB can be defined using several different MIB Definition Files, each providing the definition of a subset of your entire MIB. For example, the MIB-II Management Information Base defined in RFC-1213 is the standard MIB used to manage TCP/IP based devices. These MIB-II variables are usually included as part of the MIB for each such managed device. For this reason, the KwikNet SNMP Option includes a prebuilt MIB Definition File, *SNMP1213.MIB*, to complement your custom MIB definition.

SNMP Variables Module

The KwikNet MIB Compiler is used to translate and merge your MIB Definition Files into a single SNMP Variables Module named *SNMPVARS.C*. This C source file must compiled and linked with your KwikNet application. It forms the foundation for your entire MIB.

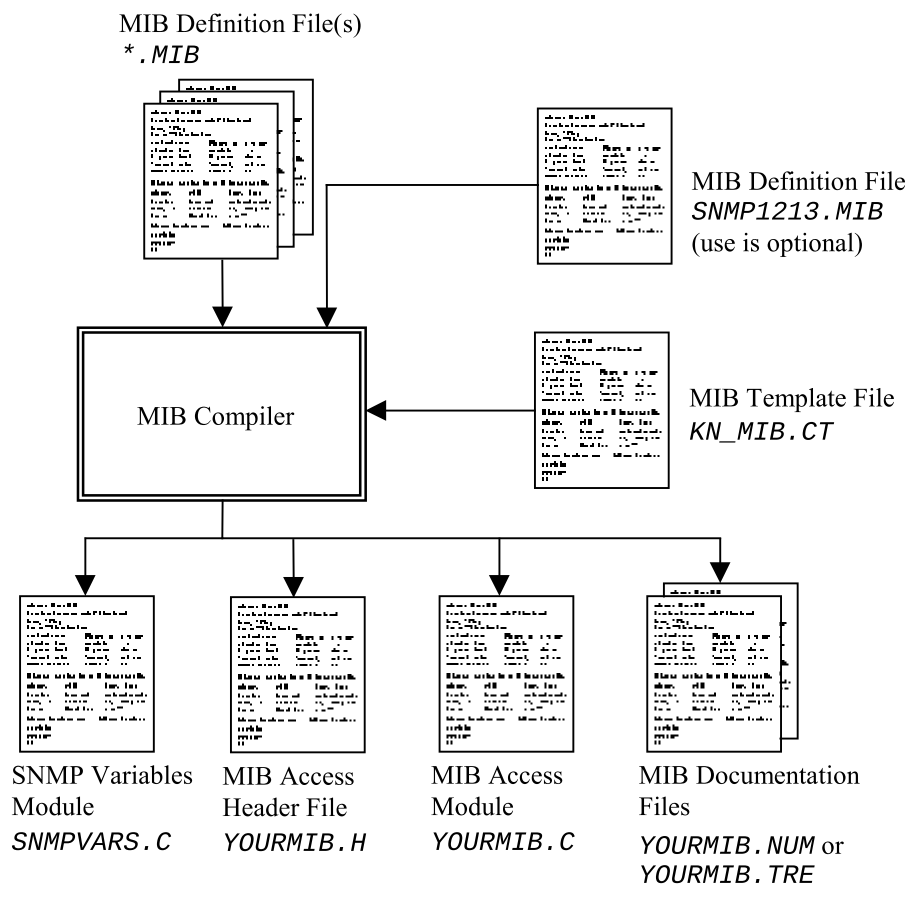


Figure 2.1-1 Creating a Custom MIB

MIB Access Modules

The SNMP Variables Module does not actually contain the data corresponding to your MIB variables. Instead, the SNMP agent calls a MIB Finder Procedure to locate a MIB variable. A separate MIB Finder Procedure exists for each MIB group and table in your entire MIB. Once the SNMP agent has located a MIB variable, it can manipulate its value.

So where are these MIB Finder Procedures and how do they find the actual MIB values? Each MIB Finder Procedure resides in a MIB Access Module which is a C source file produced by the KwikNet MIB Compiler from information in your MIB Definition Files. For each MIB group and table, the MIB Compiler generates the C code for a raw MIB Finder Procedure. You must edit these raw MIB Finder Procedures, adding the code to locate the MIB variables in the MIB group or table for which the procedure is responsible.

The KwikNet SNMP Library includes the prebuilt MIB Access Module *SNMP1213.C* which provides the MIB Finder Procedures for all of the MIB-II groups and tables specified by RFC-1213. Hence, you can always include the *SNMP1213.MIB* Definition File in the construction of your MIB without having to edit MIB Finder Procedures to access the corresponding MIB variables.

The KwikNet MIB Compiler is used to generate one or more MIB Access Modules from your MIB Definition Files. If you only have one MIB Definition File, you will create a single MIB Access File. If you have several MIB Definition Files, you can create one MIB Access File for each. Alternatively, you can generate a single MIB Access Module from several MIB Definition Files. The choice is yours. For example, if you had three MIB Definition Files, you could produce one MIB Access Module from the first two MIB Definition Files and one MIB Access Module from the third MIB Definition File.

MIB Access Header Files

The KwikNet MIB Compiler also generates a MIB Access Header File for each of your MIB Access Modules. The header file is required to compile the MIB Access Module. It can also be used by other application modules which provide actual instances of the MIB variables whose access is controlled by the MIB Access Module.

MIB Documentation Files

Finally, the KwikNet MIB Compiler can also be used to generate a MIB Documentation File describing the content of the MIB defined by one or more MIB Definition Files. This file is a text file which lists the MIB variables in either of two formats.

The **numeric format** provides the following information for each node and variable in the MIB tree: the object identifier, the text name equivalent and the node or variable type.

The **tree format** shows the MIB tree in a graphical form with each node and variable in the tree identified by its sub-identifier number and its text name equivalent.

Building a Custom MIB

The process of building a custom MIB is quite simple.

1. Define the MIB variables in your managed device and describe their organizational structure.
2. Create a custom MIB Definition File which implements this MIB. If warranted by the complexity of your MIB data organization, its definition can be split into multiple MIB Definition Files.
3. Use the MIB Compiler to generate a MIB Access Header File and a MIB Access Module from the MIB Definition File(s) created in step 2.
4. Edit the MIB Access Module created in step 3. Edit the raw MIB Finder Procedures to provide access to the MIB variables in your MIB groups and tables.
5. If your managed device is best described using separate MIBs for different subsets of its MIB variables, repeat steps 1 through 4 for each of the separate MIBs.
6. Use the MIB Compiler to generate the SNMP Variables Module *SNMPVARS.C* from the entire collection of MIB Definition Files created in step 2. If your managed device must also include the MIB-II variables defined by RFC-1213, be sure to include the prebuilt MIB Definition File *SNMP1213.MIB* provided with KwikNet.
7. Compile the MIB Access Module(s) which you edited in step 4.
8. Compile the SNMP Variables Module *SNMPVARS.C* generated in step 6.
9. Link the object modules from steps 7 and 8 with your KwikNet application.

SNMP MIB Sample

The construction of a custom MIB for a managed device is best illustrated with a simple example. The managed device shown in Figure 2.1-2 provides a visible text display and error counter and has a set of seven of toggle switches used to control the device.

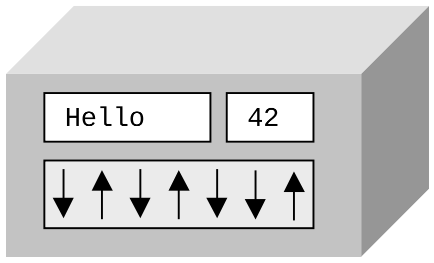


Figure 2.1-2 Sample SNMP Managed Device

The SNMP manager must be able to read and modify the text presented in the display. The SNMP manager must be able to read, but not modify, the error count and the position of each individual toggle switch.

The SNMP sample device is assumed to be one of many such sample devices manufactured by KADAK Products Ltd. It is also assumed that KADAK classifies all such devices as samples to distinguish them from all other products manufactured by KADAK. This particular sample device is to be identified as device number 2 within the entire collection of sample devices available from KADAK.

Figure 2.1-3 shows one particular implementation of a custom MIB which meets the requirements of this simple device. Other equally valid organizations of the MIB data could be devised. The sample MIB Definition File *`KNSAMMIB.MIB`* which implements this MIB is described in Chapter 2.2.

The MIB tree shown in Figure 2.1-3 has been simplified by omitting the detail of the first seven branches of the standard MIB tree. The first entry in the illustration represents the entire path down the MIB tree to the enterprise node identified by KADAK's enterprise number 4001.

```
kadak=iso(1)org(3)dod(6)internet(1)private(4)enterprises(1) 4001
└kdkprod(1)                  All of KADAK's SNMP managed products
 └kdksamples(1)              All of KADAK's sample products
  └kdksample(2)              This particular sample device
   └ksamStatus(1)
    ├ksamStatDisplay(1)      OCTET STRING ┬── ksamStatus group
    ├ksamStatErrs(2)         COUNTER      ┘
    └ksamSwitchTable(3)
     └ksamSwitchEntry(1)
      ├ksamSwitchEntIndex(1) INTEGER      ┬── ksamSwitchEntry table
      └ksamSwitchEntState(2) INTEGER      ┘

ksamStatDisplay    = 1.3.6.1.4.1.4001.1.1.2.1.1.0
ksamStatErrs       = 1.3.6.1.4.1.4001.1.1.2.1.2.0
ksamSwitchEntIndex = 1.3.6.1.4.1.4001.1.1.2.1.3.1.1.row
ksamSwitchEntState = 1.3.6.1.4.1.4001.1.1.2.1.3.1.2.row
                                                    └── 1 to 7
```

Figure 2.1-3 MIB Structure for Sample Device

The MIB Access Header File *KNSAMMIB.H* and MIB Access Module *KNSAMMIB.C* for this sample device are generated by invoking the KwikNet MIB Compiler with the following command line.

```
KN_MIB KNSAMMIB.MIB -hc
```

The MIB Finder Procedures in the MIB Access Module *KNSAMMIB.C* must be edited to provide access to the MIB variables in the groups and tables specified in MIB Definition File *KNSAMMIB.MIB*. This process is described in Chapter 2.3.

The SNMP Variables Module *SNMPVARS.C* for this sample device is generated by invoking the KwikNet MIB Compiler with the following command line. Note that the optional KwikNet MIB Definition File *SNMP1213.MIB* has been included in the MIB compilation so that the sample managed device will also support the MIB-II variables defined by RFC-1213.

```
KN_MIB KNSAMMIB.MIB SNMP1213.MIB -v
```

The full operation of the KwikNet MIB Compiler is described in Chapter 2.4.

The MIB Access Module *KNSAMMIB.C* and the SNMP Variables Module *SNMPVARS.C* must be compiled and linked with the KwikNet application for this sample device as described in Chapter 2.5.

2.2 MIB Definition Files

A MIB Definition File is a text file which describes a collection of MIB variables, their attributes and the manner in which they are organized into a MIB tree. The MIB definition must follow the organizational rules established for the Structure of Management Information (SMI) defined in RFC-1155. The MIB definition must be expressed using the Abstract Syntax Notation (ASN.1).

