User's Manual

R-IN32 Series User's Manual

(µNet3/SNMP edition)

- R-IN32M3-EC
- R-IN32M3-CL
- R-IN32M4-CL2

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (http://www.renesas.com)

Document number: R18UZ0063EJ0100
Issue date: Sep 5, 2016

Renesas Electronics
www.renesas.com

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.

   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.
(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

Instructions for the use of product

In this section, the precautions are described for over whole of CMOS device.
Please refer to this manual about individual precaution.
When there is a mention unlike the text of this manual, a mention of the text takes first priority.

1. Handling of Unused Pins
   Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.
   - The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on
   The state of the product is undefined at the moment when power is supplied.
   - The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
     In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
     In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses
   Access to reserved addresses is prohibited.
   - The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals
   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.
   - When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

How to Use This Manual

1. Purpose and Target Readers

This manual is intended for users who wish to understand the functions of an Ethernet communication LSI "R-IN32M4-CL2" for designing application of it. It is assumed that the reader of this manual has general knowledge in the fields of electrical engineering, logic circuits, and microcontrollers.

Particular attention should be paid to the precautionary notes when using the manual. These notes occur within the body of the text, at the end of each section, and in the Usage Notes section.

The revision history summarizes the locations of revisions and additions. It does not list all revisions. Refer to the text of the manual for details.
The mark "<R>" means the updated point in this revision. The mark "<R>" let users search for the updated point in this document.

Literature Literature may be preliminary versions. Note, however, that the following descriptions do not indicate "Preliminary". Some documents on cores were created when they were planned or still under development. So, they may be directed to specific customers. Last four digits of document number (described as ****) indicate version information of each document. Please download the latest document from our web site and refer to it.

The document related to R-IN32 Series

Document Name	Document Number
R-IN32M3 Series Datasheet	R18DS0008EJ****
R-IN32M3-EC User's Manual	R18UZ0003EJ****
R-IN32M3-CL User's Manual	R18UZ0005EJ****
R-IN32M3 Series User's Manual (Peripheral function)	R18UZ0007EJ****
R-IN32M3 Series Programming Manual (OS edition)	R18UZ0011EJ****
R-IN32M3 Series Programming Manual (Driver edition)	R18UZ0009EJ****
R-IN32M4-CL2 User's Manual	R18UZ0032EJ****
R-IN32M4-CL2 User's Manual (Peripheral Modules)	R18UZ0034EJ****
R-IN32M4-CL2 User's Manual (Gigabit Ethernet PHY edition)	R18UZ0044EJ****
R-IN32M4-CL2 Programming Manual (Driver edition)	R18UZ0036EJ****
R-IN32M4-CL2 Programming Manual (OS edition)	R18UZ0040EJ****
R-IN32 Series User's Manual (μNet3/SNMP edition)	R18UZ00xxEJ****

2. Notation of Numbers and Symbols

Weight in data notation: Left is high-order column, right is low-order column

Active low notation:

xxxZ (capital letter Z after pin name or signal name)
or xxx_N (capital letter _N after pin name or signal name)
or xxnx (pin name or signal name contains small letter n)

Note:

Explanation of (Note) in the text

Caution:

Item deserving extra attention

Remark:

Supplementary explanation to the text

Numeric notation:

Binary ··· xxxx , xxxxB or n’bxxxx (n bits)
Decimal ··· xxxx
Hexadecimal ··· xxxxH or n’hxxxx (n bits)

Prefixes representing powers of 2 (address space, memory capacity):

K (kilo)··· $2^{10} = 1024$
M (mega)··· $2^{20} = 1024^{2}$
G (giga)··· $2^{30} = 1024^{3}$

Data Type:

Word ··· 32 bits
Halfword ··· 16 bits
Byte ··· 8 bits

Contents

List of Figures

List of Tables

1. Introduction

The µNet3-SNMP is software which provides an SNMP agent role for the µNet3 TCP/IP protocol stack. This software responds to GetRequest or other packets sent from the manager as shown in the figure below. Also, it is capable of sending notifications such as traps to the manager. Using this software allows monitoring of the state of incorporated devices (agents) which are connected to the Ethernet through an SNMP manager.

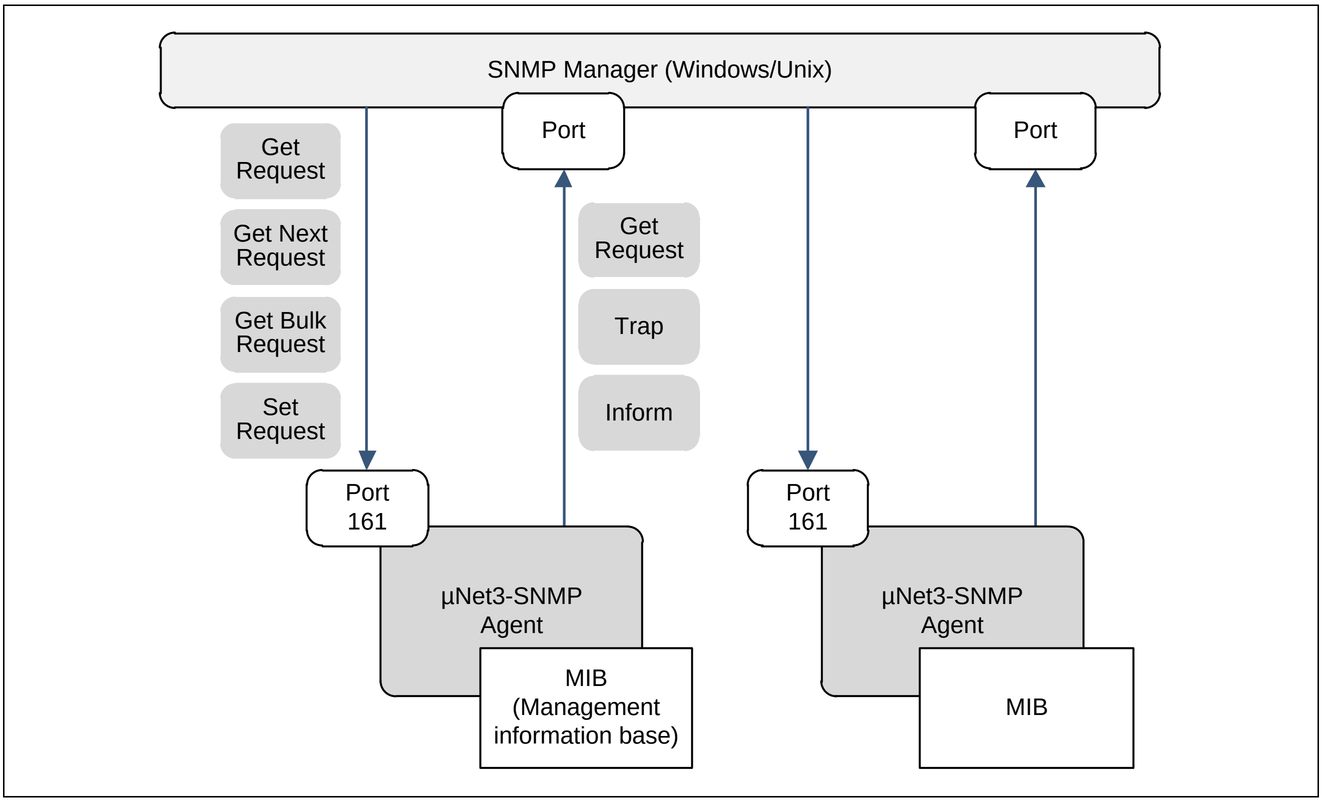


Figure 1.1 SNMP Manager and Agents

This software is for use with the µNet3 (TCP/IP protocol stack) which supports the SNMP protocol.

1.1 Restrictions

The following restrictions apply to this system.

- This system supports part of the MIB-II objects of SNMP but not all of them. For details, see Section 2.2, Supported MIB-II Objects.
- The data types available for vendor-specific MIB objects are integer, counter (32), gauge (32), time ticks, IP address, and octet string (character string). Other types are not supported.
- The vendor-specific private MIB tree cannot be modified while data are processing. In other words, addition or removal of objects (nodes) in the tree while the connected device is running is not allowed.
- This system does not support interfaces with PPP (point-to-point protocol) but with Ethernet.
- This system does not support IPv6.
- This system does not support multiple Ethernet ports. If the agent device has two LAN ports, transmission and reception of SNMP packets are handled only through a single port. For this reason, this system cannot transmit a linkDown trap because there will be no port available. When the system searches for a link to send a linkUp trap, it only detects the port assigned to the transmission and reception of SNMP packets.

  On the other hand, the MIB-II can manage information from up to two ports, which means that the MIB objects in the Interface group and the ipAddrTable (in the IP group) can contain information from two ports.

The names of the companies and products are the trademarks or the registered trademarks of individual companies.

Contents described in this document may be changed without prior notice.

2. Specification Outline

The outline of the specifications of this system is described here.

2.1 Specifications

The specifications of this system are shown below.

Table 2.1 Specifications

Item	Content		Remark
Role in SNMP	Agent		Does not work as a manager.
Supported SNMP versions	SNMPv1 and SNMPv2c		SNMPv3 is not supported.
Supported IP versions	IPv4		
Supported MIBs	MIB-II	System Group	Note
		Interfaces Group	
		Address Translation Group	
		IP Group	
		ICMP Group	
		TCP Group	
		UDP Group	
		EGP Group	Not supported
		Transmission Group	Not supported
		SNMP Group	Note
	The enterprise group in the vendor’s private subtree		The vendor-specific extended MIB

Supported traps	Standard trap		Supports the traps listed below;
			coldStart(0)
			warmStart(1)
			linkUp(3)
			authenticationFailure(4),
			but not linkDown(2).

	Extended trap		enterpriseSpecific(6) (vendor-specific trap)
Others	Function used for enabling and disabling standard traps		Function name: ena_trp/dis_trp
	Function used for issuing extended traps		Function name: snd_trp

Note: Part of the MIB-II objects of each group are supported but not all of them. For details on unsupported objects, see the next section.

2.2 Supported MIB-II Objects

This system supports part of the MIB-II objects but not all of them. Supported objects in each group are listed in the table below. The cells in gray show the objects which do not reflect data immediately. The shortest interval at which those objects are updated is every 100 milliseconds. Details on updating data in objects are described in the next section. Note that inaccessible objects such as tcpConnTable and tcpConnEntry are omitted from the table below.

Table 2.2 Supported MIB-II Objects (1/5)

Group Name	Supported Object	Restrictions	
System group	sysDescr		Objects with IDs greater than that of sysServices are not supported.
	sysObjectID		
	sysUpTime		
	sysContact		
	sysName		
	sysLocation		
	sysServices		
Interfaces group	ifNumber	The value should be 1 or 2.	Up to two network interfaces (for the device number 1 and 2) are supported.
	ifTable		
	ifEntry		
	ifIndex		
	ifDescr		
	ifType		
	ifMtu		
	ifSpeed		
	ifPhysAddress		
	ifAdminStatus	The value should always be 1. Only read access is allowed (not read-write). Link states of the network cannot be changed through the SNMP manager.	
	ifOperStatus		
	ifLastChanges		
	ifInOctets	Whether these are supported or not depends on the implementation of the Ethernet driver in use. The driver for AM335x supports these objects.	
	ifInUcastPkts		
	ifInNUcastPkts		
	ifInDiscards		
	ifInErrors		
	ifInUnknownProtos		
	ifOutOctets		
	ifOutUcastPkts		
	ifOutNUcastPkts		
	ifOutDiscards		
	ifOutErrors		
	ifOutQLen	The value should always be 0.	
	ifSpecific	The value should always be “0.0” (no detail is provided).	

Table 2.2 Supported MIB-II Objects (2/5)

Group Name	Supported Object	Restrictions	
Address translation group	atIfIndex		
	atPhysAddress		
	atNetAddress		
IP group	ipForwarding	The value should always be 2 (notForward-ing). Only read access is allowed (not read-write).	Objects with IDs greater than that of ipRoutingDiscards are not supported.
	ipDefaultTTL	Only read access is allowed (not read-write). The values cannot be modified through the SNMP manager.	
	ipInReceives		
	ipInHdrErrors		
	ipInAddrErrors		
	ipForwDatagrams		
	ipInUnknownProtos		
	ipInDiscards		
	ipInDelivers		
	ipOutRequests		
	ipOutDiscards		
	ipOutNoRoutes		
	ipReasmTimeout		
	ipReasmReqds		
	ipReasmOKs		
	ipReasmFails		
	ipFragOKs		
	ipFragFails		
	ipFragCreates		
	ipAdEntAddr	These objects are generated when the system starts up and will not be deleted even at link down, for example.	
	ipAdEntIfIndex		
	ipAdEntNetMask		
	ipAdEntBcastAddr		
	ipAdEntReasmMaxSize		
	ipNetToMediaIfIndex	Only read access is allowed (not read-write). The values cannot be modified through the SNMP manager.	
	ipNetToMediaPhysAddress		
	ipNetToMediaNetAddress		
	ipNetToMediaType		
	ipRoutingDiscards		

Table 2.2 Supported MIB-II Objects (3/5)

Group Name	Supported Object	Restrictions	
ICMP group	icmpInMsgs		Objects with IDs greater than that of icmpOutAddrMaskReps are not supported.
	icmpInErrors		
	icmpInDestUnreachs		
	icmpInTimeExcds		
	icmpInParmProbs		
	icmpInSrcQuenchs		
	icmpInRedirects		
	icmpInEchos		
	icmpInEchoReps		
	icmpInTimestamps		
	icmpInTimestampReps		
	icmpInAddrMasks		
	icmpInAddrMaskReps		
	icmpOutMsgs		
	icmpOutErrors		
	icmpOutDestUnreachs		
	icmpOutTimeExcds		
	icmpOutParmProbs		
	icmpOutSrcQuenchs		
	icmpOutRedirects		
	icmpOutEchos		
	icmpOutEchoReps		
	icmpOutTimestamps		
	icmpOutTimestampReps		
	icmpOutAddrMasks		
	icmpOutAddrMaskReps		

Table 2.2 Supported MIB-II Objects (4/5)

Group Name	Supported Object	Restrictions	
TCP group	tcpRtoAlgorithm		Objects with IDs greater than that of tcpOutRsts are not supported.
	tcpRtoMin		
	tcpRtoMax		
	tcpMaxConn		
	tcpActiveOpens		
	tcpPassiveOpens		
	tcpAttemptFails		
	tcpEstabResets		
	tcpCurrEstab		
	tcpInSegs		
	tcpOutSegs		
	tcpRetransSegs		
	tcpConnState	Only read access is allowed (not read-write). The values cannot be modified through the SNMP manager. For example, you cannot rewrite tcpConnState with deleteTCB.	
	tcpConnLocalAddress		
	tcpConnLocalPort		
	tcpConnRemAddress		
	tcpConnRemPort		
	tcpInErrs		
	tcpOutRsts		
UDP group	udpInDatagrams		Objects with IDs greater than that of udpLocalPort are not supported.
	udpNoPorts		
	udpInErrors		
	udpOutDatagrams		
	udpLocalAddress		
	udpLocalPort		

Table 2.2 Supported MIB-II Objects (5/5)

Group Name	Supported Object	Restrictions	
SNMP group	snmpInPkts		Objects with IDs greater than that of snmpEnableAuthenTraps are not supported.
	snmpOutPkts		
	snmpInBadVersions		
	snmpInBadCommunityNames		
	snmpInBadCommunityUses		
	snmpInASNParseErrs		
	snmpInTooBigs		
	snmpInNoSuchNames		
	snmpInBadValues		
	snmpInReadOnlys		
	snmpInGenErrs		
	snmpInTotalReqVars		
	snmpInTotalSetVars		
	snmpInGetRequests		
	snmpInGetNexts		
	snmpInSetRequests		
	snmpInGetResponses		
	snmpInTraps		
	snmpOutTooBigs		
	snmpOutNoSuchNames		
	snmpOutBadValues		
	snmpOutGenErrs		
	snmpOutGetRequests		
	snmpOutGetNexts		
	snmpOutSetRequests		
	snmpOutGetResponses		
	snmpOutTraps		
	snmpEnableAuthenTraps	Only read access is allowed (not read-write). The values cannot be modified through the SNMP manager.	

2.3 Updating Data in MIB Objects

The timing of updating MIB objects is described here. The MIB objects are held in the TCP/IP protocol stack, for which the shortest interval of updating the objects is every 100 milliseconds, as described in the previous section, and in the SNMP module, for which the objects are updated immediately (Figure 2.1).

As shown in the figure, the objects related to the link state of the interface groups are, as exceptions, updated immediately by a callback. On reception of a callback which signifies the detection of linkage, the SNMP module returns a linkUp trap (if this is enabled) in response.

On the other hand, the data in the TCP/IP protocol stack are updated at interval in order to prevent the transmission rate from decreasing. Updating of the objects, for which the shortest interval is every 100 milliseconds, is based on the timer task in the protocol stack, which runs every 100 milliseconds.

Users can set a longer interval by setting a desired value in millisecond (in multiples of 100) in the macro CFG_STS_UPD_RES for configuring the protocol stack. The default value of this sample program (net_cfg.c) is two seconds as explained below;

E.g., if you want an interval of 100 milliseconds, define CFG_STS_UPD_RES with the value 100.

