User's Manual

RZ/T1 グループ

μNet3/SNMP ユーザーズマニュアル

・RZ/T1

資料番号：R01US0202JJ0200
発行年月：2020.11.1

ルネサス エレクトロニクス
www.renesas.com

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット
   高品質水準：輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）
   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等） を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等

8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有･使用を規制するRoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。
注2. 本資料において使用されている「当社製品」とは、注1 において定義された当社の開発、製造製品をいいます。

製品ご使用上の注意事項

ここでは、CMOS デバイスの一般的注意事項について説明します。個別の使用上の注意事項については、本文を参照してください。なお、本マニュアルの本文と異なる記載がある場合は、本文の記載が優先するものとします。

1. 未使用端子の処理
   【注意】未使用端子は、本文の「未使用端子の処理」に従って処理してください。
   CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】電源投入時は、製品の状態は不定です。
   電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。
   外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。
   同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. リザーブアドレスのアクセス禁止
   【注意】リザーブアドレスのアクセスを禁止します。
   アドレス領域には、将来の機能拡張用に割り付けられているリザーブアドレスがあります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
4. クロックについて
   【注意】リセット時は、クロックが安定した後、リセットを解除してください。
   プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。
   リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。

このマニュアルの使い方

1. 目的と対象者

このマニュアルは、SNMPインタフェースライブラリの機能を理解し、それを用いた応用システムを設計するユーザを対象にしています。

このマニュアルは、SNMPインタフェースライブラリの機能、動作、使用方法を理解していただくことを目的としています。

このマニュアルを読むにあたっては、マイクロコンピュータに関する一般知識を必要とします。

本LSIは、注意事項を十分確認の上、使用してください。注意事項は、各章の本文中、各章の最後、注意事項の章に記載しています。

改訂記録は旧版の記載内容に対して訂正または追加した主な箇所をまとめたものです。改訂内容すべてを記録したものではありません。詳細は、このマニュアルの本文でご確認ください。

目　次

1. はじめに

μNet3-SNMPは μNet3（TCP/IPプロトコルスタック）上にSNMPのエージェントの機能を提供するソフトウェアです。本ソフトウェアは次の図のようにマネージャーからのGetRequestなどのパケットに対して応答を返し、さらにトラップ（Trap）などをマネージャーに送信するエージェントの機能を持っています。つまり、本ソフトウェアを使用するとSNMPのマネージャーからイーサネットに接続している組み込み機器（エージェント）の状態を監視することができます。

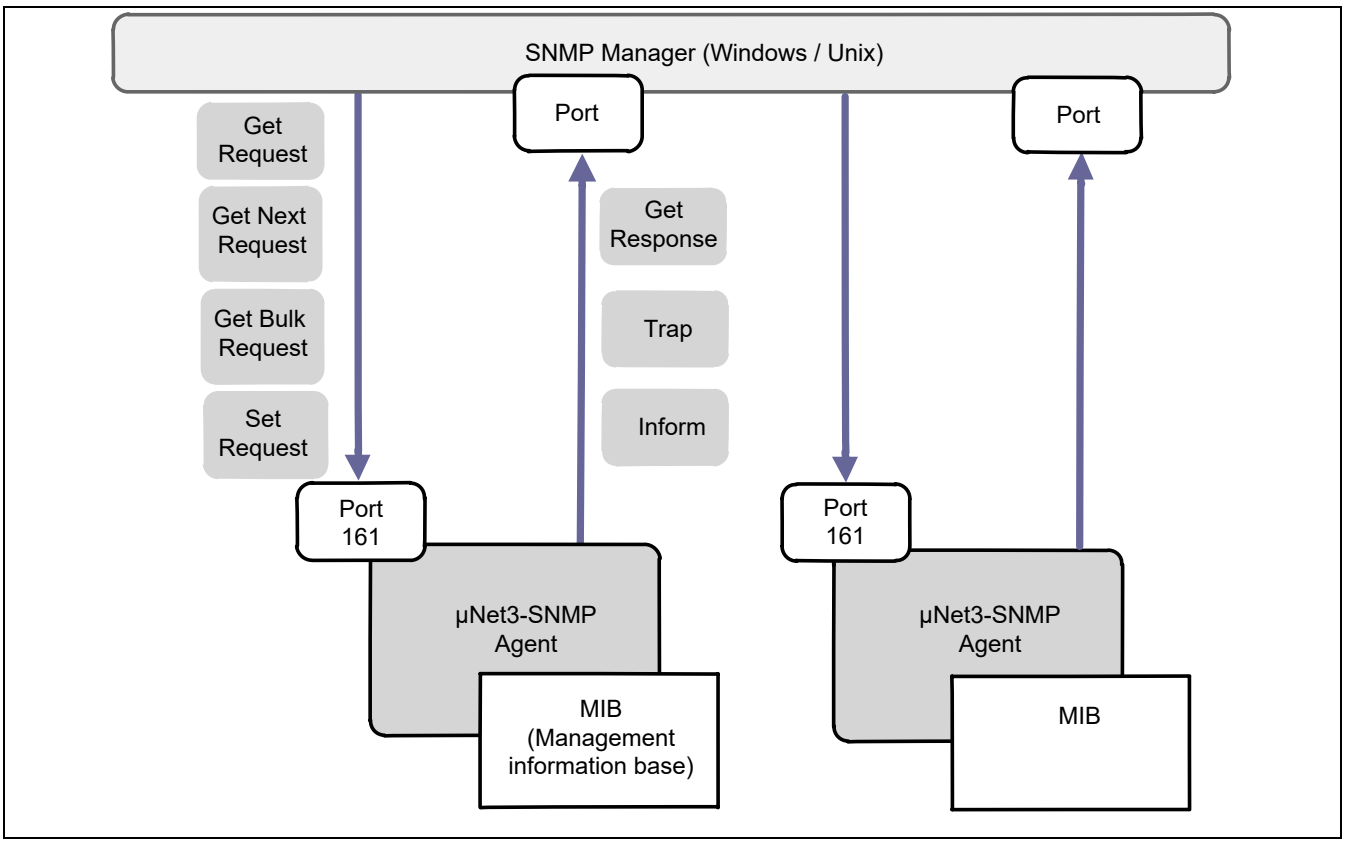

図1.1　SNMPのマネージャーとエージェント

なお、本ソフトウェアを使用する場合はSNMP 対応版のμNet3（TCP/IPプロトコルスタック）が必要です。

1.1 制限事項

制限事項は次のようになります。

- 本システムはSNMPのMIB-IIのオブジェクトに対応していますが、すべてのオブジェクトには対応していません。詳細は「2.2 MIB-II一覧」をお読みください。
- ベンダー固有のプライベートMIBで使用可能なMIBのオブジェクトのデータ型は Integer/Counter(32)/Gauge(32)/TimeTicks/IP Address/Octet String（文字列）のみです。それ以外のデータ型には対応していません。
- PPPのインタフェースには対応していません。イーサネットのインタフェースに対応しています。
- 複数のネットワークのデバイス（LANのポート）を持つ機器に対応していますが、最大で16 個のポートまでの対応になります。
- IPv6には対応していません。
- 関数の snd_trpでトラップの送信ができます。このトラップに追加するVariable bindingsには固定の（初期化時に作成した）ベンダーMIBのオブジェクトを指定できます。可変の（動作中に追加した）ベンダーMIBのオブジェクトは指定できません。

2. 仕様概要

本システムの仕様の概要を説明します。

2.1 仕様一覧

本システムの仕様の一覧は次のようになります。

表 2.1 仕様一覧

項目	内容		備考
SNMP 機能	エージェント		マネージャー機能なし
対応 SNMP バージョン	SNMPv1/SNMPv2c		v3 には非対応
対応 IP バージョン	IPv4		
対応 MIB	MIB-II	System group	注 1
		Interfaces group	
		Address translation group	
		IP group	
		ICMP group	
		TCP group	
		UDP group	
		EGP group	非対応
		Transmission group	非対応
		SNMP group	注 1
	private サブツリーの enterprises Group		ベンダー固有の拡張 MIB
対応トラップ	標準トラップ		coldStart(0) warmStart(1) linkDown(2) linkUp(3) authenticationFailure(4) のみ対応
	拡張トラップ		enterpriseSpecific(6) （ベンダー固有)
その他	標準トラップの有効・無効化用関数		関数名：ena_trp/dis_trp
	拡張トラップの発行用関数		関数名：snd_trp

注1. MIB-IIの各グループのすべてのオブジェクトには対応していません。非対応のオブジェクトの説明は次節をお読みください。

2.2 MIB-II一覧

本システムはMIB-IIに対応していますが、すべてのオブジェクトには対応していません。対応しているオブジェクトは次のようになります。なお、下記の表の“対応オブジェクト”の欄で網掛けをしているオブジェクトはデータを即時更新しません。データの更新間隔は最小で100msecになります。更新に関する説明は次節をお読みください。また、下記の表ではアクセスできないテーブル（tcpConnTableなど）やエントリ（tcpConnEntryなど）のオブジェクトは省略しています。

表2.2 MIB-II一覧(1 / 5)

グループ名	対応オブジェクト	制限事項	
System Group	sysDescr		sysServices より大きい ID のオブジェクトは非対応
	sysObjectID		
	sysUpTime		
	sysContact		
	sysName		
	sysLocation		
	sysServices		
Interfaces Group	ifNumber	µNet3 のデバイス番号（最大 16）	ネットワーク・インタフェースは 16 個まで対応
	ifTable		
	ifEntry		
	ifIndex		
	ifDescr		
	ifType		
	ifMtu		
	ifSpeed		
	ifPhysAddress		
	ifAdminStatus	値は常に 1。アクセスは read-only（read-write ではありません）。ネットワークのリンク状態を SNMP のマネージャーから変更することはできません。	
	ifOperStatus		
	ifLastChanges		
	ifInOctets	各オブジェクトの対応はイーサネットドライバの実装に依存。 ● AM335x 用のドライバではすべてのオブジェクトに対応しています。ただし、値はポート 1 とポート 2 の合計値になります。イーサネットのコントローラがポートごとの統計情報を提供していないため。 ● その他ドライバの対応は各ドライバのドキュメント(uNet3_XXX_ETH.txt)を参照してください。	
	ifInUcastPkts		
	ifInNUcastPkts		
	ifInDiscards		
	ifInErrors		
	ifInUnknownProtos		
	ifOutOctets		
	ifOutUcastPkts		
	ifOutNUcastPkts		
	ifOutDiscards		
	ifOutErrors		
	ifOutQLen	値は常に 0	
	ifSpecific	値は常に"0.0"	
Address Translation Group	atIfIndex		
	atPhysAddress		
	atNetAddress		

表2.2 MIB-II一覧 (2 / 5)

グループ名	対応オブジェクト	制限事項	
IP Group	ipForwarding	値は常に 2（notForwarding） アクセスは read-only（read-write ではありません）。	ipRoutingDiscards より大きい ID のオブジェクトは非対応。
	ipDefaultTTL	アクセスは read-only（read-write ではありません）。SNMP のマネージャーから値の変更はできません。	オブジェクトの ipRouteTable は非対応
	ipInReceives		
	ipInHdrErrors		
	ipInAddrErrors		
	ipForwDatagrams		
	ipInUnknownProtos		
	ipInDiscards		
	ipInDelivers		
	ipOutRequests		
	ipOutDiscards		
	ipOutNoRoutes		
	ipReasmTimeout		
	ipReasmReqds		
	ipReasmOKs		
	ipReasmFails		
	ipFragOKs		
	ipFragFails		
	ipFragCreates		
	ipAdEntAddr	本システムの起動時に生成し、削除することはありません。例えばリンクダウン時に本オブジェクトを削除しません。	
	ipAdEntIfIndex		
	ipAdEntNetMask		
	ipAdEntBcastAddr		
	ipAdEntReasmMaxSize		
	ipNetToMediaIfIndex	アクセスは read-only（read-write ではありません）。 SNMP のマネージャーから値の変更はできません。	
	ipNetToMediaPhysAddress		
	ipNetToMediaNetAddress		
	ipNetToMediaType		
	ipRoutingDiscards		

表2.2 MIB-II一覧 (3 / 5)

グループ名	対応オブジェクト	制限事項	
ICMP Group	icmpInMsgs		icmpOutAddrMaskReps より大きい ID のオブジェクトは非対応
	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		

表2.2 MIB-II一覧 (4 / 5)

グループ名	対応オブジェクト	制限事項	
TCP Group	tcpRtoAlgorithm		tcpOutRsts より大きい ID のオブジェクトは非対応
	tcpRtoMin		
	tcpRtoMax		
	tcpMaxConn		
	tcpActiveOpens		
	tcpPassiveOpens		
	tcpAttemptFails		
	tcpEstabResets		
	tcpCurrEstab		
	tcpInSegs		
	tcpOutSegs		
	tcpRetransSegs		
	tcpConnState	アクセスは read-only（read-write ではありません）。SNMP のマネージャーから値の変更はできません（例えば tcpConnState を deleteTCB に書き換えることはできません）	
	tcpConnLocalAddress		
	tcpConnLocalPort		
	tcpConnRemAddress		
	tcpConnRemPort		
	tcpInErrs		
	tcpOutRsts		
UDP Group	udpInDatagrams		udpLocalPortより大きいIDのオブジェクトは非対応
	udpNoPorts		
	udpInErrors		
	udpOutDatagrams		
	udpLocalAddress		
	udpLocalPort		

表2.2 MIB-II一覧 (5 / 5)

グループ名	対応オブジェクト	制限事項	
SNMP Group	snmpInPkts		snmpEnableAuthenTraps より大きい ID のオブジェクトは非対応
	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	アクセスは read-only（read-write ではありません）。SNMP のマネージャーから値の変更はできません。	

2.3 MIBオブジェクトデータの更新

MIBのオブジェクトのデータの更新間隔について説明します。MIBの各オブジェクトのデータはTCP/IPプロトコルスタックと本SNMPのモジュールの2カ所で保持します。前節の説明の最小100msecの間隔で更新するデータはプロトコルスタックが保持するデータです。また、即時更新するデータはSNMPのモジュールが保持するデータです（図2.1）。図2.1のようにインタフェースグループのリンク状態に関係するオブジェクトのデータは特別にコールバックで即時更新します。SNMPのモジュールではリンク検出のコールバックを受け取るとlinkUpのトラップを送信します（linkUpのトラップが有効な場合のみ）。

TCP/IPプロトコルスタックが保持するデータを一定の間隔で更新する理由は通信速度の低下を防ぐためです。プロトコルスタック内部ではタイマー用のタスクがあります。このタスクは100msec間隔で処理を行います。よって、データの更新間隔の最小は100msecになります。ユーザーはプロトコルスタックの設定用のマクロのCFG_STS_UPD_RESを使用してデータの更新間隔を調整することができます。マクロのCFG_STS_UPD_RESには更新間隔をmsecの単位（100の倍数）で指定してください。サンプルプログラム（net_cfg.h）では次のように2秒に設定にしています。例えば更新間隔を100msecにしたい場合はCFG_STS_UPD_RESを100で定義します。

–Sample/***.SNMP/net_cfg.c–（プロトコルスタックの設定ファイル）

```
#define CFG_STS_UPD_RES     2000    /* 2秒 */
```

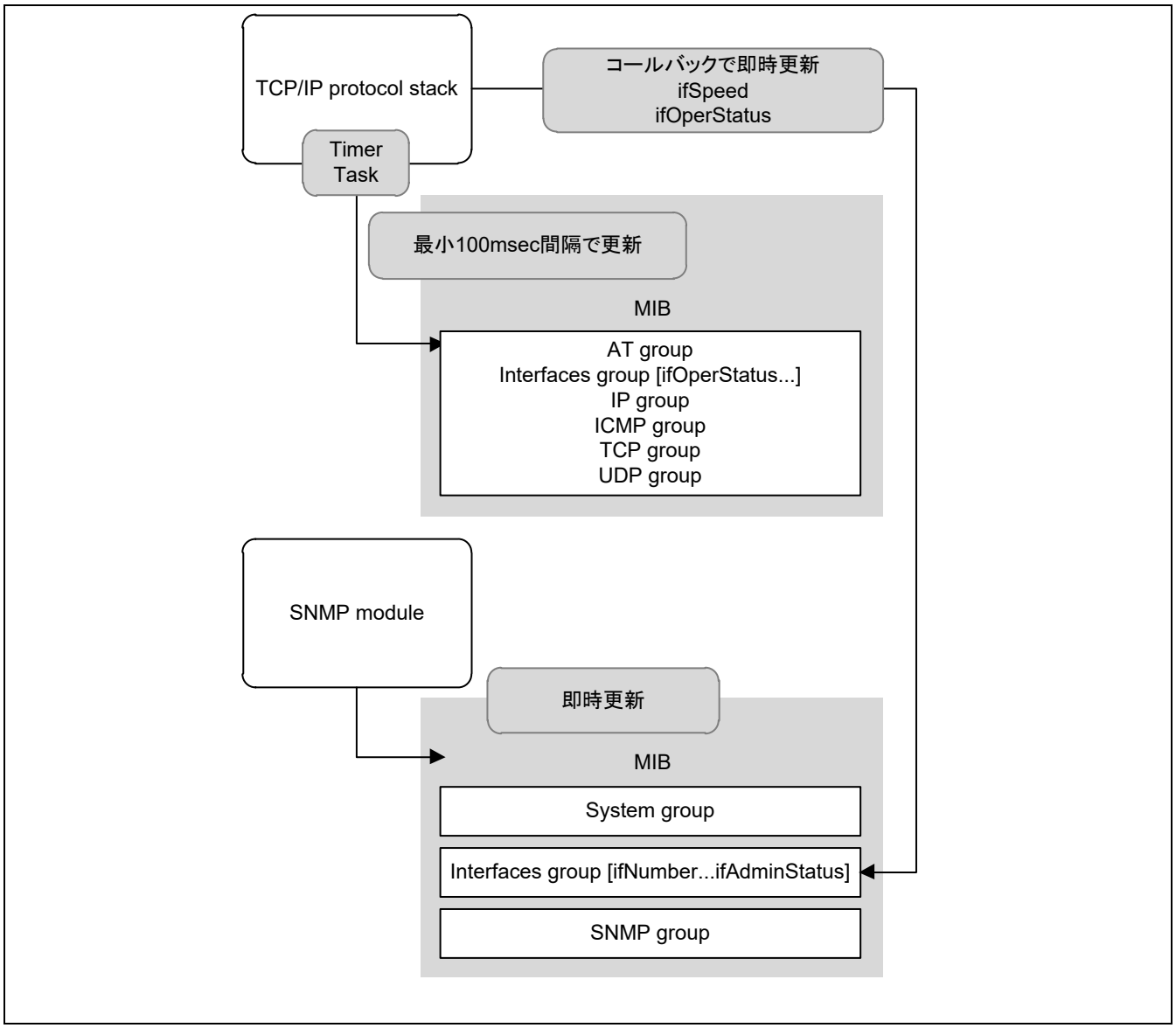

図2.1 MIB-IIのデータ更新

2.4 MIBツリーの作成

SNMPを実装するためには一般にMIBのツリーを作成する必要があります。本システムでも初期化時MIBのツリー（双方向のリスト）をメモリ（RAM）上に作成します。MIBのツリーを作成するためには、MIBの各オブジェクトのOID（“1.3.6.1.2.1.1.1”など）が必要になります。本システムでは設定するMIBのOIDを“ドット区切りの文字列”で指定します。たとえば、ベンダー固有のプライベートMIBのあるOIDが“1.3.6.1.4.1.1234.1.1”の場合は、次の網かけの文字列で設定します。

```
/* MIBのOIDのプリフィックス（最後に.を追加）*/
const VB snmp_mib_ven_pre_1[] = “1.3.6.1.4.1.1234.”;   /* Prefix OID */

/* プリフィックスに続くMIBのOID（最後に.0を追加）*/
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) */
```