This manual makes no attempt to describe the detailed structure of a MIB Definition File. It is assumed that you have a working knowledge of the SMI data organization rules and the ASN.1 language. Reference materials are provided in Appendix A of the KwikNet TCP/IP Stack User's Guide.

The MIB Definition File for the sample device introduced in Chapter 2.1 is illustrated in Figure 2.2-1. The MIB tree structure of this sample MIB was shown in Figure 2.1-3.

MIB Organization

All of the data within a MIB tree is contained in MIB variables which are organized into groups and tables. A group contains a single instance of zero or more MIB variables. A table consists of zero or more sets of MIB variables in which the variables define the columns in the table and each instance of the set of variables forms a row in the table.

Each MIB variable in a group is identified using the object identifier of the group followed by the sub-identifier for the MIB variable and a mandatory trailing sub-identifier of 0. The MIB shown in Figure 2.1-3 defines a single group, `ksamStatus`, which contains two members, `ksamStatDisplay` and `ksamStatErrs`.

Each MIB variable in a table is identified using the object identifier of the table followed by the sub-identifier for that variable. Each MIB variable which forms a column in the table is defined in this fashion. To specify a particular instance of a MIB variable in a table, the values for each of the `INDEX` objects specified in the table description are appended, in their order of definition, to the MIB variable object identifier as sub-identifiers. These extra sub-identifiers act as the row identifier, providing access to a particular instance of the MIB variable in the table.

The MIB shown in Figure 2.1-3 defines a single table, `ksamSwitchTable`, containing a collection of `ksamSwitchEntry` objects, each of which contains two members, `ksamSwitchEntIndex` and `ksamSwitchEntState`. Each object fully describes one of the seven switches in the managed device. The MIB variable `ksamSwitchEntState` provides the state of one switch. The particular switch is identified by the MIB variable `ksamSwitchEntIndex` which is used as the `INDEX` to select a specific row of the table.

Figure 2.2-1 MIB Definition File for Sample Device

```
KADAK-SAMPLE-MIB DEFINITIONS ::= BEGIN

-- External symbols

IMPORTS
           enterprises, Counter FROM RFC1155-SMI
           OBJECT-TYPE FROM RFC-1212;

-- Base Object Identifiers for kdksample MIB

kadak      OBJECT IDENTIFIER ::= { enterprises 4001 }
kdksample  OBJECT IDENTIFIER ::=
                { kadak kdkprod(1) kdksamples(1) 2 }

-- Define the status group (ksamStatus)

ksamStatus OBJECT IDENTIFIER ::= { kdksample 1 }

ksamStatDisplay OBJECT-TYPE
           SYNTAX  OCTET STRING
           ACCESS  read-write
           STATUS  mandatory
           DESCRIPTION
                "Currently displayed text string."
           ::= { ksamStatus 1 }

ksamStatErrs OBJECT-TYPE
           SYNTAX  Counter
           ACCESS  read-only
           STATUS  mandatory
           DESCRIPTION
                "Running count of device errors."
           ::= { ksamStatus 2 }
```

...more

...continued

```
-- Define the switch table (ksamSwitchTable)

ksamSwitchTable OBJECT-TYPE
          SYNTAX  SEQUENCE OF KsamSwitchEntry
          ACCESS  not-accessible
          STATUS  mandatory
          DESCRIPTION
                "A list of switches in the device."
          ::= { ksamStatus 3 }

ksamSwitchEntry OBJECT-TYPE
          SYNTAX  KsamSwitchEntry
          ACCESS  not-accessible
          STATUS  mandatory
          DESCRIPTION
                "A switch entry containing objects
                describing a particular switch."
          INDEX  { ksamSwitchEntIndex }
          ::= { ksamSwitchTable 1 }

KsamSwitchEntry ::=
          SEQUENCE {
                ksamSwitchEntIndex
                  INTEGER,
                ksamSwitchEntState
                  INTEGER
          }

ksamSwitchEntIndex OBJECT-TYPE
          SYNTAX  INTEGER
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                "A unique value for each switch in the device.
                Its value ranges between 1 and the number of
                switches in the device."
          ::= { ksamSwitchEntry 1 }

ksamSwitchEntState OBJECT-TYPE
          SYNTAX  INTEGER {
                up(1),
                down(2)
          }
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
                "The position of a switch in the device."
          ::= { ksamSwitchEntry 2 }

END
```

Figure 2.2-1 MIB Definition File for Sample Device

Predefined Symbols used in MIB Definitions

The KwikNet MIB Compiler recognizes the key elements of a MIB Definition. To reduce the number of repeated definitions which you might otherwise require in your MIB Definition Files, the commonly occurring SMI constructs and keywords listed in Figure 2.2-2 are predefined by the KwikNet MIB Compiler.

```
The macro OBJECT-TYPE from RFC-1212
The macro TRAP-TYPE from RFC-1215
The following defined object types from RFC-1155:
        NetworkAddress
        IpAddress
        Counter
        Gauge
        TimeTicks
        Opaque
The following identifiers from RFC-1155 and RFC-1213:
        iso             OBJECT IDENTIFIER ::= { 1 }
        org             OBJECT IDENTIFIER ::= { iso 3 }
        dod             OBJECT IDENTIFIER ::= { org 6 }
        internet        OBJECT IDENTIFIER ::= { dod 1 }
        directory       OBJECT IDENTIFIER ::= { internet 1 }
        mgmt            OBJECT IDENTIFIER ::= { internet 2 }
        experimental    OBJECT IDENTIFIER ::= { internet 3 }
        private         OBJECT IDENTIFIER ::= { internet 4 }
        enterprises     OBJECT IDENTIFIER ::= { private 1 }
        mib-2           OBJECT IDENTIFIER ::= { mgmt 1 }
```

Figure 2.2-2 MIB Compiler Predefined Symbols

Imported Symbols

The ASN.1 *IMPORTS* directive is intended to provide access within one MIB Definition File to object, symbol, type or macro definitions located in another MIB Definition File. The KwikNet MIB Compiler does not support this use of the *IMPORTS* directive. Instead, when an *IMPORTS* directive is encountered, the MIB Compiler simply checks that the imported symbols have been defined. Therefore, you must ensure that all symbols in the *IMPORTS* section are either predefined by the KwikNet MIB Compiler or defined in your MIB Definition File prior to the *IMPORTS* directive.

2.3 MIB Finder Procedures

A MIB Finder Procedure is an application procedure which the KwikNet SNMP Agent calls to locate each MIB variable. A separate MIB Finder Procedure exists for each MIB group and table in your entire MIB. These procedures are created for you in your MIB Access Module(s), the C source file(s) produced by the KwikNet MIB Compiler from information in your MIB Definition Files.

For each MIB group and table, the MIB Compiler generates the C code for a raw MIB Finder Procedure. You must edit these raw MIB Finder Procedures, adding the code to locate the MIB variables in the MIB group or table for which the procedure is responsible.

Prebuilt MIB Finder Procedures are provided in the KwikNet SNMP Library for all of the MIB-II variables specified by RFC-1213. The source code for these MIB Finder Procedures is provided in file *`SNMP1213.C`* in the KwikNet SNMP installation directory.

In addition to the MIB Access Module, the KwikNet MIB Compiler also generates a MIB Access Header File, a C header file which can be used to create and manipulate instances of your custom MIB data. For each group or table which is defined in the MIB Definition File, the MIB Access Header File contains a structure definition for the group or for one entry (row) in the table, a set of magic constants which can be used to identify members of the structure and a prototype for the MIB Finder Procedure which will be called to access that group or table.

For a group or table named *`XXX`*, the MIB Finder Procedure is named *`var_XXX`*. The structure defining the group or table is named *`XXX_mib`*. Each member of structure *`XXX_mib`* is given the name of the MIB variable which it implements. The group structure does not include members for tables, if any, which are considered to belong to the group.

There is a magic constant defined for each member of structure *`XXX_mib`*. The constant name is derived by converting the name of the structure member to upper case. The constant is assigned a value equal to the offset of the member within structure *`XXX_mib`*. As will be shown, the magic constants can be used by the MIB Finder Procedure to readily identify the MIB variable of interest without having to interpret its object identifier.

Warning

The MIB Access Header File contains information which is duplicated in the SNMP Variables Module.

Do NOT edit your MIB Access Header File(s).

Figure 2.3-1 illustrates the MIB Access Header File *KNSAMMIB.H* generated by the KwikNet MIB Compiler from the sample MIB Definition File provided in Figure 2.2-1.

Figure 2.3-2 shows the corresponding MIB Access Module *KNSAMMIB.C* with its raw MIB Finder Procedures ready to be edited.

```
/* Recommended MIB structures */


/* 'ksamStatus' group      */
struct ksamStatus_mib {
         void*             ksamStatDisplay;
         unsigned long     ksamStatErrs;
         };

/* 'ksamSwitchEntry' table */
struct ksamSwitchEntry_mib {
         long              ksamSwitchEntIndex;
         long              ksamSwitchEntState;
         };


/* Variable Access Constants */

/* 'ksamStatus' group      */
#define KSAMSTATDISPLAY    0
#define KSAMSTATERRS       KSAMSTATDISPLAY+4

/* 'ksamSwitchEntry' table */
#define KSAMSWITCHENTINDEX 0
#define KSAMSWITCHENTSTATE KSAMSWITCHENTINDEX+4


/* Prototypes for MIB Finder Procedures */
/* 'ksamStatus' group      */
extern unsigned char *var_ksamStatus(
         const struct knsa_var *, knsa_subid *, int *, int, int *);

/* 'ksamSwitchEntry'       */
extern unsigned char *var_ksamSwitchEntry(
         const struct knsa_var *, knsa_subid *, int *, int, int *);
```

Figure 2.3-1 MIB Access Header File for Sample Device

```
/* MIB Finder Procedure for 'ksamStatus' group                          */
unsigned char *var_ksamStatus(
         const struct knsa_var *vp,
         knsa_subid *name, int *length,
         int oper, int *var_len)
{
         /* >>>> TODO: Add code here */
         return NULL;     /* Default FAIL return                        */
         }


/* MIB Finder Procedure for 'ksamSwitchEntry' table                     */
unsigned char *var_ksamSwitchEntry(
         const struct knsa_var *vp,
         knsa_subid *name, int *length,
         int oper, int *var_len)
{
         /* >>>> TODO: Add code here */
         return NULL;     /* Default FAIL return                        */
         }
```

Figure 2.3-2 MIB Access Module for Sample Device

Operation of a MIB Finder Procedure

A MIB Finder Procedure is called by the KwikNet SNMP Agent to locate one of the MIB variables in the group or table for which the MIB Finder Procedure is responsible. The procedure is presented with enough information to permit it to locate the MIB variable of interest. If the MIB variable exists, the procedure presents the SNMP agent with a pointer to the memory location at which the MIB variable data is maintained. The SNMP agent can read and/or write at that memory location as required to manipulate the MIB variable's value. Of course the agent will only do either if the MIB access rights permit.