– Network/inc/net_cfg.h – (the header file for setting default values for the protocol stack)

```
#define DEF_STS_UPD_RES        2000       /* default value (2 seconds) */
```

– Sample/***.SNMP/net_cfg.c – (the configuration file for the protocol stack)

```
#define CFG_STS_UPD_RES        DEF_STS_UPD_RES       /* 2 seconds */
```

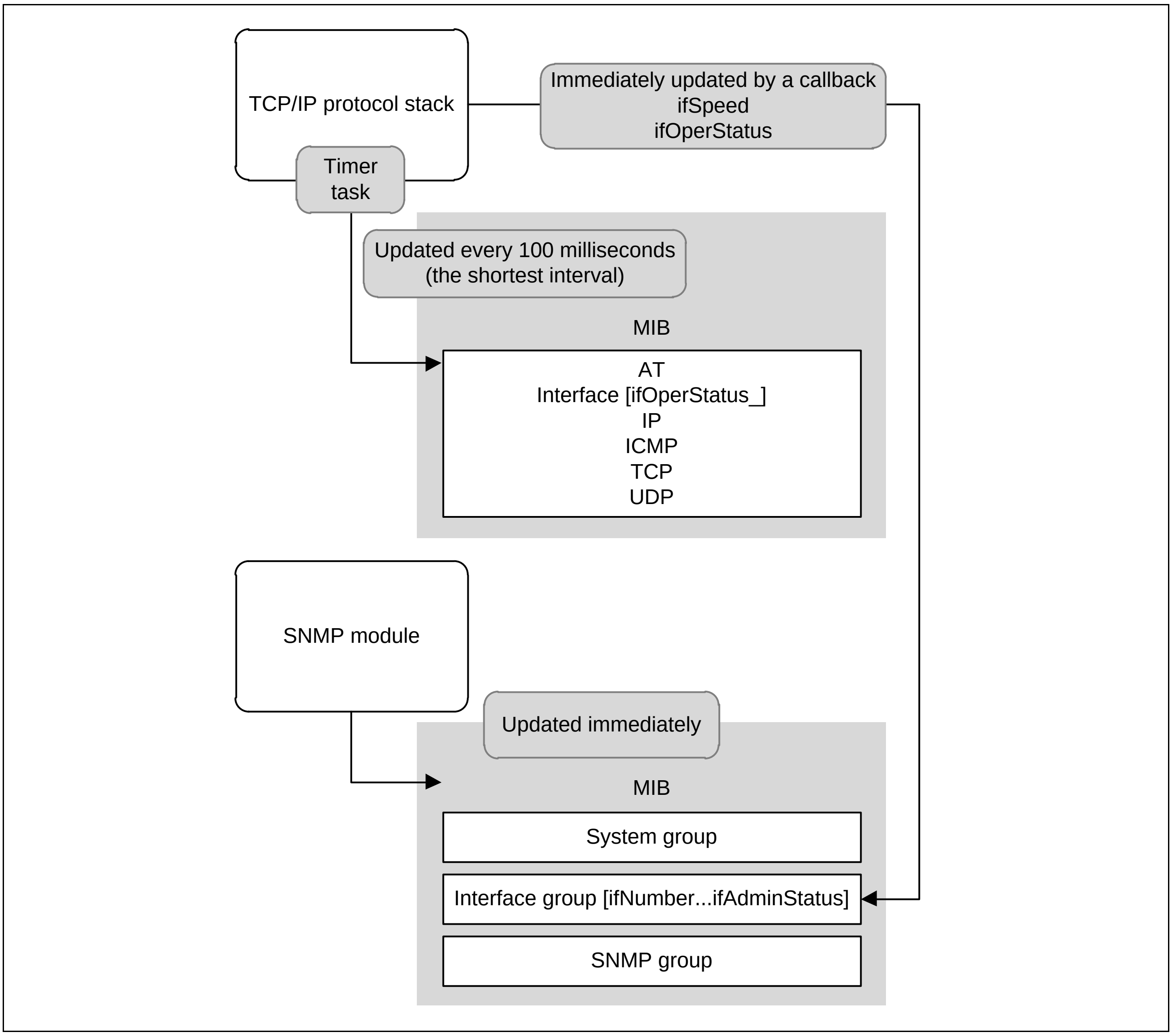


Figure 2.1 Updating Data in the MIB-II Objects

2.4 Generating MIB Trees

Users are generally required to generate an MIB tree to implement SNMP in their systems. In this system, a tree is generated in memory (RAM) by using a two-way list when the system is initialized. The tree needs the OIDs of individual objects (such as “1.3.6.1.2.1.1.1”), which take the form of numerals separated by “.” in this system. For example, a vendor-specific private MIB object with the OID “1.3.6.1.4.1.1234.1.1” is configured as follows (highlighted in gray).

```
/* The prefix for the MIB OID (add a period at the end) */
const VB snmp_mib_ven_pre_1[] = "1.3.6.1.4.1.1234.";     /* Prefix OID */

/* The MIB OIDs following the prefix (add a period at the end) */
const VB snmp_mib_1234_1_1[] = "1.1.0";     /* Descr      (1.3.6.1.4.1.1234.1.1) */
const VB snmp_mib_1234_1_2[] = "1.2.0";     /* Version    (1.3.6.1.4.1.1234.1.2) */
const VB snmp_mib_1234_1_3[] = "1.3.0";     /* Status     (1.3.6.1.4.1.1234.1.3) */
const VB snmp_mib_1234_1_4[] = "1.4.0";     /* User name  (1.3.6.1.4.1.1234.1.4) */
```

As shown above, in the configuration of vendor-specific private MIBs, users need to include the information in the form of the OID strings, data types of individual objects, and their initial values in the C-language source file.

This system reads the strings in the source file and form a tree of MIB on memory as shown in Figure 2.2. The MIB-II tree includes objects associated with TCP or UDP sockets. These objects (nodes in the tree) should be generated and deleted as users generate and delete sockets in their applications. This means that the generated MIB-II tree is modified while data are being processed. However, it is not possible to add objects (nodes) in a vendor-specific private MIB.

Furthermore, the number of nodes to be used in the tree is only figured out after it has been generated, which means that the user cannot tell how much memory will be used in advance. Therefore, the function for initializing this system (snmp_ini) is configured to return the number of nodes (amount of memory) needed to generate a tree for debugging.

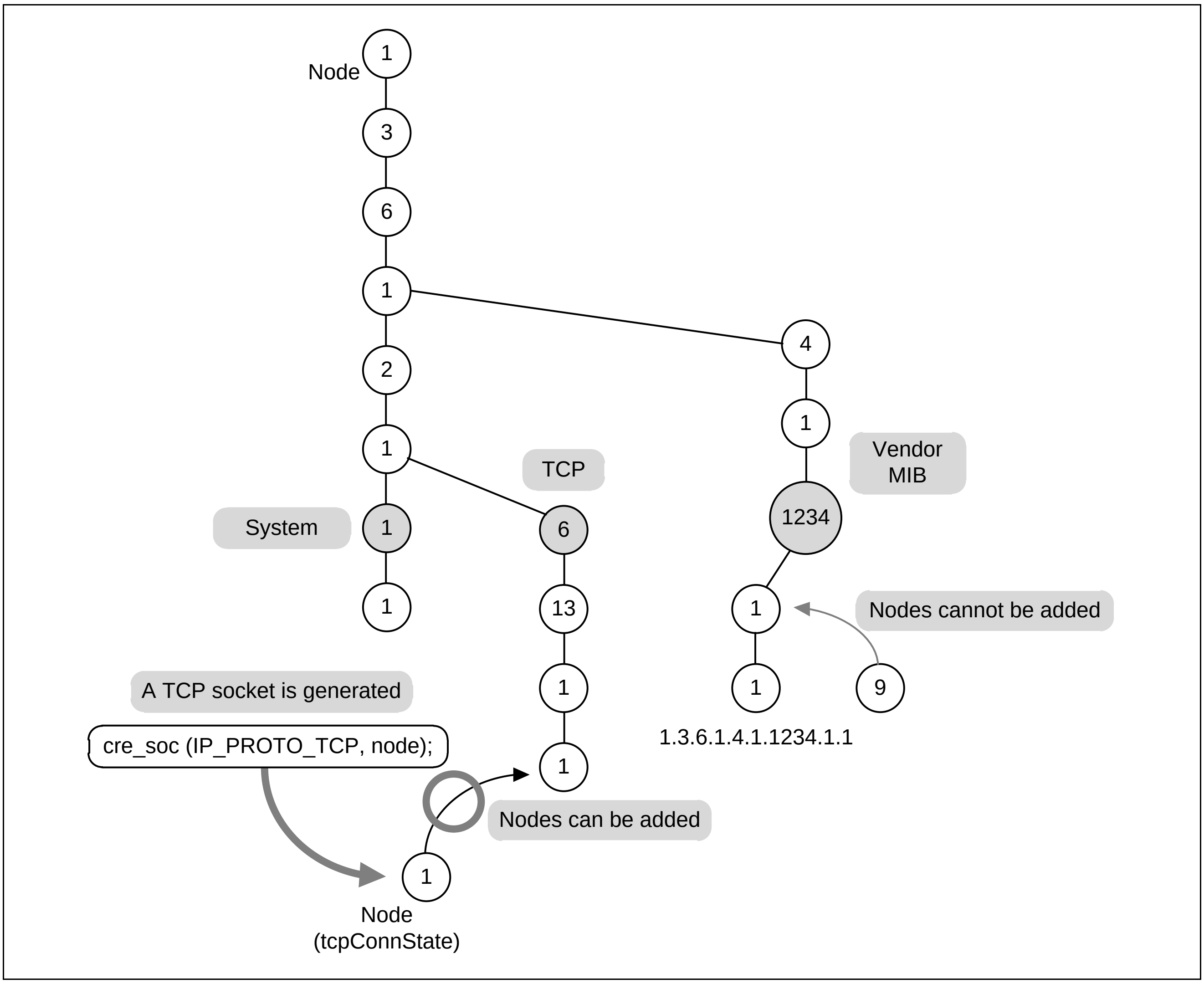


Figure 2.2 MIB Tree

2.5 Vendor-Specific MIB and Callback Function

Values for the objects in a vendor-specific private MIB are changed in two ways. One way is to obtain the target value by calling the get_mib_obj function and change it by calling the set_mib_obj function from a user task (process 1 in Figure 2.3). The other way is to change the value in the callback function issued by the receiving task in the system, in response to packets such as GetRequest from the manager (process 2 in Figure 2.3). With the latter approach, when this system receives GetRequest, for example, from the manager, the user can change the argument of the callback function to be returned to the desired value and exit the function. Then, the system returns the callback with the new value to the manager.

In summary, if you want to change the value in an object of a vendor-specific private MIB at any time, use the functions get_mib_obj and set_mib_obj, and if you want to change the value on reception of a request from the manager, use a callback function.

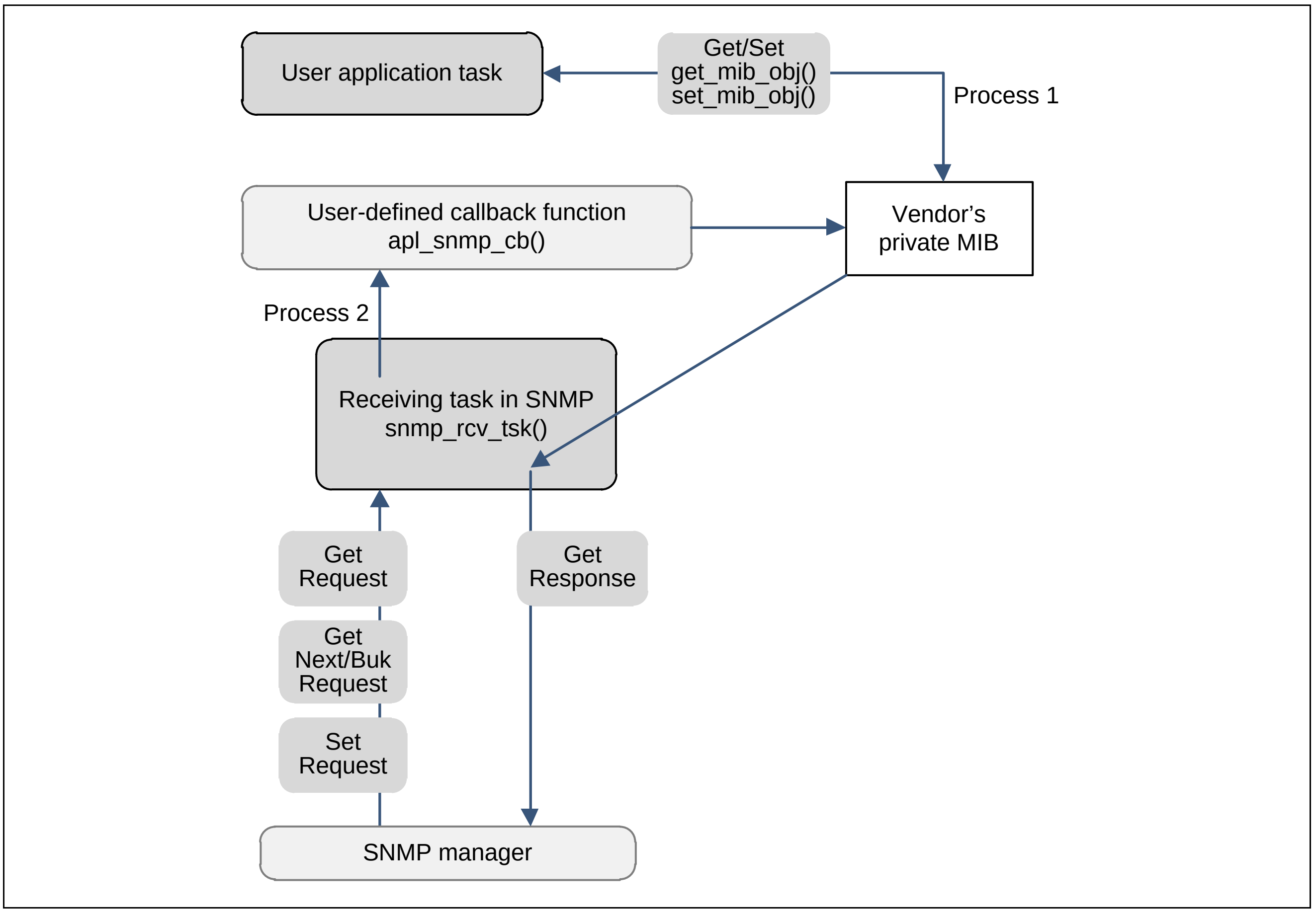


Figure 2.3 Vendor-Specific MIB and Callback Functions

3. Outline of the Structure

3.1 File Structure

The installer of this system copies the files to the μC3/Compact/SNMP folder (for the compact version of the operating system) or the μC3Std/SNMP folder (for the standard version of the operating system). The file structure of the system is shown below.

Table 3.1 File Structure

Folder	File Name	Description
SNMP/doc	uNet3_SNMP.txt	Update history
	uNet3_SNMAUsersGuide.pdf	User's guide
SNMP/inc (header file)	snmp.h	User-defined function
	snmp_ber.h	BER (basic encoding rules) of ASN.1 (Abstract Syntax Notation One)
	snmp_def.h	Internal definition
	snmp_lib.h	For creating libraries
	snmp_mac.h	Macro for configuration
	snmp_mib.h	Macro for defining MIB-II IDs
	snmp_net.h	Fixed values for MIB-II
SNMP/src (source file)	snmp.c	User-defined functions and functions for tasks
	snmp_ber.c	Encoding and decoding in BER
	snmp_mib.c	For processing the MIB tree
	snmp_mib_dat.c	Data in MIB-II
	snmp_tcp.c	For TCP/IP protocol stack
SNMP/lib/ (library)	SNMP[processor name, etc.].*	Libraries (excluding snmp_mib_dat.c)
	[processor name]/ SNMP[processor name, etc.]. [extension of the project]	Project file for building libraries

An application which uses the API functions of this system requires snmp.h among its files of source code. The other header files are for use in the system or in the user's configuration files (snmp_cfg.c, snmp_mib_cfg.c).

The source files composing the library of this system do not include snmp_mib_dat.c because this file contains variable data (sysDescr of MIB-II for defining the name and version identifier of the device, for example). Therefore, the user will need to create this file.

To use this system, settings are required by using the files listed in the table below. These files are included in the folders for the sample program. For example, the configuration files are included in the Sample/EVMAM3358.SNMP folder for the Cortex-A8 (AM335x).

Table 3.2 Configuration Files

File Name	Content
snmp_cfg.h	Macros used for configuring the SNMP
snmp_cfg.c	Variables for configuring the SNMP
snmp_mib_cfg.h	Macros used for configuring the vendor's private MIB
snmp_mib_cfg.c	Variables for configuring the vendor's private MIB

3.2 Libraries

The libraries of this system are built by using the same compiler options as for the operating systems and the TCP/IP protocol stack. For example, this system, when used with a Cortex-A8 (AM335x), includes four libraries representing the four combinations of the ARM and Thumb states and whether VFP is or is not present. Each file name starts with “SNMP”, followed by the same strings as those of the operating systems or the protocol stack.

[Code Composer Studio]

ARM/Thumb	Endian	VFP	Library Name
ARM	Little	—	SNMPcortexal.lib
Thumb	Little	—	SNMPcortexatl.lib
ARM	Little	VFPv3	SNMPcortexafl.lib
Thumb	Little	VFPv3	SNMPcortexaftl.lib

The libraries are built into this system without including debugging information. The libraries having already been built in this way means that this system cannot be traced by a debugger. If you want to trace the source code of this system, rebuild the library with debugging information.

3.3 Module Structure Overview

This section describes major structures of the modules used in this system. Three tasks are provided in this system and implemented in the function snmp.c as shown below.

Table 3.3 Tasks

Number	Function for Each Task	Description
1	snmp_rcv_tsk	Receive SNMP packets and send responses
2	snmp_tim_tsk	Count running time
3	snmp_trp_tsk	Send traps and Inform packets

This system also requires memory area where the MIB (MIB-II and vendor-specific private MIB) information are to be stored. The TCP/IP protocol stack also sums up data for the MIB-II and updates this area. Behavior of each task is described from the next section.

3.3.1 Task for Receiving SNMP Packets and Sending Responses

This task receives SNMP packets and sends response packets. A task waits for incoming SNMP packets at port 161 as UDP by issuing the rcv_soc command of the μNet3 stack, and, on receiving data, it returns a response to the manager from the same port by issuing the snd_soc command of μNet3.