よって、ユーザーがベンダー固有のプライベートMIBを設定するためには、このような文字列や各オブジェクトのデータ型、初期値などをC言語のソースファイルに実装する必要があります。

そして、本システムはこのOIDの文字列を読み出して、初期化時に図2.2のようにMIBのツリーをメモリ上に作成します。また、MIB-IIのツリーの中にはTCPやUDPのソケットに関連したオブジェクトがあります。これらのオブジェクトはユーザーがアプリケーションでTCP/UDPのソケットを作成／削除すると、ツリーのオブジェクト（ノード）も追加／削除されます。つまり、MIB-IIのツリーは実行中に変更することがあります。

ベンダー固有の拡張（プライベート）MIBも初期化時にMIBのツリーを作成します。ここで初期化時に作成したベンダーのMIBは変更できません。しかし、関数の add_val_mib_nod を使用すると、ベンダーMIBのオブジェクトを追加することができます。さらに、関数の del_val_mib_nod を使用すると追加したベンダーMIBのオブジェクトを削除することができます。

また、このMIBのツリーは一度ツリーを作成してみないと、そのツリーで使用するノードの個数、つまりメモリ使用量がわかりません。よって、本システムの初期化用関数（snmp_ini）はデバッグ用にツリーの作成のために必要なノードの個数（メモリの使用量）を返す仕様になっています。ただし、この必要なノードの個数には関数の add_val_mib_nodで追加するベンダーのMIBのノードは含まれていません。

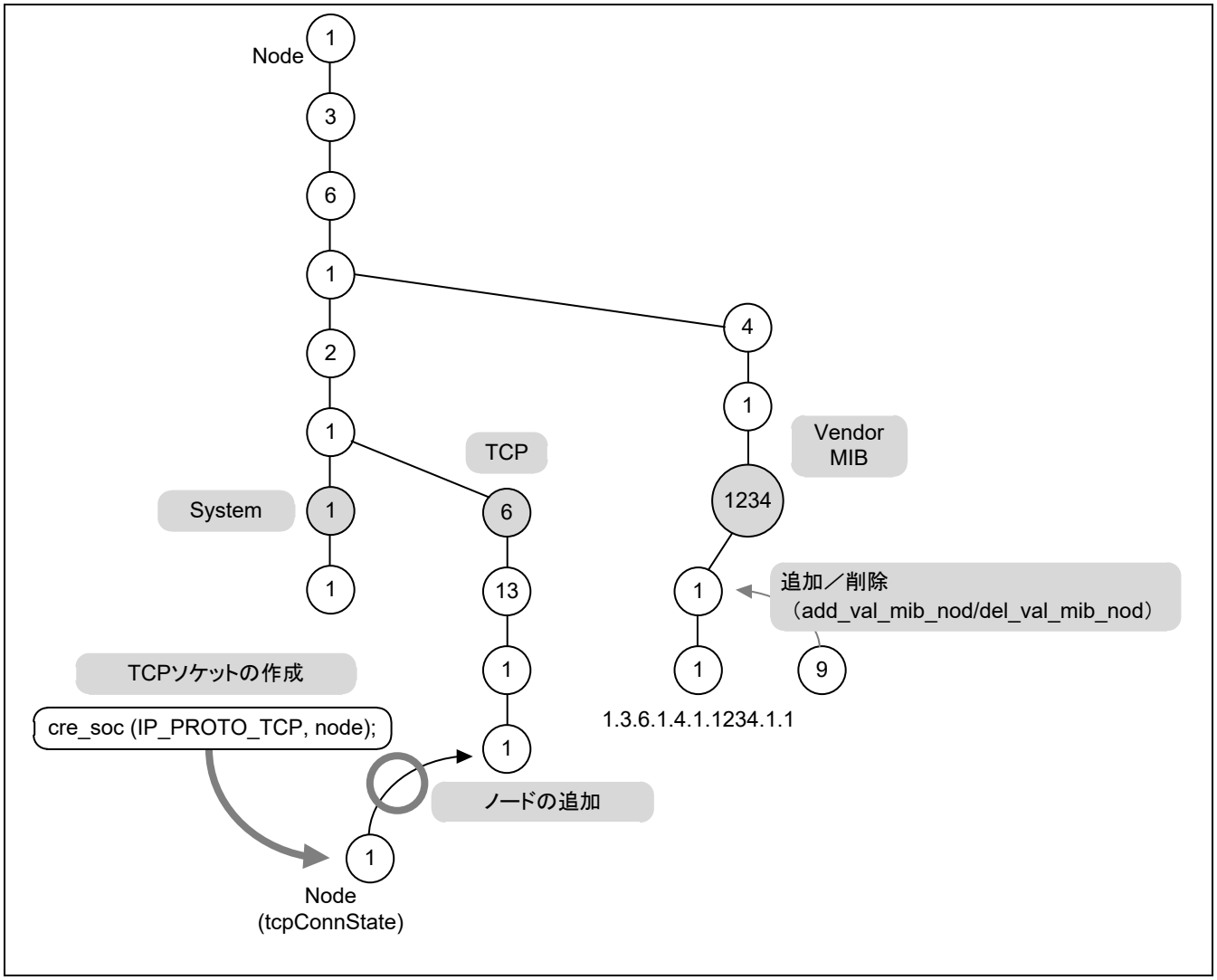
Node
1
3
6
1
2
1
1
1
System
TCP
6
13
1
1
TCPソケットの作成
cre_soc (IP_PROTO_TCP, node);
ノードの追加
Node
(tcpConnState)
4
1
Vendor
MIB
1234
1
1
9
追加／削除
（add_val_mib_nod/del_val_mib_nod）
1.3.6.1.4.1.1234.1.1

図2.2 MIBのツリー

2.5 ベンダー固有MIBとコールバック関数

ベンダー固有のプライベートMIBのオブジェクトの値は二つの方法で変更することができます。変更方法のひとつは①ユーザーのタスクで関数の get_mib_objと set_mib_obj、またはget_val_mib_objと set_val_mib_objを発行し、値の取得と変更を実行する方法です。もうひとつの方法は、次の図のように②マネージャーからGetRequestなどのパケットを受信したときに、本システムの受信用タスクがユーザー定義のコールバック関数を発行するので、そのコールバック関数の中で値を変更する方法です。後者の方法は、たとえばGetRequestを受信した場合は、コールバック関数の引数に現在のオブジェクトの値が格納されているので、ユーザーはその値を別の値に変更してコールバック関数を終了すると、本システムはその変更した値をマネージャーに返します。つまり、ベンダーのプライベートMIBのオブジェクトの値を任意のタイミングで更新したい場合は get_mib_obj/set_mib_obj、または get_val_mib_obj/set_val_mib_objを使用し、マネージャーからリクエストが来たときに更新したい場合はコールバック関数を使用してください。

なお、コールバック関数の中で関数のset_mib_obj/set_val_mib_objなどは発行できません。値の変更はコールバック関数の説明のようにcbk_dat->bufのデータを変更してください。

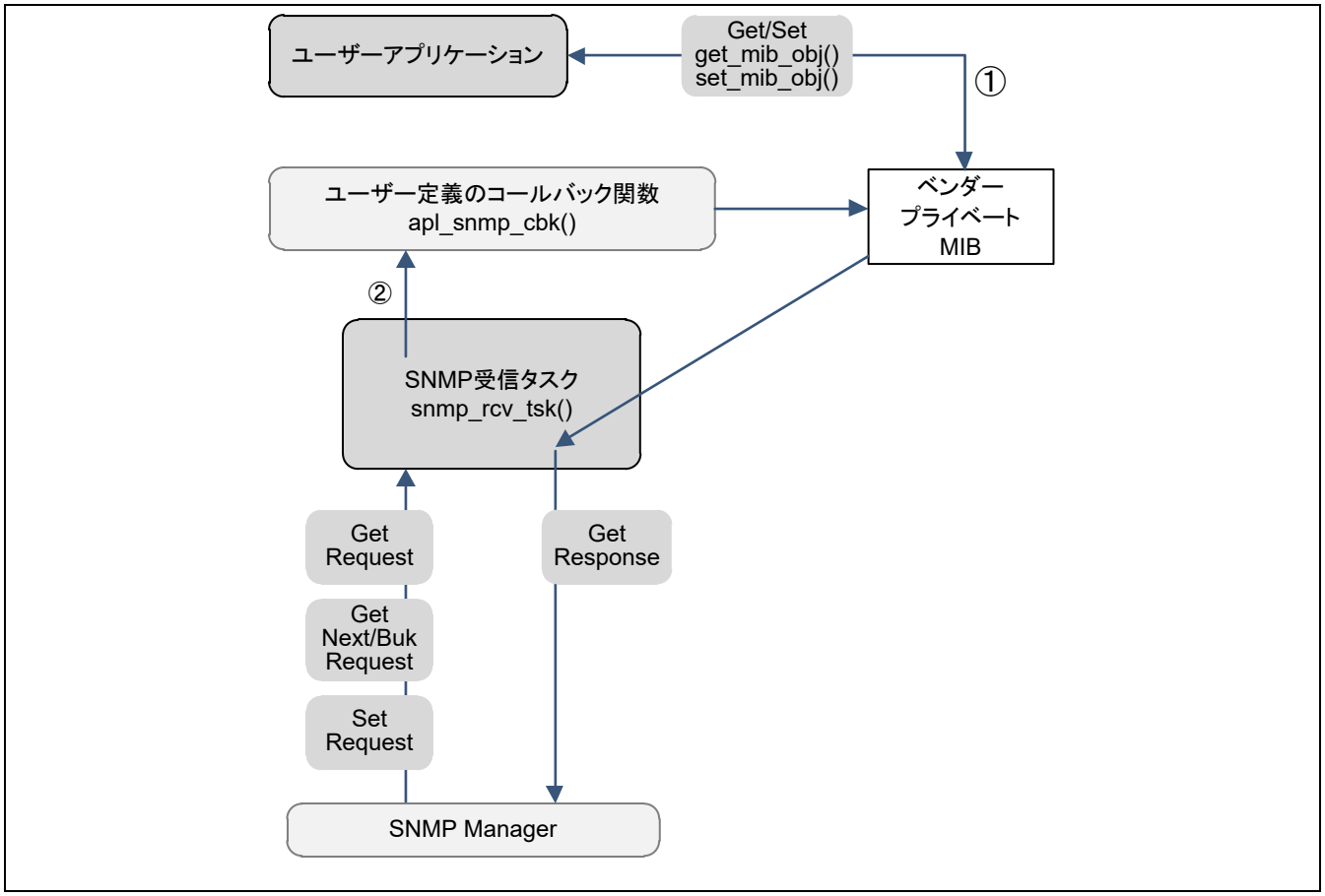

図 2.3 ベンダーMIBのコールバック関数

3. 全体構成

3.1 ファイル構成

本システムのインストーラはフォルダの uC3/Compact/SNMP（Compact版OS用）、またはuC3Std/SNMP（Standard 版 OS用）へファイルをコピーします。本システムのファイルの構成は次のようになります。

表 3.1 ファイル構成

フォルダ	ファイル名	内容
SNMP/doc	uNet3_SNMP.txt	更新履歴
	uNet3_SNMAUsersGuide.pdf	ユーザーズガイド
SNMP/inc（ヘッダファイル）	snmp.h	ユーザー関数
	snmp_ber.h	BER(Basic encoding rules) [ASN.1]
	snmp_def.h	内部定義
	snmp_lib.h	ライブラリ作成用
	snmp_mac.h	設定用マクロ
	snmp_mib.h	MIB-II の ID のマクロ
	snmp_net.h	MIB-II の固定値
SNMP/src（ソースファイル）	snmp.c	ユーザー関数とタスク用関数
	snmp_ber.c	BER のエンコード／デコード処理
	snmp_mib.c	MIB ツリーの処理
	snmp_mib_dat.c	MIB-II のデータ
	snmp_tcp.c	TCP/IP プロトコルスタック用
SNMP/lib/（ライブラリ）	SNMP[プロセッサ名など].*	ライブラリ（snmp_mib_dat.c を除く）
	[プロセッサ名]/SNMP[プロセッサ名など].[プロジェクトの拡張子]	ライブラリのビルド用プロジェクトファイル

本システムのAPIの関数を使用するアプリケーションはソースファイルで snmp.h をインクルードしてください。その他のヘッダファイルは本システムの内部やユーザーの設定用ファイル（snmp_cfg.c、snmp_mib_cfg.c）で使用します。

アプリケーションの作成時にTCP/IPプロトコルスタックに付属するStringライブラリ（Network/NetApp/net_strlib.c）とファイルのsnmp_mib_dat.cをビルドしリンクしてください。本システムのライブラリはファイルのsnmp_mib_dat.cを除くソースファイルのライブラリです。ファイルのsnmp_mib_dat.cには可変のデータ（例えばMIB-IIのSystemの sysDescr（機器の名前やバージョン））を含むのでライブラリから除外しています。よって、ファイルのsnmp_mib_dat.cはユーザーがビルドします。なお、ライブラリのビルドでは OS、TCP/IPプロトコルスタック、さらにTCP/IPプロトコルスタックに付属するStringライブラリ（Network/NetApp/net_strlib.h）のヘッダが必要になります。

また、本システムを使用するためには次の表の設定用のファイルが必要になります。設定用のファイルはサンプルプログラムのフォルダに収録しています。例えば Cortex-A8 (AM335x)用ではフォルダの Sample/EVMAM3358.SNMPに設定用ファイルを収録しています。

表 3.2 設定用ファイル一覧

ファイル名	内容
snmp_cfg.h	SNMP の設定用マクロ
snmp_cfg.c	SNMP の設定用変数
snmp_mib_cfg.h	ベンダーのプライベート MIB の設定用マクロ
snmp_mib_cfg.c	ベンダーのプライベート MIB の設定用変数

3.2 ライブラリ

本システムのライブラリは OS やTCP/IPプロトコルスタックと同じコンパイルオプションでビルドしています。たとえば、Cortex-A8 (AM335x)用の本システムではArmステートとThumbステート、VFPの有無の組み合わせで次の4種類のライブラリを収録しています。ライブラリのファイル名は先頭に“SNMP”が付きます。“SNMP”の後の文字列は OS やプロトコルスタックの文字列と同じです。

【Code Composer Studio版】

Arm/Thumb	エンディアン	VFP	ライブラリ名
Arm	Little	なし	SNMPcortexal.lib
Thumb	Little	なし	SNMPcortexatl.lib
Arm	Little	VFPv3	SNMPcortexafl.lib
Thumb	Little	VFPv3	SNMPcortexaftl.lib

なお、収録しているライブラリはデバッグ情報を含めずにビルドしています。よって、ビルド済みのライブラリを使用した場合、本システムの内部をデバッガでトレースすることはできません。本システムのソースコードをトレースする場合はライブラリをデバッグ情報付きで再度ビルドしてください。

3.3 モジュール構成概要

本システムの主なモジュール構成を説明します。本システムは次のように3個のタスクを使用します。これらのタスクの関数は snmp.c に実装しています。

表3.3 タスク一覧

番号	タスクの関数名	内容
1	snmp_rcv_tsk	SNMP パケットの受信と応答
2	snmp_tim_tsk	稼働時間の集計
3	snmp_trp_tsk	トラップとインフォームの送信

タスクの他にはMIBの情報を格納するメモリ領域が必要です。このMIBにはMIB-IIとベンダー固有のプライベートMIBがあります。また、TCP/IPプロトコルスタックも動作中にMIB-IIに格納するデータを集計し、MIB-IIの情報を更新します。次節では各タスクの動作を説明します。

3.3.1 SNMPパケット受信応答タスク

SNMPのパケットを受信し応答するタスクはUDPのポート161でパケットの受信を待ち（µNet3のrcv_socを発行）、UDPのポート161からマネージャーへ応答を返します（µNet3の snd_socを発行）。本タスクはGetRequestなどのパケットを受信すると、SNMPの仕様に準じ、MIBのデータを参照しながら応答用のパケットを作成します。パケットの作成ではオブジェクトのデータに対して BER（Basic encoding rules）[ASN.1]の仕様に準じてエンコード／デコードを行います。また、受信したパケットのコミュニティ名がユーザーの設定と一致しない場合はトラップ（authenticationFailure）をマネージャーに返す場合もあります（実際にトラップを送信するタスクはトラップ用のタスクです）。さらに、SetRequestのパケットを受信した場合は MIBのオブジェクトの値を更新します。