The MIB Finder Procedure can call KwikNet service procedure *knsa_setvarrange()* to limit the range of values which the SNMP agent is permitted to write into a numeric MIB variable. For non-numeric MIB variables such as strings (octets) and object identifiers, the range indicates the minimum and maximum number of bytes which the MIB variable value can occupy.

The MIB Finder Procedure must follow the specifications presented in Chapter 2.3.1.

MIB Set Procedure

Normally, the KwikNet SNMP Agent takes complete responsibility for writing a MIB variable value at the memory location specified by the MIB Finder Procedure. However, the MIB Finder Procedure can override the agent by providing a MIB Set Procedure which the SNMP agent must subsequently use to alter the MIB variable value. The MIB Finder Procedure must call KwikNet service procedure *knsa_setvarfunc()* to provide the SNMP agent with a pointer to the MIB Set Procedure to be used.

The KwikNet MIB Compiler generates a single raw MIB Set Procedure in each MIB Access Module. This empty procedure, which resides within a C comment block, can be deleted or used as a template for any of the MIB Set Procedures which your MIB Finder Procedures might require.

The MIB Set Procedure must follow the specifications presented in Chapter 2.3.2.

MIB Finder Procedure Example

Figure 2.3-2 showed the sample MIB Access Module *KNSAMMIB.C* with its raw MIB Finder Procedures ready for editing.

Figure 2.3-3 shows the same MIB Access Module *KNSAMMIB.C* with the pair of MIB Finder Procedures coded to meet the requirements of the sample SNMP managed device introduced in Chapter 2.1.

Figure 2.3-3 MIB Finder Procedures for Sample Device

```
struct ksamStatus_mib KsamMib;      /* Storage for variable data   */
char KsamStatDisplay[128];

/* MIB Finder Procedure for the 'ksamStatus' group                 */
unsigned char *var_ksamStatus(const struct knsa_var *vp,
      knsa_subid *name, int *length, int oper, int *var_len)
{
      /* Do object identifier operations                           */
      if (oper != KNSA_NEXT_OP) {
            /* For KNSA_SET_OP or KNSA_GET_OP, check for exact match*/
            if (knsa_oidcmpv(name, *length, vp))
                  return NULL;      /* FAIL return                 */

            /* Note: *name and *length are already correct.        */
            }

      else {
            /* For KNSA_NEXT_OP, save name of matched variable     */
            knsa_oidcpyv(name, vp);
            *length = vp->grp->namelen;
            }

      /* Handle variable length types or special integer variables.*/
      switch (vp->magic) {
      case KSAMSTATDISPLAY:
            /* Length of current string                            */
            *var_len = strlen(KsamStatDisplay);

            /* Max length for set operation                        */
            knsa_setvarrange(0, 127);

            /* Return pointer to string storage                    */
            return ((unsigned char *)KsamStatDisplay);
            }

      /* Default return for integer types                          */
      *var_len = sizeof(long);
      return ((unsigned char *)&KsamMib + vp->magic);
      }
```

...more

...continued

```
#define NUMSWITCHES     7     /* Sample device has 7 switches      */
#define SW_LEN          15    /* SwitchEntry object identifiers    */
                              /* always have 15 sub-ids            */
                              /* (from MIB Definition)             */
long KsamSwitchStates[NUMSWITCHES]; /* Storage for variable data   */


/* MIB Finder Procedure for the 'ksamSwitchEntry' table            */
unsigned char *var_ksamSwitchEntry(const struct knsa_var *vp,
      knsa_subid *name, int *length, int oper, int *var_len)
{
      int         row;              /* Row index of switch         */
      int         result;           /* Name comparison results     */
      knsa_subid  newname[SW_LEN];  /* Buffer for new object id    */
      static long long_return;      /* Temporary return buffer     */

      /* Find requested switch and build its object identifer      */
      knsa_oidcpyv(newname, vp);
      for (row = 1; row <= NUMSWITCHES; row++) {
            newname[SW_LEN-1] = (knsa_subid)row;
            result = knsa_oidcmp(name, *length, newname, SW_LEN);
            if (((oper != KNSA_NEXT_OP) && (result == 0)) ||
                        ((oper == KNSA_NEXT_OP) && (result < 0)))
                  break;
            }

      if (row > NUMSWITCHES)
            return (NULL);          /* Not found                   */

      /* Save name of matched variable                             */
      knsa_oidcpy(name, newname, SW_LEN);
      *length = SW_LEN;

      /* Handle variable length types or special integer variables.*/
      switch (vp->magic) {
      case KSAMSWITCHENTINDEX:
            long_return = (long)row;
            *var_len = sizeof(long_return);
            /* Return pointer to temporary static storage          */
            /* This method is safe for 'read-only' variables.      */
            return ((unsigned char *)&long_return);

      case KSAMSWITCHENTSTATE:
            /* Return pointer to switch state array entry          */
            *var_len = sizeof(KsamSwitchStates[0]);
            return ((unsigned char *)(&KsamSwitchStates[row-1]));
            }

      /* All known table entries handled.  Return an error.        */
      return (NULL);
      }
```

Figure 2.3-3 MIB Finder Procedures for Sample Device

2.3.1 MIB Finder Procedure Specification

Purpose To Find a MIB Variable

Used by ■ KwikNet SNMP Agent

Setup Prototype is in a MIB Access Header File such as *KNSAMMIB.H*.

```
#include "KN_LIB.H"
#include "KN_SNMP.H"
#include "KNSAMMIB.H"
unsigned char *var_xxx(
                  const struct knsa_var *vp,
                  knsa_subid *name,
                  int   *length,
                  int   oper,
                  int   *var_len);
```

Description *Vp* is a pointer to a structure used by the KwikNet SNMP Agent to reference an SNMP MIB variable. Although the structure *knsa_var*, defined in header file *KN_SNMP.H*, is considered private to the SNMP agent, it contains the following members which can be read, but not modified, by the MIB Finder Procedure.

vp->magic	Magic constant	*(KSAMSTATERRS)*
vp->type	ASN.1 or SMI type	*(SMI_COUNTER)*
vp->acl	Access rights	*(KNSA_RONLY)*
vp->grp->namelen	Number of sub-identifiers	*(13)*

The values shown in parentheses describe the MIB variable *ksamStatErrs* defined for the sample SNMP managed device introduced in Chapter 2.1. That MIB variable is a read-only counter with an object identifier of *1.3.6.1.4.1.4001.1.1.2.1.2.0* which includes 13 sub-identifiers.

The magic constant is defined in the MIB Access Header File. It uniquely identifies the MIB variable in a group or the column of interest in a table.

The MIB variable type is one of the ASN.1 types *ASN_xxxxx* or SMI types *SMI_xxxxx* defined in header file *KN_SNMP.H*.

The MIB variable will have read-only (*KNSA_RONLY*), read-write (*KNSA_RWRITE*) or write only (*KNSA_WONLY*) access rights.

The structure referenced by parameter *vp* also contains a compressed object identifier which may be accessed using the KwikNet service procedures *knsa_oidcpyv()* and *knsa_oidcmpv()*. The last sub-identifier in the compressed object identifier is considered to be a row identifier. For MIB variables in a group, this row identifier is always *0*. For MIB variables in a table, this row identifier is *0xFFFF* for 16-bit object identifiers or *0xFFFFFFFF* for 32-bit object identifiers.

...more

Description ...continued

`Name` is a pointer to an array of sub-identifiers. The maximum number of sub-identifiers in the array is defined in header file `KN_LIB.H` to be `KN_SNMP_SUBIDS`. The value for this constant is determined by the value which you entered on the SNMP property page when you edited your KwikNet Library Parameter File.

On entry, this sub-identifier array contains the object identifier received in an SNMP request to access a particular MIB variable. On return, this sub-identifier array will contain the object identifier of the MIB variable which was actually found.

`Length` is a pointer to storage for the number of sub-identifiers present in the sub-identifier array referenced by parameter `name`. On entry, `*length` specifies the number of sub-identifiers present in the object identifier received in an SNMP request to access a particular MIB variable. On return, `*length` specifies the number of sub-identifiers in the object identifier of the MIB variable which was actually found.

`Oper` indicates the operation specified in the SNMP request. Parameter `oper` will be one of the following constants defined in header file `KN_SNMP.H`:

`KNSA_GET_OP`	Get the value of a MIB variable
`KNSA_NEXT_OP`	Get the value of the next MIB variable
`KNSA_SET_OP`	Set the value of a MIB variable

For operations `KNSA_GET_OP` and `KNSA_SET_OP`, the only MIB variable which can satisfy the request is the one with the object identifier specified by input parameter `name`. If the MIB variable is in a group, parameter `vp` already references the MIB variable of interest. If the MIB variable is in a table, parameter `vp` identifies the MIB variable of interest, but the MIB Finder Procedure must decode parameter `name` to locate the row containing the particular instance of that variable.

For operation `KNSA_NEXT_OP`, the MIB Finder Procedure must locate the MIB variable whose object identifier lexicographically follows the object identifier specified by input parameter `name`. If the MIB variable is in a group, parameter `vp` already references the MIB variable of interest. If the MIB variable is in a table, parameter `vp` identifies the MIB variable of interest, but the MIB Finder Procedure must decode parameter `name` to locate the next row containing an instance of that variable.

`Var_len` is a pointer to storage for an integer result which must be returned to the SNMP agent. The value stored at `*var_len` will specify the number of bytes of storage occupied by the value of the MIB variable.

...more

Returns If the MIB variable can be located, the MIB Finder Procedure must return a pointer to storage which contains the current value of that MIB variable. The pointer is treated by the SNMP agent as a pointer to an array of unsigned characters. The value stored at **var_len* must specify the number of bytes of storage occupied by that MIB variable value.

A numeric MIB value must be stored in 4 bytes in host endian form. A variable length MIB value, such as a string, must be stored as an array of *n* bytes where *n* is the array size. A MIB value which is an IP address must be stored as an array of 4 bytes in net endian form.

The object identifier of the MIB variable identified by the MIB Finder Procedure must be stored at **name*. The number of sub-identifiers in that object identifier must be stored at **length*.

If the MIB variable cannot be located, the MIB Finder Procedure must return the *NULL* pointer. In this case, the input parameters at **name* and **length* must NOT be modified. The value at **var_len* can be left undefined.

Notes The MIB Finder Procedure only finds the location the MIB variable value. The actual read or write operation will be performed by the SNMP agent or by your custom MIB Set Procedure.

For write operations, the range of acceptable numeric values or the allowable length of variable length values can be limited using KwikNet service procedure *knsa_setvarrange()*. Restrictions on written values can also be imposed by using a custom MIB Set Procedure.

You can override the SNMP agent write operation by providing a MIB Set Procedure which the SNMP agent will call to actually write the MIB variable value. Your MIB Finder Procedure must call KwikNet service procedure *knsa_setvarfunc()* to identify your custom MIB Set Procedure.