For example, when this task receives a packet such as GetRequest, it refers to the data in the relevant MIB and generates a response packet based on the SNMP specifications. These packets are generated by encoding or decoding the data based on the BER (basic encoding rules) of ASN.1. An authenticationFailure trap may be returned (from the task for sending traps) if the community string of the received packet does not match that set by the user.

When this task receives a packet such as SetRequest, it updates the value in the relevant MIB object.

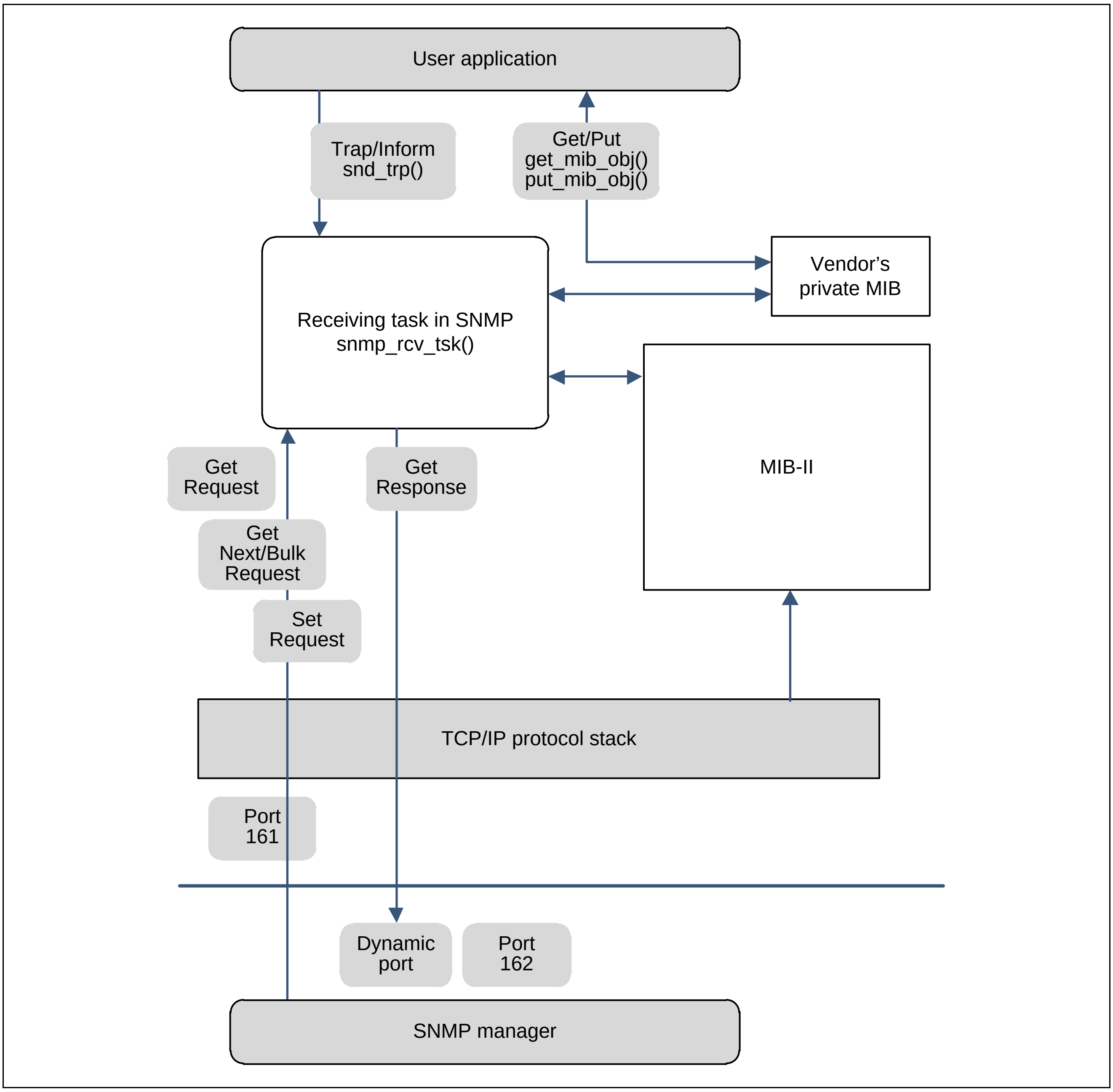


Figure 3.1 Task for Receiving and Responding Packets between MIBs

3.3.2 Task for Counting Running Time

This task obtains a value which indicates how much time has passed since power was supplied to the device. The obtained value is stored in the object “sysUpTime” in the system group. The value is obtained by using the service call API, “get_tim” (get system time). After waking up within an interval such that the value of the thirty-two lower-order bits of the returned system time value does not overflow, this task converts the returned value into the running time of the SNMP and stores it in the relevant MIB. As a whole, the execution time of this task is very short.

Do not use the set_tim command to change the system time while this task is running.

3.3.3 Task for Sending Traps

This task sends traps and Inform packets. There are two types of traps; standard traps and vendor-specific traps. The user can enable and disable standard traps by using the configuration macros or the API functions.

When a vendor-specific traps is to be sent, the user’s application task issues a call of the “snd_trp” function. Once the task sends a message to the destination, it waits until completion of the transmission of the trap, which is the time when µNet3 has finished sending the trap by snd_soc (UDP transmission).

When this task sends an Inform packet, it waits until receiving a response packet (rcv_soc) from the destination. This task is not needed for operations which do not use traps and Inform packets.

4. OS_Resources

This section describes the resources for the operating systems.

4.1 List of OS Resources

The operating system resources used in this system are listed in the table below. Among the resources, task 3, semaphore 2, event flag 2, and mailbox (ID_xxx_TRP) (those highlighted in gray in the table) are not needed for operations which do not use traps.

The table of resources is automatically generated in the standard version of the operating system. For the compact version, users are required to configure the resources listed in Table 4.1 by the configurator provided with the operating system.

Table 4.1 List of OS Resources (1/2)

No.	Resource	Settings (Default or Sample Value)		Content
1	Task 1	Defined ID name	ID_SNMP_TSK_RCV	Receiving SNMP packets and sending the response packets
		Function name of the task	snmp_rcv_tsk	
		Initial value for the priority level	6	
		Extended information	None	
		Executable state	None	
		Restrictions on the task	None	
		Stack size	768 (local stack)	
2	Task 2	Defined ID name	ID_SNMP_TSK_TIM	Counting running time
		Function name of the task	snmp_tim_tsk	
		Initial value for the priority level	6	
		Extended information	None	
		Executable state	None	
		Restrictions on the task	None	
		Stack size	512 (local stack)	
3	Task 3	Defined ID name	ID_SNMP_TSK_TRP	Sending traps and Inform packets
		Function name of the task	snmp_trp_tsk	
		Initial value for the priority level	6	
		Extended information	None	
		Executable state	None	
		Restrictions on the task	None	
		Stack size	512 (local stack)	
4	Semaphore 1	Defined ID name	ID_SNMPA_SEM_MIB	Excluding other MIBs
		Initial value for the number of resources	1	
		Maximum number of the resources 1	1	
		Attribute	TA_TFIFO	
5	Semaphore 2	Defined ID name	ID_SNMPA_SEM_TRP	Excluding other trap resources
		Initial value for the number of resources	1	
		Maximum number of the resources 1	1	
		Attribute	TA_TFIFO	

Table 4.1 List of OS Resources (2/2)

No.	Resource	Settings (Default or Sample Value)		Content
6	Event flag 1	Defined ID name	ID_SNMP_FLG_STS	State of the task
		Initial value	0x0	
		Queueing of tasks to be executed	TA_TFIFO	
		To permit multiple tasks to wait	TA_WSGL	
		Clear the flag	None	
7	Event flag 2	Defined ID name	ID_SNMP_FLG_TRP	State of handling a trap
		Initial value	0x0	
		Queueing of tasks to be executed	TA_TFIFO	
		To permit multiple tasks to wait	TA_WSGL	
		Clear the flag	None	
8	Mailbox	Defined ID name	ID_SNMP_MBX_TRP	Sending a command block of a trap
		Queueing of tasks to be executed	TA_TFIFO	
		Message queueing	TA_MFIFO	
9	UDP socket	Defined ID name	ID_SNMP_UDP_SOC	UDP socket
		Interface binding	Ethernet0 (optional)	
		IP version number	IPv4	
		Protocol	UDP	
		Local port	161	
		Timeout value for snd_soc	2000	
		Timeout value for rcv_soc	2000	

4.2 Configuring OS Resources

Specific variables are required for configuring the resources for the operating system. In this system, the variables have been already implemented in the sample program file snmp_cfg.c.

Configuration of the resources for the compact version of the operating system is as follows:

Declare the variable for the structure T_SNMP_CFG_OS as “snmp_cfg_os”. Assign the ID of each resource which is set by the configurator to the corresponding variables and initialize each of them. This system uses the values read from these variables (assigned to the ROM area).

```
const T_SNMP_CFG_OS snmp_cfg_os = {
      ID_SNMP_TSK_RCV,        /* task 1 */
      ID_SNMP_TSK_TIM,        /* task 2 */
      ID_SNMP_TSK_TRP,        /* task 3 */
      ID_SNMP_SEM_MIB,        /* semaphore 1 */
      ID_SNMP_SEM_TRP,        /* semaphore 2 */
      ID_SNMP_FLG_STS,        /* event flag 1 */
      ID_SNMP_FLG_TRP,        /* event flag 2 */
      ID_SNMP_MBX_TRP,        /* mailbox */
      ID_SNMP_UDP_SOC         /* UDP socket */
};
```

Configuration of the resources for the standard version of the operating system (such as the Cortex-A8 (AM335x)) is as follows:

The variables representing the information for generating resources are implemented in the snmp_cfg.c file as shown below. The resources for the operating system is automatically generated using these variables.

```
const T_CTSK snmp_cfg_os_tsk_rcv = {TA_HLNG, 0, (FP)snmp_rcv_tsk,   TSK_RCV_PRI, TSK_RCV_STK, 0, 0};
const T_CTSK snmp_cfg_os_tsk_tim = {TA_HLNG, 0, (FP)snmp_tim_tsk,   TSK_TIM_PRI, TSK_TIM_STK, 0, 0};
const T_CTSK snmp_cfg_os_tsk_trp = {TA_HLNG, 0, (FP)snmp_trp_tsk,   TSK_TRP_PRI, TSK_TRP_STK, 0, 0};
const T_CSEM snmp_cfg_os_sem_mib = {TA_TFIFO, 1, 1, 0};
(Omitted)
```

The priority level and the stack size of individual tasks are configured in the configurator which is provided with the compact version of the operating system, or use the macro in the snmp_cfg.f for the standard version of the operating system.

5. Configuring the SNMP

Implement the configuration macros for this system in the files snmp_cfg.h and snmp_cfg.c. Implement the identifier macro described in Section 5.1, Basic Settings in the snmp_cfg.h file and the configuration variables described from the subsequent sections in the snmp_cfg.c file.

5.1 Basic Settings

The macros defined in the file snmp_cfg.h are shown in the table below. The macros for “OS and network” in this table are not used for the compact version of the operating system. Instead, the user needs to set the priority level and the stack size for individual tasks.

Table 5.1 List of Configuration Macros (1/2)

Category	Macro Definition	Example Value	Description
SNMP	CFG_SNMP_NET_DEV_CNT	1	The number of network devices to be used, in other words, the number of LAN ports to be used. Always set 1.
	CFG_SNMP_NET_DEV_NUM	1	The number given to each network device to be used. Set 1 or 2.
	CFG_SNMP_MAX_SOC_CNT	DEF_NET_SOC_MAX	The maximum number of the TCP and UDP sockets to be generated in the µNet3, which is set by net_cfg.c (the default value of the µNet3 is DEF_NET_SOC_MAX).
	CFG_SNMP_MAX_TCP_CNT	DEF_NET_TCP_MAX	The maximum number of the TCP sockets to be generated in the µNet3, which is set by net_cfg.c (the default value of the µNet3 is DEF_NET_TCP_MAX).
	CFG_SNMP_MAX_ARP_CNT	DEF_NET_ARP_MAX	The number of entries in the ARP table to be used in the µNet3, which is set by net_cfg.c (the default value in the µNet3 is DEF_NET_ARP_MAX).
	CFG_SNMP_MAX_TRP_CNT	12	The maximum number of the traps and Inform packets (including the standard traps from this system) which are transmitted at the same time. Set a value between 0 and 32. Set 0 if no traps are to be used.
	CFG_SNMP_MSG_VAR_CNT	32	The maximum number of the variable-bindings to be added to the SNMP packet. Set an integer value greater than or equal to 4.
	CFG_SNMP_MIB_NOD_CNT	800	The maximum number of the nodes in the MIB tree.
	CFG_SNMP_MAX_MIB_DEP	32	The maximum depth of nodes in the MIB tree, in other words, the maximum number of the dotted strings of the object ID.
	CFG_SNMP_MIB_DAT_LEN	(64 + 1)	The maximum amount of data allowed in a MIB object in bytes, including the terminating null character.
	CFG_SNMP_GEN_TRP_ENA	TRP_ALL_BIT	Enables and disables the standard traps when the system is initialized.

Table 5.1 List of Configuration Macros (2/2)

Category	Macro Definition	Example Value	Description	
MIB-II	CFG_SNMP_MIB2_IF_ENA	1	Enables (1) and disables (0) the Interfaces group.	
	CFG_SNMP_MIB2_AT_ENA	1	Enables (1) and disables (0) the Address Translation group.	
	CFG_SNMP_MIB2_IP_ENA	1	Enables (1) and disables (0) the IP group.	
	CFG_SNMP_MIB2_ICMP_ENA	1	Enables (1) and disables (0) the ICMP group.	
	CFG_SNMP_MIB2_TCP_ENA	1	Enables (1) and disables (0) the TCP group.	
	CFG_SNMP_MIB2_UDP_ENA	1	Enables (1) and disables (0) the UDP group.	
	CFG_SNMP_MIB2_SNMP_ENA	1	Enables (1) and disables (0) the SNMP group.	
OS and network	TSK_RCV_PRI	6	Priority level	For task 1 (receiving SNMP packets)
	TSK_TIM_PRI	6		For task 2 (counting running times)
	TSK_TRP_PRI	6		For task 3 (sending traps)
	TSK_RCV_STK	1024	Stack sizes in bytes	For task 1 (receiving SNMP packets)
	TSK_TIM_STK	512		For task 2 (counting running times)
	TSK_TRP_STK	768		For task 3 (sending traps)
	CFG_SNMP_RCV_MSG_LEN	2048	The maximum size of receiving SNMP message in bytes	
	CFG_SNMP_SND_MSG_LEN	CFG_SNMP_RCV_MSG_LEN	The maximum size of sending SNMP message in bytes	

5.1.1 Configuring the SNMP

CFG_SNMP_NET_DEV_CNT

This macro is used to specify the number of network devices to be used, in other words, the number of LAN ports. Always set 1.

CFG_SNMP_NET_DEV_NUM

This macro is used to specify the number given to the network device to be used. This number should be same as the number (from 1) defined in the μNet3 TCP/IP protocol stack, which corresponds to the number of the LAN port to be used. For example, the value for this macro is 1 if a single LAN port is to be used. The function gNET_ADR in the net_cfg.c file is used for the configuration of LAN ports. The value obtained by adding 1 to the index (from 0) of the array variable gNET_ADR is the device number.

CFG_SNMP_MAX_SOC_CNT

This macro is used to specify the value representing the maximum number of the TCP and UDP sockets to be generated in the μNet3.

CFG_SNMP_MAX_TCP_CNT

This macro is used to specify the maximum number of the TCP sockets to be generated in the μNet3.

CFG_SNMP_MAX_ARP_CNT

This macro is used to specify the number of entries in the ARP table to be used in the μNet3.

The values for these three macros above should be same as each of those defined in the configuration file of the μNet3 (net_cfg.c), in other words, the values in CFG_NET_SOC_MAX, CFG_NET_TCP_MAX, and CFG_NET_ARP_MAX.

CFG_SNMP_MAX_TRP_CNT

This macro is used to specify the maximum number of the traps which are transmitted at the same time. Set 0 if traps are not to be used. In this case, the task for sending traps will be restrained from waking up. The maximum number of the resources for traps, which are used internally, is calculated as follows.

x = the number of destinations for sending the traps specified in snmp_cfg_trp
y = the number of managers specified by snmp_cfg_mgr or currently connecting to the device
z = the number of traps sent from multiple tasks at the same time by calling the function snd_trp

CFG_SNMP_MAX_TRP_CNT = (x * 2) + y + z

If Inform packets are also to be sent, the value is calculated as follows.

CFG_SNMP_MAX_TRP_CNT = (x * 2) + y + (z * 3)

The resources required for x in issuing the standard trap (1) cold/warmStart and (2) linkUp are doubled because these traps may be used at the same time. The amount of resources required for issuing the authenticationFailure trap is expressed by y. The amount of resources required at the time of issuing a snd_trp call is represented by z. When an Inform packet is issued, the cancellation and the response to the notification make the required resources three times the value expressed by z.

CFG_SNMP_MSG_VAR_CNT

This macro is used to specify the value representing the maximum number of variable-bindings contained in a SNMP packet to be received or transmitted. Set an integer value greater than or equal to 4.

This system returns the error code “tooBig” to the destination if the received v1 packets contain more variable-bindings than specified in this macro.

CFG_SNMP_MIB_NOD_CNT

This macro is used to specify the number of nodes in the MIB tree, which is only figured out after it is generated by the snmp_ini function. Accordingly, check if the snmp_ini function is terminating normally by setting a value greater than the expected one in this macro. If the value set in this macro is smaller than it should be, the snmp_ini function returns the error code E_NOMEM. After successful execution of the snmp_ini function, the value indicating the number of nodes for the newly generated MIB tree is stored in the buffer pointed to by the argument of the function. Users can set this value (that is the number of nodes) in this macro.

