KwikNet®

SNMP Agent

User's Guide

First Printing: February 15, 1999
Last Printing: September 15, 2005

Manual Order Number: PN303-9S

KADAK Products Ltd.
206 - 1847 West Broadway Avenue
Vancouver, BC, 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 during the product's initial support period. For technical support beyond the initial period, you must purchase a Technical Support Subscription. Contact KADAK for details. Please keep us informed of the primary user in your company to whom 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. 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, BC, Canada, V6J 1Y5

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

DISCLAIMER

KADAK Products Ltd. makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability and fitness for any particular purpose. Further, KADAK Products Ltd. reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of KADAK Products Ltd. to notify any person of such revision or changes.

TRADEMARKS

AMX in the stylized form and KwikNet are registered trademarks of KADAK Products Ltd.
AMX, AMX/FS, InSight, *KwikLook* and KwikPeg are trademarks of KADAK Products Ltd.
UNIX is a registered trademark of AT&T Bell Laboratories.
Microsoft, MS-DOS and Windows are registered trademarks of Microsoft Corporation.
All other trademarked names are the property of their respective owners.

KwikNet SNMP Agent User's Guide
Table of Contents

	Page
1. KwikNet SNMP Agent Overview	**1**
1.1 Introduction	1
Installation	2
The Treck SNMP User Manual	3
1.2 General Operation	4
Managed Data	4
Object Identifiers	5
SNMP Communities	5
SNMP Messages (Requests, Responses and Traps)	6
SNMP Security	7
1.3 KwikNet SNMP Library Configuration	8
1.4 SNMP Agent Operation	10
Multitasking Operation	10
Single Threaded Operation	11
SNMP Agent Definition	12
1.5 SNMP Traps	13
Trap Messages	13
Trap Targets	14
Authentication Traps	14
1.6 MIB Data Organization	15
MIB-II Support	17
1.7 Adding the SNMP Agent to Your Application	18
KwikNet SNMP Library	18
KwikNet Task Considerations	18
Reconstructing Your KwikNet Application	19
AMX Considerations	19
Performance Considerations	20
2. KwikNet MIB Construction	**21**
2.1 Introduction	21
MIB Definition Files	21
SNMP Variables Module	21
MIB Access Module	22
MIB Insert Module	22
Building a Custom MIB	23
2.2 SNMP MIB Sample	24
2.3 MIB Definition Files	26
MIB Organization	26

Table of Figures

Page

Figure 1.2-1 KwikNet SNMP v1 Messages (PDUs) 6
Figure 1.2-2 MIB Access Rights and Allowed Operations 7
Figure 1.6-1 Standard MIB Tree Structure 15
Figure 2.2-1 Sample SNMP Managed Device 24
Figure 2.2-2 MIB Structure for Sample Device 25
Figure 2.3-1 MIB Definition File for Sample Device 27

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 Chapter 1.7.4 and Chapter 2 of the Treck SNMP User Manual.

> Note
>
> The KwikNet SNMP Agent is founded upon the SNMP agent from Treck Inc. Hence you must become familiar with the SNMP application programming interface (API) described in the Treck SNMP User Manual.

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.

Installation

The KwikNet SNMP Agent is an optional component that is provided in three variants. The SNMP v1 Agent supports only SNMP v1. The SNMP v2 Agent supports both SNMP v1 and v2. The SNMP v3 Agent supports SNMP v1, v2 and v3.

When you install KwikNet, the standard distribution of the Turbo Treck TCP/IP Stack from Treck Inc., as delivered to you with KwikNet, will be installed in a directory named `TRECK`. The SNMP agent is installed in subdirectory `TRECK\SNMPD`.

The Treck SNMP User Manual

This manual offers an introduction to SNMP and its features as exemplified by SNMP v1. The Treck SNMP User Manual describes the SNMP agent in detail and must be used as your programming guide whether you are using SNMP v1, v2 or v3. Much of the documentation is tutorial in nature and for this reason deserves careful reading.

Chapter 1 provides an excellent introduction to the SNMP protocol, what it is and how it is used. Chapter 2 lists the RFCs from which the fundamental design of the SNMP agent has been derived.