See Also *knsa_setvarfunc()*, *knsa_setvarrange()*

2.3.2 MIB Set Procedure Specification

Purpose To Write into a MIB Variable

Used by ■ KwikNet SNMP Agent

Setup A template is provided in each MIB Access Module such as *`KNSAMMIB.C`*.

```
#include "KN_LIB.H"
#include "KN_SNMP.H"
#include "KNSAMMIB.H"
int set_xxx(struct knsa_setparms *parmp);
```

Description *`Parmp`* is a pointer to the MIB set parameter structure *`knsa_setparms`* defined in header file *`KN_SNMP.H`* as follows:

```
struct knsa_setparms {        /* 'Set' function parameters*/
  const unsigned char *varp;  /* A(ASN.1 encoded set value)*/
  const knsa_subid *name;     /* Object id of variable    */
  int              varlen;    /* Length of set value      */
  int              name_len;  /* # of subids in subid array*/
  unsigned char    vartype;   /* ASN.1 type of set value  */
  char rsv1, rsv2, rsv3;      /* Reserved for alignment   */
  unsigned char    *setp;     /* A(value storage buffer)  */
  int              setlen;    /* Length of storage buffer */
  int              rsv4;      /* Reserved for alignment   */
  const struct knsa_var *vp;  /* A(MIB variable info)     */
  };
```

Parameter *`parmp->vp`* is a pointer to the MIB variable description. This is the same pointer provided to the MIB Finder Procedure which located the MIB variable.

The object identifier of the MIB variable identified by the MIB Finder Procedure is stored as a subid array at *`parmp->name`*. The number of sub-identifiers in that object identifier is specified by *`parmp->name_len`*.

Parameter *`parmp->setp`* is a pointer to storage for the MIB variable value. This is the pointer returned to the SNMP agent by the MIB Finder Procedure when it located the MIB variable.

Parameter *`parmp->setlen`* specifies the length, in bytes, of the storage buffer referenced by parameter *`parmp->setp`*. This length is the value returned to the SNMP agent by the MIB Finder Procedure when it located the MIB variable.

The value to be written into the MIB variable is available as an ASN.1 encoded value at *`parmp->varp`*. The value is of the ASN.1 type specified by *`parmp->vartype`*. The length of ASN.1 encoded value is provided by *`parmp->varlen`*. KwikNet service procedure *`knsa_asndecode()`* can be used to extract the MIB value from the ASN.1 encoded representation.

...more

Returns One of the following error status values:

KNSA_ERR_NOERROR	Operation was successful
KNSA_ERR_NOSUCHNAME	Variable does not exist or cannot be set
KNSA_ERR_BADVALUE	Set value is not valid
KNSA_ERR_TOOBIG	Set value exceeds a limit imposed by the MIB Set Procedure
KNSA_ERR_GENERR	Other unspecified error

It is up to the MIB Set Procedure to determine if it is willing to update the MIB variable with the value provided. The operation is declared a success by returning the value *KNSA_ERR_NOERROR*. Success usually implies that the MIB Set Procedure has accepted the ASN.1 encoded MIB value and written the decoded value, or some derived equivalent value, into the MIB value storage buffer referenced by parameter *parmp->setp*. In rare cases, the operation may be deemed successful even though the MIB value is left unaltered.

If an error is reported, the MIB value should not be altered. However, it is up to your MIB Set Procedure to decide whether or not to follow this recommendation.

Typically, the MIB Set Procedure writes the new data into the buffer located by the MIB Finder Procedure and then calls an application function to notify the application of the changed value.

Restriction The SNMP agent will not call the MIB Set Procedure if the value to be written into the MIB variable exceeds the range limitation, if any, imposed on the value by the MIB Finder Procedure with a call to KwikNet service procedure *knsa_setvarrange()*.

Note The data value to be written to the MIB variable is presented as an ASN.1 encoded value referenced by parameter *parmp->varp*. The following code fragment illustrates how your MIB Set Procedure can decode this value directly into the MIB variable data buffer.

```
knsa_asndecode(parmp->varp, parmp->vartype,
               parmp->setp, parmp->setlen);
```

See Also *knsa_asndecode()*, *knsa_setvarfunc()*

2.4 Using the MIB Compiler

The KwikNet MIB Compiler is a utility program which will create a Management Information Base (MIB) for use with the KwikNet TCP/IP Stack and its SNMP Option. The MIB Compiler is provided ready for use on a PC or compatible running the Microsoft® MS-DOS® or Windows® 9x or NT operating system.

When the optional KwikNet SNMP components are installed on your hard disk, the MIB Compiler utility program and its related files are stored in directory *MIBCOMP* in your KwikNet installation directory.

File	**Purpose**
KN_MIB.EXE	KwikNet MIB Compiler (utility program)
KN_MIB.CT	KwikNet MIB Template File
SNMP1213.MIB	RFC-1213 MIB-II MIB Definition File

The MIB generation process has been illustrated in the block diagram of Figure 2.1-1.

The MIB Compiler reads the definition of your MIB from a set of one or more MIB Definition Files. It then generates a set of output files using its MIB Template File as a model for each. The output files include the MIB Access Module, the corresponding MIB Access Header File and an SNMP Variables Module. A MIB Documentation File can also be generated in either a numeric or tree form.

The MIB Access Module generated by the MIB Compiler must be edited to implement your application specific MIB Finder Procedures. Once you have edited this module, you must be very careful not to accidentally regenerate the module using the MIB Compiler. Doing so will produce a new copy of the MIB Access Module which will no longer contain your edited procedures. The MIB Compiler attempts to avoid such a disaster. If the MIB Access Module being generated, say *YOUR_MIB.C*, already exists, the MIB Compiler renames it to be *YOUR_MIB.BAK* and then generates a new copy of file *YOUR_MIB.C*. You then have an opportunity to recover your edited procedures from the backup file.

!!! Important Note !!!

If you regenerate a MIB Access Module without proper care, you may lose the edited MIB Finder Procedures which it contains.

Always duplicate or rename your MIB Access Module before generating a new version of the same file.

Running the MIB Compiler

The KwikNet MIB Compiler is a utility program which is provided ready for use on a PC or compatible running the Microsoft® MS-DOS® or Windows® 9x or NT operating system. The MIB Compiler must be started from the MS-DOS command line or from the Windows MS-DOS command prompt. It can also be started from the Windows RUN dialog box. Alternatively, you can create a Windows shortcut to the MIB Compiler's filename and then simply double click the shortcut's icon.

The directory which is in effect when the MIB Compiler is started is called the **working directory**. Hence, when the MIB Compiler is started from an MS-DOS command prompt, the MS-DOS current directory becomes the MIB Compiler's working directory. If the MIB Compiler is started from a Windows shortcut, the working directory is determined by the start path specified in the shortcut's link file. If the MIB Compiler is started from the Windows RUN dialog box, the working directory will be undefined.

The **command line** used to start the MIB Compiler is as follows:

```
KN_MIB [-f] mibdefn1 [mibdefn2 ...] [-i mibdefn3 ...] [-a cmdfile]
        [-c] [-h] [-n] [-t|-T] [-v] [-q|-q0|-q1|-q2|-q3] [-?]
```

The MIB Compiler utility program is named `KN_MIB.EXE`.

The MIB Compiler requires a list of one or more MIB Definition Files which it treats as a single MIB description. At least one MIB Definition File (`mibdefn1` in the example) must be provided.

All other parameters are optional as indicated by the enclosing `[]` brackets. Omit the `[` and `]` characters when entering an optional parameter. The symbol `|` indicates that only one of the set of optional parameters can be used. Option letters are case sensitive. The order of optional parameters is not important. The default condition for each option is defined in the option descriptions which follow.

If an option is repeated on the command line, an error message may be generated. If no error is observed, then the last (rightmost) use of an option will take precedence.

The file generation options `-c`, `-h`, `-n`, `-t`, `-T` and `-v` can be combined into a single option. For example, the option string `-nchT` is equivalent to `-n -c -h -T`. The option ordering and case sensitivity rules defined above still apply. If no file generation option is specified, the `-n` option is assumed.

Files on the Command Line

Three of the command line options provide control over file related commands. The `-f` option, if used, precedes a list of one or more MIB Definition Files which will be compiled. An implicit `-f` option is assumed if none is present.

Option `-i`, if used, precedes a list of one or more MIB Definition Files which will be parsed for object, symbol, type and macro definitions but will otherwise not be compiled.

To avoid long command lines, the "`-a cmdfile`" option can be used to inject command line parameters from a text file named `cmdfile` as though the parameters were directly entered on the command line. End of line characters are treated as whitespace.

Command Line Parameters

`mibdefnX`	Mandatory full path and file name of your MIB Definition File. A file name with extension `MIB`, such as `MY_MIB.MIB`, is recommended. The working directory is used if no path, or a relative path, is specified. Multiple MIB Definition Files may be specified.
`-c`	Generate a MIB Access Module named `mibdefn1.C`. This module will contain the raw MIB Finder Procedures which you must edit to provide access to your custom MIB variables.
`-h`	Generate a MIB Access Header File named `mibdefn1.H`. This file contains C structure and constant definitions and function prototypes for use by the MIB Access Module.
`-n`	Generate a MIB Documentation File `mibdefn1.NUM` in numeric format. This file lists the object identifier, its text name and the type of each MIB variable defined in your set of MIB Definition Files.
`-t or -T`	Generate a MIB Documentation File `mibdefn1.TRE` in tree format. This file shows the MIB tree structure containing all of the MIB variables defined in your set of MIB Definition Files. The `-t` option renders the tree using only standard ASCII characters. The `-T` option permits the use of IBM extended graphical characters.
`-v`	Generate the SNMP Variables Module `SNMPVARS.C`. This file contains the C data representation for all of the MIB variables which are available within your SNMP managed device. You must specify all of your MIB Definition Files in the input file list when you generate the SNMP Variables Module. If your device is to support the MIB-II variables defined in RFC-1213, you must include the KwikNet MIB Definition File `SNMP1213.MIB`.
`-qN`	The `-qN` option controls the display of messages by the MIB Compiler. The `-q` or `-q0` option inhibits the display of any messages. The `-q1` option enables the display of error messages only. The `-q2` option enables the display of error messages and simple progress messages. The `-q3` option enables the display of error messages and more detailed progress messages. Unless otherwise specified, the `-q2` option is assumed.
`-?`	Use the `-?` option to display a helpful summary of the command line syntax and available options.

KwikNet Sample MIB

Included with the KwikNet MIB Compiler are the sample files required to create the custom MIB for the sample managed device described in this chapter. The following files, located in directory *MIBCOMP\SAMPLE* in your KwikNet installation directory, are provided.

File	**Purpose**	
KNSAMMIB.MIB	Sample MIB Definition File	
KNSAMMIB.H	Sample MIB Access Header File	
KNSAMMIB.C	Sample MIB Access Module	(in edited form)
KNSAMMIB.NUM	Sample MIB Documentation File	(in numeric form)
KNSAMMIB.TRE	Sample MIB Documentation File	(in tree form)

File *KNSAMMIB.MIB* is the MIB Definition File from which the other files were derived. MIB Access File *KNSAMMIB.C* was generated by the MIB Compiler and then edited to implement the MIB Finder Procedures listed in Figure 2.3-3. The remaining files can be regenerated as follows.