CFG_SNMP_MAX_MIB_DEP

This macro is used to specify the maximum depth of the tree, consisting of the MIB-II and the vendor-specific MIB. For example, if the OID of the vendor-specific MIB tree is set as “1.3.6.1.4.1.1234.1.2.3.4.5.6.7”, which is composed of fourteen strings, the value to be set in this macro definition is 14. In other words, set the dotted strings of the OID in this macro.

CFG_SNMP_MIB_DAT_LEN

This macro is used to specify the maximum amount of data allowed in a MIB object in bytes. The SNMP specification allows four-byte integer data or 65,535 characters (bytes) of octet string data. The memory used for these values can be reduced, for example, by limiting the number of strings to 64 in sysDescr (defining names of the hardware and software) in the system group of the MIB. In summary, this macro is especially designed to specify the maximum string size. Note that the size includes a terminating null character. For example, set 65 to this macro for the data with the maximum number of characters as 64.

On the other hand, CFG_SNMP_MIB_SYS_DESCR_LEN in the snmp_mib_cfg.h file is used to specify the maximum size of certain objects such as the maximum number of strings for sysDescr in the system group of the MIB. This means that the value for this macro should be same or greater than the maximum amount of data specified in each object. The error code E_BOVR is returned from the function snmp_ini if the value for this macro is smaller than the maximum values specified in each macro.

CFG_SNMP_GEN_TRP_ENA

This macro is used to specify the standard traps to be enabled when the system is initialized. Example of implementation is given below. Specify 0x00 for disabling all traps.

```
/* enabling all traps */
#define CFG_SNMP_GEN_TRP_ENA      TRP_ALL_BIT

/* enabling coldStart and linkUp */
#define CFG_SNMP_GEN_TRP_ENA      (COLD_STA_BIT | LINK_UP_BIT)
```

The macros used for enabling each trap are shown below.

Table 5.2 Macros for Setting Standard Traps

Number	Macro	Trap Name	Remark
1	COLD_STA_BIT	coldStart	
2	WARM_STA_BIT	warmStart	
3	LINK_DOWN_BIT	linkDown	Not supported
4	LINK_UP_BIT	linkUp	
5	AUTH_FAIL_BIT	authenticationFailure	
6	EPG_LOSS_BIT	egpNeighborLoss	Not supported
7	TRP_ALL_BIT	All traps	

This system sends a coldStart trap when the function snmp_ena (enabling this system) is issued the first time and sends a warmStart trap when the function is issued the second and subsequent times.

5.1.2 Configuring the MIB-II

CFG_SNMP_MIB2_***_ENA is used to enable and disable individual groups in the MIB-II. For example, setting CFG_SNMP_MIB2_SNMP_ENA to 0 disables the SNMP group. Although it reduces memory usage in the system by eliminating the part occupied by the given group, when the manager requests a value from the MIB, this system returns an error indicating that the relevant MIB entry does not exist.

5.1.3 Configuring the Operating System

TSK_***_PRI specifies priority levels of individual tasks in this system. The default value for the number of tasks in the TCP/IP protocol stack is four, which is specified in the macro DEF_NET_TSK_PRI in net_cfg.h, and the priority level for the tasks in this system should be set to a value lower than that (six).

TSK_***_STK specifies the stack size for the tasks in this system in bytes. For example, TSK_RCV_STK is used for specifying the stack size of the receiving task (shown in Figure 2.3). Given that the receiving task issues a user-defined callback function, if the callback includes a process that uses a large amount of stack space, the value for this macro should be large enough to cover this. Users are not required to change other values except for the receiving task.

CFG_SNMP_RCV_MSG_LEN and CFG_SNMP_SND_MSG_LEN are used for specifying the maximum size of the SNMP messages to be received or transmitted in bytes. When this system calls the rcv_soc function (reception of UDP packets) to receive an SNMP message, the size of the buffer where the message will be stored will have been set as an argument for the function by using the value in CFG_SNMP_RCV_MSG_LEN. Note that this system always stores the message as a whole. If the reception of a message longer than the size given by this macro is attempted, the message is discarded and there is no response to the manager. In CFG_SNMP_SND_MSG_LEN, specify the size of the buffer where transmission messages are held. Generally, set the same value as the macro for reception.

5.1.4 Examples of Implementation

Examples of implementation for the basic settings are shown below.

```
/* The number of network devices (LAN ports) (always 1) */
#define CFG_SNMP_NET_DEV_CNT      1                /* Number of network devices (1) */
/* The numbers given to individual network devices to be used (1 or 2) */
#define CFG_SNMP_NET_DEV_NUM      1                /* Network devices number (1..2) */
/* The maximum number of network sockets and TCP sockets (same as the value in net_cfg.c) */
#include "net_cfg.h"
#define CFG_SNMP_MAX_SOC_CNT      DEF_NET_SOC_MAX
#define CFG_SNMP_MAX_TCP_CNT      DEF_NET_TCP_MAX
#define CFG_SNMP_MAX_ARP_CNT      DEF_NET_ARP_MAX

/* The maximum number of the traps and Inform packets which are transmitted at the same time (0 means no traps will
   be used) */
#define CFG_SNMP_MAX_TRP_CNT      12            /* Number of traps at any time (0 or 1...32) */
/* The maximum number of variable bindings to be added to the SNMP packet */
#define CFG_SNMP_MSG_VAR_CNT      32            /* Maximum number of variable bindings */
/* The maximum number of nodes in the MIB tree */
#define CFG_SNMP_MIB_NOD_CNT      680           /* Number of nodes in the MIB tree */
/* The maximum depth of the nodes in the MIB tree (the maximum number of the strings of the OID) */
#define CFG_SNMP_MAX_MIB_DEP      32            /* Maximum depth of the MIB tree */
/* The maximum amount of data allowed in a MIB object
   The maximum length of octet string data specified by using DESCR_LEN in the snmp_mib_cfg.c, including the
   terminating null character */
#define CFG_SNMP_MIB_DAT_LEN      (64 + 1)      /* Maximum size of the MIB data */

/* Generic trap enabled */
/* Specify the standard traps to be sent */
/* TRP_ALL_BIT specifies all traps (no transmission of traps when the link is down) */
#define CFG_SNMP_GEN_TRP_ENA      TRP_ALL_BIT

/* MIB2 group selector */
/* Enabling (1) and disabling (0) the groups of MIB2 */
#define CFG_SNMP_MIB2_IF_ENA      1     /* Interfaces      (1.3.6.1.2.1.2) */
#define CFG_SNMP_MIB2_AT_ENA      1     /* Address trans   (1.3.6.1.2.1.3) */
#define CFG_SNMP_MIB2_IP_ENA      1     /* IP              (1.3.6.1.2.1.4) */
#define CFG_SNMP_MIB2_ICMP_ENA    1     /* ICMP            (1.3.6.1.2.1.5) */
#define CFG_SNMP_MIB2_TCP_ENA     1     /* TCP             (1.3.6.1.2.1.6) */
#define CFG_SNMP_MIB2_UDP_ENA     1     /* UDP             (1.3.6.1.2.1.7) */
#define CFG_SNMP_MIB2_SNMP_ENA    1     /* SNMP            (1.3.6.1.2.1.11) */

/* Task priority */
/* Priority levels of the SNMP tasks for the standard version of the operating systems */
/* Priority levels of the SNMP tasks for the compact version of the operating systems are specified by the configurator
   */
#define TSK_RCV_PRI      6       /* Receive task   */
#define TSK_TIM_PRI      6       /* Timer task */
#define TSK_TRP_PRI      6       /* Trap task */
```

```
/* Task stack size */
#define TSK_RCV_STK       1024    /* Receive task (byte) */
#define TSK_TIM_STK       512     /* Timer task (byte) */
#define TSK_TRP_STK       768     /* Trap task (byte) */

/* Maximum size of an SNMP message (4-byte aligned) */
/* The maximum size of an SNMP messages to be received or transmitted */
#define CFG_SNMP_RCV_MSG_LEN    2048                    /* Message can receive */
#define CFG_SNMP_SND_MSG_LEN    CFG_SNMP_RCV_MSG_LEN    /* Message can send*/
```

5.2 Configuring Managers

This section describes how to designate managers. The user can select which managers to allow as the source of SNMP messages. If an SNMP packet from a manager other than the selected ones is received, this system discards the packet on reception. It is also possible to receive all SNMP packets without limiting the source managers.

To select managers, declare the array variable in the T_SNMP_CFG_MGR structure as “snmp_cfg_mgr”. This structure contains the following variable.

```
/* Manager */
typedef struct t_snmp_cfg_mgr {
        T_NODE* nod;        /* Remote node */
} T_SNMP_CFG_MGR;
```

Number	Type	Variable Name	Description
1	T_NODE	nod	The network device number of the manager to allow receiving messages from and its IP address. For T_NODE, specify 0 in “port” and IP_VER4 in “ver”.

Examples of implementation are given below. Add a null character at the end to terminate the array.

```
static T_NODE snmp_cfg_mgr_nod_1 = {0/*port*/, IP_VER4, NET_DEV_ID, 0xc0a8016e};
                                                      /* 0xc0a8016e = 192.168.1.110 */
static T_NODE snmp_cfg_mgr_nod_2 = {0/*port*/, IP_VER4, NET_DEV_ID, 0xc0a80165};
                                                      /* 0xc0a80165 = 192.168.1.101 */

T_SNMP_CFG_MGR snmp_cfg_mgr[] = {
        {&snmp_cfg_mgr_nod_1},
        {&snmp_cfg_mgr_nod_2},
        0
};
```

This is an example of receiving SNMP packets from all the managers. Set an empty value in the variable as shown below.

```
/* Receive SNMP packets from all managers (managers not specified) */
T_SNMP_CFG_MGR snmp_cfg_mgr[] = {
        0
};
```

5.3 Configuring Communities

This section describes how to configure the community. Declare the array variable in the structure T_SNMP_CFG_COM as “snmp_cfg_com”. This structure contains the following variables.

```
/* Community */
typedef struct t_snmp_cfg_com {
        VB* str;            /* Community strings */
        UB sts;             /* Access status */
} T_SNMP_CFG_COM;
```

Number	Type	Variable Name	Description
1	VB*	str	A string which represents the community name
2	UB	sts	Access mode STS_RO: read only STS_RW: readable and writable

Examples of implementation are given below. Configuring multiple communities is possible. Add a null character at the end to terminate the array.

```
static VB snmp_cfg_com_ro[] = "public";      /* Read only */
static VB snmp_cfg_com_rw[] = "private";     /* Read and write */

T_SNMP_CFG_COM snmp_cfg_com[] = {
        {snmp_cfg_com_ro, STS_RO},
        {snmp_cfg_com_rw, STS_RW},
        {0, 0}
};
```

5.4 Configuring Destinations for Sending Standard Traps

This section describes how to configure the destination for sending standard traps. Declare the array variable in the structure T_SNMP_CFG_TRP as “snmp_cfg_trp”. This structure contains the following variables.

```
/* Trap */
typedef struct t_snmp_cfg_trp {
    VB* str;            /* Community strings */
    T_NODE* nod;        /* Remote node */
    UB ver;             /* Protocol version */
    ID id;              /* ID (Reserve) */
} T_SNMP_CFG_TRP;
```

Number	Type	Variable Name	Description
1	VB*	str	A string which represents the community name of the destina-tion for sending traps.
2	T_NODE*	nod	The network device number of the destination and its IP address. For T_NODE, specify 0 in “port” and IP_VER4 in “ver”.
3	UB	ver	The version number of the trap Version 1: SNMP_VER_V1 Version 2c: SNMP_VER_V2C
4	ID	id	Always set 0 for the current version numbers.

Examples of implementation are given below. Add a null character at the end to terminate the array.

```
static VB snmp_cfg_trp_com_1[] = "public";
static VB snmp_cfg_trp_com_2[] = "public";
static T_NODE snmp_cfg_trp_nod_1 = {0/*port*/, IP_VER4, NET_DEV_ID, 0xc0a8016e};
                                                        /* 0xc0a8016e = 192.168.1.110 */
static T_NODE snmp_cfg_trp_nod_2 = {0/*port*/, IP_VER4, NET_DEV_ID , 0xc0a80165};
                                                        /* 0xc0a80165 = 192.168.1.101 */

T_SNMP_CFG_TRP snmp_cfg_trp[] = {
    {snmp_cfg_trp_com_1, &snmp_cfg_trp_nod_1, SNMP_VER_V2C, 0},
    {snmp_cfg_trp_com_2, &snmp_cfg_trp_nod_2, SNMP_VER_V1,  0},
    {0, 0, 0, 0}
};
```

The configuration described here applies to standard traps which are sent within this system such as coldStart and linkUp. Destinations for vendor’s traps, which users send by calling the API function snd_trp, are specified in the respective arguments.

5.5 Configuring Standard Callbacks for Vendor’s Private MIB

This section describes how to configure the standard callback functions for the vendor’s private MIB (Figure 2.3). Declare the array variable in the structure T_SNMP_CFG_CBK as “snmp_cfg_cbk”. This structure contains the following variables.

```
/* Callback functions */
typedef struct t_snmp_cfg_cbk {
    ER (*fnc)(T_SNMP_CFG_CBK_DAT*);
} T_SNMP_CFG_CBK;
```

Number	Type/Variable Name	Description
1	ER (*fnc)(T_SNMP_CFG_CBK_DAT*)	A pointer to the standard callback function

Examples of implementation are given below. Only a single function may be recorded. Add a null character at the end to terminate the array.

```
extern ER apl_snmp_cbk_0(T_SNMP_CFG_CBK_DAT*);

T_SNMP_CFG_CBK snmp_cfg_cbk[] = {
    apl_snmp_cbk_0,
    0
};
```

This is an example of implementation when callback function is not used. Set an empty value for the variable as shown below.

```
T_SNMP_CFG_CBK snmp_cfg_cbk[] = {
    0
};
```

The configuration described here applies to the standard callback function. In addition to the standard callbacks, users can configure multiple callback functions for individual vendor-specific extended MIB objects. The standard callback function is not issued for objects for which a separate callback function has been set. See the subsequent sections for how to set the callback functions for each object.

6. Configuring Vendor-Specific MIBs

This section describes how to configure the vendor-dependent MIBs. The system group of the MIB-II is macro-defined in the configuration file snmp_mib_cfg.h. Vendor-specific extended MIBs are configured by setting variables in the configuration file snmp_mib_cfg.c.

6.1 Configuring System Groups of the MIB-II

This section describes how to configure the system group of the MIB-II. This group contains sysDescr (defining the name and version identifier of the hardware and software) and sysObjectID (vendor's object ID). Define values for these objects in the configuration file snmp_mib_cfg.h by using the macro definitions listed below.

Table 6.1 Macros for Configuring the System Group

Target	Macro Definition	Example Value	Description
System group	CFG_SNMP_MIB_SYS_DESCR_LEN	(32 + 1)	The maximum number of characters allowed in sysDescr including a null terminator
	CFG_SNMP_MIB_SYS_DESCR	"HW:Ver.1.0.0 SW:Ver.1.0.0"	sysDescr (1.3.6.1.2.1.1.1) The name and version identifier of the hardware and software
	CFG_SNMP_MIB_SYS_OBJECTID_LEN	(32 + 1)	The maximum number of characters allowed in sysObjectID including a null terminator
	CFG_SNMP_MIB_SYS_OBJECTID	"1.3.6.1.4.1.1234"	sysObjectID (1.3.6.1.2.1.1.2) The vendor's object ID (the ID of the enterprise field on the trap (v1))
	CFG_SNMP_MIB_SYS_CONTACT_LEN	(32 + 1)	The maximum number of characters allowed in sysContact including a null terminator
	CFG_SNMP_MIB_SYS_CONTACT	"Email address"	sysContact (1.3.6.1.2.1.1.4) The contact of the device manager (e-mail address)
	CFG_SNMP_MIB_SYS_NAME_LEN	(32 + 1)	The maximum number of characters allowed in sysName including a null terminator
	CFG_SNMP_MIB_SYS_NAME	"System name"	sysName (1.3.6.1.2.1.1.5) Domain name of the device
	CFG_SNMP_MIB_SYS_LOCATION_LEN	(32 + 1)	The maximum number of characters allowed in sysLocation including a null terminator
	CFG_SNMP_MIB_SYS_LOCATION	"First floor"	sysLocation (1.3.6.1.2.1.1.6) Physical location of the device
	CFG_SNMP_MIB_SYS_SERVICES	64	sysServices (1.3.6.1.2.1.1.7) A value which indicates the set of services that this device may potentially offer

Examples of implementation are given below.

```
/* System sysDescr (1.3.6.1.2.1.1.1) */
/* The name and version identifier of the hardware and software */
#define CFG_SNMP_MIB_SYS_DESCR_LEN          (32 + 1)    /* The maximum length including a terminating null
    character */
#define CFG_SNMP_MIB_SYS_DESCR              "HW:Ver.1.0.0 SW:Ver.1.0.0"

/* System sysObjectID (1.3.6.1.2.1.1.2) */
/* Vendor's Object ID */
/* sysObjectID in the system group of the MIB and the enterprise field of the trap (v1) */
#define CFG_SNMP_MIB_SYS_OBJECTID_LEN       (32 + 1)
#define CFG_SNMP_MIB_SYS_OBJECTID           "1.3.6.1.4.1.1234"

/* System sysContact (1.3.6.1.2.1.1.4) */
/* Contact of the device manager (e-mail address) */
#define CFG_SNMP_MIB_SYS_CONTACT_LEN        (32 + 1)
#define CFG_SNMP_MIB_SYS_CONTACT            "Email address"

/* System sysName (1.3.6.1.2.1.1.5) */
/* Domain name of the device */
#define CFG_SNMP_MIB_SYS_NAME_LEN           (32 + 1)
#define CFG_SNMP_MIB_SYS_NAME               "Evalution board"

/* System sysLocation (1.3.6.1.2.1.1.6) */
/* Physical location of the device */
#define CFG_SNMP_MIB_SYS_LOCATION_LEN       (32 + 1)
#define CFG_SNMP_MIB_SYS_LOCATION           "First floor"

/* System sysServices (1.3.6.1.2.1.1.7) */
/* A value which indicates the set of services that this device may potentially offer */
#define CFG_SNMP_MIB_SYS_SERVICES           64          /* Application layer */
```

6.2 Configuring Vendor's Private MIBs

Users can add vendor-specific extended MIBs in the enterprise group under the private subtree of the MIB tree. This section describes how to configure the extended MIBs.