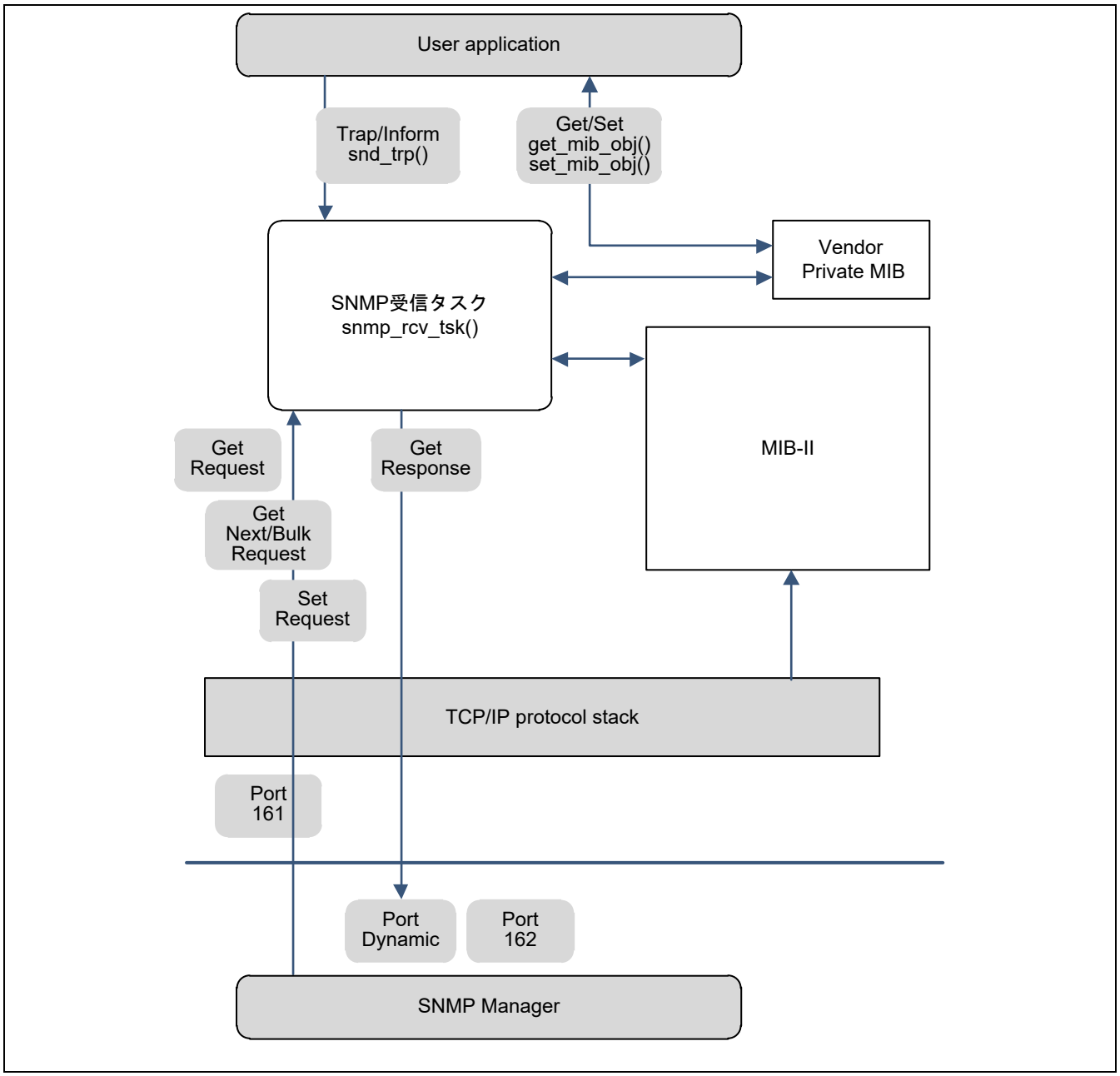

図3.1 パケット受信応答タスクとMIB

3.3.2 稼働時間集計タスク

時間集計のタスクは機器の稼働時間の取得のために使用します。この稼働時間はMIB-IIのSystemのsysUpTime の値になります。稼働時間の取得は OS のサービスコールのget_tim（システム時刻の参照）を使用します。本タスクはget_timが返すシステム時刻の下位32ビットの値がオーバーフローしない間隔で起床します。起床後、get_timで取得したシステム時刻をSNMPの稼働時間に変換してMIB へ保存します。したがって本タスクはごくわずかな時間のみ実行状態になります。なお、本タスクはget_timを使用するので、本システムの動作中はset_timを使用してOSのシステム時刻を変更しないでください。

3.3.3 トラップ用タスク

本タスクはトラップまたはインフォームを送信します。トラップは標準トラップとベンダー固有のトラップがあります。標準トラップは設定用のマクロやAPIの関数で有効・無効に変更できます。

ベンダー固有のトラップを送信する場合はユーザーのアプリケーションのタスクで関数のsnd_trpを発行します。関数のsnd_trpは本タスクにメッセージを送り、本タスクがトラップの送信を完了するまで待ち状態になります。トラップの送信の完了とはμNet3の snd_soc（UDP 送信）でトラップを発行し終わった時です。また、インフォームの場合はインフォームの送信先から応答パケットを受信（rcv_soc）するまで待ち状態になります。なお、トラップとインフォームを使用しない場合、本タスクは不要です。

4. 資源の設定

このシステムが使用するOSとネットワークの資源を説明します。

4.1 OS用資源

4.1.1 資源一覧

このシステムで使用する OS の資源は次のようになります。トラップを使用しない場合、タスク3／セマフォ2／イベントフラグ2／メイルボックス（ID_xxx_TRP）は必要ありません。本システムのコールバック関数はタスク1の関数の snmp_rcv_tskが発行します。よって、コールバック関数の中でスタックを大きく消費する処理を実行する場合は、タスク 1 のスタックサイズを大きくしてください。

使用するOSがCompact 版の場合、ユーザーは表4.1の資源をOSに付属するコンフィギュレータで設定する必要があります。Standard 版の OSでは本システムが自動で生成するのでユーザーの作業は不要です。

表 4.1 OSの資源一覧 (1 / 2)

番号	資源	設定値（デフォルト値／サンプルの設定値）		処理内容
1	タスク 1	ID の定義名	ID_SNMP_TSK_RCV	SNMP パケットの受信と応答
		タスクの関数名	snmp_rcv_tsk	
		優先度の初期値	6	
		拡張情報	なし	
		実行可能状態	なし	
		制約タスク	なし	
		スタックサイズ	1024 以上（1536 など） （ローカルスタック）	
2	タスク 2	ID の定義名	ID_SNMP_TSK_TIM	稼働時間の集計
		タスクの関数名	snmp_tim_tsk	
		優先度の初期値	6	
		拡張情報	なし	
		実行可能状態	なし	
		制約タスク	なし	
		スタックサイズ	512（ローカルスタック）	
3	タスク 3	ID の定義名	ID_SNMP_TSK_TRP	トラップとインフォームの送信
		タスクの関数名	snmp_trp_tsk	
		優先度の初期値	6	
		拡張情報	なし	
		実行可能状態	なし	
		制約タスク	なし	
		スタックサイズ	1024（ローカルスタック）	
4	セマフォ 1	ID の定義名	ID_SNMPA_SEM_MIB	MIB の排他
		資源数の初期値	1	
		最大資源数	1	
		属性	TA_TFIFO	
5	セマフォ 2	ID の定義名	ID_SNMPA_SEM_TRP	トラップ用資源の排他
		資源数の初期値	1	
		最大資源数	1	
		属性	TA_TFIFO	

表 4.1 OSの資源一覧 (2 / 2)

番号	資源	設定値（デフォルト値／サンプルの設定値）		処理内容
6	イベントフラグ 1	ID の定義名	ID_SNMP_FLG_STS	タスクの状態
		初期値	0x0	
		タスクの待ち行列	TA_TFIFO	
		複数タスクの待ちの許可	TA_WMUL	
		クリア	なし	
7	イベントフラグ 2	ID の定義名	ID_SNMP_FLG_TRP	トラップの処理状態
		初期値	0x0	
		タスクの待ち行列	TA_TFIFO	
		複数タスクの待ちの許可	TA_WMUL	
		クリア	なし	
8	メイルボックス	ID の定義名	ID_SNMP_MBX_TRP	トラップの命令ブロックの送信
		タスクの待ち行列	TA_TFIFO	
		メッセージのキュー	TA_MFIFO	

4.1.2 資源の設定

OSの資源の設定のために決められた変数を実装する必要があります。設定用の変数はサンプルプログラムのファイルの snmp_cfg.cにすでに実装してあります。

使用するOSがCompact版の場合は、次のように構造体のT_SNMP_CFG_OSの変数を“snmp_cfg_os”という名前で宣言します。そして、コンフィギュレータで設定した OS の各資源のIDを代入して変数を初期化します。本システムはこの変数（ROM領域に配置）の値を読み出して使用します。

```
const T_SNMP_CFG_OS snmp_cfg_os = {
    ID_SNMP_TSK_RCV,        /* タスク１ */
    ID_SNMP_TSK_TIM,        /* タスク２ */
    ID_SNMP_TSK_TRP,        /* タスク３ */
    ID_SNMP_SEM_MIB,        /* セマフォ１ */
    ID_SNMP_SEM_TRP,        /* セマフォ２ */
    ID_SNMP_FLG_STS,        /* イベントフラグ1 */
    ID_SNMP_FLG_TRP,        /* イベントフラグ2 */
    ID_SNMP_MBX_TRP,        /* メイルボックス */
};
```

使用するOSがStandard版（Cortex-A8(AM335x)用など）の場合は、次のようにタスクなどの生成情報を代入した変数がsnmp_cfg.cに実装されています。本システムはこの変数を使用してOSの資源を自動で生成します。

```
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};
（略）
```

なお、タスクの優先度やスタックサイズは使用するOSがCompact 版の場合はコンフィギュレータで設定してください。Standard版の場合は次章で説明するsnmp_cfg.hのマクロで設定します。

4.2 ネットワーク用資源

4.2.1 UDPソケット

このシステムはネットワークでUDPのソケットを使用します。UDPソケットの設定は次のようになります。GetRequestなどのメッセージを送受信するUDPソケット1とトラップ／インフォームを送信するUDPソケット2の2つがあります。このUDPソケットは使用するLANのポートごとに必要になります。

OSがCompact 版の場合はコンフィギュレータで表4.2のソケットを設定してください。Standard 版のOSでは本システムが内部で生成するのでユーザーの作業は不要です。

表 4.2 ネットワークの資源

1	UDP ソケット 1	ID の定義名	ID_SNMP_UDP_SOC1 ～	UDP ソケット（メッセージ送受信用）
		インタフェースのバンディング	Ethernet0（使用するネットワークデバイス）	
		IP バージョン	IPv4	
		プロトコル	UDP	
		ローカルポート	161	
		snd_soc のタイムアウト	2000	
		rcv_soc のタイムアウト	2000	
2	UDP ソケット 2	ID の定義名	ID_SNMP_UDP_TRP_SOC1 ～	UDP ソケット（トラップ送信用）
		インタフェースのバンディング	Ethernet0（使用するネットワークデバイス）	
		IP バージョン	IPv4	
		プロトコル	UDP	
		ローカルポート	PORT_ANY(0)（任意）	
		snd_soc のタイムアウト	2000	
		rcv_soc のタイムアウト	2000	

4.2.2 UDPソケットの設定

このシステムはSNMPで使用するネットワークのデバイス（LANのポート）を制限することができます。たとえば、4個のLANポートを持つターゲット機器で、1のポートではSNMPのメッセージとトラップ、2と4のポートではトラップのみ、3のポートはSNMPを使用しないという設定が可能です。

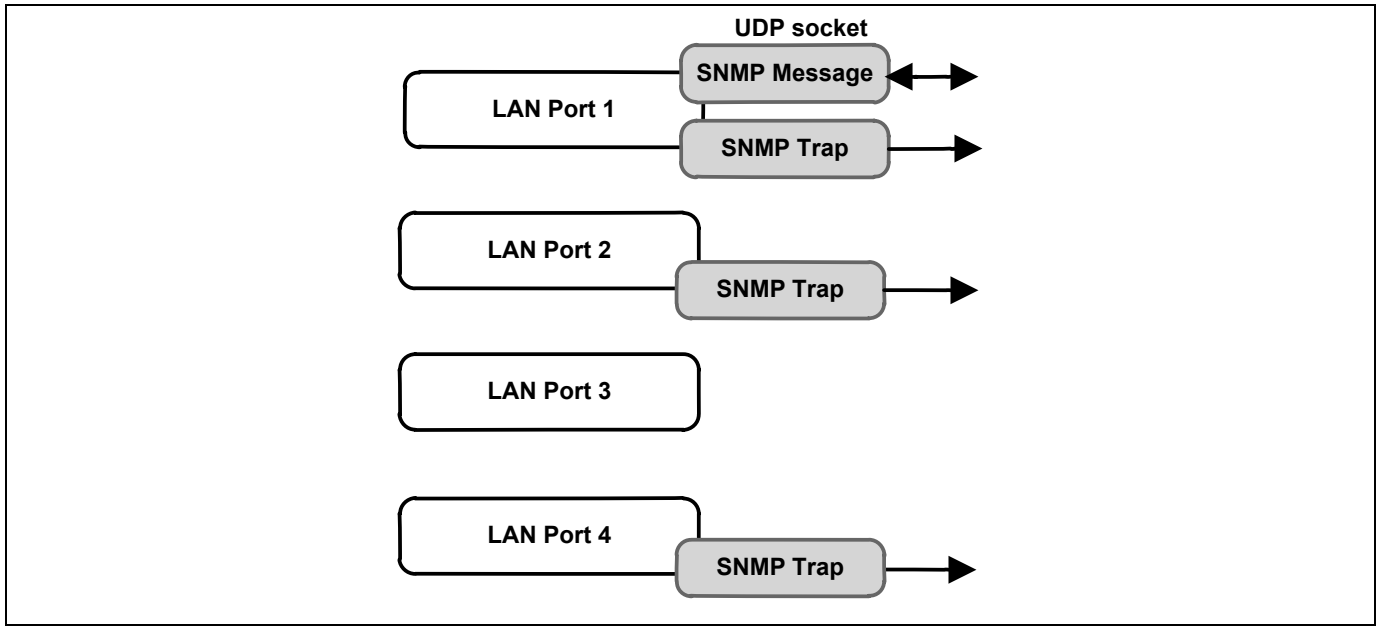

図4.1 SNMPで使用するデバイスの例

この使用するネットワークのデバイスの設定は構造体のT_SNMP_CFG_NETの配列変数を snmp_cfg_netという名前で宣言してください。この変数はファイルの snmp_cfg.cに実装してあります。この変数は constを付けて宣言し、ROM領域に配置してください。構造体は次のようになります。

```
/* Network configuration */
typedef struct t_snmp_cfg_net {
    UH dev_num;          /* Network device number (1...65535) */
    ID id_udp_soc;       /* UDP socket          (ID_SNMP_UDP_SOCx) */
    ID id_trp_soc;       /* UDP socket for trap   (ID_SNMP_UDP_TRP_SOCx) */
} T_SNMP_CFG_NET;
```

番号	型	変数名	説明
1	UH	dev_num	ネットワークデバイスの番号（1～）
2	ID	id_udp_soc	Compact 版の場合 使用する場合は UDP ソケット 1 の ID 使用しない場合は 0 Standard 版の場合 使用する場合は 1 使用しない場合は 0
3	ID	id_trp_soc	Compact 版の場合 使用する場合は UDP ソケット 2 の ID 使用しない場合は 0 Standard 版の場合 使用する場合は 1 使用しない場合は 0

図4.1のように設定する場合の実装例は次のようになります。Compact 版で使用するソケットのIDはコンフィギュレータで表4.2のように作成してください。

```
#define CFG_SNMP_NET_USE_CNT                                    3

/* Compact版 */
const T_SNMP_CFG_NET snmp_cfg_net[CFG_SNMP_NET_USE_CNT] = {
    {1U, ID_SNMP_UDP_SOC1,          ID_SNMP_UDP_TRP_SOC1},
    {2U, 0,                         ID_SNMP_UDP_TRP_SOC2},
    {4U, 0,                         ID_SNMP_UDP_TRP_SOC3}
};

/* Standard OS */
const T_SNMP_CFG_NET snmp_cfg_net[CFG_SNMP_NET_USE_CNT] = {
    {1U, 1, 1},
    {2U, 0, 1},
    {4U, 0, 1}
};
```

5. SNMPの設定

本システムの設定は設定ファイルのsnmp_cfg.hとsnmp_cfg.cに実装します。「5.1 基本設定」で説明している識別子（マクロ）はsnmp_cfg.hに実装してください。それ以降の節で説明している設定用変数はsnmp_cfg.c に実装してください。

5.1 基本設定

ファイルのsnmp_cfg.hに定義するマクロの一覧は次のようになります。OSとネットワークの設定は使用するOSがCompact版の場合は使用しません。Compact版の場合はコンフィギュレータでタスクの優先度とスタックサイズを設定してください。

表 5.1 設定用マクロ一覧 (1 / 2)

種類	マクロ定義	設定例	説明
SNMP の設定	CFG_SNMP_NET_DEV_CNT	1	ターゲット機器で使用するネットワークデバイス（LAN のポート）の数。1 ～16 が設定可能
	CFG_SNMP_NET_DEV_NUM	1	SNMP で使用する（メッセージの送受信を行う）ネットワークデバイス（LAN のポート）の数。 1～16 が設定可能
	CFG_SNMP_MAX_SOC_CNT	CFG_NET_SOC_MAX	net_cfg.h で設定した µNet3 で作成する TCP と UDP 用ソケットの最大個数
	CFG_SNMP_MAX_TCP_CNT	CFG_NET_TCP_MAX	net_cfg.h で設定した µNet3 で作成する TCP 用ソケットの最大個数
	CFG_SNMP_MAX_ARP_CNT	CFG_NET_ARP_MAX	net_cfg.h で設定した µNet3 で使用する ARP テーブルのエントリ数
	CFG_SNMP_MAX_TRP_CNT	12	システムの内部で同時に送信するトラップ／受信するインフォーム応答の最大個数（0 の場合は標準トラップとインフォームを使用しない）
	CFG_SNMP_VEN_TRP_CNT	4	関数の snd_trp で同時に送信するトラップ／インフォームの数 0～32 が設定可能。（0 の場合は関数の snd_trp を使用しない）
	CFG_SNMP_MSG_VAR_CNT	32	SNMP のパケットに追加する Variable-bindings のデータの最大個数（4 以上の整数）
	CFG_SNMP_MIB_NOD_CNT	800	MIB のツリーのノードの最大個数
	CFG_SNMP_MAX_MIB_DEP	32	MIB のツリーのノードの最大の深さ（オブジェクト ID のドット区切りの数字の最大個数）
	CFG_SNMP_MIB_DAT_LEN	(64 + 1)	MIB のオブジェクトのデータの最大サイズ（バイト）（終端の NULL 文字を含む）
	CFG_SNMP_GEN_TRP_ENA	TRP_ALL_BIT	本システム初期化時の標準トラップの有効・無効の指定
	CFG_SNMP_MAX_OID_DEP	10	OID 型 MIB オブジェクトの最大階層（ドット数）
MIB-II の設定	CFG_SNMP_MIB2_IF_ENA	1	Interfaces の有効 (1) ／無効(0)
	CFG_SNMP_MIB2_AT_ENA	1	Address Translation の有効 (1) ／無効(0)
	CFG_SNMP_MIB2_IP_ENA	1	IP の有効 (1) ／無効(0)
	CFG_SNMP_MIB2_ICMP_ENA	1	ICMP の有効 (1) ／無効(0)
	CFG_SNMP_MIB2_TCP_ENA	1	TCP の有効(1)／無効 (0)
	CFG_SNMP_MIB2_UDP_ENA	1	UDP の有効 (1) ／無効 (0)
	CFG_SNMP_MIB2_SNMP_ENA	1	SNMP の有効 (1) ／無効(0)