In order to avoid losing the edited MIB Finder Procedures in file *KNSAMMIB.C*, rename the file *KNSAMMIB.MFP*. Then generate the raw MIB Access Module *KNSAMMIB.C* and its MIB Access Header File. The MIB Documentation Files *KNSAMMIB.NUM* in numeric form and *KNSAMMIB.TRE* in tree form can be generated at the same time. The *-T* option permits the use of IBM graphical characters to draw the tree structure.

```
RENAME KNSAMMIB.C KNSAMMIB.MFP
KN_MIB KNSAMMIB.MIB -hcnT
```

You should then save the raw MIB Access Module *KNSAMMIB.C* and restore the original edited version of the file provided with KwikNet.

```
RENAME KNSAMMIB.C KNSAMMIB.RAW
RENAME KNSAMMIB.MFP KNSAMMIB.C
```

To generate the SNMP Variables Module *SNMPVARS.C* for this application, use the following command line. Note that in this case, support for MIB-II variables per RFC-1213 is not being included.

```
KN_MIB KNSAMMIB.MIB -v
```

In this simple example, the SNMP Variables Module *SNMPVARS.C* could have been generated together with the MIB Access Module using the following command.

```
KN_MIB KNSAMMIB.MIB -hcnTv
```

To include support for MIB-II variables per RFC-1213 for this application, you must generate the SNMP Variables Module *SNMPVARS.C* using the following command line. The MIB Definition File *SNMP1213.MIB* must be included in the list of such files to provide MIB-II support.

```
KN_MIB KNSAMMIB.MIB SNMP1213.MIB -v
```

It is important to note that when MIB-II support is required, the MIB Definition File *SNMP1213.MIB* must NOT be included in the generation of your MIB Access Module. Hence the SNMP Variables Module *SNMPVARS.C* must be generated separately from the MIB Access Module as just illustrated.

Warning

Your MUST NOT include the RFC-1213 MIB Definition File *SNMP1213.MIB* in the generation of any MIB Access Header File or MIB Access Module. Doing so will duplicate information already present in the KwikNet SNMP header file *SNMP1213.H* and library module *SNMP1213.C*.

The MIB Documentation Files generated in this example do not include descriptions of the MIB-II variables. If you include MIB-II support in your SNMP Variables Module *SNMPVARS.C* and want the MIB-II variable descriptions to appear in your MIB Documentation File(s), create the MIB Documentation File(s) separate from all other generated files as follows.

```
KN_MIB KNSAMMIB.MIB SNMP1213.MIB -nT
```

Using Multiple MIBs

A complex SNMP managed device may have several components which must be managed separately. For example, a robotic controller might also include a front panel display. You decide that it is best to define two MIBs for this device: one for the robot and one for the display panel. You therefore create MIB Definition Files *RBOT_CTL.MIB* for the robot and *RBOT_PNL.MIB* for the display panel.

The MIB for this device can be generated as follows.

```
KN_MIB RBOT_CTL.MIB -hc
KN_MIB RBOT_PNL.MIB -hc
```

The MIB Access Modules *RBOT_CTL.C* and *RBOT_PNL.C* must be edited to implement the MIB Finder Procedures for the MIB groups and tables which these MIBs describe.

To generate the SNMP Variables Module *SNMPVARS.C* for this device, use the following command line. The MIB Definition File *SNMP1213.MIB* must be included in the file list if the device must also provide MIB-II support.

```
KN_MIB RBOT_CTL.MIB RBOT_PNL.MIB SNMP1213.MIB -v
```

Managing a Library of MIBs

A complex SNMP managed device may have several components or operating modes which must be managed separately. For example, a device might be configurable to serve one of several different purposes. You decide that each use of the device is best defined with a unique MIB which is unencumbered by the other possible uses of the device. You then have a library (i.e. a collection) of individual MIBs from which the complete MIB for a device can be generated. The MIB for a specific device is constructed from the set of MIBs which describe that device's actual capabilities.

Assume that your MIB library includes a collection of MIB Definition Files named `LIBMIBnn.MIB`, where `nn` varies from `01` to `99`. You create a custom MIB Definition File `DVCA_MIB.MIB` which describes the MIB variables, if any, which are unique to that particular device. The device also needs MIBs `LIBMIB05.MIB`, `LIBMIB12.MIB` and `LIBMIB36.MIB` from your MIB library.

The simplest implementation of the MIB for this device can be generated as follows.

```
KN_MIB DVCA_MIB.MIB -hc
KN_MIB LIBMIB05.MIB -hc
KN_MIB LIBMIB12.MIB -hc
KN_MIB LIBMIB36.MIB -hc
KN_MIB DVCA_MIB.MIB LIBMIB05.MIB LIBMIB12.MIB LIBMIB36.MIB -nT
```

The MIB Access Modules `DVCA_MIB.C`, `LIBMIB05.C`, `LIBMIB12.C` and `LIBMIB36.C` must be edited to implement the MIB Finder Procedures for the MIB groups and tables which these MIBs describe. Of course, once you have edited and tested the MIB Access Modules for your standard MIBs, you will probably save them in your MIB library along with the corresponding MIB Access Header Files.

To generate the SNMP Variables Module `SNMPVARS.C` for this device, use the following command line. The MIB Definition File `SNMP1213.MIB` must be included in the file list if the device must also provide MIB-II support.

```
KN_MIB DVCA_MIB.MIB LIBMIB05.MIB LIBMIB12.MIB LIBMIB36.MIB
         SNMP1213.MIB -v
```

If the list of MIB Definition Files results in an excessively long command line, you can concatenate the MIB Definition Files into a single temporary file which you present to the MIB Compiler. For example, the SNMP Variables Module `SNMPVARS.C` produced above could also be generated as follows.

```
COPY DVCA_MIB.MIB+LIBMIB05.MIB+LIBMIB12.MIB TMP1_MIB.MIB
COPY TMP1_MIB.MIB+LIBMIB36.MIB+SNMP1213.MIB TMP2_MIB.MIB
KN_MIB TMP2_MIB.MIB -v
```

Note

If the SNMP agent cannot access the standard MIB-II variables, you may have omitted the MIB Definition File `SNMP1213.MIB` when you generated the SNMP Variables Module, `SNMPVARS.C`.

2.5 Compiling and Linking a MIB

The Management Information Base (MIB) for your SNMP managed device is constructed from one or more MIB Access Modules and the SNMP Variables Module *SNMPVARS.C*. These modules are created for you by the KwikNet MIB Compiler. Each MIB Access Module must have been edited to provide MIB Finder Procedures for each of the MIB groups and tables described in the module.

The MIB Access Modules and the SNMP Variables Module are C source files which must be compiled to produce object modules which are then linked as part of your KwikNet application.

In order to compile these modules, the following KwikNet header files must be present in the same directory as the C source files. Alternatively, you may choose to define the path to the header files using compiler switches or environment variables.

KN_LIB .H	KwikNet Library Configuration Module
KN_API .H	KwikNet Application Interface definitions
KN_COMN .H	KwikNet Common Interface definitions
KN_OSIF .H	KwikNet OS Interface definitions
KNZZZCC .H	KwikNet compiler specific definitions
KN_SNMP .H	KwikNet SNMP Application Interface definitions

Header file *KN_LIB.H* is a copy of your KwikNet Library Configuration Module from your KwikNet library directory. This file is created as a byproduct of the KwikNet Library construction process described in Chapter 3.2 of the KwikNet TCP/IP Stack User's Guide.

Header files *KN_API.H*, *KN_COMN.H* and *KN_OSIF.H* are the KwikNet files with which all application modules must be compiled. These files will be found in KwikNet installation directory *INET*.

Header file *KNZZZCC.H* is the compiler specific file which will be found in KwikNet installation directory *TOOLXXX*, where *XXX* is KADAK's three character mnemonic for a particular vendor's C tools.

Header file *KN_SNMP.H* is the KwikNet SNMP file with which all SNMP application modules must be compiled. This file will be found in KwikNet installation directory *SNMP*.

Finally, note that when compiling a MIB Access Module, the C compiler must have access to the corresponding MIB Access Header File. If necessary, copy the MIB Access Header File to the directory containing the MIB Access Module being compiled.

The SNMP MIB modules are compiled using exactly the same C command line switches as are used for compiling the C modules in the KwikNet libraries. These command line switches are defined in the tailoring file *KNZZZCC.INC* which you used to create your KwikNet libraries with your particular C compiler. Tailoring files are described in Chapter 3.2 of the KwikNet TCP/IP Stack User's Guide.

Linking the MIB with your Application

Your application link and/or locate specification files must be updated to include your MIB Access Module(s), the SNMP Variables Module and the KwikNet SNMP Library.

The object module for the SNMP Variables Module, *SNMPVARS.OBJ*, must be linked prior to the KwikNet libraries. It is recommended that it be inserted along with your KwikNet Network Configuration Module.

The object module for your MIB Access Module must be linked prior to the KwikNet libraries. It is recommended that it be linked after the SNMP Variables Module *SNMPVARS.OBJ*. If you have more than one MIB Access Module, collect the corresponding object files and link them following the SNMP Variables Module *SNMPVARS.OBJ*. If your MIB Finder Procedures reference other application specific support modules, be sure to link those modules explicitly or link with the library containing them.

Link the KwikNet SNMP Library file *KNnnnSNM.LIB* after all object modules but prior to the KwikNet IP Library.

With these changes to your application link and/or locate specification files in place, you can link and create an updated KwikNet application with SNMP support included.

Note that object and library files may have extensions *.O* or *.A* or some other extension as dictated by the toolset which you are using.

Using a MIB Library File

If your MIB has more than one MIB Access Module, you can combine the compiled object modules into one or more custom MIB library files using your object module librarian. Do NOT include the object module for the SNMP Variables Module, *SNMPVARS.OBJ*, in any such library.

When linking a MIB library file, insert it following the KwikNet SNMP Library *KNnnnSNM.LIB*.

Note

If, when linking your application, the MIB Finder Procedure *var_XXX* for the MIB group or table named *XXX* is undefined, you probably omitted the MIB Access Module which provides that procedure or inadvertently altered its name when editing the procedure.

This page left blank intentionally.

3. KwikNet SNMP Services

3.1 Introduction to SNMP Services

KwikNet provides a set of service procedures which control the operation of your KwikNet SNMP Agent. These service procedures reside in the KwikNet SNMP Library which you must link with your application. A description of these service procedures is provided in Chapter 3.2. The descriptions are ordered alphabetically for easy reference.

Italics are used to distinguish programming examples. Procedure names and variable names which appear in narrative text are also displayed in italics. Occasionally a lower case procedure name or variable name may appear capitalized if it occurs as the first word in a sentence.