Chapter 3 describes the SNMP agent in detail and must be mastered for proper use of the agent. Chapter 3.1 is devoted to the porting and configuration process. With KwikNet, the porting has been done and the SNMP agent is ready for use. The choice of SNMP features to be used in your application is easily made using the KwikNet Configuration Builder with its built-in help service to guide you to the proper selection.

The MIB Compiler which is used to create the SNMP Management Information Base (MIB) for your embedded product is described in Chapter 3.7. After the Treck TCP/IP Stack has been installed, you must unpack and install the MIB Compiler as instructed in Chapter 3.7.2.

Chapter 4 presents the Programmer's Reference, the application programming interface (API) giving you complete access to the SNMP v1, v2 and v3 features you will need to customize the SNMP agent to meet your particular requirements.

Note

The Treck TCP/IP User Manual describes how to port the Turbo Treck TCP/IP Stack for use with any RTOS and how to configure it for your own use. The Treck stack is ready for use with KwikNet and the SNMP agent.

The Treck port has been done. Furthermore, the KwikNet Configuration Builder can be used to configure KwikNet and the SNMP agent for your system without having to edit the Treck *TRSYSTEM.H* header file.

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 8-bit, 16-bit or 32-bit sub-identifiers, the variable type *`oid`* is introduced in Treck header file *`TRASN1.H`*. Using this definition, a subid array is as an array of *`n`* sub-identifiers of type *`oid`*.

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 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 SNMP v1 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 v1 Messages (PDUs)

SNMP Security

A limited security scheme is provided with SNMP v1. 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.

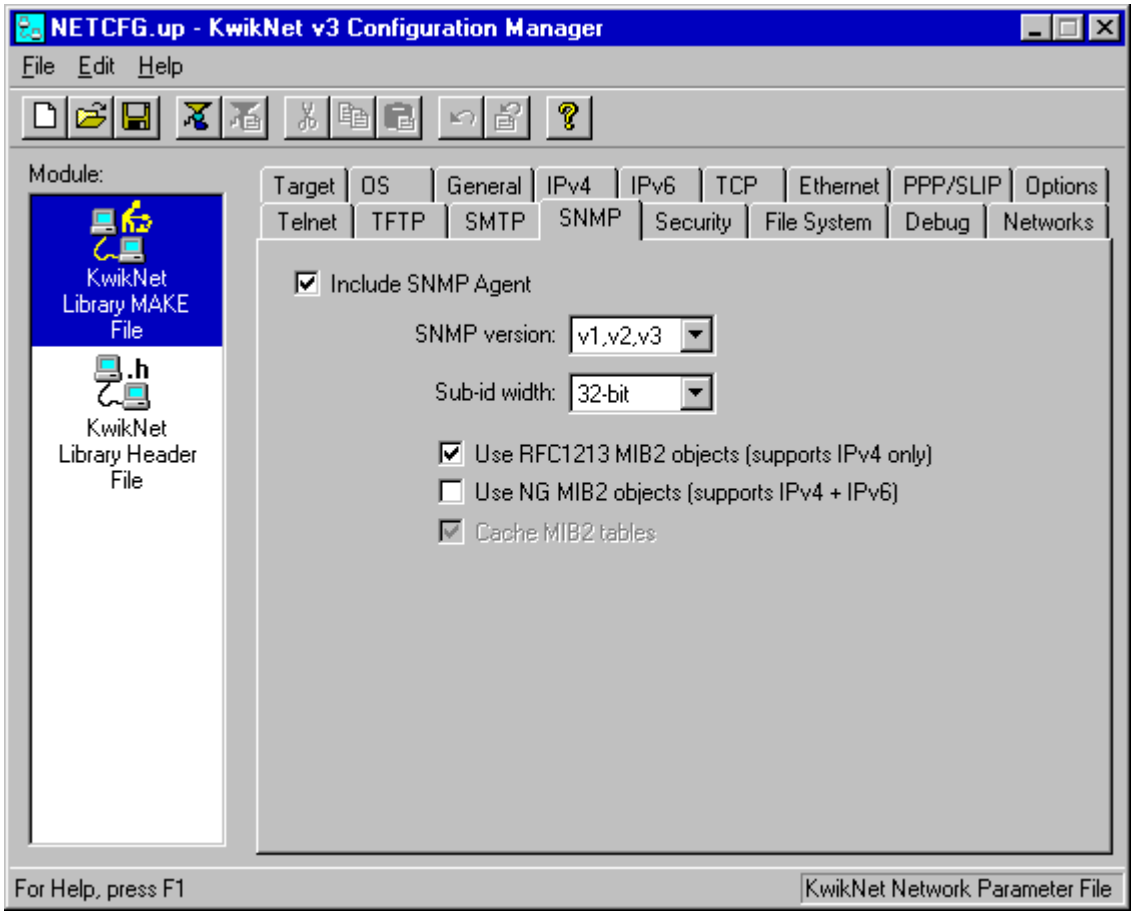
When the SNMP agent receives a request to fetch or modify the value of a MIB variable, it checks the access rights of the community and MIB variable specified in the request. If the community does not have access rights appropriate to the request, then the SNMP agent does not respond to the request. 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 Network Parameter File. The KwikNet SNMP Library parameters are edited on the SNMP property page. The layout of the window is shown below.