表 5.1 設定用マクロ一覧 (2 / 2)

種類	マクロ定義	設定例	説明	
OS とネットワークの設定	TSK_RCV_PRI	6	優先度	タスク 1 SNMP パケットの受信
	TSK_TIM_PRI	6		タスク 2 稼働 時間の集計
	TSK_TRP_PRI	6		タスク 3 トラップの送信
	TSK_RCV_STK	1024	スタックサイズ （バイト）	タスク 1 SNMP パケットの受信
	TSK_TIM_STK	512		タスク 2 稼働時間の集計
	TSK_TRP_STK	1024		タスク 3 トラップの送信
	CFG_SNMP_RCV_MSG_LEN	2048	受信可能な SNMP のメッセージの最大サイズ （バイト）	
	CFG_SNMP_SND_MSG_LEN	CFG_SNMP_RCV_MSG_LEN	送信可能な SNMP のメッセージの最大サイズ （バイト）	

5.1.1 SNMPの設定

CFG_SNMP_NET_DEV_CNT

CFG_SNMP_NET_DEV_CNTの値はターゲット機器で使用するネットワークデバイス（LANのポート）の個数を設定します。最大で16個まで設定可能です。たとえば、このマクロを2に設定すると、Interfaceグループとと ipAddrTable（IPグループ）のMIBのオブジェクトはポート2個分の情報を提供することができます。このマクロの値はnet_cfg.hのマクロのCFG_NET_DEV_MAXと同じ値を設定してください。

CFG_SNMP_NET_USE_CNT

CFG_SNMP_NET_USE_CNTの値はSNMPが使用するネットワークデバイス（LANのポート）の個数を設定します。最大で16個まで設定可能です。たとえば、ターゲット機器に2個のLANポートがあっても、SNMPのメッセージ通信は1個のポートのみで行う場合は、このマクロを1に設定してください。

CFG_SNMP_MAX_SOC_CNT

CFG_SNMP_MAX_SOC_CNT値はμNet3で作成するTCPとUDPのソケットの最大個数を設定してください。

CFG_SNMP_MAX_TCP_CNT

CFG_SNMP_MAX_TCP_CNTの値はμNet3で作成するTCPのソケットの最大個数を設定してください。

CFG_SNMP_MAX_ARP_CNT

CFG_SNMP_MAX_ARP_CNT の値はμNet3で使用するARPテーブルのエントリ数を設定してください。

上記 3 つのマクロの値はμNet3の設定ファイル（net_cfg.h）で設定した値と同じ値を設定します。つまり、net_cfg.hのマクロの CFG_NET_SOC_MAX、CFG_NET_TCP_MAX、CFG_NET_ARP_MAX と同じ値を設定してください。

CFG_SNMP_MAX_TRP_CNT

CFG_SNMP_MAX_TRP_CNTの値はシステムの内部で同時に使用するトラップ用資源の最大個数を指定します。このマクロの資源は標準トラップの送信とインフォームの応答で使用されます。標準トラップと関数のsnd_trpでインフォームを使用しない場合はこのマクロを 0 にしてください。この資源の最大個数の見込みは次のように算出します。

x＝snmp_cfg_trpで指定したトラップの送信先の数
y＝snmp_cfg_mgrで指定した、または接続しているマネージャーの数
z＝関数の snd_trpを使用して複数のタスクから同時にインフォームを送信する数

CFG_SNMP_MAX_TRP_CNT = {(x * 2) + y + z} * 2 ～ 4

標準トラップの①cold/warmStartと②linkUpを発行するための資源は同時に使用する可能性があるのでxの2倍分の資源が必要になります。authenticationFailureのトラップをマネージャーに発行するときにy個分の資源が必要になります。snd_trpでインフォームを送信すると、その応答用にz個分の資源が必要になります。ただし、xに関しては異常などでイーサネットのlinkUpとlinkDownが連続して多数回発生した場合に不足する可能性があります。また、インフォームの再送の指定回数によって、zも再送回数分だけ増加する可能性があります。よって、全体を2～4倍して余裕のある個数を設定してください。

CFG_SNMP_VEN_TRP_CNT

CFG_SNMP_VEN_TRP_CNTの値は関数の snd_trpで複数のタスクから同時に送信するトラップ／インフォームの最大個数を指定します。インフォームを送信する場合はこのマクロとCFG_SNMP_MAX_TRP_CNTにも最大個数を追加してください。このマクロは0から32までの値が設定できます。関数のsnd_trpを使用しない場合はこのマクロを0に設定してください。

CFG_SNMP_MSG_VAR_CNT

CFG_SNMP_MSG_VAR_CNTは受信／送信するSNMPのパケット内のVariable-bindingの個数の最大値を指定します。本マクロは4以上の整数を設定してください。受信したSNMPのv1のパケットに最大値を超えた個数のVariable-bindingが含まれていた場合、本システムはパケットの送り先にエラー（tooBig）を返します。

CFG_SNMP_MIB_NOD_CNT

CFG_SNMP_MIB_NOD_CNTはMIBのツリーのノードの数を設定します。MIBのツリーは初期化用関数のsnmp_iniで作成し、その作成後にMIBのツリーのノードの数が判明します。よって、ユーザーは最初にCFG_SNMP_MIB_NOD_CNTに大きな値を代入し、snmp_iniが正常終了することを確認してください。CFG_SNMP_MIB_NOD_CNTの値が小さい場合はsnmp_iniがエラーのE_NOMEMを返します。snmp_iniが正常に終了した場合は、snmp_iniの引数のバッファに、内部で作成したMIBのツリーのノード数が代入されています。よって、最後にユーザーはCFG_SNMP_MIB_NOD_CNTの値をその引数の値（ノード数）に設定しなおしてください。

ただし、関数のsnmp_iniは動作中に追加するベンダー固有の拡張MIBのノード数は返しません。関数のadd_val_mib_nodでMIBを追加するときに必要なノード数はユーザー自身で計算し、マクロのCFG_SNMP_MIB_NOD_CNTに追加してください。

CFG_SNMP_MAX_MIB_DEP

CFG_SNMP_MAX_MIB_DEPはMIB-IIとベンダー固有MIBのツリーの最大の深さを設定してください。例えばユーザーがベンダー固有MIBのOIDを“1.3.6.1.4.1.1234.1.2.3.4.5.6.7”と設定した場合は、数字の数が14個あるので、CFG_SNMP_MAX_MIB_DEPの値は14になります。つまり、OIDのドット(.)で区切られた数字の個数を設定してください。

CFG_SNMP_MIB_DAT_LEN

CFG_SNMP_MIB_DAT_LENはMIBのオブジェクトのデータの最大サイズ（バイト）を指定します。

SNMPの仕様ではデータの最大サイズはデータの型がIntegerの場合は4バイト、データ型がOctet String（文字列）の場合は65535文字（バイト）です。しかし、例えばMIBのSystemのsysDescr（機器のハードウェア、ソフトウェアの名前）などの文字列を64文字に制限すれば、本システムの内部で使用するメモリ量を抑えることができます。よって、本マクロは特に文字列のデータの最大サイズを指定します。なお、本マクロに設定する数値は文字列の終端のNULL文字を含めてください。たとえば、最大の文字数が 64 個の場合は本マクロに65を設定します。なお、オブジェクトの各データの最大値、たとえばMIBのSystemのsysDescrの文字列の最大値はsnmp_mib_cfg.hのCFG_SNMP_MIB_SYS_DESCR_LENで設定します。よって、本マクロの数値はオブジェクトの各データの最大値と同じ、または、より大きい値にする必要があります。もし、本マクロの値が各データの最大値より小さい場合は初期化関数のsnmp_iniがエラー（E_BOVR）を返します。

CFG_SNMP_GEN_TRP_ENA

CFG_SNMP_GEN_TRP_ENAには初期化時に有効にする標準トラップを指定してください。実装例は次のようになります。なお、すべてのトラップを無効にする場合は本マクロに0x00を指定してください。

```
/* 全トラップを有効化 */
#define CFG_SNMP_GEN_TRP_ENA   TRP_ALL_BIT

/* coldStart と linkUp のみを有効化 */
#defineCFG_SNMP_GEN_TRP_ENA    (COLD_STA_BIT|LINK_UP_BIT)
```

指定するマクロは次のようになります。

表5.2 標準トラップの設定用マクロ

番号	マクロ	トラップ名	備考
1	COLD_STA_BIT	coldStart	
2	WARM_STA_BIT	warmStart	
3	LINK_DOWN_BIT	linkDown	
4	LINK_UP_BIT	linkUp	
5	AUTH_FAIL_BIT	authenticationFailure	
6	EPG_LOSS_BIT	egpNeighborLoss	非対応
7	TRP_ALL_BIT	すべてのトラップ	

本システムは最初に関数の snmp_ena（本システムの有効化）を発行したときにcoldStartのトラップを送信します。2度目以降のsnmp_enaの発行ではwarmStartのトラップを発行します。

CFG_SNMP_MAX_OID_DEP

CFG_SNMP_MAX_OID_DEPにはオブジェクト識別子型のベンダーMIBに設定可能なOIDの階層を定義します。オブジェクト識別子型のMIBでは、この値より長いOIDを設定することはできません。つまり設定可能なOIDのドット(.)数の上限値を表します。MIBの値はドットを含む文字列として表現されるため、MIBの値が実際にとり得る最大長は、NULLも含めてCFG_SNMP_MAX_OID_DEP*6文字になります（各ノードの最大値は65536（5桁））。

また文字列型のMIBオブジェクトが格納できるデータのサイズは CFG_SNMP_MIB_DAT_LENで決定しますので、たとえ OIDの要素数がCFG_SNMP_MAX_OID_DEPに満たない場合でも、文字列表現がCFG_SNMP_MIB_DAT_LENを超える場合は格納できません。

5.1.2 MIB-IIの設定

CFG_SNMP_MIB2_***_ENAのマクロはMIB-IIの各グループの有効／無効を設定します。たとえば、CFG_SNMP_MIB2_SNMP_ENAの値を0にするとSNMPのグループを無効化し、その分だけ本システムのメモリ使用量を抑えることができます。ただし、マネージャーからSNMPのMIBの値をリクエストされた場合、本システムはそのMIBが存在しないことを示すエラーを返します。

5.1.3 OSの設定

TSK_***_PRIは本システムのタスクの優先度です。TCP/IPプロトコルスタック内部のタスクの優先度のデフォルト値は4です（DEF_NET_TSK_PRI（net_cfg.h））。本システムのタスクの優先度はTCP/IPプロトコルスタックのタスクより低い優先度（6）を設定します。

TSK_***_STK は本システムのタスクのスタックサイズ（バイト単位）です。TSK_RCV_STKは図2.3の受信タスクのスタックサイズを設定します。受信タスクはユーザーのコールバック関数を発行するので、もしコールバックの中でスタックを大きく消費する処理を実行する場合はTSK_RCV_STKの値を大きくしてください。受信タスク以外のスタックサイズは一般に変更する必要はありません。

CFG_SNMP_RCV_MSG_LENとCFG_SNMP_SND_MSG_LENはSNMPのメッセージの受信と送信の最大サイズをバイト単位で設定します。つまり、本システムは rcv_soc（UDPの受信）でSNMPのメッセージを受け取りますが、そのrcv_socの引数に指定するバッファのサイズが本マクロになります。本システムは必ずメッセージ全体をバッファ格納するので、本マクロより長いSNMPのメッセージは受信できません。長いメッセージを途中まで受信し、全体を受信できなかった場合、本システムはそのメッセージを破棄し、マネージャーへ応答を返しません。また、CFG_SNMP_SND_MSG_LENは送信用のバッファのサイズを設定します。一般にこのマクロは受信用のマクロと同じ値を設定してください。

5.1.4 実装例

基本設定の実装例は次のようになります。

```
/* ネットワークデバイス（LANポート）の個数 */
#define CFG_SNMP_NET_DEV_CNT        2           /* Number of network devices */
/* SNMPで使用するネットワークデバイスの個数 */
#define CFG_SNMP_NET_USE_CNT        1           /* Number of network devices for SNMP */
/* ネットワークのソケットとTCPのソケットの最大個数（net_cfg.hと同じ値）*/
#include "net_cfg.h"
#define CFG_SNMP_MAX_SOC_CNT        CFG_NET_SOC_MAX
#define CFG_SNMP_MAX_TCP_CNT        CFG_NET_TCP_MAX
#define CFG_SNMP_MAX_ARP_CNT        CFG_NET_ARP_MAX

/* 同時に送信するトラップ／インフォームの最大個数（0の場合はトラップを使用しない）*/
#define CFG_SNMP_MAX_TRP_CNT        12          /* Number of traps at any time (0 or 1...32) */
/* SNMPのパケットに追加するvariable bindingsデータの最大個数 */
#define CFG_SNMP_MSG_VAR_CNT        32          /* Maximum number of variable bindings */
/* MIBのツリーのノードの最大個数 */
#define CFG_SNMP_MIB_NOD_CNT        680         /* Number of nodes in the MIB tree */
/* MIBのツリーのノードの最大の深さ（オブジェクトIDの数字の最大個数）*/
#define CFG_SNMP_MAX_MIB_DEP        32          /* Maximum depth of the MIB tree */
/* MIBのオブジェクトのデータの最大サイズ
   snmp_mib_cfg.cのDESCR_LENなどで設定したOctet String（文字列）のデータの最大長（末端のNULL文字を含む）*/
#define CFG_SNMP_MIB_DAT_LEN        (64 + 1)    /* Maximum size of the MIB data */
/* オブジェクト識別子 (OID) 型のMIBオブジェクトの最大の深さ */
#define CFG_SNMP_MAX_OID_DEP        10          /* Maximum number of OID objects as MIB data */
/* Generic trap enabled */
/* 送信する一般トラップの登録 */
/* TRP_ALL_BITはすべてのトラップ（ただし、リンクダウン時はトラップを送信しない）*/
#define CFG_SNMP_GEN_TRP_ENA        TRP_ALL_BIT

/* MIB2 group selector */
/* MIB2のグループの有効(1), 無効(0) */
#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 */
/* SNMPのタスクの優先度（Standard版OSの場合のみ）*/
/* Compact版OSの場合、タスクの優先度はコンフィギュレータで設定 */
#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                 1024            /* Trap task (byte) */

/* Maximum size of an SNMP message (4-byte aligned) */
/* 受信, 送信可能なSNMPメッセージの最大サイズ */
#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 マネージャー設定

マネージャーの設定方法を説明します。本システムが受信を許可するSNMPのメッセージの送信元（マネージャー）を設定することが可能です。本設定以外のマネージャーのSNMPのパケットは受信しても破棄します。また、マネージャーを制限しないで、すべてのSNMPのパケットを受信する設定も可能です。

設定は構造体のT_SNMP_CFG_MGRの配列変数を“snmp_cfg_mgr”という名前で宣言してください。構造体の中には次の変数があります。

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

番号	型	変数名	説明
1	T_NODE	nod	受信を許可するマネージャーのネットワークデバイスの番号(1～)と IP アドレス (T_NODE の port は 0、ver は IP_VER4 を設定)

実装例は次のようになります。配列の最後の値は終端用の0を代入してください。

```
static T_NODE snmp_cfg_mgr_nod_1 = {0/*port*/, IP_VER4, 1, 0xc0a8016e};
                                                        /* 0xc0a8016e = 192.168.1.110 */
static T_NODE snmp_cfg_mgr_nod_2 = {0/*port*/, IP_VER4, 1, 0xc0a80165};
                                                        /* 0xc0a80165 = 192.168.1.101 */
static T_NODE snmp_cfg_mgr_nod_3 = {0/*port*/, IP_VER4, 2, 0xac100065};
                                                        /* 0xac100065 = 172.16.0.101 */
T_SNMP_CFG_MGR snmp_cfg_mgr[] = {
  {&snmp_cfg_mgr_nod_1},
  {&snmp_cfg_mgr_nod_2},
  {&snmp_cfg_mgr_nod_3},
  0                                                     /* 0で終端 */
};
```

すべてのマネージャーからのSNMPのパケットを受信する場合は、次のように空の変数を実装してください。

```
/* すべてのSNMPパケットの受信する（マネージャーの制限なし）*/
T_SNMP_CFG_MGR snmp_cfg_mgr[] = {
    0
};
```

5.3 コミュニティ設定

コミュニティの設定方法を説明します。設定は構造体のT_SNMP_CFG_COMの配列変数を“snmp_cfg_com”という名前で宣言してください。構造体の中には次の変数があります。

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

番号	型	変数名	説明
1	VB*	str	コミュニティ名の文字列
2	UB	sts	アクセス状態（モード）STS_RO：読み込みのみ STS_RW：読み書き可能

実装例は次のようになります。複数個のコミュニティを設定することができます。配列の最後の値は終端用の0を代入してください。

```
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 標準トラップ送信先の設定

標準トラップの送信先の設定方法を説明します。設定は構造体のT_SNMP_CFG_TRPの配列変数を“snmp_cfg_trp”という名前で宣言してください。構造体の中には次の変数があります。

```
/* 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;
```