The structure of the extended MIBs described in this section is shown below.

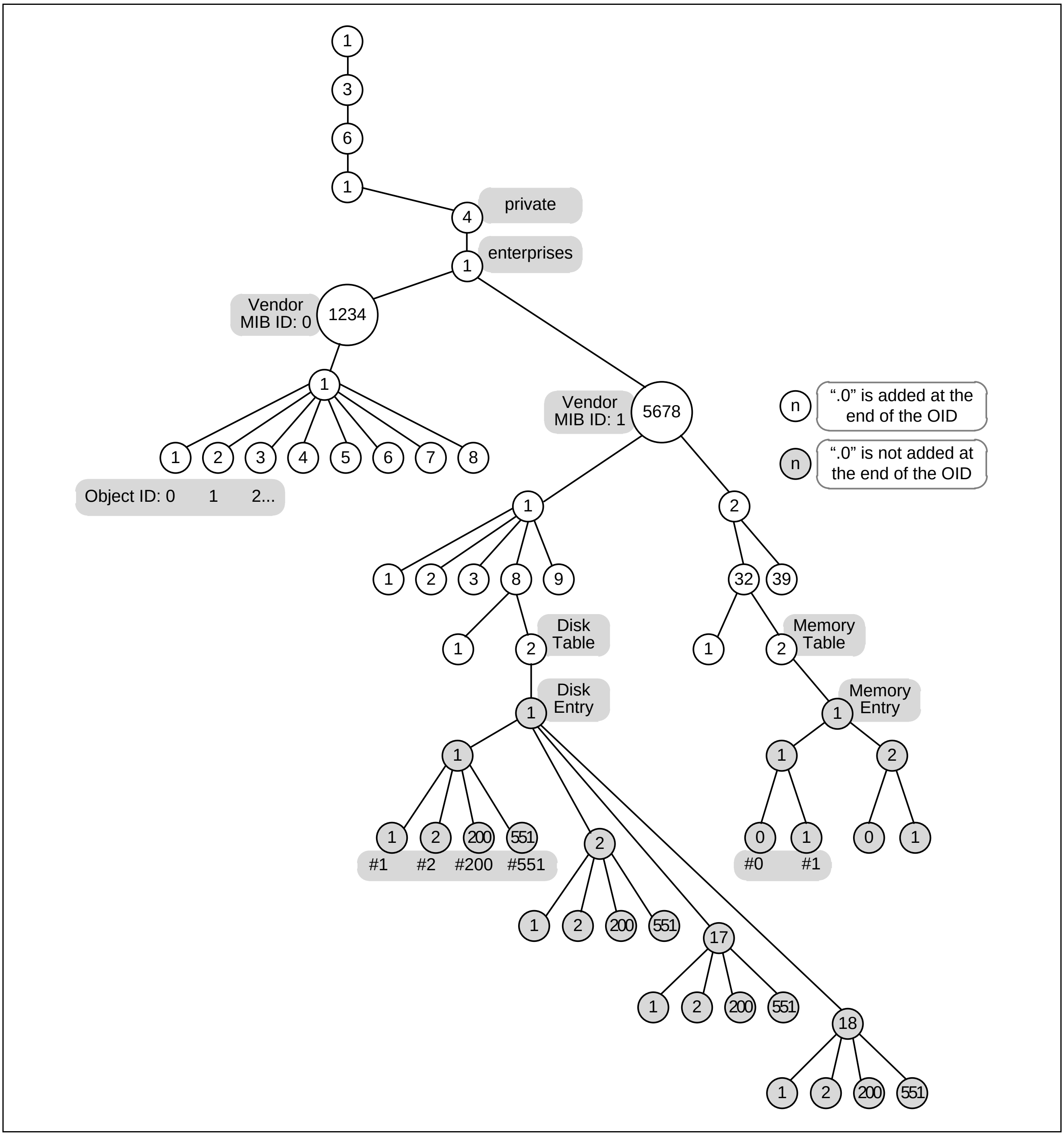
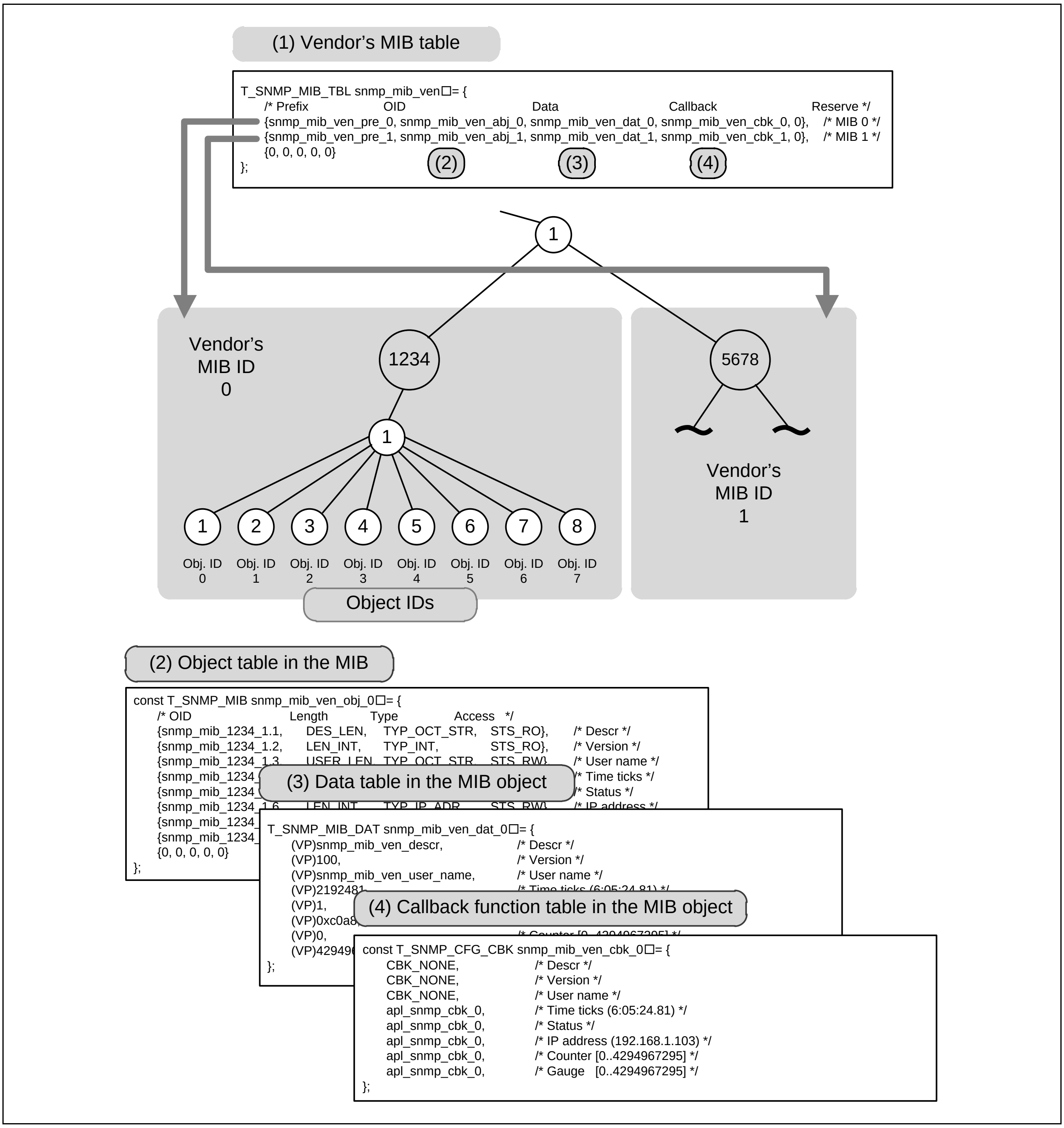


Figure 6.1 Example of Implementing Vendor's Extended MIBs

6.2.1 MIB IDs and Object IDs

In general, an object in a MIB is represented by numerals separated by "." such as "1.3.6.1.4.1.1234.1.1". In this system, the objects are recognized by using ID codes (16-bit values of type UH). If, as shown in Figure 6.1, the MIB tree is split into two groups with OIDs "1.3.6.1.4.1.1234.*" and "1.3.6.1.4.1.5678.*", an MIB ID (8-bit value of UB type) is applied to each group.

In configuration of an extended MIB tree, (1) declare the MIB table in the figure below in an array and then, (2) declare the object table, (3) the data table, and (4) the callback function table in the table (1). Here, the index for the table (1) (index to the array) selects an 8-bit MIB ID and the index for table (2) (index to the array) selects an object ID.



```
T_SNMP_MIB_TBL snmp_mib_ven□= {
    /* Prefix                OID                    Data                   Callback               Reserve */
    {snmp_mib_ven_pre_0, snmp_mib_ven_abj_0, snmp_mib_ven_dat_0, snmp_mib_ven_cbk_0, 0},    /* MIB 0 */
    {snmp_mib_ven_pre_1, snmp_mib_ven_abj_1, snmp_mib_ven_dat_1, snmp_mib_ven_cbk_1, 0},    /* MIB 1 */
    {0, 0, 0, 0, 0}
};
```

```
const T_SNMP_MIB snmp_mib_ven_obj_0□= {
    /* OID                  Length        Type            Access   */
    {snmp_mib_1234_1.1,     DES_LEN,      TYP_OCT_STR,    STS_RO},       /* Descr */
    {snmp_mib_1234_1.2,     LEN_INT,      TYP_INT,        STS_RO},       /* Version */
    {snmp_mib_1234_1.3,     USER_LEN,     TYP_OCT_STR,    STS_RW},       /* User name */
    {snmp_mib_1234_                                                      /* Time ticks */
    {snmp_mib_1234_                                                      /* Status */
    {snmp_mib_1234_1.6,     LEN_INT,      TYP_IP_ADR,     STS_RW},       /* IP address */
    {snmp_mib_1234_
    {snmp_mib_1234_
    {0, 0, 0, 0, 0}
};
```

```
T_SNMP_MIB_DAT snmp_mib_ven_dat_0□= {
    (VP)snmp_mib_ven_descr,                 /* Descr */
    (VP)100,                                /* Version */
    (VP)snmp_mib_ven_user_name,             /* User name */
    (VP)2192481,                            /* Time ticks (6:05:24.81) */
    (VP)1,
    (VP)0xc0a8
    (VP)0,                                  /* Counter [0..4294967295] */
    (VP)42949
};
```

```
const T_SNMP_CFG_CBK snmp_mib_ven_cbk_0□= {
    CBK_NONE,               /* Descr */
    CBK_NONE,               /* Version */
    CBK_NONE,               /* User name */
    apl_snmp_cbk_0,         /* Time ticks (6:05:24.81) */
    apl_snmp_cbk_0,         /* Status */
    apl_snmp_cbk_0,         /* IP address (192.168.1.103) */
    apl_snmp_cbk_0,         /* Counter [0..4294967295] */
    apl_snmp_cbk_0,         /* Gauge   [0..4294967295] */
};
```

Figure 6.2 Configuration of Vendor's Extended MIBs

IDs are used for recognizing individual MIBs and objects in the API functions and callback functions of this system with the variable names "mib_id"and "obj_id". The capacity of individual MIB ID and object ID is 8 bits and 16 bits, respectively.

One MIB table can contain up to 256 groups and one object table can contain up to 65,536 objects.

An extended MIB is configured by declaring the array variable in the configuration file snmp_mib_cfg.c. The array variables for the MIB table ((1) in above figure) and the data table ((3) in above figure) are assigned to the RAM area because values to the variables are to be changed. Other array variables are assigned to the ROM area. Implementation of array variables are described in the subsequent sections.

6.2.2 MIB Tables

As shown in Figure 6.1, the extended MIB is split into two groups. The MIB IDs are 0 for the group with the OID "1.3.6.1.4.1.1234.*" and 1 for the other with the OID "1.3.6.1.4.1.5678.*". At the beginning of the configuration, specify pointers to the variables which will configure the objects at the group level. Declare the array variable in the T_SNMP_MIB_TBL structure as "snmp_mib_ven". This structure contains the following variables.

```
/* Vendor MIB table */
typedef struct t_snmp_mib_tbl {
    const VB* pre;                    /* Prefix */
    const T_SNMP_MIB* mib;            /* Objects */
    T_SNMP_MIB_DAT* dat;              /* Data */
    T_SNMP_CFG_CBK* cbk;              /* Callback functions */
    UH cnt;                           /* Predefined */
} T_SNMP_MIB_TBL;
```

Number	Type	Variable Name	Description
1	const VB*	pre	A pointer to the prefix (strings) of the MIB OID
2	const T_SNMP_MIB*	mib	A pointer to the configuration variable for the T_SNMP_MIB structure (MIB OID, size, type, access restriction)
3	T_SNMP_MIB_DAT*	dat	A pointer to the T_SNMP_MIB_DAT structure (object data)
4	T_SNMP_CFG_CBK*	cbk	A pointer to the configuration variable for the T_SNMP_CFG_CBK structure (callback function for each object). Set 0x00 if a callback function is not to be set.
5	UH	cnt	Predefined variable (used inside this system)

An example of implementation is given below. Two groups are configured in this example. Up to 255 MIB groups can be configured in the snmp_mib_ven structure. Add a null character at the end to terminate the array.

```
/* Vendor MIB table */
/* The table of vendor-specific MIB groups (add {0, 0, 0, 0, 0} at the end) */
T_SNMP_MIB_TBL snmp_mib_ven[] = {
    /* Prefix             OID               Data              Callback            Reserve */
    {snmp_mib_ven_pre_0, snmp_mib_ven_obj_0, snmp_mib_ven_dat_0, 0x00,                 0},
    {snmp_mib_ven_pre_1, snmp_mib_ven_obj_1, snmp_mib_ven_dat_1, snmp_mib_ven_cbk_1, 0},
    {0, 0, 0, 0, 0}
};
```

The subsequent section describes how to configure the variables to the T_SNMP_MIB_TBL structure.

6.2.3 OID Prefix

This section describes how to configure the variable “pre” of the T_SNMP_MIB_TBL structure. Specify the pointer to the prefix (strings) of an OID in pre. Here, the prefix means the common header of the dotted strings of OIDs. In the following example, there are two sub-groups in the enterprise group under the private subtree, one with the OID “1.3.6.1.4.1.1234.*” and the other with the OID “1.3.6.1.4.1.5678.*”. Declare the prefix and set it in the variable pre as follows. Add a period at the end of each string of the prefix.

```
/* Prefix of the MIB OID (add a period at the end) */
const VB snmp_mib_ven_pre_0[] = "1.3.6.1.4.1.1234.";     /* Prefix OID (MIB 0) */
const VB snmp_mib_ven_pre_1[] = "1.3.6.1.4.1.5678.";     /* Prefix OID (MIB 1) */

const T_SNMP_MIB_TBL snmp_mib_ven[] = {
    /* Prefix               OID                 Data                        Callback                Reserve */
    {snmp_mib_ven_pre_0, snmp_mib_ven_obj_0, snmp_mib_ven_dat_0, 0x00,                      0},
    {snmp_mib_ven_pre_1, snmp_mib_ven_obj_1, snmp_mib_ven_dat_1, snmp_mib_ven_cbk_1, 0},
    {0, 0, 0, 0, 0}
};
```

6.2.4 Object Table

This section describes how to configure the variable “mib” in the T_SNMP_MIB_TBL structure. Specify the pointer to the variable for the T_SNMP_MIB structure in mib, which are, the OID string following the prefix, the maximum amount of data for the object, data type, and access mode. This structure contains the following variables.

```
/* MIB object */
typedef struct t_snmp_mib {
    const VB* str;    /* Object string */
    UH len;           /* Size (byte) */
    UB typ;           /* Type */
    UB acs;           /* Accsess */
} T_SNMP_MIB;
```

Number	Type	Variable Name	Description
1	const VB*	str	A string of numerals separated by a period for the OID following the prefix of the object
2	UH	len	The maximum amount of data for the object in bytes
3	UB	typ	Data type of the object
4	UB	acs	Access mode of the object

In the variable “typ” (data type of the object) of this structure, set values by using the macros below.

Table 6.2 Data Types of Objects

Number	Macro	Size (len) in bytes	Description	Remark
1	TYP_NONE	4	Type is not defined.	For the Entry object in the table
2	TYP_INT	4	Integer	32bit
3	TYP_OCT_STR	1 to CFG_SNMP_MIB_DAT_LEN	Octet String	Character string
4	TYP_SEQ	4	SEQUENCE	For the Table object
5	TYP_IP_ADR	4	Ip Address	32 bits (for IPv4)
6	TYP_CNT	4	Counter	32 bits
7	TYP_GAUGE	4	Gauge	32 bits
8	TYP_TIM_TIC	4	Time ticks	32 bits

In the variable “acs” (access mode) of this structure, set values by using the macros below.

Table 6.3 Access Mode of Objects

Number	Macro	Description	Remark
1	STS_NO	Reference not allowed	The object is not-accessible. This is used only for the Table and Entry objects.
2	STS_RO	Read only	
3	STS_WO	Write only	
4	STS_RW	Readable and writable	

In the variable “str” of this structure, specify the OID string following the prefix, which was specified in the previous section.