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.

SNMP Version

Select the version of the SNMP protocol that the agent will support.

Sub-id Width

Specify the number of bits to be used for the sub-identifier numbers in a MIB object identifier. Pick one of 8-bit, 16-bit or 32-bit from the pull down list. If you choose 8-bit sub-ids, the maximum sub-identifier value allowed in any MIB object identifier will be 255. If you choose 16-bit sub-ids, the maximum sub-identifier value allowed in any MIB object identifier will be 65535. If you require sub-ids greater than 65535, you must use 32-bit sub-ids.

Although the use of 8-bit or 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 8-bit or 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.

Use RFC1213 MIB2 Objects

Check this box to enable support for the RFC1213 MIB-II objects. Otherwise, leave this box unchecked.

This option only supports IPv4 objects. It does not support IPv6 MIB objects.

Use Next Generation MIB2 Objects

Check this box to enable support for the Next Generation MIB2 objects. Otherwise, leave this box unchecked.

This option supports IPv4 and IPv6 MIB objects.

Cache MIB2 Tables

Check this box if the SNMP agent must support GETNEXT requests with the following MIB2 tables: ipNetToMediaTable (ARP cache), ipRouteTable (Routing table), udpTable (UDP listening sockets) and tcpConnTable (TCP sockets).

This option is automatically enabled if you have selected either of the RFC1213 or Next Generation MIB2 Object options.

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.

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 are handled by the SNMP agent as described in Chapter 1.5.

Multitasking Operation

When used with a real-time operating system (RTOS) such as KADAK's AMX Real-Time Multitasking Kernel, the SNMP agent usually operates as an application task. Such a task is referred to as the agent task. Only one agent task is allowed.

An agent task is created and started just like any other application task. The agent task must properly configure itself using the services provided in the KwikNet SNMP Library so that it can act as an SNMP agent. When ready to begin operation, the agent task simply calls procedure *`tfSnmpdMain()`* to establish itself as the SNMP agent.

Such an application agent task ***must*** operate in blocking mode and at a priority below that of the KwikNet Task. There is no return from procedure *`tfSnmpdMain()`* until some other application task calls procedure *`tfSnmpdStop()`* requesting the agent to stop. All requests from SNMP managers will be serviced in the context of the agent task.

In a multitasking system, the SNMP agent can operate in an alternate fashion. The agent can execute within the context of the KwikNet Task, eliminating the need for an additional application task. Do not use this approach unless all MIB variables are readily accessible without delay. An application task must first properly configure the agent and then call procedure *`tfSnmpdMain()`* to start the SNMP agent, forcing it to operate in non-blocking mode. There will be an immediate return to the caller as soon as the SNMP agent has been started. Thereafter, all requests from SNMP managers will be serviced in the context of the KwikNet Task.

> Note
>
> In multitasking systems which implement the SNMP agent as an application task, that task MUST execute at a priority below that of the KwikNet Task.

Single Threaded Operation

When used with a single threaded operating system, the SNMP agent operates in the KwikNet domain in the context of the KwikNet Task as described in Chapter 1.2 of the KwikNet TCP/IP Stack User's Guide. Only one SNMP agent is supported.

Your App-Task must properly configure the agent and then call procedure *`tfSnmpdMain()`* to start the SNMP agent, forcing it to operate in non-blocking mode. There will be an immediate return to your App-Task as soon as the SNMP agent has been started. Thereafter, all requests from SNMP managers will be serviced in the context of the KwikNet Task.