番号	型	変数名	説明
1	VB*	str	送信先のコミュニティ名の文字列
2	T_NODE*	nod	送信先のネットワークデバイスの番号(1～)と IP アドレス （T_NODE の port は 0、ver は IP_VER4 を設定）
3	UB	ver	トラップのバージョン バージョン 1：SNMP_VER_V1 バージョン 2c：SNMP_VER_V2C
4	ID	id	現在のバージョンでは必ず 0 を設定

実装例は次のようになります。配列の最後の値は終端用の0を代入してください。

```
static VB snmp_cfg_trp_com_1[] = "public";
static VB snmp_cfg_trp_com_2[] = "public";
static VB snmp_cfg_trp_com_3[] = "public";
static T_NODE snmp_cfg_trp_nod_1 = {0/*port*/, IP_VER4, 1, 0xc0a8016e};
                                                    /* 0xc0a8016e = 192.168.1.110 */
static T_NODE snmp_cfg_trp_nod_2 = {0/*port*/, IP_VER4, 1, 0xc0a80165};
                                                    /* 0xc0a80165 = 192.168.1.101 */
static T_NODE snmp_cfg_trp_nod_3 = {0/*port*/, IP_VER4, 2, 0xac100065};
                                                    /* 0xac100065 = 172.16.0.101 */

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},
    {snmp_cfg_trp_com_3, &snmp_cfg_trp_nod_3, SNMP_VER_V2C, 0},
    {0, 0, 0, 0}                    /* 終端用 */
};
```

なお、本設定は本システムの内部で送信する標準トラップ（coldStart/linkUpなど）の送信先の設定です。APIの関数の snd_trp を使用してユーザーが送信するベンダーのトラップの送信先は関数の引数で個々に設定します。

5.5 ベンダーMIB用標準コールバック関数の設定

本システムのベンダーMIB用の標準コールバック関数（図2.3）の設定方法を説明します。設定は構造体のT_SNMP_CFG_CBKの配列変数を“snmp_cfg_cbk”という名前で宣言してください。構造体の中には次の変数があります。

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

番号	型／変数名	説明
1	ER (*fnc)(T_SNMP_CFG_CBK_DAT*)	標準コールバック関数のポインタ

実装例は次のようになります。登録できる関数は1個のみです。配列の最後の値は終端用の0を代入してください。

```
extern ER apl_snmp_cbk_0(T_SNMP_CFG_CBK_DAT*);

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

コールバック関数を使用しない場合は次のように空の変数を実装してください。

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

なお、本設定は標準のコールバック関数の設定です。コールバック関数は標準コールバック関数以外にベンダー固有の拡張MIBのオブジェクトごとに複数設定することができます。オブジェクトに対するコールバック関数が設定されていた場合は本設定の標準コールバック関数は発行しません。オブジェクトごとのコールバック関数の設定方法は次章をお読みください。

System Groupのオブジェクトに対するコールバック関数は標準のコールバック関数を発行します。

6. ベンダーMIBの設定

ベンダーに依存するMIBの設方法を説明します。MIB-IIのSystemグループの設定はファイルのsnmp_mib_cfg.hにマクロを定義します。ベンダー固有の拡張MIBの設定はファイルのsnmp_mib_cfg.cに変数を実装します。

6.1 MIB-IIのSystemの設定

MIB-IIのSystemグループの設定方法を説明します。MIB-IIのSystemには sysDescr（機器のハードウェア、ソフトウェアの名前やバージョン）やsysObjectID（ベンダーのオブジェクトID）などがあります。それらの値はファイルのsnmp_mib_cfg.hに下記の表のマクロを定義してください。

表 6.1 Systemの設定用マクロ

種類	マクロ定義	設定例	説明
System の設定	CFG_SNMP_MIB_SYS_DESCR_LEN	(32 + 1)	sysDescr の最大文字数 （終端の NULL 文字を含む）
	CFG_SNMP_MIB_SYS_DESCR	"HW:Ver.1.0.0 SW:Ver.1.0.0"	sysDescr (1.3.6.1.2.1.1.1) 機器のハードウェア、ソフトウェアの名前や
	CFG_SNMP_MIB_SYS_OBJECTID_LEN	(32 + 1)	sysObjectID の最大文字数 （終端の NULL 文字を含む）
	CFG_SNMP_MIB_SYS_OBJECTID	"1.3.6.1.4.1.1234"	sysObjectID (1.3.6.1.2.1.1.2) ベンダーのオブジェクト ID （トラップ(v1)の enterprise フィールドの ID）
	CFG_SNMP_MIB_SYS_CONTACT_LEN	(32 + 1)	sysContact の最大文字数 （終端の NULL 文字を含む）
	CFG_SNMP_MIB_SYS_CONTACT	"Email address"	sysContact (1.3.6.1.2.1.1.4) 機器の管理者の連絡先（メールアドレス）
	CFG_SNMP_MIB_SYS_NAME_LEN	(32 + 1)	sysName の最大文字数 （終端の NULL 文字を含む）
	CFG_SNMP_MIB_SYS_NAME	"System name"	sysName (1.3.6.1.2.1.1.5) 機器のドメインネーム
	CFG_SNMP_MIB_SYS_LOCATION_LEN	(32 + 1)	sysLocation の最大文字数 （終端の NULL 文字を含む）
	CFG_SNMP_MIB_SYS_LOCATION	"First floor"	sysLocation (1.3.6.1.2.1.1.6) 機器の物理的な位置
	CFG_SNMP_MIB_SYS_SERVICES	64	sysServices (1.3.6.1.2.1.1.7) 機器が提供するサービスの値

実装例は次のようになります。

```
/* System sysDescr (1.3.6.1.2.1.1.1) */
/* 機器のハードウェア、ソフトウェアの名前やバージョン */
#define CFG_SNMP_MIB_SYS_DESCR_LEN          (32 + 1)      /* 最大長（末端のNULL文字を含む）*/
#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) */
/* ベンダーのオブジェクトID */
/* MIBのSystemのsysObjectIDとトラップ(v1)のenterpriseフィールド */
#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) */
/* 機器の管理者の連絡先（メールアドレス）*/
#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) */
/* 機器のドメインネーム */
#define CFG_SNMP_MIB_SYS_NAME_LEN           (32 + 1)
#define CFG_SNMP_MIB_SYS_NAME               "Evaluation board"

/* System sysLocation (1.3.6.1.2.1.1.6) */
/* 機器の物理的な位置 */
#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) */
/* 機器が提供するサービスの値 */
#define CFG_SNMP_MIB_SYS_SERVICES           64            /* アプリケーション層 */
```

6.2 ベンダーのプライベートMIBの設定

MIBのprivateサブツリーのenterprises Groupにはベンダー固有の拡張MIBを追加することができます。本節では拡張MIBの設定方法を説明します。本説明で実装する拡張MIBのツリーは次のようになります。このMIBは初期化時（snmp_iniの発行時）に作成されます。作成後に変更することはできません。

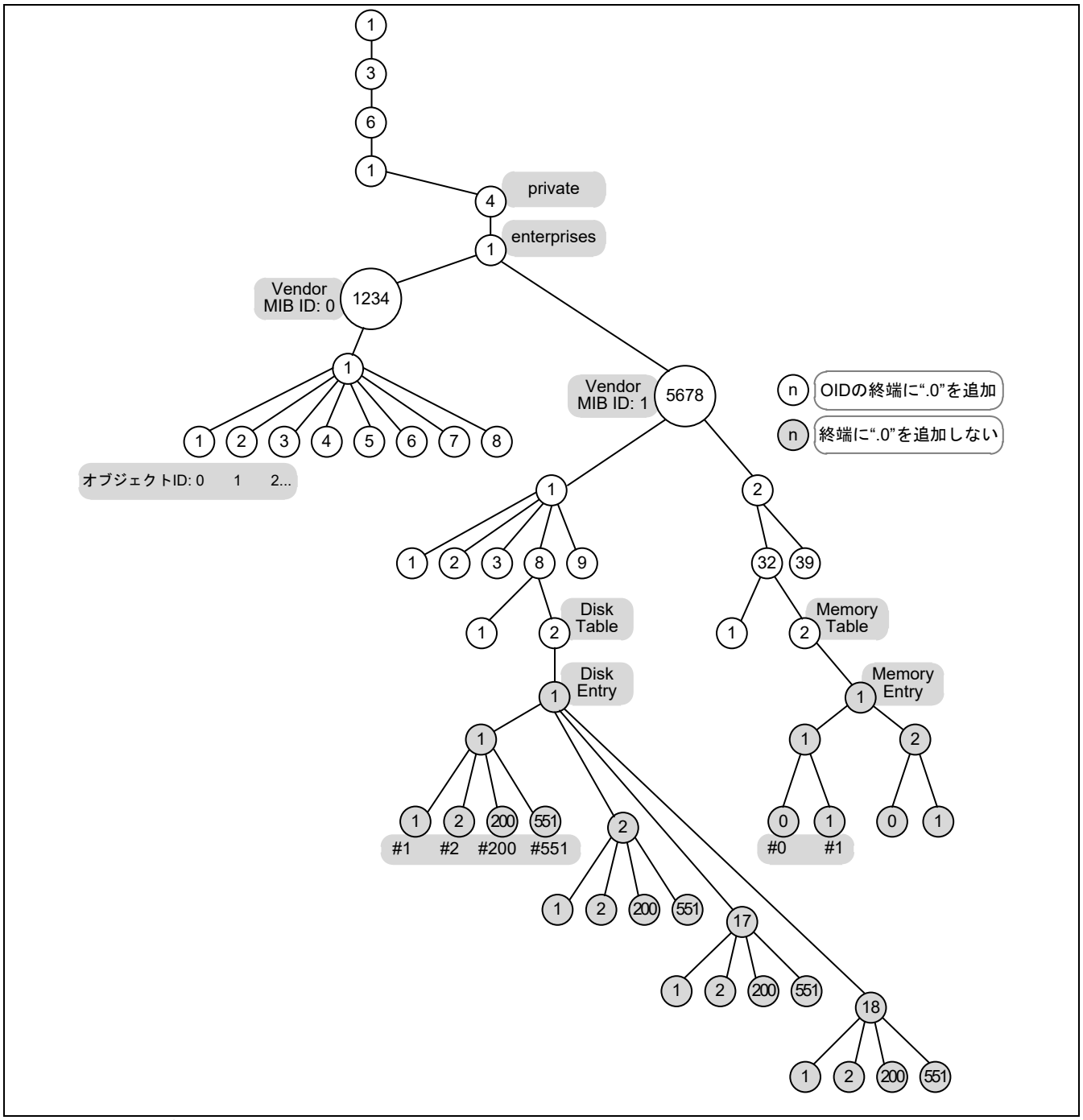

図6.1 ベンダーの拡張MIBの実装例

6.2.1 MIBとオブジェクトのID

MIBのオブジェクトは一般にはドット区切りの文字列（“1.3.6.1.4.1.1234.1.1”など）で表現します。しかし、本システムではID（16ビットの数値／UH型）でオブジェクトを識別します。また、図6.1の拡張MIBの実装例のように、MIBのツリーがOIDの“1.3.6.1.4.1.1234.*”と“1.3.6.1.4.1.5678.*”の二つのグループに分かれている場合、個々のグループに対してMIBのID（8ビットの数値／UB型）が付きます。

拡張MIBの設定では、はじめに①MIBテーブルを配列で宣言し、さらに、そのテーブルの中に②オブジェクトのテーブル、③データのテーブル、④コールバック関数のテーブルを宣言します。ここで、最初の①MIBテーブルの添字（配列の添字）がMIBのID（８ビット）になります。また、②オブジェクトのテーブルの添字（配列の添字）がオブジェクトのIDになります。

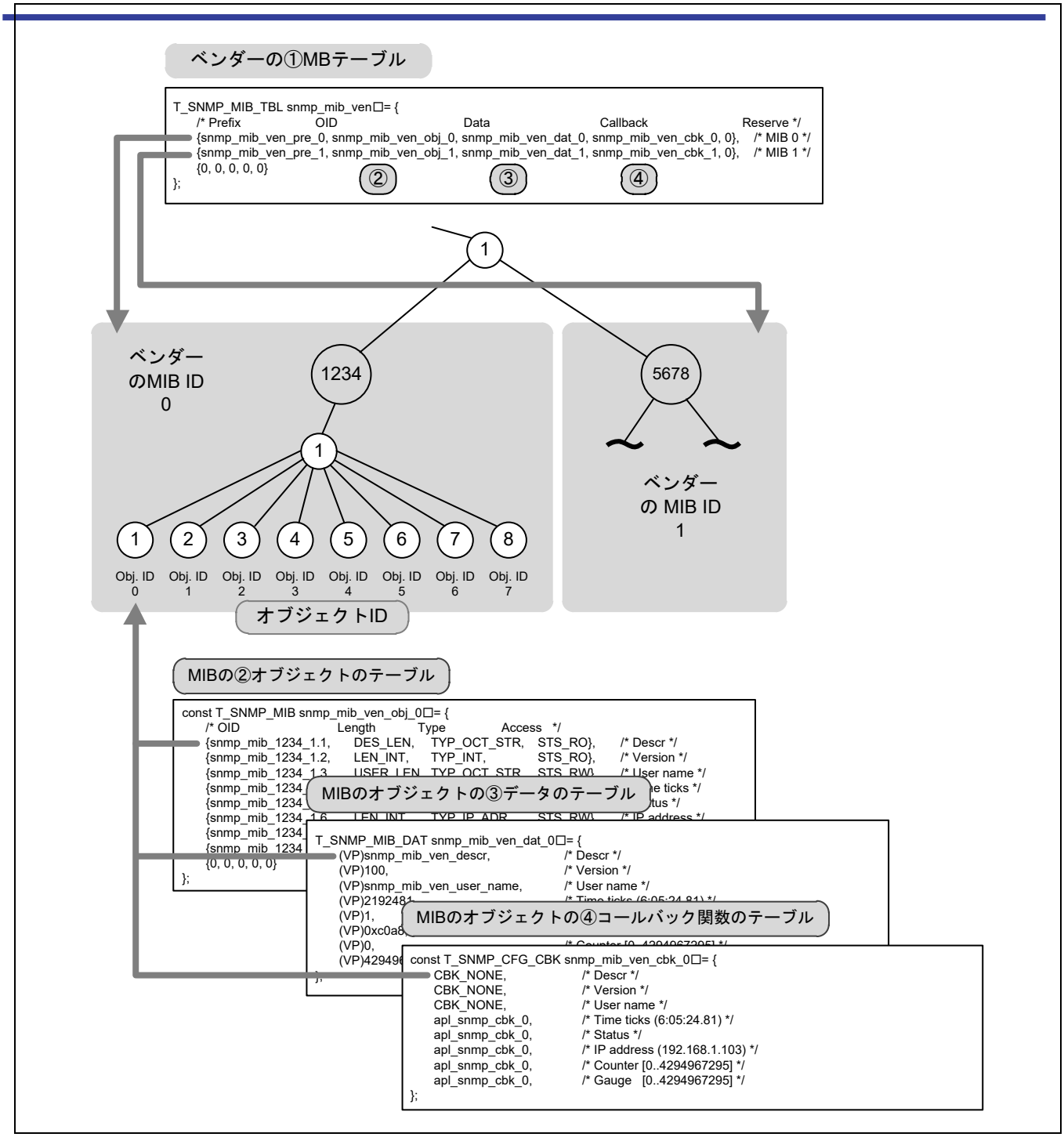

図6.2 ベンダーの拡張MIBの設定

これらのMIBとオブジェクトのIDは本システムのAPIの関数やコールバック関数でオブジェクトを識別するときに使用します。このとき、関数内で使用する変数名はそれぞれ“mib_id”と“obj_id”です。また、MIBのIDは8ビット、オブジェクトのIDは16ビットの制限があります。さらに、MIBテーブルは最大で254 個のグループを登録することができます。また、オブジェクトのテーブルには最大で 65534 個のオブジェクトを登録することができます。

なお、拡張MIBの設定はファイルのsnmp_mib_cfg.cに配列変数を宣言します。ここで、①MIBテーブルと③データ（バッファ）の配列変数は値を変更するのでRAM領域に配置します。その他の配列変数はROM領域に配置します。次節以降は各配列変数の実装方法を説明します。

6.2.2 MIBテーブル

図6.1では拡張MIBが2つのグループに分かれています。OIDが“1.3.6.1.4.1.1234.*”のグループと“1.3.6.1.4.1.5678.*”のグループはそれぞれMIBのIDが0と1になります。拡張MIBの設定では初めにグループごとにオブジェクトの設定用変数のポインタを設定します。この設定は構造体のT_SNMP_MIB_TBLの配列変数を“snmp_mib_ven”という名前で宣言してください。構造体の中には次の変数があります。

```
/* 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;                     /* 予約 */
} T_SNMP_MIB_TBL;
```

番号	型	変数名	説明
1	const VB*	pre	MIB の OID のプリフィックス（文字列）のポインタ
2	const T_SNMP_MIB*	mib	T_SNMP_MIB（MIB の OID、サイズ、型、アクセス制限）の設定変数のポインタ
3	T_SNMP_MIB_DAT*	dat	T_SNMP_MIB_DAT（オブジェクトのデータ）のポインタ
4	T_SNMP_CFG_CBK*	cbk	T_SNMP_CFG_CBK（オブジェクトごとのコールバック関数）の設定変数のポインタ。コールバック関数を設定しない場合は 0x00 を設定
5	UH	cnt	予約変数（本システムが内部で使用します）

実装例は次のようになります。実装例では2個のグループを登録しています。snmp_mib_ven には最大254個までのMIBのグループが登録できます。配列の最後の値は終端用の0を代入してください。

```
/* Vendor MIB table */
/* ベンダー固有のMIBのグループの表（末尾に {0, 0, 0, 0, 0} を追加
）*/ 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}
};
```

次に本構造体の各変数の設定方法を説明します。

6.2.3 OIDのプリフィックス

構造体のT_SNMP_MIB_TBL の pre の設定方法を説明します。preはOIDのプリフィックス（文字列）のポインタを設定します。ここでプリフィックスとはOID（ドット区切りの文字列）の先頭の共通部分です。実装例ではprivateサブツリーの enterprisesグループにOIDが“1.3.6.1.4.1.1234.*”のグループと“1.3.6.1.4.1.5678.*”のグループがあります。よって、次のようにプリフィックスを宣言し、変数の pre に代入します。プリフィックスの文字列の最後にはドット（“.”）を付けてください。

```
/* MIBのOIDのプリフィックス（最後に.を追加）*/
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 オブジェクトのテーブル