An example of implementation for the group with the prefix “1.3.6.1.4.1.1234.*” is given below. Here, only the OID strings following the prefix are declared. Add “0” (instance identifier) at the end of the each string if the target object is not in the table.

```
const VB snmp_mib_ven_pre_0[] = "1.3.6.1.4.1.1234.";      /* Prefix OID (MIB 0) */

/* OID of MIB 0 (add a period at the end) */
const VB snmp_mib_1234_1_1[] = "1.1.0";      /* Descr          (1.3.6.1.4.1.1234.1.1) */
const VB snmp_mib_1234_1_2[] = "1.2.0";      /* Version        (1.3.6.1.4.1.1234.1.2) */
const VB snmp_mib_1234_1_3[] = "1.3.0";      /* User name      (1.3.6.1.4.1.1234.1.3) */
const VB snmp_mib_1234_1_4[] = "1.4.0";      /* Time ticks     (1.3.6.1.4.1.1234.1.4) */
... (The rest are omitted.)
```

An example of implementation for the group with the prefix “1.3.6.1.4.1.5678.*” is given below. The objects “disk table” and “memory table” are tables. The disk table contains the ifIndex objects with the values 1, 2, 200, and 551. The memory table contains the ifIndex objects with values 0 and 1. In configuration of the t_snmp_mib.str structure, if the object to be configured is not in a table, “.0” is added to the OID. If the object to be configured is a table, do not add “.0”, including entry and the lower-order objects (the circles in gray in Figure 6.1).

```
/* OID of MIB 1 (add ".0" at the end) */
/* However, ".0" is not added to the OIDs of entry and the lower-order objects if the object is a table */
const VB snmp_mib_5678_1_1[]            = "1.1.0";       /* Descr                   (1.3.6.1.4.1.5678.1.1) */
const VB snmp_mib_5678_1_2[]            = "1.2.0";       /* Version                 (1.3.6.1.4.1.5678.1.2) */
const VB snmp_mib_5678_1_3[]            = "1.3.0";       /* Status                  (1.3.6.1.4.1.5678.1.3) */
const VB snmp_mib_5678_1_8[]            = "1.8.0";       /* Disk                    (1.3.6.1.4.1.5678.1.8) */
const VB snmp_mib_5678_1_8_1[]          = "1.8.1.0";     /* Disk number             (1.3.6.1.4.1.5678.1.8.1) */
const VB snmp_mib_5678_1_8_2[]          = "1.8.2.0";     /* Disk table              (1.3.6.1.4.1.5678.1.8.2) */
                                                         /* The beginning of the entry table (".0" is not added) */
const VB snmp_mib_5678_1_8_2_1[]        = "1.8.2.1";     /* Disk entry              (1.3.6.1.4.1.5678.1.8.2.1) */
const VB snmp_mib_5678_1_8_2_1_1_1[]    = "1.8.2.1.1.1"; /* Disk #1    ifIndex      (1.3.6.1.4.1.5678.1.8.2.1.1.1) */
const VB snmp_mib_5678_1_8_2_1_1_2[]    = "1.8.2.1.1.2"; /* Disk #2    ifIndex      (1.3.6.1.4.1.5678.1.8.2.1.1.2) */
const VB snmp_mib_5678_1_8_2_1_1_200[]  = "1.8.2.1.1.200"; /* Disk #200ifIndex      (1.3.6.1.4.1.5678.1.8.2.1.1.200) */
const VB snmp_mib_5678_1_8_2_1_1_551[]  = "1.8.2.1.1.551"; /* Disk #551ifIndex      (1.3.6.1.4.1.5678.1.8.2.1.1.551) */
... (The rest are omitted.)
```

In the variable "len" of this structure, set the maximum amount of data of the object in bytes. Set four in this variable except for the following case; the data type of the object is TYP_OCT_STR (a character string), as shown in Table 6.2. In this case, set the maximum size for the strings including the terminating null character. For example, if the value in "len" is (32 + 1), the maximum number of strings allowed in response to the SetRequest command from the manager is thirty-two strings. Note that the value in "len" cannot exceed the value specified in the macro CFG_SNMP_MIB_DAT_LEN (the maximum amount of data of the object), which is described in Section 5.1, Basic Settings.

An example of implementation of the T_SNMP_MIB structure is given below. Add a null character at the end to terminate the array. The index of these arrays (0 to 7) represent the object IDs (obj_id).

```
#define LEN_INT         4           /* Data length of the data types INT, CNT, GAUGE, and IP_ADR */

/* Vendor Descr */
#define DESCR_LEN       (16 + 1)    /* The maximum length of the strings including a terminating null character */
/* User name */
#define USER_LEN        (32 + 1)

/* MIB 0 */
/* Configuration of the vendor-specific MIB (add {0, 0, 0, 0} at the end) */
const T_SNMP_MIB snmp_mib_ven_obj_0[] = {
    /* OID                  Length         Type            Access    */
    {snmp_mib_1234_1_1,     DESCR_LEN,     TYP_OCT_STR,    STS_RO},      /* Descr */
    {snmp_mib_1234_1_2,     DESCR_LEN,     TYP_OCT_STR,    STS_RO},      /* Version */
    {snmp_mib_1234_1_3,     USER_LEN,      TYP_OCT_STR,    STS_RW},      /* User name */
    {snmp_mib_1234_1_4,     LEN_INT,       TYP_TIM_TIC,    STS_RW},      /* Time ticks */
    {snmp_mib_1234_1_5,     LEN_INT,       TYP_INT,        STS_RW},      /* Status */
    {snmp_mib_1234_1_6,     LEN_INT,       TYP_IP_ADR,     STS_RW},      /* IP address */
    {snmp_mib_1234_1_7,     LEN_INT,       TYP_CNT,        STS_RO},      /* Counter */
    {snmp_mib_1234_1_8,     LEN_INT,       TYP_GAUGE,      STS_RO},      /* Gauge */
    {0, 0, 0, 0}
};
```

6.2.5 Data Table

This section describes how to configure the variable “dat” in the T_SNMP_MIB_TBL structure. Specify the buffer for the object data with an initial value in “dat”. The data are stored in an array variable of the T_SNMP_MIB_DAT type, with its VP (void*) being converted to a new name.

```
/* MIB data or data pointer */
typedef   VP   T_SNMP_MIB_DAT;
```

An example of implementation is given below.

If the data type is TYP_OCT_STR (a character string), convert the pointer to the beginning of the buffer where the string is stored to the VP type before setting in the array of the T_SNMP_MIB_DAT type. The buffer for the strings needs to be large enough to allocate the maximum amount of data specified in the variable “len” of the T_SNMP_MIB structure.

If the data type is other than TYP_OCT_STR such as four bytes of type TYP_INT, convert the initial value of the data into the VP type and set it in the array.

Declare a buffer for strings with an access mode other than read-only and the array variable of the T_SNMP_MIB_DAT type in the RAM area. These variables hold the pointers to the buffers where the object data are to be stored and may be overwritten while data are being processed. The array variables do not need a terminating null character at the end.

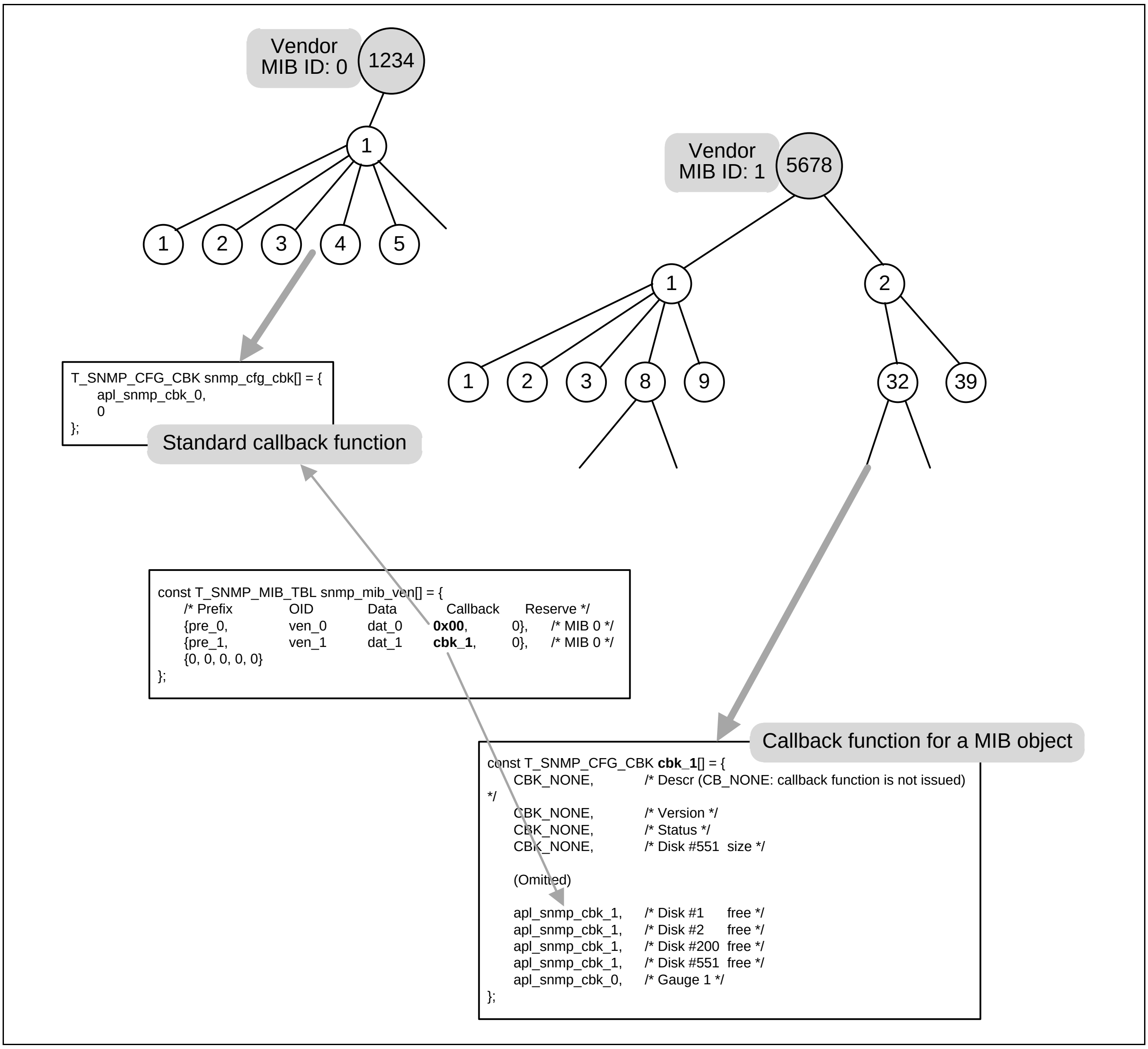
```
/* Vendor Descr */
static const VB snmp_mib_ven_descr[DESCR_LEN] = {
    "Vendor MIB"
};

/* User name */
static VB snmp_mib_ven_user_name[USER_LEN] = {
    "User name"
}
;
/* MIB 0 data */
/* Buffer where the vendor-specific MIB data are stored */
T_SNMP_MIB_DAT snmp_mib_ven_dat_0[] = {
    (VP)snmp_mib_ven_descr,         /* Descr strings */
    (VP) snmp_mib_ven_ver,          /* Version */
    (VP)snmp_mib_ven_user_name,     /* User name strings */
    (VP)2192481,                    /* Time ticks (6:05:24.81) */
    (VP)1,                          /* Status */
    (VP)0xc0a80167,                 /* IP address (192.168.1.103) */
    (VP)0,                          /* Counter [0..4294967295] */
    (VP)10                          /* Gauge [0..4294967295] */
};
```

6.2.6 Callback Function Table

This section describes how to configure the variable “cbk” of the T_SNMP_MIB_TBL structure, in other words, how to configure the callback function to each object.

As described in Section 2.5, Vendor-Specific MIB and Callback Function, this system issues an user-defined callback function in response to the request to the vendor-specific extended MIB object from the manager. In addition to the standard callback function described in Section 5.5, Configuring Standard Callbacks for Vendor’s Private MIB, users can configure callback functions for individual objects.


Vendor
MIB ID: 0
1234
1
1
2
3
4
5
Vendor
MIB ID: 1
5678
1
2
1
2
3
8
9
32
39
Standard callback function
Callback function for a MIB object

```
T_SNMP_CFG_CBK snmp_cfg_cbk[] = {
      apl_snmp_cbk_0,
      0
};
```

```
const T_SNMP_MIB_TBL snmp_mib_ven[] = {
      /* Prefix          OID          Data          Callback      Reserve */
      {pre_0,            ven_0        dat_0         0x00,         0},      /* MIB 0 */
      {pre_1,            ven_1        dat_1         cbk_1,        0},      /* MIB 0 */
      {0, 0, 0, 0, 0}
};
```

```
const T_SNMP_CFG_CBK cbk_1[] = {
      CBK_NONE,           /* Descr (CB_NONE: callback function is not issued)
*/
      CBK_NONE,           /* Version */
      CBK_NONE,           /* Status */
      CBK_NONE,           /* Disk #551  size */

      (Omitted)

      apl_snmp_cbk_1,     /* Disk #1      free */
      apl_snmp_cbk_1,     /* Disk #2      free */
      apl_snmp_cbk_1,     /* Disk #200  free */
      apl_snmp_cbk_1,     /* Disk #551  free */
      apl_snmp_cbk_0,     /* Gauge 1 */
};
```

Figure 6.3 Standard Callback Function and Callback Function for Each Object

Here, how to switch the standard callback function and the callback function for each object is described. In an example of implementation, the group with the OID “1.3.6.1.4.1.1234.*” is configured to issue standard callback functions and the other with the OID “1.3.6.1.4.1.5678.*” is configured to issue a callback function to each object. The mode of callback is judged by the value in “cbk” of the T_SNMP_MIB_TBL structure (Figure 6.3). The value 0x00 (null) in cbk is for the default callback function and other values are for cbk for the callback function for objects specified in the array variables of the T_SNMP_CFG_CBK structure.

How to declare the array variable to this structure is shown in the example below. Note that no callback function is issued to the relevant object if the element value in the array variable is 0x00 (null). At this time, no standard callback is issued as well.

This array variable does not need a terminating null character at the end.

```
#define CBK_NONE     0x00        /* Callback function not defined */

const T_SNMP_CFG_CBK snmp_mib_ven_cbk_1[] = {
        CBK_NONE,              /* Descr          (No callback functions are issued at all) */
        CBK_NONE,              /* Version        (No standard callback functions are issued as well) */
        CBK_NONE,              /* Status */
        (Omitted))
        CBK_NONE,              /* Disk #2        size */
        CBK_NONE,              /* Disk #200      size */
        CBK_NONE,              /* Disk #551      size */
        apl_snmp_cbk_1,        /* Disk #1        free (The callback function for the object is apl_snmp_cbk_1) */
        apl_snmp_cbk_1,        /* Disk #2        free */
        apl_snmp_cbk_1,        /* Disk #200      free */
        (Omitted))
        apl_snmp_cbk_0,        /* Gauge 2 */
}
```

7. Interfaces

This section describes how to use the API functions of this system and their callback functions.

7.1 List of Functions

This system provides the following functions.

Category	Function Name	Description
Initialization	snmp_ini	Initialize the system
	snmp_ext	Exit the system
	snmp_ena	Enable the system
	snmp_dis	Disable the system
Management information	get_mib_obj	Read data from a vendor's MIB object
	set_mib_obj	Write data to a vendor's MIB object
Trap	ena_trp	Enable standard traps
	dis_trp	Disable standard traps
	snd_trp	Send vendor-specific traps

Use set_mib_obj or a callback function to rewrite the data in a vendor-specific MIB object. If the user directly rewrites the data in an object declared in the snmp_mib_cfg.c file, the system may return a half-written value to the manager. Directly rewriting the data in the objects is still possible if the system is yet to be started.

7.2 Specification of Functions

Details on the functions used in this system are described in this section.

7.2.1 *snmp_ini* (Initialization)

Format

```
ER snmp_ini(UH* mib_nod_cnt)
```

Parameters		
UH*	mib_nod_cnt	A pointer to the variable where the number of nodes in the MIB tree is stored
Returned value		
ER	ercd	E_OK for a normal termination or an error code
Error codes		
E_PAR	An error in the configuration file	
E_NOMEM	Insufficient memory (insufficient number of nodes in the MIB tree)	
E_BOVR	Maximum amount of data for the object is small.	
E_OBJ	Other error	

Description

This function is used for initializing this system. Issue this function before using this system, following initialization (net_ini) of the TCP/IP protocol stack.

This function handles the initialization of internal variables, initialization of internal buffers, generation of OS resources (except for the compact version of the operating system), generation of the MIB tree, and generation of network sockets.

Once the MIB tree is generated, this function returns the value for the number of nodes to be used in the tree in the mib_nod_cnt argument. Users are required to obtain this value and set it to the macro CFG_SNMP_MIB_NOD_CNT in the basic settings. If this value is not necessary, specify 0x00 (null) in the mib_nod_cnt argument.

The error code E_PAR is returned for an error in the configuration files (xxx_cfg.h and xxx_cfg.c). The error code E_NOMEM is returned if the value in CFG_SNMP_MIB_NOD_CNT is too small to create a MIB tree. The error code E_BOVR is returned if the value of CFG_SNMP_MIB_DAT_LEN is small.

7.2.2 *snmp_ext* (Exit)

Format

```
ER snmp_ext(void)
```

Parameters

None

Returned value

ER	ercd	E_OK for a normal termination

Error codes

None

Description

This function causes a normal termination of the system. The resources generated in the function are freed, except for that for the compact version of the operating system. When the system is initialized (snmp_ini) and enabled (snmp_ena) again after issuing this function, it sends a coldStart trap.