Vertical ellipses are used in program examples to indicate which a portion of the program code is missing. Most frequently this will occur in examples where fragments of application dependent code are missing.

```
:
: /* Continue processing */
:
```

Capitals are used for all defined KwikNet file names, constants and error codes. All KwikNet procedure, structure and constant names can be readily identified according to the nomenclature introduced in Chapter 1.3 of the KwikNet TCP/IP Stack User's Guide.

KwikNet Procedure Descriptions

A consistent style has been adopted for the description of the KwikNet SNMP service procedures. The procedure name is presented at the extreme top right and left as in a dictionary. This method of presentation has been chosen to make it easy to find procedures since they are ordered alphabetically.

Purpose A one-line statement of purpose is always provided.

Used by ■ Task □ ISP □ Timer Procedure □ Restart Procedure □ Exit Procedure □ Other

This block is used to indicate which application procedures can call the KwikNet procedure. A filled in box indicates that the procedure is allowed to call the KwikNet procedure. In the above example, only tasks would be allowed to call the procedure.

For AMX users, this block is used to indicate which of your AMX application procedures can call the KwikNet procedure. You are reminded that the term ISP refers to the Interrupt Handler of a conforming ISP.

...more

KwikNet Procedure Descriptions (continued)

Used by	■ Task ◻ ISP ◻ Timer Procedure ◻ Restart Procedure ◻ Exit Procedure ◻ Other
	For other multitasking systems, a task is any application task executing at a priority below that of the KwikNet Task. A Timer procedure is a function executed by a task of higher priority than the KwikNet Task. An ISP is a KwikNet device driver interrupt handler called from an RTOS compatible interrupt service routine. The other procedures do not exist.
	For a single threaded system, your App-Task (see glossary in Appendix A of the KwikNet TCP/IP Stack User's Guide) is the only task. An ISP is a KwikNet device driver interrupt handler called from an interrupt service routine. Timer, restart and exit procedures do not exist.
	The category Other is rarely present. When the category is needed, the word Other is replaced with a more meaningful term which is then explained in the body of the procedure description. Procedures which can only be called by the MIB Finder Procedures and MIB Set Procedures in your MIB Access Modules are identified in this fashion.
Setup	The prototype of the KwikNet procedure is shown. The KwikNet header file in which the prototype is located is identified. Include KwikNet header files *KN_LIB.H* and *KN_SNMP.H* for compilation.
	File *KN_LIB.H* is the KwikNet include file which corresponds to the KwikNet Libraries which your application uses. This file is created for you by the KwikNet Configuration Manager when you create your KwikNet Libraries. File *KN_LIB.H* automatically includes the correct subset of the KwikNet header files for a particular target processor.
	File *KN_SNMP.H* is the KwikNet include file which you must include if your application uses any SNMP services. This file is located in KwikNet installation directory *SNMP*.
Description	Defines all input parameters to the procedure and expands upon the purpose or method if required.
Returns	The outputs, if any, produced by the procedure are always defined. Most KwikNet procedures return an integer error status.
Restrictions	If any restrictions on the use of the procedure exist, they are described.
Note	Special notes, suggestions or warnings are offered where necessary.
See Also	A cross reference to other related KwikNet procedures is always provided if applicable.

3.2 SNMP Service Procedures

KwikNet provides a set of SNMP service procedures which your application can use to interact with the KwikNet SNMP Agent. These service procedures reside in the KwikNet SNMP Library which you must link with your application. Some of these procedures must only be called by the MIB Finder Procedures and MIB Set Procedures in your MIB Access Modules.

The following list summarizes these KwikNet SNMP service procedures. They are grouped functionally for easy reference.

SNMP Agent Operations

knsa_control	Control the operation of KwikNet SNMP Agent
knsa_address	Get the IP address and port number of the KwikNet SNMP Agent
knsa_config	Configure the KwikNet SNMP Agent
knsa_community	Modify the access rights of an SNMP community
knsa_traptarget	Add/remove an SNMP trap target to/from the trap target list
knsa_trap	Send an SNMP trap message to all known trap targets

General SNMP Operations

knsa_oidcmp	Compare two object identifiers
knsa_oidcpy	Copy an object identifier
knsa_oidfrombytes	Convert a byte array to an object identifier
knsa_oidtobytes	Convert an object identifier to a byte array

MIB Access Operations

knsa_asndecode	Decode an ASN.1 encoded MIB value
knsa_setvarrange	Specify the valid range of values for a MIB variable
knsa_setvarfunc	Specify the MIB Set Procedure which is to write to a MIB variable
knsa_oidcmpv	Compare an object identifier with a compressed object identifier
knsa_oidcpyv	Copy (extract) a compressed object identifier

Related KwikNet Services

kn_netstats	Log a statistics summary for the KwikNet SNMP Agent (See Chapter 4.6 of the KwikNet TCP/IP Stack User's Guide.)

Purpose **Get the IP address and Port Numbers of the KwikNet SNMP Agent**

Used by ■ Task □ ISP □ Timer Procedure □ Restart Procedure □ Exit Procedure

Setup Prototype is in file *KN_SNMP.H*.

```
#include "KN_LIB.H"
#include "KN_SNMP.H"
int knsa_address(struct in_addr *inadrp,
                    int *snmpportp, int *trapportp);
```

Description *Inadrp* is a pointer to a structure into which the SNMP agent will store the IP address, in net endian form, which SNMP managers can use to communicate with the agent. The BSD structure *in_addr* is defined in file *KN_API.H* as follows:

```
struct in_addr {
  unsigned long s_addr;        /* IP address (net endian)  */
  };
```

Snmpportp is a pointer to storage for the UDP port number on which the KwikNet SNMP Agent will accept SNMP messages.

Trapportp is a pointer to storage for the UDP port number to which the KwikNet SNMP Agent will direct SNMP trap messages.

Returns The value *0* is always returned.
The agent's IP address is stored at *inadrp->s_addr*.
The agent's SNMP port number is stored at **snmpportp*.
The agent's SNMP trap port number is stored at **trapportp*.

See Also *knsa_control()*

Purpose **Decode an ASN.1 Encoded MIB Value**

Used by ◻ MIB Finder Procedure ■ MIB Set Procedure

Setup Prototype is in file *KN_SNMP.H*.

```
#include "KN_LIB.H"
#include "KN_SNMP.H"
void knsa_asndecode(const unsigned char *var_valp,
                    int var_val_type,
                    unsigned char *destp,
                    int destlen);
```

Description *Var_valp* is a pointer to the ASN.1 encoded MIB variable value.

Var_val_type is the ASN.1 type of the encoded variable.

Destp is a pointer to storage for the decoded MIB variable value.

Destlen is the length, in bytes, of the destination buffer referenced by parameter *destp*.

If *var_val_type* specifies a numeric type for the MIB variable, then four bytes of storage must be provided. If *var_val_type* specifies a MIB variable whose type permits values of variable length, then sufficient storage must be provided to hold the longest expected value.

Note 1 MIB variables which are strings (ASN.1 type *ASN_OCTET_STR*) are a special case. The storage length *destlen* specifies the maximum allowed string length, excluding a terminating *'\0'* character. However, the buffer referenced by *destp* must provide *destlen+1* bytes of storage to allow for the terminating *'\0'* character which procedure *knsa_asndecode()* always appends to a string value.

...more

Note 2 The MIB Set Procedure calling *knsa_asndecode()* is executed by the SNMP agent as a consequence of a request from a MIB Finder Procedure. The MIB Finder Procedure uses procedure *knsa_setvarfunc()* to make such a request.

The MIB Finder Procedure can call *knsa_setvarrange()* to establish the range of permissible values for the MIB variable. This action can influence the subsequent operation of procedure *knsa_asndecode()* when eventually called by the MIB Set Procedure. The effect is as follows.

If the MIB variable has a variable length value (ASN.1 types *ASN_OCTET_STR*, *SMI_IPADDRESS* or *SMI_OPAQUE*), then the upper limit of the MIB value established by the call to *knsa_setvarrange()* will be used by *knsa_asndecode()* as the length of the MIB storage buffer, overriding your parameter *destlen*.

Returns The decoded MIB value is stored in the data buffer referenced by parameter *destp*. If the MIB variable is a string, a terminating *'\0'* character will have been appended to the string by this procedure.

Example See the usage illustrated in the MIB Set Procedure specification provided in Chapter 2.3.2.

See Also *knsa_setvarrange()*, *knsa_setvarfunc()*

Purpose **Modify the Access Rights of an SNMP Community**

Used by ■ Task □ ISP □ Timer Procedure □ Restart Procedure □ Exit Procedure

Setup Prototype is in file *KN_SNMP.H*.

```
#include "KN_LIB.H"
#include "KN_SNMP.H"
int knsa_community(const char *community, unsigned int access);
```

Description *Community* is a pointer to a text string which specifies the name of an SNMP community for which the access rights are to be adjusted.

Access is one of the following constants which determine the action to be taken. These constants are defined in header file *KN_SNMP.H*.

KNSA_DELETE	Delete the SNMP agent from the community
KNSA_RONLY	Grant read-only access to the community
KNSA_WONLY	Grant write-only access to the community
KNSA_RWRITE	Grant read/write access to the community

If the SNMP agent already belongs to the specified community, the access rights for the community will be adjusted per parameter *access*. If you grant access to a community to which the SNMP agent does not already belong, the SNMP agent will be added to that community. The agent can only be deleted from a community to which it belongs.

Returns If successful, a value of *0* is returned.

The access rights for the specified community are adjusted per parameter *access*. If you grant access to a community to which the SNMP agent does not already belong, the SNMP agent is added to that community. If you deny access by a community to which the SNMP agent belongs, the agent will be deleted from that community.

On failure, the error status *-1* is returned. Reasons for failure include:

The *community* pointer is *NULL*.
The *community* string is too long or is of zero length.
The *access* value is not one of the specified constants.
The SNMP agent cannot be added to the community without exceeding the agent's community membership limit.
The SNMP agent cannot be deleted from a community to which it does not belong.

Note The maximum length of a community name string and the maximum number of communities to which the KwikNet SNMP Agent can belong are specified in your KwikNet Library Parameter File (see Chapter 1.3).