次に構造体のT_SNMP_MIB_TBLのmibの設定方法を説明します。mibには構造体のT_SNMP_MIBの変数のポインタを代入します。T_SNMP_MIBは各オブジェクトの設定、つまりプリフィックスに続くOIDの文字列、オブジェクトのデータの最大サイズ、データの型、アクセスを設定します。この構造体の中には次の変数があります。

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

番号	型	変数名	説明
1	const VB*	str	オブジェクトのプリフィックスに続く OID のドット区切りの文字列
2	UH	len	オブジェクトのデータの最大サイズ（バイト）
3	UB	typ	オブジェクトのデータ型
4	UB	acs	オブジェクトのアクセス

構造体のtyp（オブジェクトのデータ型）を設定するマクロは次のようになります。

表6.2　オブジェクトのデータ型

番号	マクロ	サイズ（len）（バイト）	説明	備考
1	TYP_NONE	4	なし	テーブルの Entry のオブジェクト用
2	TYP_INT	4	Integer	32 ビット
3	TYP_OCT_STR	1 ～ CFG_SNMP_MIB_DAT_LEN	Octet String	文字列
4	TYP_SEQ	4	SEQUENCE	Table のオブジェクト用
5	TYP_IP_ADR	4	IP Address	32 ビット（IPv4 用）
6	TYP_CNT	4	Counter	32 ビット
7	TYP_GAUGE	4	Gauge	32 ビット
8	TYP_TIM_TIC	4	Time ticks	32 ビット
9	TYP_OBJ_ID	6 ～ CFG_SNMP_MIB_DAT_LEN	OID の文字列表現	文字列

構造体のacs（アクセス）を設定するマクロは次のようになります。

表6.3　オブジェクトのアクセス

番号	マクロ	説明	備考
1	STS_NO	参照不可	not-accessible Table や Entry のオブジェクトで使用
2	STS_RO	読み込みのみ	read-only
3	STS_WO	書き出しのみ	write-only
4	STS_RW	読み書き可能	read-write

本構造体のstrには前節で設定したプリフィックスに続くOIDの文字列を設定します。プリフィックスが“1.3.6.1.4.1.1234.*”のグループの実装例は次のようになります。プリフィックスに続くOIDの文字列のみを宣言します。対象のオブジェクトがテーブルでは無い場合、OIDの最後に“.0”（インスタンス識別子）を追加してください。

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

/* MIB 0のOID（最後に.0を追加）*/
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) */
…（略）
```

また、プリフィックスが“1.3.6.1.4.1.5678.*”のグループの実装例は次のようになります。このグループの“Disk Table”と“Memory Table”のオブジェクトはテーブルです。DiskにはifIndexが1、2、200、551のオブジェクトがあります。MemoryにはifIndexが0と1のオブジェクトがあります。t_snmp_mib.strの文字列の設定では対象のオブジェクトがテーブルでは無い場合は OID の最後に“.0”を追加しますが、対象のオブジェクトがテーブルの場合はEntryを含めて下位のオブジェクトには“.0”を追加しないでください。図 6.1では網掛けのある丸で示したオブジェクトが“.0”を追加しないオブジェクトになります。また、テーブルを設定する場合は先に（配列の若い方に）Entry を代入し、続いてEntryの下位のオブジェクトを代入してください。

```
/* MIB 1のOID（最後に.0を追加）*/
/* ただし、テーブルの場合はEntryを含む下位のOIDの最後は.0を追加しない*/
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)*/
                                                       /* テーブルのEntry開始（.0を追加しない）*/
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 #200 ifIndex (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 #551 ifIndex (1.3.6.1.4.1.5678.1.8.2.1.1.551)*/
…（略）
```

本構造体のlenにはオブジェクトのデータの最大サイズをバイト単位で設定します。ただし、lenには表6.2のようにオブジェクトのデータ型がTYP_OCT_STR, TYP_OBJ_ID（文字列）以外の場合は4を設定してください。データ型がTYP_OCT_STR.TYP_OBJ_IDの場合は末端のNULL文字を含めて文字列の最大サイズを設定してください。たとえば、lenが（32 + 1）の場合はマネージャーからSetRequestで設定できる文字列の最大長は32文字になります。なお、lenは「5.1　基本設定」で説明したマクロのCFG_SNMP_MIB_DAT_LEN（オブジェクトのデータの最大サイズ）より大きなサイズを設定することはできません。

構造体のT_SNMP_MIBの変数の実装例は次のようになります。配列の最後の値は終端用の0を代入してください。なお、この実装例の配列の添字（0～7）がオブジェクトのID（obj_id）になります。

```
#define LEN_INT         4               /* INT,CNT,GAUGE,IP_ADR型のデータの長さ */

/* Vendor Descr */
#define DESCR_LEN       (16 + 1)        /* 文字列の最大長（末端のNULL文字を含む）*/
/* User name */
#define USER_LEN        (32 + 1)

/* Obect identifier */
#define OIDSTR_LEN      (60)            /* OIDを文字列表現した場合の最大長( ドット (.) と末端のNULL文字を含む)*/

    /* MIB 0 */
    /* ベンダー固有のMIBの設定（末尾に{0,  0,  0,  0}を追加）*/
    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 */
            {snmp_mib_1234_1_9,     OIDSTR_LEN,     TYP_OBJ_ID,     STS_RO},    /* Identifier*/
            {0, 0, 0, 0}
    };
```

6.2.5 データのテーブル

次に構造体のT_SNMP_MIB_TBLのdatの設定方法を説明します。datにはオブジェクトのデータのバッファ（初期値付き）を設定します。T_SNMP_MIB_DAT型の配列変数がオブジェクトのデータの配列になります。T_SNMP_MIB_DATは次のように型のVP(void*)を新しい名前に変換しています。

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

実装例は次のようになります。この実装例のようにデータの型がTYP_OCT_STR やTYP_OBJ_ID（文字列）の場合はその文字列のバッファの先頭のポインタをVP型に変換してからT_SNMP_MIB_DAT型の配列に代入してください。文字列のバッファはT_SNMP_MIBのlenで指定したデータの最大サイズ分の領域が必要です。データ型が文字列以外、つまりTYP_INTのようにデータのサイズが4バイトの場合はデータの初期値をVP型に変換して配列に代入してください。また、アクセスがread-only以外の文字列のバッファとT_SNMP_MIB_DAT型の配列変数はRAM 領域に宣言してください。これらの変数は各オブジェクトのデータのバッファです。このバッファの内容は動作中に書き換えることがあります。なお、この配列変数の最後の値は0で終端する必要はありません。

```
/* 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"
};

/* Obect identifier */
#define OIDSTR_LEN (60)
static VB snmp_mib_ven_obj_id[OIDSTR_LEN] = {
    "1.22.333.4444.55555.6.7.8.9.10"
};
/* MIB 0 data */
/* ベンダー固有のMIBのデータバッファ */
T_SNMP_MIB_DAT snmp_mib_ven_dat_0[] = {
    (VP)snmp_mib_ven_descr,          /* Descr文字列 */
    (VP) snmp_mib_ven_ver,           /* Version */
    (VP)snmp_mib_ven_user_name,      /* User name文字列 */
    (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] */
    (VP)snmp_mib_ven_obj_id          /* Identifier */
};
```

6.2.6　コールバック関数のテーブル

次に構造体のT_SNMP_MIB_TBLの cbk、つまり各オブジェクトのコールバック関数の設定方法を説明します。「2.5　ベンダー固有MIBとコールバック関数」の説明のように、本システムはベンダー固有の拡張MIBのオブジェクトに対してマネージャーからリクエストを受け取ると、ユーザー定義のコールバック関数を発行します。「5.5　ベンダーMIB用標準コールバック関数の設定」では標準コールバック関数を説明しました。標準のコールバック関数とは別に、個々のオブジェクトに対してコールバック関数を設定することができます。

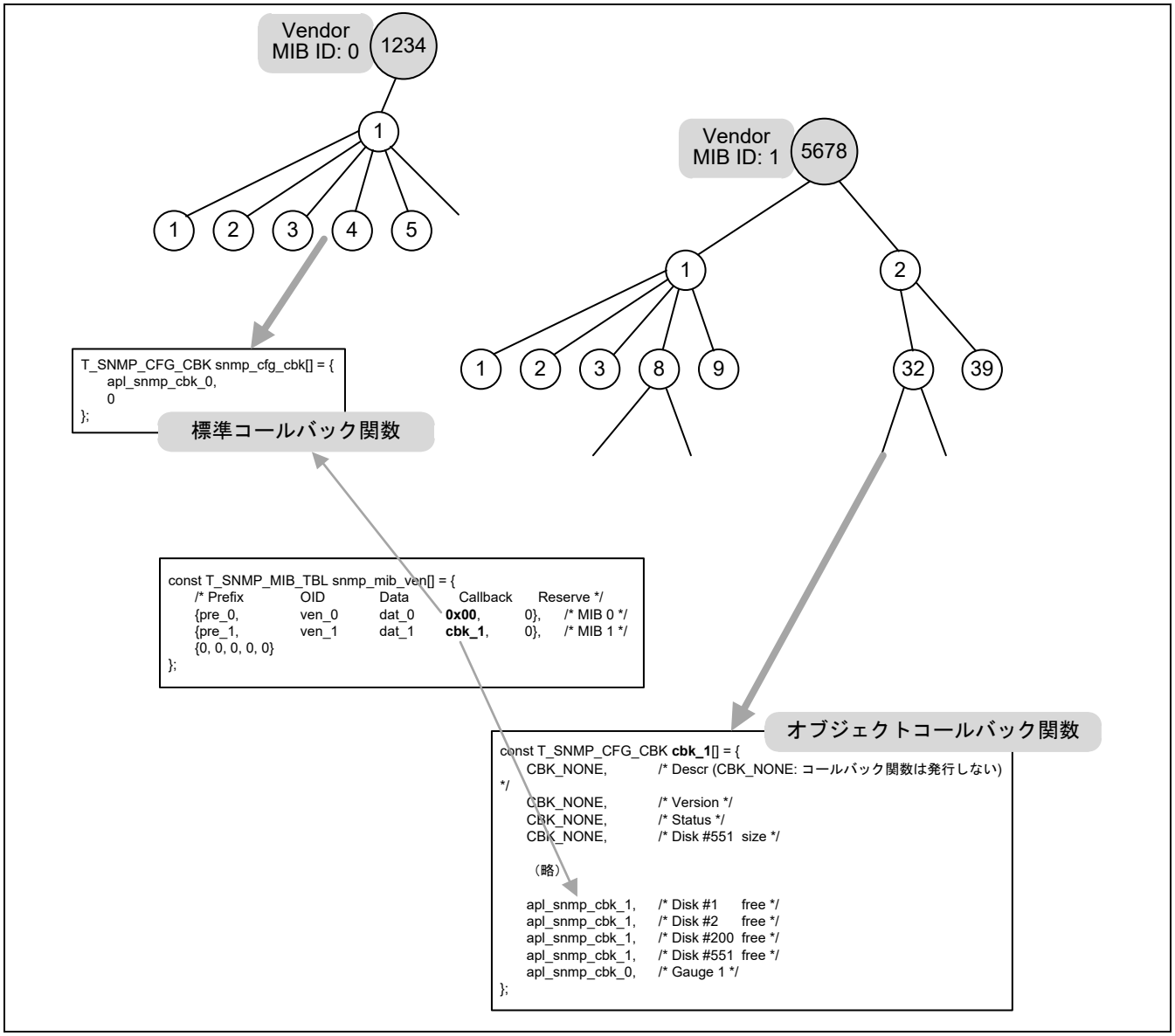

図6.3　標準コールバック関数とオブジェクトのコールバック関数

標準コールバック関数と各オブジェクトのコールバック関数の切り替えについて説明します。実装例では“1.3.6.1.4.1.1234.*”のグループは標準コールバック関数、“1.3.6.1.4.1.5678.*”のグループはオブジェクトごとのコールバックを発行するように設定しています。この切り替えは構造体のT_SNMP_MIB_TBLの cbkの値で判定します（図 6.3）。つまり、cbkが 0x00（NULL）の場合はデフォルトコールバック関数を発行し、0x00以外の場合は cbk に代入されたT_SNMP_CFG_CBKの配列変数のコールバック関数を発行します。

T_SNMP_CFG_CBKの配列変数は次の実装例のように宣言します。また、この配列変数の要素の値が0x00（NULL）の場合はそのオブジェクトに対してコールバック関数を発行しません。このとき、標準コールバック関数も発行しません。なお、この配列変数の最後の値は0で終端する必要はありません。

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

const T_SNMP_CFG_CBK snmp_mib_ven_cbk_1[] = {
    CBK_NONE,               /* Descr        （コールバック関数を全く発行しない）*/
    CBK_NONE,               /* Version      （標準コールバック関数も発行しない）*/
    CBK_NONE,               /* Status */
    （略）
    CBK_NONE,               /* Disk #2      size */
    CBK_NONE,               /* Disk #200    size */
    CBK_NONE,               /* Disk #551    size */
    apl_snmp_cbk_1,         /* Disk #1      free（オブジェクトのコールバック関数はapl_snmp_cbk_1）*/
    apl_snmp_cbk_1,         /* Disk #2      free */
    apl_snmp_cbk_1,         /* Disk #200    free */
    （略）
    apl_snmp_cbk_0,         /* Gauge 2 */
    }
```

6.3 ベンダーの可変プライベートMIBの設定

前節では初期化時に固定で作成するベンダー固有の拡張MIBの設定方法を説明しました。この節では動作中に追加し、さらに削除が可能なベンダー固有のMIBの設定方法を説明します。

6.3.1 可変拡張MIBの無効

ベンダー固有の拡張MIBを動作中に変更する必要がない場合は、次のように配列変数を宣言してください。この場合、関数のadd_val_mib_nod/del_val_mib_nodとget_val_mib_obj/set_val_mib_objは使用できません。

```
T_SNMP_MIB_TBL snmp_mib_ven_val[] = {          /* 追加変更するベンダー MIB無し */
    {0, 0, 0, 0, 0}                            /* 終端マーク */
};
```

6.3.2 可変拡張MIBのMIB テーブル

動作中に追加可能な拡張MIBの設定方法は6.2で説明した固定の拡張 MIB の設定方法と同じです。ただし、次のように設定用の配列変数の変数名が異なります。末尾に“_val”が付きます。

テーブル	固定 MIB	可変 MIB
MIB テーブル	snmp_mib_ven	snmp_mib_ven_val
オブジェクト	snmp_mib_ven_obj	snmp_mib_ven_obj_val
データ	snmp_mib_ven_dat	snmp_mib_ven_dat_val
コールバック関数	snmp_mib_ven_cbk	snmp_mib_ven_cbk_val

MIBテーブルの snmp_mib_ven_valは必ずこの変数名で宣言してください。その他の変数名は別の名前でも構いません。

関数のadd_val_mib_nodやdel_val_mib_nodなどの引数にMIBのID(val_mib_id)とオブジェクトのID(val_obj_id)がありますが、このIDは設定用変数のsnmp_mib_ven_valで設定したデータのIDになります。

この設定用の配列変数の内容は一部を除いてユーザーが直接変更することはできません。変更できる設定用データは図6.4のように、可変拡張MIB用のOIDの文字列とT_SNMP_MIB_DATのデータの値です。これらのデータは関数のadd_val_mib_nodでノードを追加する前であれば、ユーザーが直接変更することができます。ただし、ノードの追加後は関数のdel_val_mib_nodで削除するまで値を変更しないでください。なお、T_SNMP_MIB_DATの配列の要素数は変更できません。

例えば図6.4のように追加するノードの末尾のOIDが不明の場合は“1.3.6.1.4.1.5678.1.8.2.1.1.xxx”のように仮のOIDで設定し、関数のadd_val_mib_nodでノードを追加する直前で“xxx”の部分を“701”に変更することが可能です。

ノードの追加後にT_SNMP_MIB_DATのデータの値を変更する場合は関数の set_val_mib_obj を使用してください。

6.3.3 可変拡張MIBのノード用資源

関数のadd_val_mib_nodを発行してベンダーの拡張MIBを追加するとノードの資源（RAMメモリ）を消費します。ここでノードとは図6.4の丸印です。図6.4ではOIDの“1.3.6.1.4.1.5678.1.8.2.1.1.552”を追加し

ています。このテーブルのオブジェクトの追加で新規に増えたノードは552の可変ノードの1個のみです。その他のノードは初期化時に作成した固定のMIBの固定ノードを使用しています。