7.2.3 *snmp_ena* (Enable)

Format

```
ER snmp_ena(void)
```

Parameters

None

Returned value

ER	ercd	E_OK for a normal termination or the error code

Error codes

E_OBJ	The system is not initialized (snmp_ini is not issued).

Description

This function enables the system and wakes up the tasks in the system. This function receives SNMP packets while a task is running. This function sends a coldStart trap when it is issued the first time and sends a warmStart trap the second and subsequent times. Basically, issue this function before issuing the function for initializing the Ethernet driver (net_dev_ini). If the Ethernet driver has been initialized and connected to the network before issuing this function, a linkUp trap will not be issued.

The error code E_OBJ is returned if this function is issued before the system is initialized (snmp_ini).

7.2.4 *snmp_dis* (Disable)

Format

```
ER snmp_dis(void)
```

Parameters

None

Returned value

ER	ercd	E_OK for a normal termination or the error code

Error codes

E_OBJ	The system has not been disabled (snmp_ena has not been issued).

Description

This function disables the system and terminates the tasks in the system. It may take up to two seconds to terminate all the tasks in this function.

The error code E_OBJ is returned if this function is issued before the system is enabled (snmp_ena).

7.2.5 *get_mib_obj* (Read Data from a Vendor's MIB Object)

Format

```
ER get_mib_obj(VP buf, UH* len, UH mib_id, UH obj_id)
```

Parameters		
VP	buf	A pointer to the buffer where data is stored
UH*	len	Size of the buffer and data (in bytes)
UH	mib_id	MIB ID
UH*	obj_id	Object ID
Returned value		
ER	ercd	E_OK for a normal termination or the error code
Error codes		
E_PAR	Argument error	
E_BOVR	Insufficient buffer length	
E_OBJ	Other error	

Description

This function reads values from the vendor-specific MIB objects specified in the arguments mib_id and obj_id and stores them in the buf argument. Buffer size for the data is specified in the len argument in bytes.

The error code E_BOVR is returned for insufficient buffer size (buf). In this case, the len argument with the value for necessary buffer size is returned. This system also returns len with the value for the read data size in a successful reading process. The content in the buffer (buf) is undefined in the case of an error.

The variable for buf should be 4 or more when the type of the object data is Integer, Counter32, Gauge32, Time Ticks, or four-byte IP Address. When the type of the object data is octet string, the buf argument must have the area which is large enough to allocate the number of strings to be obtained (without including the terminating null character). In this case, the string stored in the buf does not need a null character (\0) at the end.

An example of implementation is given below.

```
#define MAX_STR_LEN     32        /* Maximum string buffer size */
#define MAX_DAT_LEN     32        /* Maximum buffer size */
static UW apl_str_buf[MAX_STR_LEN / sizeof(UW)];
static UW apl_dat_buf[MAX_DAT_LEN / sizeof(UW)];

VB* str;
UW* dat;
UH len;
UH mib_id;
UH obj_id;

str = (VB*)apl_str_buf;
dat = apl_dat_buf;
len = MAX_DAT_LEN;

mib_id = 0;
obj_id = 2;
ercd = get_mib_obj(dat, &len, mib_id, obj_id);
```

```
if (ercd == E_OK) {
        if (snmp_mib_ven[mib_id].mib[obj_id].typ == TYP_OCT_STR) {
                /* TYP_OCT_STR (a character string) */
                ((VB*)dat)[len] = '\0';        /* add a null string before printf */
                printf((const VB*)dat);
        } else if (snmp_mib_ven[mib_id].mib[obj_id].typ == TYP_IP_ADR) {
                /* TYP_IP_ADR (four-byte IP address) */
                ip_ntoa(str, *dat);
                printf(str);
        } else {
                /* TYP_INT, TYP_CNT, TYP_GAUGE, TYP_TIM_TIC (four-byte value) */
                printf("0x%x", *dat);
        }
}
```

7.2.6 *set_mib_obj* (Write Data to a Vendor’s MIB Object)

Format

```
ER set_mib_obj(VP buf, UH len, UH mib_id, UH obj_id)
```

Parameters			
	VP	buf	A pointer to the buffer where data is stored
	UH*	len	Size of the data in bytes
	UH	mib_id	MIB ID
	UH*	obj_id	Object ID
Returned value			
	ER	ercd	E_OK for a normal termination or the error code
Error codes			
	E_PAR	Argument error	
	E_OBJ	Data overflow or underflow, or other error	

Description

This function writes values from the buf argument to the vendor-specific MIB objects specified in the arguments mib_id and obj_id. Buffer size for the data is specified in the len argument in bytes. The error code E_OBJ is returned if the buffer overflows or underflows.

The variable for “len” should be 4 or more when the type of the object data is integer, counter32, gauge32, time ticks, or four-byte IP address.

When the type of the object data is octet string, specify the number of the strings of the data (without including the terminating null character) in the len argument. In this case, the string to be stored in buf does not need a null character (\0) at the end.

Note that this function also updates data in objects for which only read access is allowed.

Example of implementation is given below.

```
#define MAX_STR_LEN       32            /* Maximum string buffer size */
static UW apl_str_buf[MAX_STR_LEN / sizeof(UW)];

VB* str;
UW dat;
UH len;
UH mib_id;
UH obj_id;

str = (VB*)apl_str_buf;

mib_id = 0;
obj_id = 2;

if (snmp_mib_ven[mib_id].mib[obj_id].typ == TYP_OCT_STR) {
        /* TYP_OCT_STR (a character string) */
        strcpy(str, "test1234");
        len = strlen(str);
        ercd = set_mib_obj(str, len, mib_id, obj_id);
} else if (snmp_mib_ven[mib_id].mib[obj_id].typ == TYP_IP_ADR) {
        /* TYP_IP_ADR (four-byte IP address) */
        dat = 0xC0A80167;
        ercd = set_mib_obj(&dat, 4, mib_id, obj_id);
} else {
        /* TYP_INT, TYP_CNT, TYP_GAUGE, TYP_TIM_TIC (four-byte integer value) */
        dat = 1234;
        ercd = set_mib_obj(&dat, 4, mib_id, obj_id);
}
```

7.2.7 *ena_trp* (Enable Standard Traps)

Format

```
ER ena_trp(UH trp_bit)
```

Parameters		
UH	trp_bit	The macro for the standard trap
Returned value		
ER	ercd	E_OK for a normal termination or the error code
Error codes		
E_OBJ	Other error	

Description

This function enables standard traps used in this system. Specify the macro or macros for the trap to be enabled in the argument trp_bit.

Trap Number	Identifier (Macro)	Value	Trap Name
0	COLD_STA_BIT	0x0001	coldStart
1	WARM_STA_BIT	0x0002	warmStart
2	Unsupported	—	linkDown
3	LINK_UP_BIT	0x0008	linkUp
4	AUTH_FAIL_BIT	0x0010	authenticationFailure
5	Unsupported	—	egpNeighborLoss
—	TRP_ALL_BIT	0x003f	All traps

An example of implementation is given below.

```
/* enabling coldStart and linkUp */
ercd = ena_trp(COLD_STA_BIT | LINK_UP_BIT);
```

7.2.8 *dis_trp* (Disable Standard Traps)

Format

```
ER dis_trp(UH trp_bit)
```

Parameters		
UH	trp_bit	The macro for the standard trap
Returned value		
ER	ercd	E_OK for a normal termination or the error code
Error codes		
E_OBJ	Other error	

Description

This function disables standard traps used in this system. Specify the macro or macros for the trap to be disabled in the argument trp_bit.

Trap Number	Identifier (Macro)	Value	Trap Name
0	COLD_STA_BIT	0x0001	coldStart
1	WARM_STA_BIT	0x0002	warmStart
2	Unsupported	—	linkDown
3	LINK_UP_BIT	0x0008	linkUp
4	AUTH_FAIL_BIT	0x0010	authenticationFailure
5	Unsupported	—	egpNeighborLoss
—	TRP_ALL_BIT	0x003f	All traps

An example of implementation is given below.

```
/* disabling warmStart and linkUp */
ercd = dis_trp(WARM_STA_BIT | LINK_UP_BIT);
```

7.2.9 *snd_trp* (Send Vendor-Specific Traps)

Format

```
ER snd_trp(T_NODE* nod, T_SNMP_TRP* trp, TMO tmo)
```

Parameters

T_NODE*	nod	A pointer to the transmission destination node
T_SNMP_TRP*	trp	A pointer to the command of the trap
TMO	tmo	Time until expiration of the monitoring period (in milliseconds)

Returned value

ER	ercd	E_OK for a normal termination or the error code

Error codes

E_PAR	An invalid parameter was specified.
E_TMO	Timeout
E_OBJ	Other error

Description

This function sends a vendor-specific trap or Inform packet to a particular destination. In the transmission of traps, once this function generates a trap packet, it waits until transmission of the trap by the UDP is completed. In transmission of Inform packets, once this function sends the notification to the destination, it waits until the response packet is received, so this takes longer than the time for waiting in the transmission of traps.

Designate the destination of transmission in the nod argument including its IP address but not the port (nod.port) because the system specifies it (port 162).

Specify the value for timeout in the tmo argument. Specify the pointer to the variable for the T_SNMP_TRP structure in the trp argument according to the following table for variables.

Number	Type	Variable Name	Content
1	UB	ver	The macro for the version number of the protocol: SNMP_VER_V1: for v1 SNMP_VER_V2C: for v2c
2	VB*	com	Strings which represent the community name
3	UH	flg	Option flag
4	VB*	ent_oid	The string which represents the OID for the enterprise (for v1) The string which represents the OID for snmpTrapOID (for v2c)
5	INT	gen_trp	The value which represents a standard trap (v1) Always set TRP_ENT_SPEC.
6	INT	spc_trp	The value which represents trap specification (for v1)
7	UH	tmo	Timeout value (msec) (in sending Inform packets)
8	UH	rty_cnt	The number of retrials (in sending Inform packets)
9	VP	var_oid	The object ID or IDs of the variable binding or bindings to be added
10	UH	var_cnt	The number of added variable binding or bindings

Specify the version number of the protocol for the trap in the ver argument, SNMP_VER_V1 for v1 and SNMP_VER_V2C for v2c.

Specify the community name where the trap is to be sent in the com argument.

Specify the options associated with trap transmission by using the macro below. Set 0x00 in the variable to select transmission of a trap.

Number	Identifier (Macro)	Description
1	TRP_INF_ENA	Send an inform packet instead of a trap

For v1 traps, specify the string of the enterprise OID, for example, “1.3.6.1.4.1.1234”, in the argument ent_oid. Specifying 0x00 (null) in this argument uses the OID which was specified by the configuration macro CFG_SNMP_MIB_SYS_OBJECTID in the snmp_mib_cfg.h file. For v2c traps, specify the second variable binding, the OID string for snmpTrapOID in the argument ent_oid.

v1 traps uses values in the variables gen_trp and spc_trp. Specify the macro TRP_ENT_SPEC (6) in gen_trp and the number which indicates the detailed trap information in spc_trp.

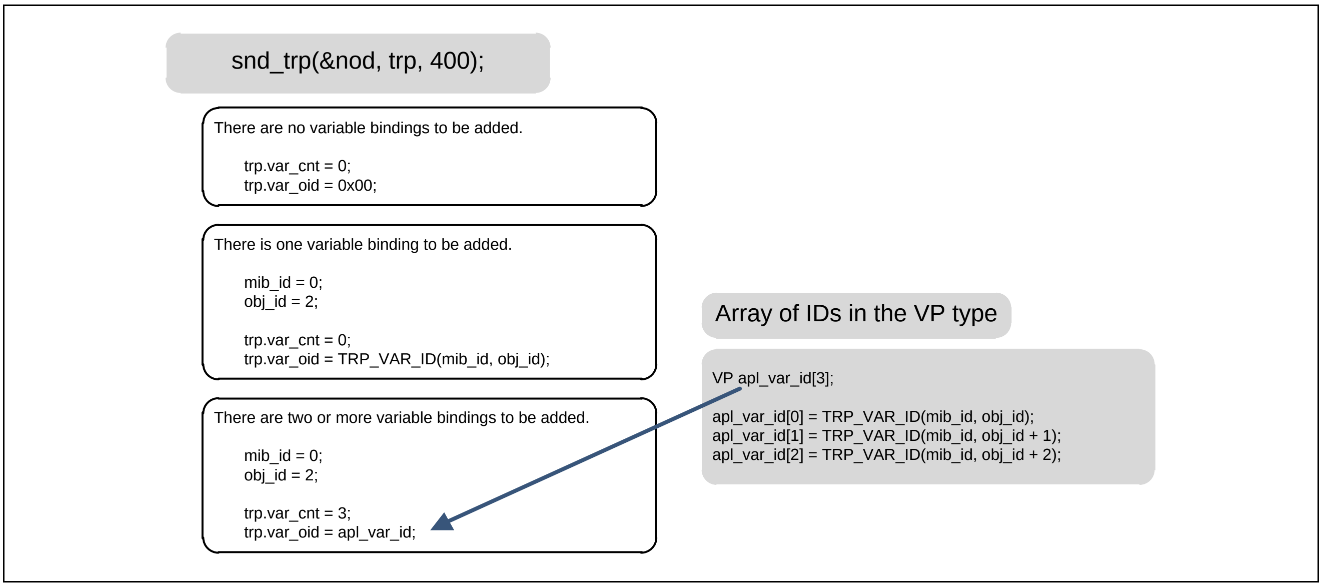
Values in the variables tmo and rty_cnt are used to send Inform packets. trp.tmo is the time until timeout expiration and trp.rty_cnt is the number the times sending of an Inform packet is retried. If there is no response from the destination after the time set in trp.tmo has elapsed, the Inform packet is resent the number of times set in trp.rty_cnt. Note that timeout detection proceeds every second (1000 ms), so the value in trp.tmo should be a multiple of 1000, for example, 8000 (eight seconds).

Specify the variable bindings to be added to a tarp in the variables var_oid and var_cnt. If there are no variable bindings to be added, specify 0 and 0x00 in vat_cnt and var_oid, respectively. If there is one variable binding to be added, specify 0 in vat_cnt and the vendor-specific MIB ID and OID in var_oid after converting them into the VP type, by setting the former in the sixteen higher-order bits and the latter in the sixteen lower-order bits.

A configuration macro is provided in the header file snmp.h as follows.

```
#define SNMP_TRP_VAR_ID(x, y)      ((VP)(((UH)(x) & 0x00ff) << 16 | (UH)(y)))
```

If two or more variable bindings are to be added, specify the number of targets in var_cnt and the pointer to the array of their IDs in var_oid in the VP type. In summary, set the MIB ID and OIDs of the variable bindings in an array of the VP type and set the pointer to the array in var_oid (Figure 7.1).



```
snd_trp(&nod, trp, 400);
```

There are no variable bindings to be added.

```
trp.var_cnt = 0;
trp.var_oid = 0x00;
```

There is one variable binding to be added.

```
mib_id = 0;
obj_id = 2;

trp.var_cnt = 0;
trp.var_oid = TRP_VAR_ID(mib_id, obj_id);
```

There are two or more variable bindings to be added.

```
mib_id = 0;
obj_id = 2;

trp.var_cnt = 3;
trp.var_oid = apl_var_id;
```

Array of IDs in the VP type

```
VP apl_var_id[3];

apl_var_id[0] = TRP_VAR_ID(mib_id, obj_id);
apl_var_id[1] = TRP_VAR_ID(mib_id, obj_id + 1);
apl_var_id[2] = TRP_VAR_ID(mib_id, obj_id + 2);
```

Figure 7.1 Adding Variable-Bindings to a Trap

An example of implementation for sending a v1 trap is given below.

```
T_SNMP_TRP trp;

memset(&trp, 0, sizeof(trp));
trp.ver = SNMP_VER_V1;            /* The trap version is v1 */
trp.com = "public";               /* Community name */
trp.gen_trp = TRP_ENT_SPEC;       /* Vendor-specific trap (fixed value) */
trp.spc_trp = 1234;               /* Detailed trap information (any integer value) */
ercd = snd_trp(nod, trp, TRP_TMO);
    /* If the value in trp.ent_oid is 0, the enterprise OID uses CFG_SNMP_MIB_SYS_OBJECTID (snmp_mib_cfg.h) */
    /* If the values in trp.var_cnt and trp.var_oid are 0, no variable bindings will be added */
```

An example of implementation for sending a v2c trap is given below.

```
memset(&trp, 0, sizeof(trp));
trp.ver = SNMP_VER_V2C;                 /* The trap version is v2c */
trp.com = "public";                     /* Community name */
trp.ent_oid = "1.3.6.1.4.1.1234.1.2";   /* snmpTrapOID vendor's private MIB */
ercd = snd_trp(nod, trp, TRP_TMO);
        /* If the values in trp.var_cnt and trp.var_oid are 0, there are no variable-bindings to be added */
```

An example of implementation for sending v1 trap with one variable binding is given below.

```
memset(&trp, 0, sizeof(trp));
trp.ver = SNMP_VER_V1;                  /* The trap version is v1 */
trp.com = "public";                     /* Community name */
trp.gen_trp = TRP_ENT_SPEC;             /* Vendor-specific trap (fixed value) */
trp.spc_trp = 1234;                     /* Detailed trap information (any value) */
trp.ent_oid = "1.3.6.1.4.1.9876.1234";  /* Set the enterprise OID strings */
trp.var_cnt = 0;                        /* The number of variable bindings to be added to a trap (set 0 if there is one) */
trp.var_oid = TRP_VAR_ID(0, 2);         /* The IDs of the variable bindings to be added */
ercd = snd_trp(nod, trp, TRP_TMO);
```

An example of implementation for sending a v2c trap with three variable bindings is given below.

```
VP apl_var_id[8];                       /* Variable binding ID */

memset(&trp, 0, sizeof(trp));
trp.ver = SNMP_VER_V2C;                 /* The trap version is v2c */
trp.com = "public";                     /* Community name */
trp.ent_oid = "1.3.6.1.4.1.1234.1.2";   /* snmpTrapOID vendor-specific MIB */
trp.var_cnt = 3;                        /* The number of variable bindings to be added to the trap */
apl_var_id[0] = TRP_VAR_ID(0, 2);       /* Element 2 of snmp_mib_ven_0 */
apl_var_id[1] = TRP_VAR_ID(1, 5);       /* Element 5 of snmp_mib_ven_1 */
apl_var_id[2] = TRP_VAR_ID(1, 6);       /* Element 6 of snmp_mib_ven_1 */
trp.var_oid = apl_var_id;               /* Array of the ID of the variable bindings */
ercd = snd_trp(nod, trp, TRP_TMO);
```

An example of implementation for sending an Inform packet is given below.

```
memset(&trp, 0, sizeof(trp));
trp.ver = SNMP_VER_V2C;                 /* The trap version is v2c */
trp.com = "public";                     /* Community name */
trp.ent_oid = "1.3.6.1.4.1.1234.1.2";   /* snmpTrapOID vendor-specific MIB */
trp.flg = TRP_INF_ENA;                  /* Select an Inform packet instead of a trap */
trp.tmo = 8000;                         /* Timeout for sending an Inform packet (msec) */
trp.rty_cnt = 4;                        /* The number of times sending of the Inform packet is retried */
ercd = snd_trp(&nod, &trp, TRP_TMO);    /* Send to the server specified in nod */
```

7.3 Callback Functions

This section describes the specification of callback function provided in this system. This system issues callback functions in response to the reception of packets GetRequest, GetNextRequest, GetBulkRequest, and SetRequest to the vendor-specific private MIB objects from the manager.