See Also *knsa_config()*

Purpose **Configure the KwikNet SNMP Agent**

Used by ■ Task □ ISP □ Timer Procedure □ Restart Procedure □ Exit Procedure

Setup Prototype is in file *KN_SNMP.H*.

```
#include "KN_LIB.H"
#include "KN_SNMP.H"
int knsa_config(const struct knsa_snmpinfo *cfgp);
```

Description *Cfgp* is a pointer to a structure which contains SNMP agent configuration information. The configuration information which you provide will override the default configuration used to initialize the SNMP agent. The default parameters are located in source module *SNMPUSER.C* in KwikNet directory *SNMP*. The structure *knsa_snmpinfo* is defined in header file *KN_SNMP.H* as follows:

```
struct knsa_snmpinfo {
  /* MIB-II System Group Information (RFC-1213)            */
  const char   *sysDescr;    /* System description         */
  const char   *sysContact;  /* Contact information        */
  const char   *sysName;     /* Name of the node           */
  const char   *sysLocation; /* Location of the node       */
  const knsa_subid *sysObjID; /* Vendor ID subid array     */
  int          sysObjIDlen;  /* # of subids in Vendor ID   */
  int          sysServices;  /* Services provided by node  */

  /* Array of Community Access Definitions                 */
  const struct knsa_accessdef *commAcl;
  int          commCount;    /* # of communities defined   */
  int          rsv1;         /* Reserved for alignment     */

  /* Default trap community name string                    */
  const char *trapDefaultComm;

  /* Array of Trap Target Definitions                      */
  const struct knsa_trapdef *trapTargets;
  int          trapCount;    /* # of trap targets defined  */
  int          rsv2;         /* Reserved for alignment     */
  };
```

The MIB-II System Group Information is defined by RFC-1213. A brief description of these parameters can be found in the KwikNet MIB Definition File *SNMP1213.MIB* located in directory *MIBCOMP* in your KwikNet installation directory. For examples, review the default configuration in source module *SNMPUSER.C* in KwikNet directory *SNMP*.

If, within this structure, a pointer parameter is set to *NULL* or an integer parameter is set to *0*, the corresponding SNMP agent configuration parameter will not be altered. You can therefore use this procedure to selectively alter specific configuration parameters without altering the configuration as a whole.

...more

Description ...continued

Member *commAcl* provides a pointer to an array of Community Access Control Definitions. Member *commCount* specifies the number of communities defined in the array. These definitions specify the communities to which the SNMP agent is to belong. This list of communities completely replaces the list of communities, if any, to which the SNMP agent currently belongs. Each definition is a *knsa_accessdef* structure which is defined in header file *KN_SNMP.H* as follows:

```
struct knsa_accessdef {
  const char *name;       /* Community name string      */
  unsigned short access;  /* Community access rights    */
                          /* One of KNSA_RONLY,         */
                          /* KNSA_WONLY or KNSA_RWRITE  */
  short       rsv1;       /* Reserved for alignment     */
  };
```

Member *trapDefaultComm* is a pointer to a text string specifying the community name to be used in any SNMP trap message sent to a trap target which has a *NULL* community name pointer in its definition.

Member *trapTargets* provides a pointer to an array of Trap Target Definitions. Member *trapCount* specifies the number of trap targets defined in the array. These definitions specify the trap targets to which the SNMP agent is to direct SNMP trap messages. This list of trap targets completely replaces the list of targets, if any, currently in use by the SNMP agent. Each definition is a *knsa_trapdef* structure which is defined in header file *KN_SNMP.H* as follows:

```
struct knsa_trapdef {
  const char *name;       /* Community name string      */
                          /* If NULL, the default trap  */
                          /* community will be used     */
  struct in_addr fhost;   /* IP address of trap target  */
  };
```

Returns If successful, a value of *0* is returned.

On failure, the error status *-1* is returned. Failure indicates that at least one of the configuration parameters is invalid.

Note The maximum length of a community name string and the maximum number of communities to which the KwikNet SNMP Agent can belong are specified in your KwikNet Library Parameter File (see Chapter 1.3).

If you have not enabled SNMP traps in your KwikNet Library Parameter File, structure members *trapDefaultComm*, *trapTargets* and *trapCount* will be ignored.

See Also *knsa_community()*, *knsa_traptarget()*

Purpose **Control the Operation of the KwikNet SNMP Agent**

Used by ■ Task ◻ ISP ◻ Timer Procedure ◻ Restart Procedure ◻ Exit Procedure

Setup Prototype is in file *KN_SNMP.H*.

```
#include "KN_LIB.H"
#include "KN_SNMP.H"
int knsa_control(int cmask, int cvalue);
```

Description This procedure is used to control the operation of the KwikNet SNMP Agent. Your application can call this procedure to enable or disable the agent from servicing SNMP requests or generating SNMP traps.

When KwikNet first starts and the SNMP agent is initialized, SNMP trap generation and the service of SNMP requests are both disabled. However, if you have enabled the automatic start option in your KwikNet Library Parameter File, the SNMP agent immediately enables both operations.

Cmask is a bit mask which identifies the SNMP agent features which are to be enabled or disabled. Only the features specified by this mask will be affected. Parameter *cmask* is created by ORing one or more of the following constants whose definitions are in header file *KN_SNMP.H*.

KNSA_CTRL_REQUESTS	Service SNMP requests
KNSA_CTRL_TRAPS	Generate SNMP traps

Cvalue is a bit mask which specifies whether the feature identified by parameter *cmask* is to be enabled or disabled. A value of *0* will disable all of the features specified by *cmask*. Add any of the above constants to *cvalue* to enable the corresponding feature.

Set *cmask* and *cvalue* to *0* to read the current control state of the SNMP agent without affecting its operation.

Returns The previous control state of the SNMP agent is always returned. The bit mask constants defined above can be used to isolate the current state of the feature controlled by the mask bit.

The special mask *KNSA_CTRL_WARM* can be applied to the result to determine if the SNMP agent has ever been permitted to service SNMP requests. If the masked result is *0*, then the SNMP agent has never been active.

Enabling an already enabled feature or disabling an already disabled feature has no effect.

...more

Note 1 Attempting to control the generation of SNMP traps will have no effect unless the SNMP trap feature is enabled in your KwikNet Library Parameter File (see Chapter 1.3).

Note 2 If SNMP trap generation is enabled when you call this procedure to enable the servicing of SNMP requests, the SNMP agent will generate an SNMP trap. The trap will only be generated if the servicing of SNMP requests by the agent is not already enabled.

A cold start trap (*`KNSA_TRAP_COLDSTART`*) will be generated the first time the servicing of SNMP requests is enabled. Thereafter, a warm start trap (*`KNSA_TRAP_WARMSTART`*) will be generated.

See Also *`knsa_address()`*

Purpose **Compare Two Object Identifiers**

Used by ■ Task ■ ISP ■ Timer Procedure ■ Restart Procedure ■ Exit Procedure ■ MIB Finder Procedure ■ MIB Set Procedure

Setup Prototype is in file *`KN_SNMP.H`*.

```
#include "KN_LIB.H"
#include "KN_SNMP.H"
int knsa_oidcmp(const knsa_subid *name1, int length1,
                const knsa_subid *name2, int length2);
```

Description *`Name1`* is a pointer to the subid array specifying the first of two object identifiers to be compared.

`Length1` is the number of sub-identifiers in the subid array referenced by parameter *`name1`*.

`Name2` is a pointer to the subid array specifying the second of the two object identifiers to be compared.

`Length2` is the number of sub-identifiers in the subid array referenced by parameter *`name2`*.

Returns

`-1` if object identifier per *`name1`* < object identifier per *`name2`*
`0` if object identifier per *`name1`* = object identifier per *`name2`*
`1` if object identifier per *`name1`* > object identifier per *`name2`*

See Also *`knsa_oidcmpv(), knsa_oidcpy(), knsa_oidcpyv(), knsa_oidfrombytes(), knsa_oidtobytes()`*

Purpose **Compare an Object Identifier with a Compressed Object Identifier**

Used by ■ MIB Finder Procedure ■ MIB Set Procedure

Setup Prototype is in file *KN_SNMP.H*.

```
#include "KN_LIB.H"
#include "KN_SNMP.H"
int knsa_oidcmpv(const knsa_subid *name1, int length1,
                 const struct knsa_var *vp);
```

Description *Name1* is a pointer to the subid array specifying the first of two object identifiers to be compared.

Length1 is the number of sub-identifiers in the subid array referenced by parameter *name1*.

Vp is a pointer to a private KwikNet MIB variable description which contains the compressed object identifier to which the object identifer specified by parameter *name1* is to be compared. This pointer is presented by the SNMP agent as a parameter to your MIB Finder Procedure or MIB Set Procedure.

Returns

-1 if object identifier per *name1* < object identifier per **vp*
0 if object identifier per *name1* = object identifier per **vp*
1 if object identifier per *name1* > object identifier per **vp*

See Also *knsa_oidcmp(), knsa_oidcpy(), knsa_oidcpyv(), knsa_oidfrombytes(), knsa_oidtobytes()*

Purpose **Copy an Object Identifier**

Used by ■ Task ■ ISP ■ Timer Procedure ■ Restart Procedure ■ Exit Procedure ■ MIB Finder Procedure ■ MIB Set Procedure

Setup Prototype is in file *`KN_SNMP.H`*.

```
#include "KN_LIB.H"
#include "KN_SNMP.H"
knsa_subid *knsa_oidcpy(knsa_subid *destp,
                   const knsa_subid *srcp, int length);
```

Description *`Destp`* is a pointer to storage for a subid array into which an object identifier can be copied.

`Srcp` is a pointer to the subid array containing the object identifier which is to be copied.

`Length` is the number of sub-identifers in the subid array referenced by parameter *`srcp`*.

Returns The destination subid array pointer *`destp`*.

The subid array at *`*srcp`* is copied to *`*destp`*.

Restriction There must be sufficient storage at the location referenced by *`destp`* to hold a subid array with the number of sub-identifiers specified by parameter *`length`*.

See Also *`knsa_oidcmp(), knsa_oidcmpv(), knsa_oidcpyv(), knsa_oidfrombytes(), knsa_oidtobytes()`*

Purpose **Copy (Extract) a Compressed Object Identifer**

Used by ■ MIB Finder Procedure ■ MIB Set Procedure

Setup Prototype is in file *KN_SNMP.H*.

```
#include "KN_LIB.H"
#include "KN_SNMP.H"
knsa_subid *knsa_oidcpyv(knsa_subid *destp,
                         const struct knsa_var *vp);
```

Description *Destp* is a pointer to storage for a subid array into which an object identifier can be copied.

Vp is a pointer to a private KwikNet MIB variable description which contains the compressed object identifier which is to be extracted and stored into the subid array specified by parameter *destp*. This pointer is presented by the SNMP agent as a parameter to your MIB Finder Procedure or MIB Set Procedure.

The number of sub-identifiers in the compressed object identifier can be accessed as *vp->grp->namelen*.

Returns The destination subid array pointer *destp*.

The compressed object identifier in the MIB variable description referenced by parameter *vp* is converted to an object identifier and stored in the subid array referenced by parameter *destp*.

Restriction There must be sufficient storage at the location referenced by *destp* to hold the subid array extracted from the MIB variable description.

See Also *knsa_oidcmp(), knsa_oidcmpv(), knsa_oidcpy(), knsa_oidfrombytes(), knsa_oidtobytes()*

Purpose **Convert a Byte Array to an Object Identifier**

Used by ■ Task ■ ISP ■ Timer Procedure ■ Restart Procedure ■ Exit Procedure ■ MIB Finder Procedure ■ MIB Set Procedure

Setup Prototype is in file *KN_SNMP.H*.

```
#include "KN_LIB.H"
#include "KN_SNMP.H"
void knsa_oidfrombytes(knsa_subid *destp,
                       const void *srcp, int count);
```

Description *Destp* is a pointer to storage for a subid array into which an object identifier can be stored.