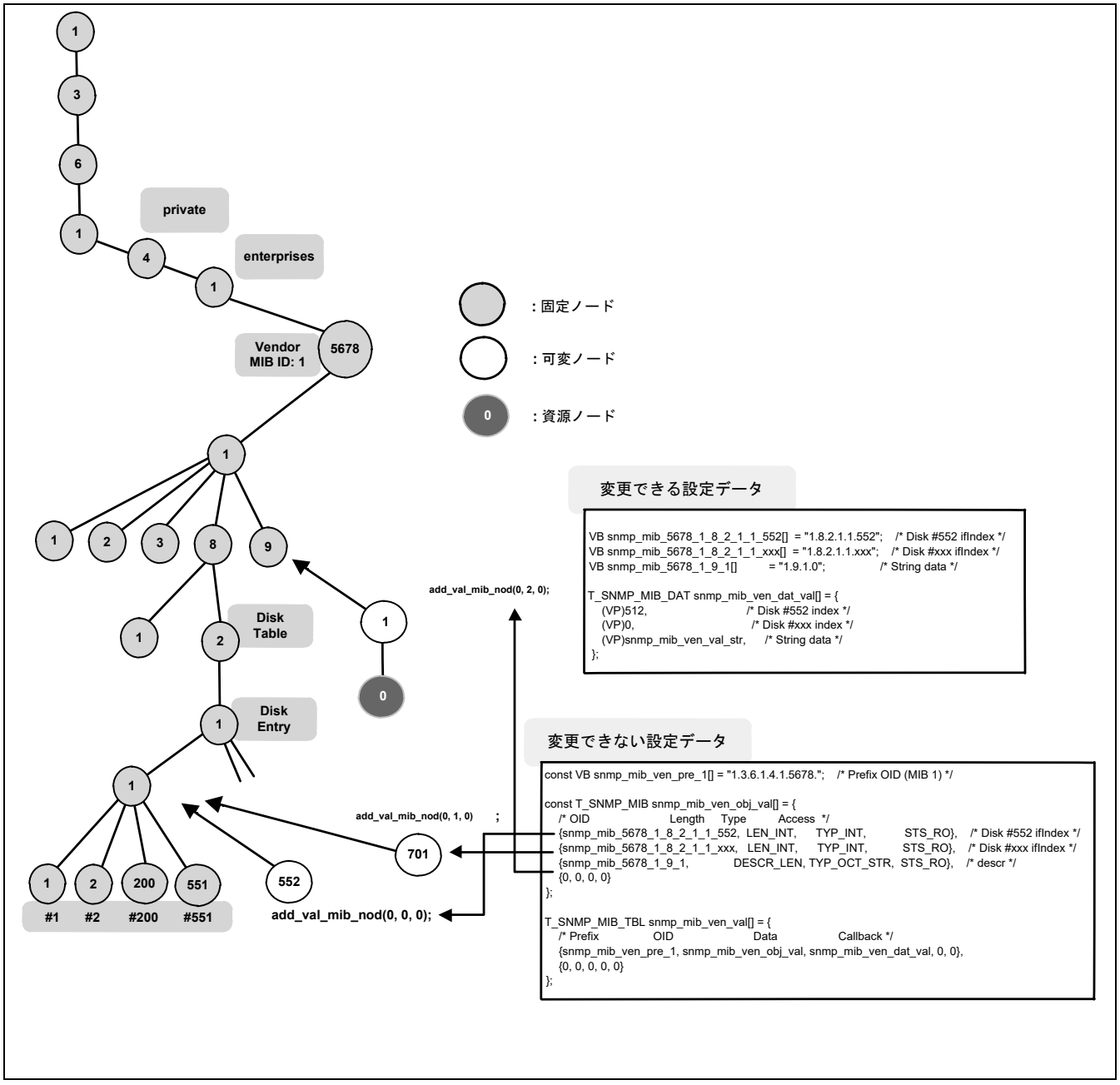

図6.4 拡張MIBオブジェクトの追加

また、次にOIDの“1.3.6.1.4.1.5678.1.9.1.0”を追加しています。この追加で新規に増えたノードは①の可変ノードと末尾0の資源ノードの2個です。このOIDの“1.3.6.1.4.1.5678.1.9.1.0”は末尾が0なのでテーブルのオブジェクトではありません。この場合は末尾の0も資源ノードとして必要になります。一般のMIBのツリー表示ではこの末尾が0のノードは記述されていません。しかし、本システムでは末尾の0もノードとして内部で消費します。

関数のadd_val_mib_nodで必要になる追加のノードの総数はユーザーで設計時に調べておく必要があります。図6.4では4個のノードが追加されます。よって、追加で必要になるノード数(4)を設定用マクロのCFG_SNMP_MIB_NOD_CNTに追加する必要があります。

なお、関数のadd_val_mib_nodは第3引数に追加したノードの個数を代入してユーザーに返します。この値を使用すれば追加のノードの総数を求めることができます。

7. インタフェース

本システムのAPIの関数とコールバック関数の使用方法を説明します。

7.1 関数一覧

本システムが提供する関数の一覧は次のようになります。

種類	関数名	機能
初期化	snmp_ini	初期化
	snmp_ext	終了
	snmp_ena	有効化
	snmp_dis	無効化
管理情報	get_mib_obj	ベンダーオブジェクトのデータの読み込み
	set_mib_obj	ベンダーオブジェクトのデータの書き出し
トラップ	ena_trp	標準トラップの有効化
	dis_trp	標準トラップの無効化
	snd_trp	ベンダートラップの送信
可変のベンダー拡張 MIB	add_val_mib_nod	可変 MIB の追加
	del_val_mib_nod	可変 MIB の削除
	get_val_mib_obj	可変 MIB のオブジェクトのデータの読み込み
	set_val_mib_obj	可変 MIB のオブジェクトのデータの書き込み

ベンダー固有のMIBのオブジェクトのデータを書き換える場合はAPIの関数の（set_mib_obj/set_val_mib_obj）、またはコールバック関数を使用してください。ファイルの snmp_mib_cfg.cで宣言したオブジェクトのデータを直接書き換えると、書き換え途中のデータをマネージャーに返す場合があります。ただし、本システムの初期化前にsnmp_mib_cfg.cのデータを直接書き換えることは可能です。

7.2 関数

本システムが提供する関数の詳細は次のようになります。

snmp_ini（初期化）

【書式】

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

【パラメータ】

UH*	mib_nod_cnt	MIBのツリーのノード数を格納する変数のポインタ

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	設定ファイルの誤り
E_NOMEM	メモリ不足（MIBのツリーのノード数不足）
E_BOVR	オブジェクトのデータの最大サイズが小さい
E_SYS	OSとネットワークの資源の作成が失敗
E_OBJ	その他のエラー

【解説】

本システムを初期化します。本システムを使用する場合は本関数を最初に発行してください。なお、本関数を発行する前にTCP/IPプロトコルスタックを初期化（net_ini）してください。

本関数の内部では、内部変数の初期化、内部バッファの初期化、Compact 版以外のOSではOSの資源の作成、MIBのツリーの作成、さらにネットワークのソケットの作成を行います。MIBのツリーを作成すると、ツリーで使用するノードの数が分かります。その数を引数のmib_nod_cntに代入して返します。ユーザーは、実際に本システムを運用する前に本関数の引数の値を入手し、その値を基本設定のマクロの CFG_SNMP_MIB_NOD_CNT に設定してください。また、この値が不要な場合は引数の mib_nod_cntに 0x00（NULL）を指定してください。

設定ファイル（xxx_cfg.hとxxx_cfg.c）に誤りがある場合、本関数はE_PARのエラーを返します。CFG_SNMP_MIB_NOD_CNTの値が小さくてMIBのツリーが作成できない場合、本関数は E_NOMEMのエラーを返します。CFG_SNMP_MIB_DAT_LENの値が小さい場合、本関数はE_BOVRのエラーを返します。Standard版のOSを使用し、本関数がE_SYSのエラーを返す場合はOSの資源の最大数の設定（一般に関数main()の tskpri_maxなどの値）を見直してください。

snmp_ext (終了)

【書式】

```
ER snmp_ext(void)
```

【パラメータ】

なし

【戻り値】

ER	ercd	正常終了（E_OK）

【エラーコード】

E_OBJ	無効化（snmp_disを発行）していない

【解説】

本システムを終了します。この関数を発行する前に関数のsnmp_disを発行してシステムを無効化してください。Compact版以外のOSでは本関数の内部で作成したOSの資源を解放します。本関数を発行した後に、再度初期化（snmp_ini）と有効化（snmp_ena）を行うと、本システムはcoldStartのトラップを送信します。

snmp_ena（有効化）

【書式】

```
ER snmp_ena(void)
```

【パラメータ】

なし

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_SYS	タスクの起床に失敗
E_OBJ	初期化（snmp_iniを発行）していない、その他のエラー

【解説】

本システムを有効にします。本関数の内部では本システムのタスクを起床します。タスクの実行中はSNMPのパケットを受信します。

本関数を最初に発行した場合はcoldStartのトラップを送信します。2度目以降の発行ではwarmStartのトラップを送信します。ただし、この関数を発行したときにトラップで使用するイーサネットのポートの接続が完了していない場合は、接続の完了後にcoldStartまたはwarmStartのトラップを送信します。

本システムを初期化（snmp_ini）していない状態で本関数を発行した場合、本関数はE_OBJのエラーを返します。

snmp_dis（無効化）

【書式】

```
ER snmp_dis(void)
```

【パラメータ】

なし

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_SYS	タスクの停止に失敗
E_OBJ	有効化（snmp_enaを発行）していない、その他のエラー

【解説】

本システムを無効にします。本関数の内部では本システムのタスクを終了します。本システムのすべてのタスクが終了するまで、本関数の内部で最大で3秒間待ち状態になる場合があります。

本システムを有効化（snmp_ena）していない状態で本関数を発行した場合、本関数はE_OBJのエラーを返します。

get_mib_obj（オブジェクトのデータの読み込み）

【書式】

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

【パラメータ】

VP	buf	データを格納するバッファのポインタ
UH*	len	バッファとデータのサイズ（バイト）
UH	mib_id	MIBのID
UH	obj_id	オブジェクトのID

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	引数の誤り
E_NOSPT	データ型が対応していない
E_BOVR	バッファの長さの不足
E_OBJ	その他のエラー

【解説】

本関数は mib_idとobj_idで指定されたベンダー固有のMIBのオブジェクトのデータを読み出してbufに格納します。lenはバッファ（buf）のサイズ（バイト）を指定してください。バッファ（buf）のサイズが足りない場合はE_BOVRのエラーを返し、さらに必要なバッファのサイズをlenに代入して返します。また、正常にデータを読み出した場合もlenに読み出したデータのサイズを代入して返します。エラーの場合、バッファ（buf）の内容は不定です。

データのデータ型がInteger／Counter32／Gauge32／Time Ticks／IP Address（4バイトの数値）の場合、bufには４バイト以上のバッファ領域（lenが4以上）が必要です。データ型がOctet String／Object ID（文字列）の場合、bufには取得する文字列の文字数（終端のNULL文字は含まない）以上のバッファ領域が必要です。また、データ型がOctet String／Object IDの場合、bufに格納した文字列の最後に NULL 文字（¥0）を追加することはありません。

実装例は次のようになります。

```
#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 ||
        snmp_mib_ven[mib_id].mib[obj_id].typ == TYP_OBJ_ID) {
        /* TYP_OCT_STR (文字列) */
        ((VB*)dat)[len] = '¥0';      /* printfの前にNULL文字を追加 */
        printf((const VB*)dat);
    } else if (snmp_mib_ven[mib_id].mib[obj_id].typ == TYP_IP_ADR) {
        /* TYP_IP_ADR (IPアドレス(4バイト)) */
        ip_ntoa(str, *dat);
        printf(str);
    } else {
        /* TYP_INT, TYP_CNT, TYP_GAUGE, TYP_TIM_TIC (4バイトの数値) */
        printf("0x%x", *dat);
    }
}
```

set_mib_obj（オブジェクトのデータの書き出し）

【書式】

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

【パラメータ】

VP	buf	データを格納したバッファのポインタ
UH	len	データのサイズ（バイト）
UH	mib_id	MIBのID
UH	obj_id	オブジェクトのID

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	引数の誤り
E_NOSPT	データ型が対応していない
E_OBJ	データのサイズの過不足、その他のエラー

【解説】

本関数はmib_idとobj_idで指定したベンダー固有のMIBのオブジェクトのデータを書き出します。書き出すデータはbufに格納してください。lenにはbufに格納したデータのサイズ（バイト）を指定してください。データのサイズが足りない、またはデータのサイズが大きすぎる場合はE_OBJのエラーを返します。

データのデータ型がInteger／Counter32／Gauge32／Time Ticks／IP Address（4バイトの数値）の場合、lenは4を指定してください。データ型がOctet String／Object ID（文字列）の場合、lenは文字列の文字数（終端のNULL文字は含まない）を指定してください。bufに文字列を格納する場合、その文字列の最後にNULL文字（¥0）を追加する必要はありません。

なお、本関数はオブジェクトのアクセスがread-onlyのデータも更新することができます。

実装例は次のようになります。

```
#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 ||
     snmp_mib_ven[mib_id].mib[obj_id].typ == TYP_OBJ_ID) {
    /* TYP_OCT_STR (文字列) */
    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 (IPアドレス(4バイト)) */
    dat = 0xC0A80167;
    ercd = set_mib_obj(&dat, 4, mib_id, obj_id);
} else {
    /* TYP_INT, TYP_CNT, TYP_GAUGE, TYP_TIM_TIC (4バイトの数値) */
    dat = 1234;
    ercd = set_mib_obj(&dat, 4, mib_id, obj_id);
}
```

ena_trp（標準トラップの有効化）

【書式】

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

【パラメータ】

UH	trp_bit	標準トラップのマクロ

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_OBJ	その他のエラー

【解説】

本システムの標準トラップを有効にします。trp_bitには有効にするトラップのマクロを指定してください。マクロは複数指定することができます。

トラップ番号	識別子（マクロ）	値	トラップ名
0	COLD_STA_BIT	0x0001	coldStart
1	WARM_STA_BIT	0x0002	warmStart
2	非対応	—	linkDown
3	LINK_UP_BIT	0x0008	linkUp
4	AUTH_FAIL_BIT	0x0010	authenticationFailure
5	非対応	—	egpNeighborLoss
—	TRP_ALL_BIT	0x003f	すべてのトラップ

実装例は次のようになります。

```
/* coldStartとlinkUpの有効化 */
ercd = ena_trp(COLD_STA_BIT | LINK_UP_BIT);
```

dis_trp（標準トラップの無効化）

【書式】

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

【パラメータ】

UH	trp_bit	標準トラップのマクロ

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_OBJ	その他のエラー

【解説】

本システムの標準トラップを無効にします。trp_bitには無効にするトラップのマクロを指定してください。マクロは複数指定することができます。

トラップ番号	識別子（マクロ）	値	トラップ名
0	COLD_STA_BIT	0x0001	coldStart
1	WARM_STA_BIT	0x0002	warmStart
2	非対応	—	linkDown
3	LINK_UP_BIT	0x0008	linkUp
4	AUTH_FAIL_BIT	0x0010	authenticationFailure
5	非対応	—	egpNeighborLoss
—	TRP_ALL_BIT	0x003f	すべてのトラップ

実装例は次のようになります。

```
/* warmStartとlinkUpの無効化 */
ercd = dis_trp(WARM_STA_BIT | LINK_UP_BIT);
```

snd_trp（ベンダーのトラップの送信）

【書式】

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

【パラメータ】

T_NODE*	nod	送信先のノードのポインタ
T_SNMP_TRP*	trp	トラップ用コマンドのポインタ
TMO	tmo	タイムアウト（msec）

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	引数の誤り
E_QOVR	トラップ用資源（CFG_SNMP_MAX_TRP_CNT）の不足
E_TMOUT	タイムアウト
E_OBJ	その他のエラー

【解説】

ベンダー固有のトラップまたはインフォームを送信します。本関数はトラップのパケットを作成し、そのトラップをUDPで送信し終わるまで待ち状態になります。また、インフォームの場合は送信先からの応答のパケットを受信するまで待ち状態になります。

引数のnodにはトラップの送信先のIPアドレスなどを指定してください。ただし、ポート（nod.port）の設定は不要です。ポート（162）は本システムが設定します。

tmoはタイムアウトの値を指定してください。このtmoの値はトラップを送信するソケット（snd_soc）のタイムアウトになります。

trpは構造体のT_SNMP_TRPの変数のポインタを指定してください。この構造体は次のようになります。

番号	型	変数名	説明
1	UB	ver	プロトコルのバージョンのマクロ SNMP_VER_V1 : v1 の場合 SNMP_VER_V2C : v2c の場合
2	VB*	com	コミュニティ名の文字列
3	UH	flg	オプションのフラグ
4	VB*	ent_oid	エンタープライズの OID の文字列（v1 の場合） snmpTrapOID の OID の文字列（v2c の場合）
5	INT	gen_trp	標準トラップの値（v1 のみ） （必ず TRP_ENT_SPEC を指定）
6	INT	spc_trp	詳細なトラップの値（v1 のみ）
7	UH	tmo	タイムアウト(msec)（インフォーム送信時のみ）
8	UH	rty_cnt	再送回数（インフォーム送信時のみ）
9	VP	var_oid	Variable binding に追加するオブジェクトの ID
10	UH	var_cnt	Variable binding の追加個数

verはトラップのプロトコルのバージョンをマクロで指定してください。v1 の場合は SNMP_VER_V1、v2c の場合はSNMP_VER_V2Cになります。

comはトラップのコミュニティ名を文字列で指定してください。

flgはトラップの送信に関するオプションをマクロで指定します。マクロは次のようになります。トラップを送信する場合はflgに0x00を指定してください。

番号	識別子（マクロ）	機能
1	TRP_INF_ENA	トラップではなくインフォームを送信する

ent_oidはトラップのプロトコルのバージョンがv1の場合は、トラップのエンタープライズのOIDの文字列、例えば“1.3.6.1.4.1.1234”などを指定してください。ent_oidに0x00（NULL）を指定した場合は設定用マクロのCFG_SNMP_MIB_SYS_OBJECTID（snmp_mib_cfg.h）で指定した OIDの文字列を使用します。また、プロトコルのバージョンがv2cの場合はent_oidにはトラップのVariable-bindingsの2つ目の値、つまりsnmpTrapOIDのOIDを文字列で指定してください。

トラップのプロトコルのバージョンがv1の場合はgen_trpとspc_trpを設定してください。gen_trpは必ずベンダー固有のトラップを示すマクロのTRP_ENT_SPEC（6）を指定してください。spc_trpには詳細なトラップ情報を示す番号を指定してください。

トラップの送信では、たとえば送信先の機器が存在しない場合は snd_socのタイムアウト（引数の tmo）の時間だけ待ってから、E_TMOUTのエラーを返します。

インフォームを送信する場合はtrp.tmo（1000以上で1000の倍数）とtrp.rty_cntを設定してください。trp.tmoはインフォームのタイムアウトの値、trp.rty_cntはインフォームの再送回数です。つまり、trp.tmoの時間が経過しても送信先からインフォームの応答がない場合はインフォームを再送信します。この再送信はtrp.rty_cntで指定された回数だけ繰り返します。trp.rty_cntが 0の場合は再送しません。ただし、インフォームのタイムアウトは１秒（1000）間隔で検出します。よって、trp.tmoには1000の倍数で1000以上の値、たとえば4000（4秒）などを設定してください。インフォームの送信では送信先の機器からの応答を待ちます。待ち時間が経過しても送信先から返信が無い場合はE_TMOUTのエラーを返します。

ただし、タイムアウトは指定した時間通りにならない場合があります。トラップ用のタスクで先に別のトラップの送信処理を行っている場合は、その処理の終了を待つため、指定よりも長い待ち時間になります。

var_oidとvar_cntはトラップに追加するVariable-bindingsを指定します。追加するVariable-bindingsが0個の場合はvat_cntに0、var_oidに0x00を指定してください。

追加するVariable-bindingsが１個の場合はvat_cntに0を指定し、var_oidには固定の（初期化時に作成した）ベンダー固有のMIBのIDとオブジェクトのIDをVP型に変換して代入してください。変換ではMIBのIDは上位の16ビット、オブジェクトのIDは下位の16ビットに設定してください。次のようにヘッダファイルのsnmp.hでは設定用のマクロを用意しています。