Once the SNMP agent is operational, your App-Task must regularly call KwikNet procedure *`kn_yield()`* to let KwikNet and all server tasks, including your agent task, operate.

The SNMP agent will operate until your App-Task calls procedure *`tfSnmpdStop()`* requesting the agent to stop.

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 generated, the SNMP agent must be able to identify all of the trap targets (SNMP managers) to whom the trap messages are to be directed.

Before you start your SNMP agent, your application must specify the agent's operating parameters. Services within the KwikNet SNMP Library are available for this purpose. Refer to Chapter 4 in the Treck SNMP User Manual.

You must define the communities to which the managed device belongs and the MIB access rights granted to each such community.

You must identify the SNMP managers to which all SNMP generated trap messages will be sent. Each trap target entry specifies the IP address of an SNMP manager to whom a trap message can be sent.

Once the SNMP agent has been started, your application can still reconfigure some of the agent's operating parameters. Communities and trap targets can be added, removed or modified.

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.

There are two kinds of SNMP traps generated by the KwikNet SNMP Agent during its normal course of operation. The SNMP agent generates a cold start trap whenever it is started. If the SNMP agent is stopped and restarted by your application, the agent also generates a cold 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 by calling any of the trap generation functions which correspond to the trap definitions in your enterprise MIB. 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.

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	(not generated)
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.

By default, the maximum number of trap targets is defined in header file *TRREGSTR.H* to be 5. 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. Be sure to call Treck function *tfNgAddTrapEntry()* to specify your particular set of trap targets.

Once the KwikNet SNMP Agent has been started, your application can dynamically revise the trap target list maintained by the agent. Trap targets can be added, deleted or modified. 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.