Srcp is a pointer to the array of bytes which is to be converted into an object identifer.

Count is the number of bytes in the source byte array. This number therefore also specifies the number of sub-identifiers which will be present in the resulting object identifier.

Returns Nothing

The array of *count* bytes at **srcp* is treated as an array of unsigned characters. Each byte is converted to a sub-identifier of type *knsa_subid* in the range 0 to 255. The resulting array of sub-identifiers is stored in the subid array at **destp*.

See Also *knsa_oidcmp(), knsa_oidcpy(), knsa_oidtobytes()*

Purpose **Convert an Object Identifier to a Byte Array**

Used by ■ Task ■ ISP ■ Timer Procedure ■ Restart Procedure ■ Exit Procedure ■ MIB Finder Procedure ■ MIB Set Procedure

Setup Prototype is in file *KN_SNMP.H*.

```
#include "KN_LIB.H"
#include "KN_SNMP.H"
void knsa_oidtobytes(void *destp,
                    const knsa_subid *srcp, int count);
```

Description *Destp* is a pointer to storage for the array of bytes which are to be derived from an object identifer.

Srcp is a pointer to the subid array which contains an object identifier which is to be converted into an array of bytes.

Count is the number of sub-identifiers in the object identifier referenced by parameter *srcp*. This number also determines the number bytes which must be available in the destination byte array referenced by parameter *destp*.

Returns Nothing

Each of the sub-identifiers in the subid array referenced by *srcp* is masked to 8 bits and stored in the byte array at **destp*. The most significant 8 or 24 bits of each sub-identifier are ignored.

See Also *knsa_oidcmp(), knsa_oidcpy(), knsa_oidfrombytes()*

Purpose **Install a MIB Set Procedure to be Used to Set a MIB Value**

Used by ■ MIB Finder Procedure □ MIB Set Procedure

Setup Prototype is in file *KN_SNMP.H*.

```
#include "KN_LIB.H"
#include "KN_SNMP.H"
int knsa_setvarfunc(KNSA_ACCESS setfunc);
```

Description This procedure is called by a MIB Finder Procedure to install a MIB Set Procedure which the SNMP agent will call to set the value of the MIB variable located by the MIB Finder Procedure.

Parameter *setfunc* is a pointer to the MIB Set Procedure.

The MIB Finder Procedure can cancel a MIB Set Procedure which it has already installed by calling this procedure again with parameter *setfunc* set to *NULL*.

The type definition for a pointer to a MIB Set Procedure is specified in header file *KN_SNMP.H* as follows:

```
typedef int (*KNSA_ACCESS)(struct knsa_setparms *parmp);
```

The MIB Set Procedure must be coded according to the specification provided in Chapter 2.3.2. The specification includes a complete description of the parameters provided to the function in the structure referenced by *parmp*.

Returns A value of *0* is always returned.

Note The SNMP agent always cancels any previously installed MIB Set Procedure prior to calling a MIB Finder Procedure to locate a MIB variable. Hence any use of procedure *knsa_setvarfunc()* by the MIB Finder Procedure can only affect writes to the MIB variable which it successfully locates for the agent.

See Also *knsa_setvarrange()*

Purpose **Specify the Valid Range of Values for an SNMP Variable**

Used by ■ MIB Finder Procedure □ MIB Set Procedure

Setup Prototype is in file *KN_SNMP.H*.

```
#include "KN_LIB.H"
#include "KN_SNMP.H"
int knsa_setvarrange(long lovalue, long hivalue);
```

Description *Lovalue* is the smallest acceptable value for the SNMP variable which the MIB Finder Procedure has located for the SNMP agent.

Hivalue is the largest acceptable value for the SNMP variable which the MIB Finder Procedure has located for the SNMP agent.

If *lovalue* is *0* and *hivalue* is *-1*, then any range previously specified by the MIB Finder Procedure is discarded. This feature can be used by the MIB Finder Procedure to cancel an erroneous range specification.

If the MIB variable located by the MIB Finder Procedure has a numeric value, then the value of the MIB variable must be in the range *lovalue ≤ value ≤ hivalue*. The following ASN.1 types defined in header file *KN_SNMP.H* specify numeric MIB variables.

ASN_INTEGER, *SMI_COUNTER*, *SMI_GAUGE*, *SMI_TIMETICKS*

If the MIB variable located by the MIB Finder Procedure has a variable length value, then the length of the value of the MIB variable must be in the range *lovalue ≤ length ≤ hivalue*. The following ASN.1 types defined in header file *KN_SNMP.H* specify variable length MIB variables.

ASN_OBJECT_ID, *ASN_OCTET_STR*, *SMI_IPADDRESS*, *SMI_OPAQUE*

Returns If the range is accepted, a value of *0* is returned.

On failure, the error status *-1* is returned.
Failure indicates that *lovalue* is greater than *hivalue.*

Note The SNMP agent always cancels range checking prior to calling a MIB Finder Procedure to locate a MIB variable. Hence any use of procedure *knsa_setvarrange()* by the MIB Finder Procedure can only affect writes to the MIB variable which it successfully locates for the agent.

If a range check is specified with *knsa_setvarrange()* and a MIB Set Procedure is installed with *knsa_setvarfunc()*, then the range check will be performed by the SNMP agent upon return from the MIB Finder Procedure prior to calling the MIB Set Procedure.

See Also *knsa_setvarfunc()*, *knsa_asndecode()*

Purpose **Send an SNMP Trap Message to All Trap Targets**

Used by ■ Task ◻ ISP ◻ Timer Procedure ◻ Restart Procedure ◻ Exit Procedure ■ MIB Finder Procedure ■ MIB Set Procedure

Setup Prototype is in file *KN_SNMP.H*.

```
#include "KN_LIB.H"
#include "KN_SNMP.H"
int knsa_trap(int trapType,
                  int specificType,
                  const knsa_subid *objId,
                  int objLen,
                  int varCount,
                  const struct knsa_trapvar *vars);
```

Description *TrapType* specifies the SNMP trap type to be used in the trap message. Trap types are defined and described in RFC-1157. The following trap types are defined in header file *KN_SNMP.H*:

```
KNSA_TRAP_COLDSTART
KNSA_TRAP_WARMSTART
KNSA_TRAP_LINKDOWN
KNSA_TRAP_LINKUP
KNSA_TRAP_AUTHFAIL
KNSA_TRAP_EGPNEIGHBORLOSS
KNSA_TRAP_ENTERPRISESPECIFIC
```

SpecificType is a specific code which you must provide when you generate a trap of type *KNSA_TRAP_ENTERPRISESPECIFIC*. Specific codes are defined in an enterprise specific manner which is not defined in the RFCs. If you are not generating such a trap, this parameter will be ignored.

ObjId is a pointer to a subid array which contains an object identifier which uniquely indicates the managed device which is generating the trap. If this parameter is *NULL*, then the SNMP agent's system object identifier (as specified by MIB variable *sysObjectID*) will be used. Note that you can use procedure *knsa_config()* to adjust this SNMP agent configuration parameter.

ObjLen is the number of sub-identifiers in the subid array referenced by parameter *ObjId*. If *ObjId* is *NULL*, this parameter is ignored.

VarCount is the number of MIB variables which are to be included in the SNMP trap message sent to each trap target. This parameter therefore specifies the number of entries in the array of trap variables referenced by parameter *vars*. If no MIB variables are to be included in the SNMP trap message, set this parameter to *0*.

...more

Description …continued

Vars is a pointer to an array of trap variables. Each trap variable describes a MIB variable which is to be embedded in the SNMP trap message. If no MIB variables are to be included in the SNMP trap message, set this parameter to *NULL*.

Each trap variable is described in a structure of type *knsa_trapvar* which is defined in header file *KN_SNMP.H* as follows:

```
struct knsa_trapvar {
                                /* ObjectId of variable    */
  knsa_subid varName[KN_SNMP_SUBIDS];
  unsigned int   varNameLen;   /* Sub-ids in varName      */
  unsigned int   varValLen;    /* Octets in variable data */
  unsigned char  varType;      /* ASN.1 type of variable  */
  char rsv1, rsv2, rsv3;       /* Reserved for alignment  */
  const unsigned char *varBuf; /* A(actual variable data) */
  };
```

Member *varName* is a subid array which provides the object identifier of the MIB variable. The number of sub-identifiers in the object identifier is specified by member *varNameLen*.

Member *varBuf* is a pointer to the actual value of the MIB variable. The length of the MIB variable value, in bytes, is specified by member *varValLen*.

Member *varType* specifies the MIB variable type. This parameter must be one of the following ASN.1 or SMI types which are defined in header file *KN_SNMP.H*:

```
ASN_BOOLEAN        SMI_IPADDRESS
ASN_INTEGER        SMI_COUNTER
ASN_OCTET_STR      SMI_GAUGE
ASN_NULL           SMI_TIMETICKS
ASN_OBJECT_ID      SMI_OPAQUE
```

Returns If successful, a value of *0* is returned.

The SNMP trap message will have been sent to all trap targets currently known to the SNMP agent.

On failure, the error status *-n* is returned where *n* indicates the number of errors which occurred while sending the SNMP trap message to all trap targets.

See Also *knsa_config()*, *knsa_traptarget()*

Purpose **Add/Remove a Trap Target to/from the Trap Target List**

Used by ■ Task □ ISP □ Timer Procedure □ Restart Procedure □ Exit Procedure ■ MIB Finder Procedure ■ MIB Set Procedure

Setup Prototype is in file *KN_SNMP.H*.

```
#include "KN_LIB.H"
#include "KN_SNMP.H"
int knsa_traptarget(const char *community,
                    const struct in_addr *inadrp, int oper);
```

Description *Community* is a pointer to a text string which specifies the name of the SNMP community which is to be embedded in any SNMP trap message sent to this trap target. If this parameter is *NULL*, then the SNMP agent's default trap community will be used. Note that you can use procedure *knsa_config()* to adjust SNMP agent's default trap community.

Inadrp is a pointer to a structure which contains the IP address, in net endian form, to which the SNMP agent will send SNMP trap messages. The BSD structure *in_addr* is defined in file *KN_API.H* as follows:

```
struct in_addr {
  unsigned long s_addr;      /* IP address (net endian)  */
  };
```

Oper is one of the following constants which determine the action to be taken. These constants are defined in header file *KN_SNMP.H*.

KNSA_ADD	Add the trap target
KNSA_DELETE	Delete the trap target

Returns If successful, a value of *0* is returned. The trap target is added or deleted to the SNMP agents's list of trap targets.

On failure, the error status *-1* is returned. Reasons for failure include:

The *community* string is too long or is of zero length.
The IP address is *0.0.0.0*.
The trap target cannot be added because the trap target list is full.
The trap target cannot be deleted because it is not on the trap target list.

Note The maximum length of a community name string and the maximum number of trap targets to which the SNMP agent can send traps is specified in your KwikNet Library Parameter File (see Chapter 1.3).

To change the definition of a trap target, you must delete the trap target and then add it with the revised definitions.

See Also *knsa_config(), knsa_trap()*