The argument of this callback functions is shown below.

Format

```
ER fnc(T_SNMP_CFG_CBK_DAT* cbk_dat)
```

Parameters

T_SNMP_CFG_CBK_DAT*	cbk_dat	A pointer to the variable of the structure for callback

Returned value

ER	ercd	E_OK for a normal termination or the error code

Error codes

E_OBJ	An error occurred.

The cbk_dat argument is a pointer to the variable for the T_SNMP_CFG_CBK_DAT structure, which was declared in this system. This structure contains the following variables.

Number	Type	Variable Name	Description
1	UH	req	A macro which defines the type of SNMP request, as listed below SNMP_REQ_GET: Get Request, GetNextRequest, GetBulkRequest SNMP_REQ_SET: SetRequest
2	UH	mib_id	Vendor’s MIB ID
3	UH	obj_id	Vendor’s OID
4	UH	typ	A macro which defines the type of data in the object
5	VP	buf	The buffer where data are stored
6	UH	dat_len	Data size in bytes
7	UH	buf_len	Buffer size in bytes (valid only when the value in req is SNMP_REQ_GET)

The req variable indicates the type of the request from the manager. The value is SNMP_REQ_GET for GetRequest, GetNextRequest, and GetBulkRequest and SNMP_REQ_SET for SetRequest.

The variables mib_id and obj_id indicate the vendor-specific MIB ID and OID, respectively. For details on ID, see Section 6.2.1, MIB IDs and Object IDs.

The typ variable indicates the type of the object data by using the macros listed below.

Number	Macro	Data Type	Remark
1	TYP_INT	Integer	32 bits
2	TYP_OCT_STR	Octet String	Strings
3	TYP_IP_ADR	IP Address	32 bits (for IPv4)
4	TYP_CNT	Counter	32 bits
5	TYP_GAUGE	Gauge	32 bits
6	TYP_TIM_TIC	Time ticks	32 bits

The buf variable for the callback function holds the object data. It holds the current object data if the value in the req variable is SNMP_REQ_GET and the data in SetRequest specified by the manager if the value is SNMP_REQ_SET.

The buf variable holds strings if the value in the typ variable is TYP_OCT_STR. In this case, a terminating null character (\0) is added if the value in the req variable is SNMP_REQ_GET and not added if the value is SNMP_REQ_SET.

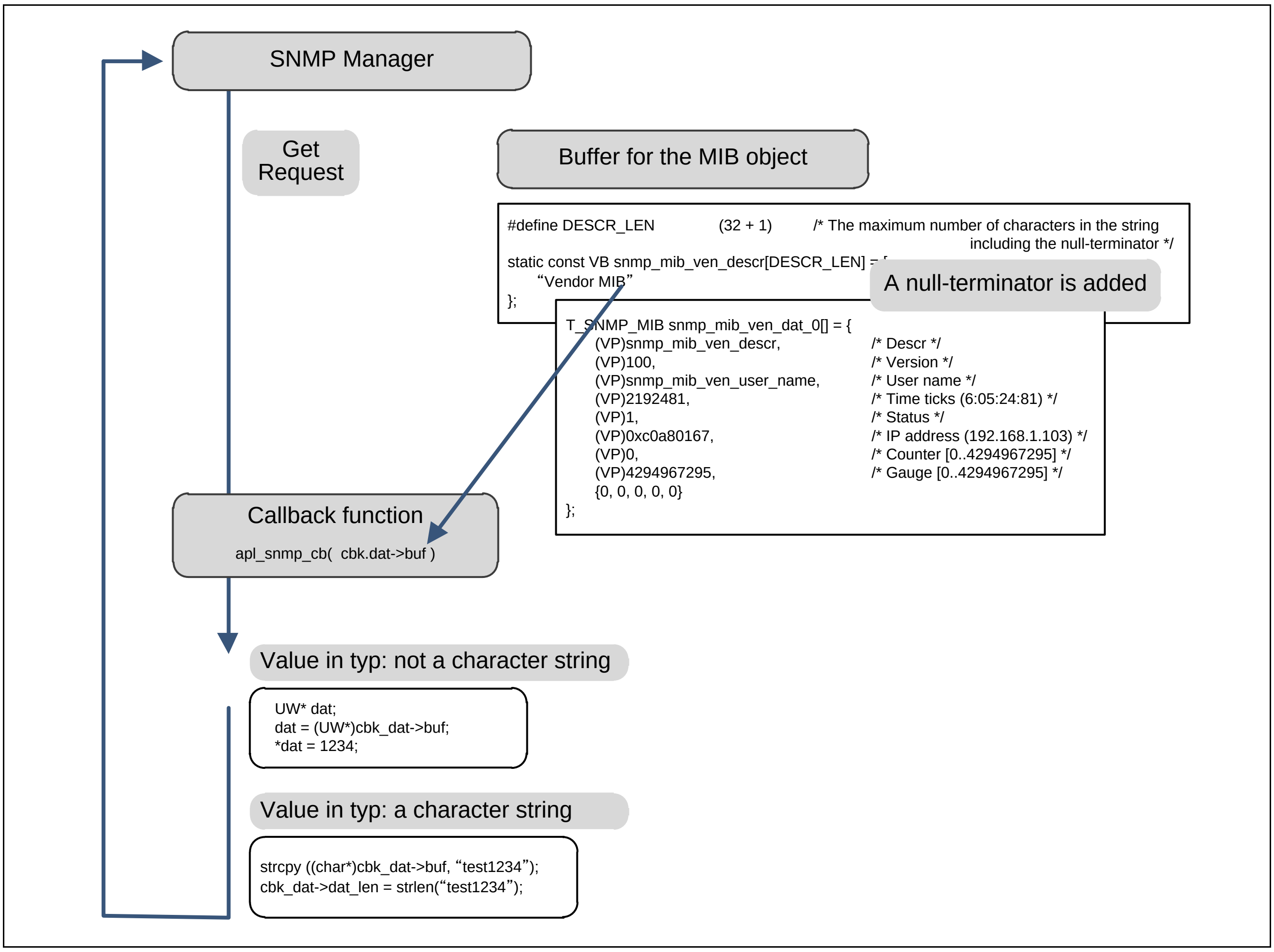
The dat_len variable indicates the length of the data stored in buf. If the value in typ is other than TYP_OCT_STR (a character string), the value in dat_len is four and the value in buf is four-byte data. If the value in typ is TYP_OCT_STR, the value in dat_len is the length of the strings without a terminating null character.

The buf_len variable indicates the size of the buffer area (buf). This variable is valid only when the value in req is SNMP_REQ_GET. If the value in typ is other than TYP_OCT_STR, the value in buf_len is 4. If the value in typ is TYP_OCT_STR (a character string), the value in buf_len is the length of the strings which is allowed in buf, including a terminating null character.

The value in req is SNMP_REQ_GET when the reception task receives a packet of GetRequest, GetNextRequest or GetBulkRequest from the manager. At this time, user can update the MIB object by setting a desired value in buf. This system returns the given value to the manager. If the value in typ is other than TYP_OCT_STR, set four-byte data in buf. If the value in typ is TYP_OCT_STR (a character string), copy the strings directly from typ. Always set the return value of the callback function as E_OK.

The value in req is SNMP_REQ_SET when the reception task receives a SetRequest packet from the manager. At this time, buf holds the data to be updated by the manager. The user can choose whether to accept the update, by setting the return value in the callback to E_OK for accepting and E_OBJ for refusing. This system does not update the object data when E_OBJ is returned. In this case, the system returns an error code commitFailed to the manager. While SNMP_REQ_SET is set in req, do not rewrite the values in buf and dat_len.

From here, details of the argument cbk_dat->buf of the callback function are described. As in Figure 7.2, when a GetRequest packet is sent, the data pointed by the argument cbk_dat->buf is the buffer for the MIB object which was configured in the snmp_mib_cfg.c file by the user.


SNMP Manager
Get Request
Buffer for the MIB object
A null-terminator is added
Callback function
apl_snmp_cb(cbk.dat->buf)
Value in typ: not a character string
Value in typ: a character string

```
#define DESCR_LEN          (32 + 1)       /* The maximum number of characters in the string
                                                                   including the null-terminator */
static const VB snmp_mib_ven_descr[DESCR_LEN] = {
      “Vendor MIB”
};
```

```
T_SNMP_MIB snmp_mib_ven_dat_0[] = {
      (VP)snmp_mib_ven_descr,              /* Descr */
      (VP)100,                             /* Version */
      (VP)snmp_mib_ven_user_name,          /* User name */
      (VP)2192481,                         /* Time ticks (6:05:24:81) */
      (VP)1,                               /* Status */
      (VP)0xc0a80167,                      /* IP address (192.168.1.103) */
      (VP)0,                               /* Counter [0..4294967295] */
      (VP)4294967295,                      /* Gauge [0..4294967295] */
      {0, 0, 0, 0, 0}
};
```

```
UW* dat;
dat = (UW*)cbk_dat->buf;
*dat = 1234;
```

```
strcpy ((char*)cbk_dat->buf, “test1234”);
cbk_dat->dat_len = strlen(“test1234”);
```

Figure 7.2 Callback Function of GetRequest

As shown above, the user can directly rewrite the buffer of the object in a callback function.

If the value in typ is other than TYP_OCT_STR, buf holds a four-byte value of the current object. The user can change this value to a desired one. If the value in typ is TYP_OCT_STR (a character string), buf holds the strings of the current object including a terminating null character. The user can change this value to a desired one.

The value of buf covers the maximum number of the characters in the string including the null-terminator for the string. This makes it possible for the user to copy a new string to buf by using the function strcpy. It is also possible to copy a string which does not include a terminating null character without using the function. When this system exits the callback function, it adds a null-terminating character to the end of the string. The user is required to return dat_len with the same value as was copied to buf (the length of the object string) to this system so that it can use the value in dat_len when adding the null-character to terminate the string. Make sure that the new value does not exceed the buffer size (cbk_dat->buf_len).

Figure 7.3 below shows the callback function for SetRequest. The value pointed by the cbk_dat->buf argument of the callback function is the internal variable and the content of buf should not be rewritten by the user.

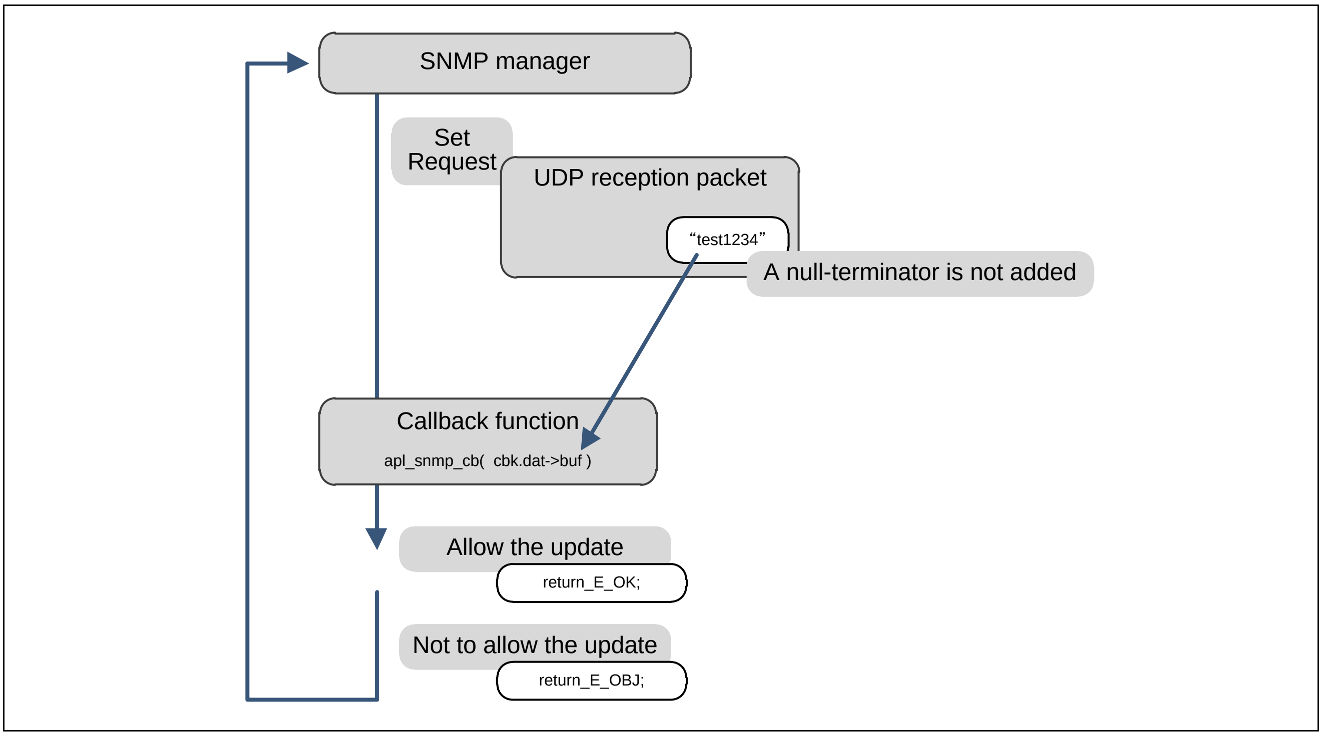


Figure 7.3 Callback Function for SetRequest

The buf variable holds the data to be updated by the manager. If the value in typ is TYP_OCT_STR (a character string), a null-terminating character is not added to the string in buf. This means that the user cannot use the function strcmp to compare strings. Instead, use the function strncmp.

An example of implementation of GetRequest is given below.

```
UW* dat;

if (cbk_dat->req == SNMP_REQ_GET) {
    /* Get request */
    if (cbk_dat->mib_id == 0) {
        /* MIB ID 0 */
        switch (cbk_dat->obj_id) {
            case 0:
                /* If the data type is a character string */
                len = strlen("New String");
                if (len < cbk_dat->buf_len) {
                    strcpy((char*)cbk_dat->buf, "New String");
                    cbk_dat->dat_len = len;
                }
                /* Data to be written should not exceed the buffer size (cbk_dat->buf_len) */
                /* Use strcpy and copy the “New String” to buf (terminating null characters can be added) */
                break;
            case 1:
                /* If the data type is strings */
                len = strlen(apl_new_str);      /* apl_new_str[] = "New String 2" */
                if (len < cbk_dat->buf_len) {
                    str = (VB*)cbk_dat->buf;
                    for (i = 0; i < len; i++) {
                        str[i] = apl_new_str[i];
                    }
                    cbk_dat->dat_len = len;
                }
                /* Use for loop and copy the strings to buf (terminating null characters can be omitted) */
                break;
            case 3:
                /* If the data type is integer value */
                dat = (UW*)cbk_dat->buf;
                *dat += 1;     /* An integer value is added */
                break;
                ....
return E_OK;     /* E_OK is returned */
```

An example of implementation for SetRequest is given below.

```
    ercd = E_OK;
if (cbk_dat->req == SNMP_REQ_SET) {
    /* Set request */
    if (cbk_dat->mib_id == 0) {
        /* MIB ID 0 */
        switch (cbk_dat->obj_id) {
            case 4:
                /* If the data type is strings */
                len = strlen("root");
                res = strncmp((const char*)cbk_dat->buf, "root", len);
                if (res == 0) {
                    ercd = E_OBJ;   /* Updating of data is not allowed if the beginning is same as the string “root” */
                }
                /* Use strncmp to compare the strings because the strings in cbk_dat->buf does not have
                    a terminating null character at the end */
                break;
            case 5:
                /* The data type is IP address */
                dat = (UW*)cbk_dat->buf;
                if ((*dat & 0xffff0000) != 0xc0a80000) {
                    ercd = E_OBJ;     /* Not to allow updating unless the IP address is 192.168.*.*   */
                }
                break;
            default:
                break;
        }
    }
}
....
return ercd;
```

REVISION HISTORY	R-IN32 Series User's Manual (μNet3/SNMP edition)

Rev.	Date	Description	
		Page	Summary
1.00	Sep 5, 2016	-	First edition issued

[MEMO]

R-IN32 Series User’s Manual
(µNet3/SNMP edition)

SALES OFFICES

Renesas Electronics Corporation

http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130
Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004
Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900
Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327
Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679
Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999
Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022
Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670
Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300
Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510
Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HALII Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777
Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141