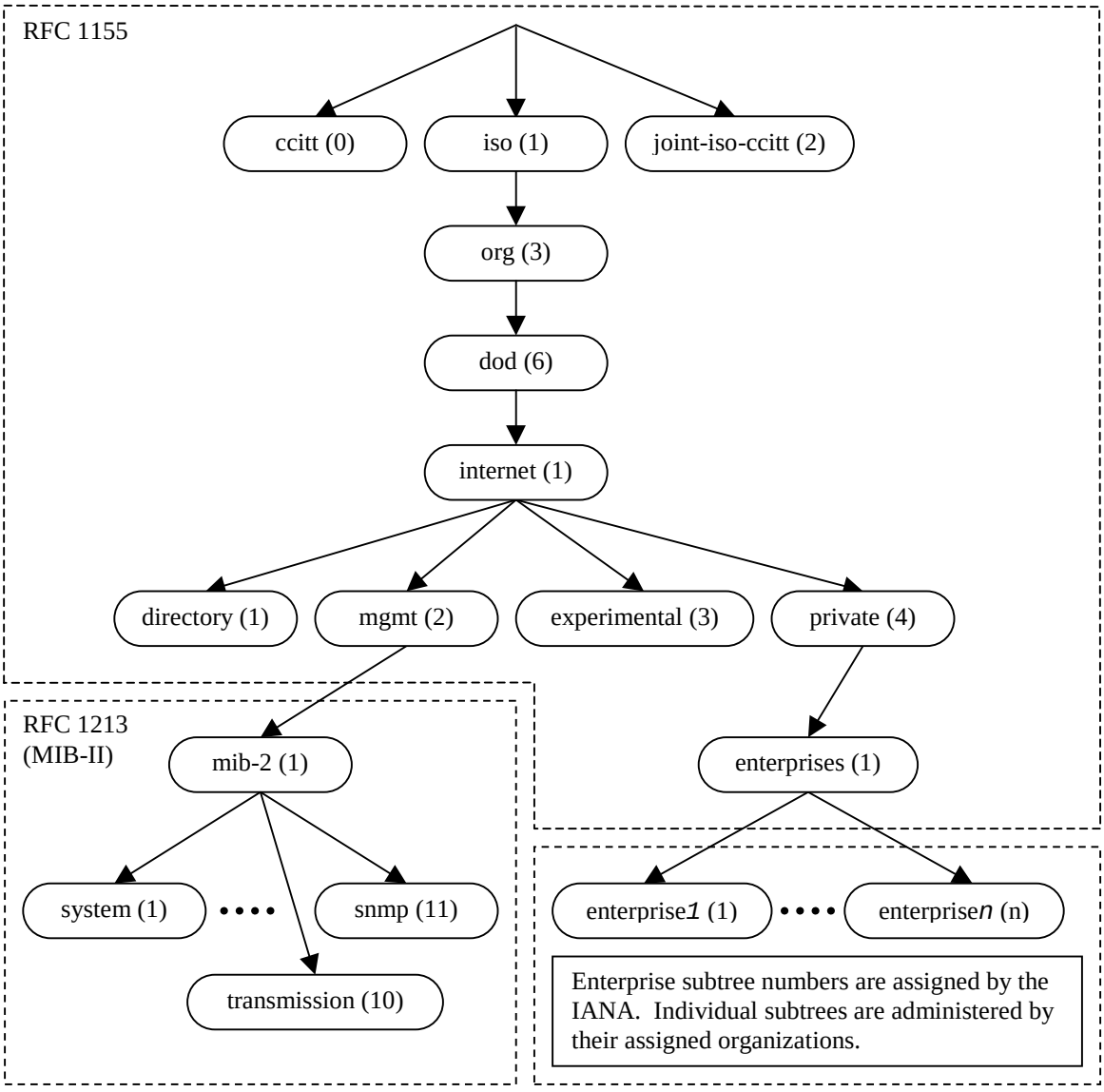


Figure 1.6-1 Standard MIB Tree Structure

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 Library includes built-in support for MIB-II variables.

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 Library 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 not supported.

1.7 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.

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

KwikNet Task Considerations

In a multitasking system, you may implement the SNMP agent to operate in blocking mode as a separate application task. In this case, you may have to increase the total number of tasks allowed by your RTOS in order to add the agent task.

A stack size of 4K to 8K bytes is considered adequate for the agent task when used with most device drivers. The stack size can be trimmed after your agent task has been tested and actual stack usage observed using your debugger.

If you start the KwikNet SNMP Agent such that it executes in non-blocking mode in the context of the KwikNet Task, you may need to increase the stack size for this task. A stack size of 4K to 8K 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 your agent task, if one exists. It must also be of higher priority than any application task which interacts with the KwikNet SNMP Agent using Treck's SNMP services.

Reconstructing Your KwikNet Application

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

To meet the memory demands of your SNMP agent, you may have to edit your KwikNet Network Parameter File to increase the memory available for allocation. If you do so, you must then rebuild your KwikNet Library.

Your application link and/or locate specification files must be updated to add the KwikNet SNMP Library file *KNnnnSNM.A* prior to the KwikNet 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, adapt the procedure just described to include the following considerations.

If you have chosen to implement your SNMP agent as a separate application task, you may have to edit your AMX User Parameter File to increase the maximum number of tasks allowed.

An agent task can be predefined in your AMX User Parameter File or it can be created dynamically at run-time. Such a task is a simple AMX trigger task without message queues.

A stack size of 4K to 8K bytes is considered adequate for use with most device drivers. The stack size can be trimmed after your agent task has been tested and actual stack usage observed using your debugger.

The agent task priority must be lower than that of the KwikNet Task.

If you edit your AMX User Parameter File, you must then rebuild and compile your AMX System Configuration Module.

No changes to your AMX Target Configuration Module are required to support SNMP unless your agent task requires special device support which is not already part of your application.

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, memory availability, sockets, 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 memory available for use by the TCP/IP stack 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.

A custom MIB is created using the Treck 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.

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 Library includes a prebuilt MIB-II implementation.

SNMP Variables Module

The Treck MIB Compiler is used to translate and merge your MIB Definition File (say file *YOURMIB*) into a single SNMP Variables Module named *yourmib_var.c*. This C source file must be compiled and linked with your KwikNet application. It forms the foundation for your entire MIB.

A header file named *yourmib_var.h*. is also generated by the MIB compiler. It must be accessible when compiling file *yourmib_var.c*.

Note that the string *yourmib* is derived from the name of your MIB Definition File, converted to lower case. The MIB Definition File name must have no extension.

MIB Access Module

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 Treck MIB Compiler from information in your MIB Definition File. The finder procedures for your MIB Definition File (say file *YOURMIB*) will be located in a MIB Access Module named *yourmib_local.c*. 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.

A header file named *yourmib_local.h*. is also generated by the MIB compiler. It must be accessible when compiling file *yourmib_local.c*.

MIB Insert Module

When the Treck MIB Compiler compiles your MIB Definition File (say file *YOURMIB*), it generates a MIB Insert Module named *yourmib.ins*. This file contains code fragments which must be inserted into Treck SNMP source files according to the instructions provided in file *yourmib.ins*.

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 *xxxmib* which implements this MIB.
3. Use the Treck MIB Compiler to generate the SNMP Variables Module *xxxmib_var.c* and MIB Access Module *xxxmib_local.c* from file *xxxmib* 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. Insert the code fragments from generated file *xxx.ins* into the SNMP source files according to the instructions provided in file *xxx.ins*.
6. If your managed device is best described using separate MIBs for different subsets of its MIB variables, repeat steps 1 through 5 for each of the separate MIBs.
7. Compile the SNMP Variables Module(s) generated in step 3.
8. Compile the MIB Access Module(s) which you edited in step 4.
9. Generate a new KwikNet SNMP Library which incorporates the SNMP source files edited in step 5.
10. Link the object modules from steps 7 and 8 and the KwikNet SNMP Library from step 9 with your KwikNet application.

2.2 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 2.2-1 provides a visible text display and error counter and has a set of seven toggle switches used to control the device.

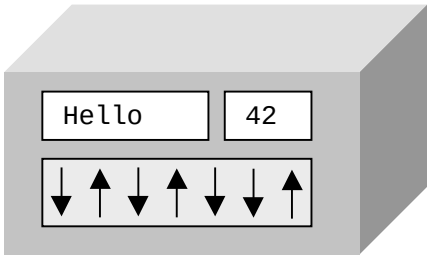


Figure 2.2-1 Sample SNMP Managed Device

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 2.2-2 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 which implements this MIB is described in Chapter 2.3.

The MIB tree shown in Figure 2.2-2 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 (example only).

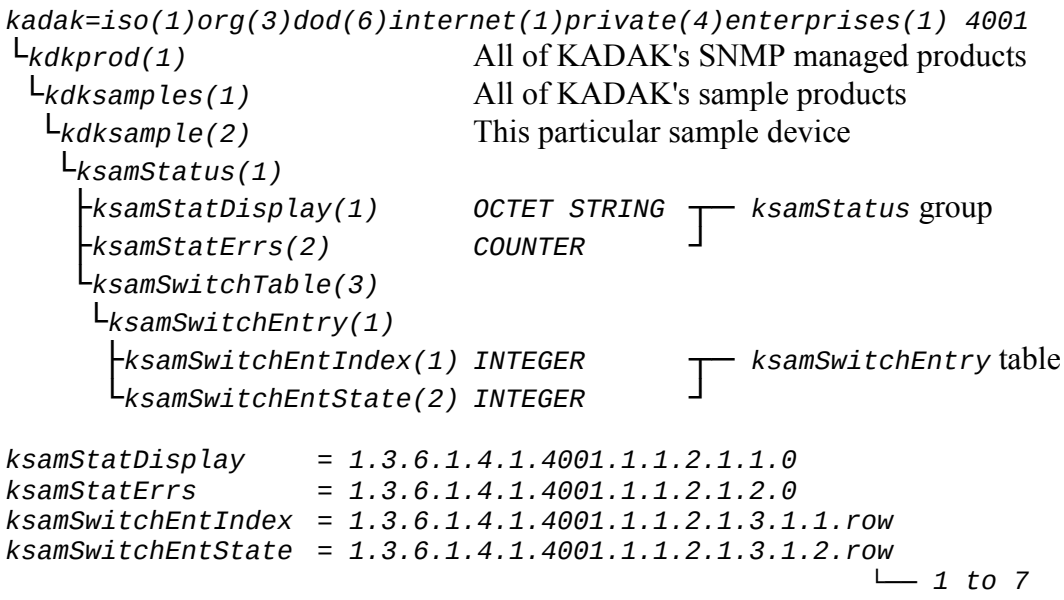


Figure 2.2-2 MIB Structure for Sample Device

2.3 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.

The MIB Definition File for the sample device introduced in Chapter 2.2 is illustrated in Figure 2.3-1. The MIB tree structure of this sample MIB was shown in Figure 2.2-2.

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 2.2-2 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 2.2-2 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 2.3-1 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 2.3-1 MIB Definition File for Sample Device