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

なお、可変の（動作中に追加した）ベンダー固有のMIBをvar_oidに指定することはできません。

Variable bindingsを2個以上指定する場合はvar_cntにその個数を指定し、var_oidにはIDをVP型の配列変数のポインタで指定してください。つまり、はじめにVP型の配列変数にVariable bindingsのMIBとオブジェクトのIDを代入します。次にその配列のポインタをvar_oidに設定します（図7.1）。

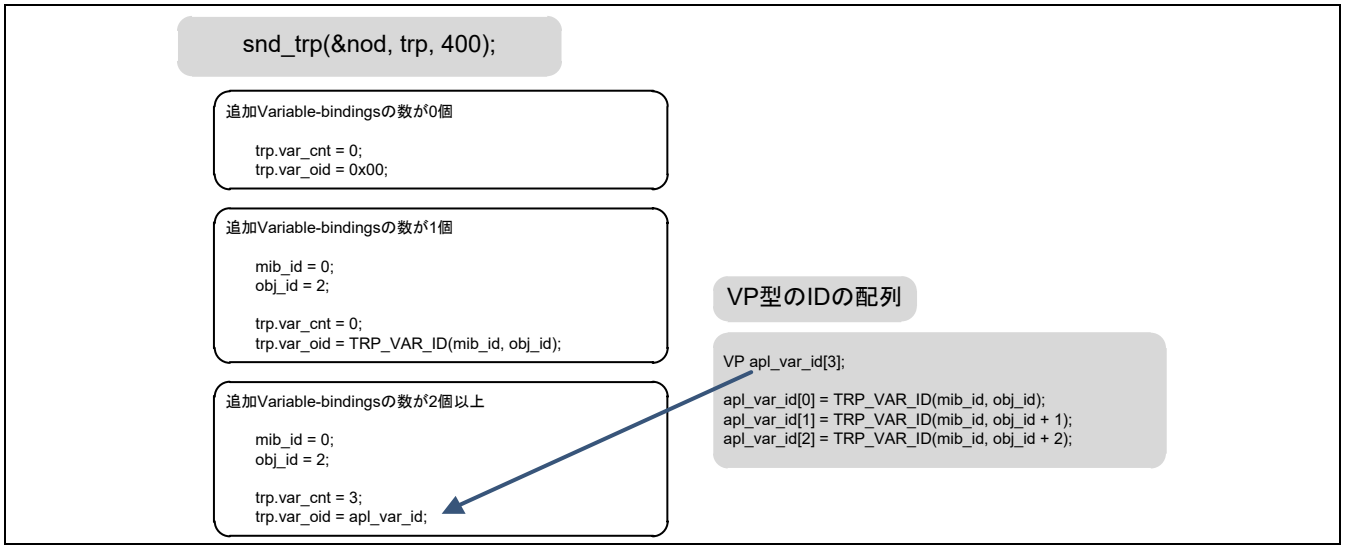

図 7.1 トラップのVariable-bindingsの追加

プロトコルのバージョンがv1のトラップを送信する場合、実装例は次のようになります。

```
T_SNMP_TRP trp;

memset(&trp, 0, sizeof(trp));
trp.ver = SNMP_VER_V1;                  /* トラップのバージョン v1 */
trp.com = "public";                     /* コミュニティ名 */
trp.gen_trp = TRP_ENT_SPEC;             /* ベンダー固有トラップ（固定値）*/
trp.spc_trp = 1234;                     /* 詳細なトラップ情報（任意の数値）*/
ercd = snd_trp(nod, trp, TRP_TMO);
        /* trp.ent_oidが0の場合、エンタープライズのOIDは
             CFG_SNMP_MIB_SYS_OBJECTID(snmp_mib_cfg.h)を使用 */
        /* trp.var_cntとtrp.var_oidが0の場合variable-bindingsは追加しない */
```

プロトコルのバージョンがv2cのトラップを送信する場合、実装例は次のようになります。

```
memset(&trp, 0, sizeof(trp));
trp.ver = SNMP_VER_V2C;                 /* トラップのバージョン v2c */
trp.com = "public";                     /* コミュニティ名 */
trp.ent_oid = "1.3.6.1.4.1.1234.1.2";   /* snmpTrapOIDベンダー固有MIB */
ercd = snd_trp(nod, trp, TRP_TMO);
      /* trp.var_cntとtrp.var_oidが0の場合variable-bindingsは追加しない */
```

プロトコルのバージョンがv1のトラップでVariable-bindingsを1個追加して送信する場合、実装例は次のようになります。

```
memset(&trp, 0, sizeof(trp));
trp.ver = SNMP_VER_V1;                          /* トラップのバージョン v1 */
trp.com = "public";                             /* コミュニティ名 */
trp.gen_trp = TRP_ENT_SPEC;                     /* ベンダー固有トラップ（固定値）*/
trp.spc_trp = 1234;                             /* 詳細なトラップ情報（任意の数値）*/
trp.ent_oid = "1.3.6.1.4.1.9876.1234";          /* エンタープライズのOID文字列を設定 */
trp.var_cnt = 0;                                /* トラップに追加するvariable-bindingsの数（1個の場合は0を設定）*/
trp.var_oid = TRP_VAR_ID(0, 2);                 /* トラップに追加するVariable-bindingsのID */
ercd = snd_trp(nod, trp, TRP_TMO);
```

プロトコルのバージョンが v2c のトラップでVariable-bindingsを3個追加して送信する場合、実装例は次のようになります。

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

memset(&trp, 0, sizeof(trp));
trp.ver = SNMP_VER_V2C;                         /* トラップのバージョン v2c */
trp.com = "public";                             /* コミュニティ名 */
trp.ent_oid = "1.3.6.1.4.1.1234.1.2";           /* snmpTrapOID ベンダー固有MIB */
trp.var_cnt = 3;                                /* トラップに追加するVariable-bindingsの数 */
apl_var_id[0] = TRP_VAR_ID(0, 2);               /* snmp_mib_ven_0の要素2 */
apl_var_id[1] = TRP_VAR_ID(1, 5);               /* snmp_mib_ven_1の要素5 */
apl_var_id[2] = TRP_VAR_ID(1, 6);               /* snmp_mib_ven_1の要素6 */
trp.var_oid = apl_var_id;                       /* variable bindingのIDの配列 */
ercd = snd_trp(nod, trp, TRP_TMO);
```

インフォームを送信する場合、実装例は次のようになります。

```
memset(&trp, 0, sizeof(trp));
trp.ver = SNMP_VER_V2C;                         /* トラップのバージョン v2c */
trp.com = "public";                             /* コミュニティ名 */
trp.ent_oid = "1.3.6.1.4.1.1234.1.2";           /* snmpTrapOID ベンダー固有MIB */
trp.flg = TRP_INF_ENA;                          /* インフォームを指定（トラップではない）*/
trp.tmo = 4000;                                 /* インフォームのタイムアウト(msec) */
trp.rty_cnt = 4;                                /* インフォームのリトライ回数 */
ercd = snd_trp(&nod, &trp, TRP_TMO);            /* nodで指定したサーバーへ送信 */
```

7.3 コールバック関数

本システムが提供するコールバック関数の仕様を説明します。マネージャーからベンダー固有のプライベートMIBのオブジェクトに対してGetRequest、GetNextRequest、GetBulkRequest、SetRequestのパケットを受信した場合に、本システムはユーザーのコールバック関数を発行します。

コールバック関数は次のような引数になります。

【書式】

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

【パラメータ】

T_SNMP_CFG_CBK_DAT*	cbk_dat	コールバック用構造体の変数のポインタ

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_OBJ	エラー

引数のcbk_datは本システムの内部で宣言した構造体のT_SNMP_CFG_CBK_DATの変数のポインタです。この構造体は次のようになります。

番号	型	変数名	説明
1	UH	req	SNMP のリクエストの種類のマクロ — 固定ベンダー MIB— SNMP_REQ_GET : Get (Next/Bulk)Request SNMP_REQ_SET : SetRequest — 標準 MIB の System Group— SNMP_REQ_SET_SYS : SetRequest — 可変ベンダー MIB— SNMP_REQ_GET_VAL : Get (Next/Bulk)Request SNMP_REQ_SET_VAL : SetRequest
2	UH	mib_id	ベンダーの MIB の ID
3	UH	obj_id	ベンダーのオブジェクトの ID
4	UH	typ	オブジェクトのデータのデータ型のマクロ
5	VP	buf	データを格納したバッファ
6	UH	dat_len	データのサイズ（バイト）
7	UH	buf_len	バッファのサイズ（バイト） （req が SNMP_REQ_GET の場合のみ有効）

req はマネージャーからのリクエストの種類です。つまり、リクエストが固定のベンダー固有のMIBのオブジェクトに対するGetRequest、GetNextRequestまたはGetBulkRequestの場合はreqにSNMP_REQ_GETが代入されています。リクエストが固定のベンダー固有のMIBのオブジェクトに対するSetRequestの場合はreqにSNMP_REQ_SETが代入されています。また、動作中に追加した可変のベンダー固有のMIBのオブジェクトに対するリクエストの場合は同様にそれぞれSNMP_REQ_GET_VALとSNMP_REQ_SET_VALが代入されています。さらに、リクエストが標準MIBのSystem Group のMIBのオブジェクトに対するSetRequestの場合はreqにSNMP_REQ_SET_SYSが代入されています。

mib_idとobj_idはベンダー固有のMIBとオブジェクトのIDです。IDの説明は「6.2.1 MIBとオブジェクトのID」の説明を参照してください。

System Groupのオブジェクトに対するコールバックの場合、mib_idは0、obj_idは3 (sysContact)、4 (sysName)または5 (sysLocation)が代入されています。ユーザーはobj_idの値でSetRequestされたオブジェクトを判定できます。このobj_idに代入される値はヘッダファイルの snmp_mib.hで次のようにマクロが定義されています。

```
#define SNMP_MIB2_SYS_CONTACT          3                   /* sysContact */
#define SNMP_MIB2_SYS_NAME             4                   /* sysName */
#define SNMP_MIB2_SYS_LOCATION         5                   /* sysLocation */
```

typにはオブジェクトのデータのデータ型がマクロで代入されています。データ型のマクロは次のようになります。

番号	マクロ	説明	備考
1	TYP_INT	Integer	32 ビット
2	TYP_OCT_STR	Octet String	文字列
3	TYP_IP_ADR	IP Address	32 ビット（IPv4 用）
4	TYP_CNT	Counter	32 ビット
5	TYP_GAUGE	Gauge	32 ビット
6	TYP_TIM_TIC	Time ticks	32 ビット
7	TYP_OBJ_ID	Object Identifier	文字列

バッファ（buf）にはオブジェクトのデータが格納されています。reqがSNMP_REQ_GET(_VAL)の場合は現在のオブジェクトのデータが格納されています。reqがSNMP_REQ_SET(_VAL)の場合はマネージャーがSetRequestで設定するデータが格納されています。typがTYP_OCT_STRの場合はbufに文字列が格納されていますが、reqがSNMP_REQ_GET(_VAL)の場合、その文字列はNULL文字（¥0）で終端されています。reqがSNMP_REQ_SET(_VAL)の場合、その文字列はNULL文字で終端されていません。

dat_lenはbufに格納されたデータの長さです。typがTYP_OCT_STR, TYP_OBJ_ID（文字列）以外の場合、dat_lenは4でbufには4バイトの数値が格納されています。typが文字列の場合、dat_len は（終端のNULL文字を含まない）文字列の長さが格納されています。

buf_lenはバッファ領域（buf）のサイズです。buf_lenはreqがSNMP_REQ_GET(_VAL)の場合のみ有効です。typが文字列以外の場合、buf_lenは4になります。typが文字列の場合、buf_lenはbufに格納可能な文字列の長さ（NULL文字を含む）になります。

マネージャーからGet(Next/Bulk)Requestを受信した場合、reqはSNMP_REQ_GET(_VAL)になります。このとき、ユーザーがオブジェクトのデータを更新したい場合は、bufに更新したい値を代入してください。本システムは更新した値をマネージャーに返します。typが文字列以外の場合はbufに4バイトの数値を格納してください。typ文字列の場合は bufに文字列をコピーしてください。コールバック関数の戻り値は必ずE_OKにしてください。

次に、マネージャーからSetRequestを受信した場合、reqはSNMP_REQ_SET(_VAL)になります。このとき、bufにはマネージャーがこれから更新するデータが格納されています。ユーザーはマネージャーが更新するデータを確認し、もしそのデータで更新することを許可する場合は本コールバック関数の戻り値をE_OKにしてください。更新を許可しない場合は戻り値をE_OBJにしてください。本コールバック関数の戻り値がE_OBJの場合、本システムはオブジェクトのデータを更新しません。また、マネージャーへエラー（commitFailed）を返します。なお、reqがSNMP_REQ_SET(_VAL)の場合はbufとdat_lenの内容を書き換えないでください。また、System Groupに対するSetRequestの場合は、本コールバック関数の戻り値を必ずE_OKにしてください。

コールバック関数の引数のcbk_dat->bufの詳細を説明します。図7.2のようにGetRequestの場合、コールバック関数の引数のcbk_dat->bufが指し示すデータはユーザーがファイルの snmp_mib_cfg.cで設定したMIBのオブジェクトのデータバッファです。

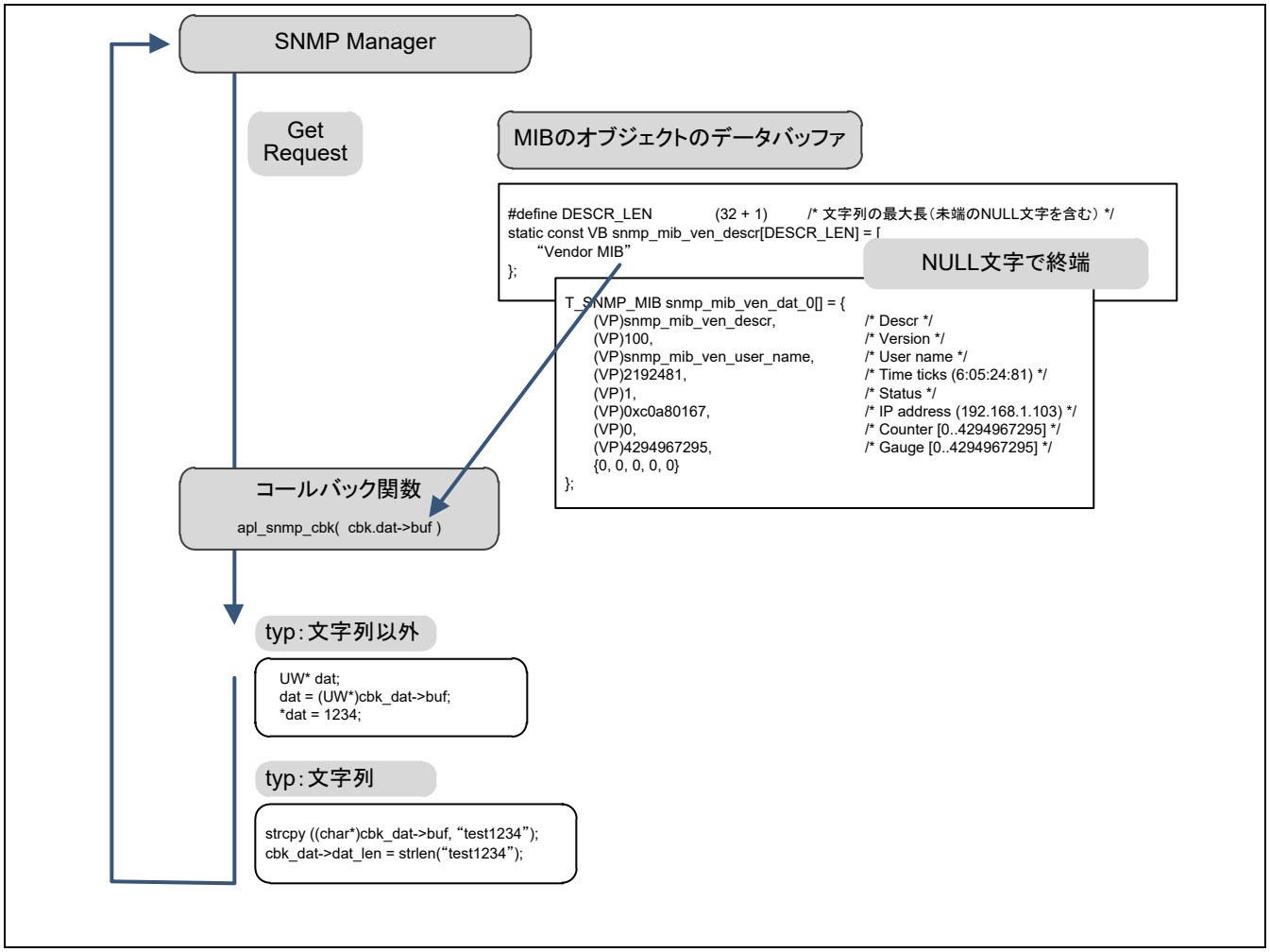

図7.2 GetRequestのコールバック関数

つまり、コールバック関数の中でユーザーは直接データバッファの内容を書き換えます。typが文字列以外の場合、bufには現在のオブジェクトの値（4バイトの数値）が代入されています。この数値をユーザーが希望する値に変更してください。typが文字列の場合、bufには現在のオブジェクトの文字列（NULL文字（¥0）で終端済み）が格納されています。この文字列をユーザーが希望する文字列に変更してください。バッファ（buf）は文字列の最大文字数の領域に加えて終端のNULL文字の領域も含まれています。よって、ユーザーは関数のstrcpyでbufに新しい文字をコピーすることが可能です。また、関数のstrcpyを使用せずにNULL文字を含まない文字列のみをコピーしてもかまいません。コールバック関数を抜けた後に、本システムが文字列の最後にNULL文字を追加します。ただし、本システムがNULL文字を追加できるように、bufにコピーした文字列の長さ（NULL文字を含まない）をdat_lenに代入して本システムに返してください。なお、文字列を書き換えるときはバッファのサイズ（cbk_dat->buf_len）を超えて書き込まないようにしてください。

次に、図7.3のようにSetRequestの場合、コールバック関数の引数のcbk_dat->bufが指し示すデータは本システムの内部変数のデータです。よって、ユーザーはbufの内容を書き換えないでください。

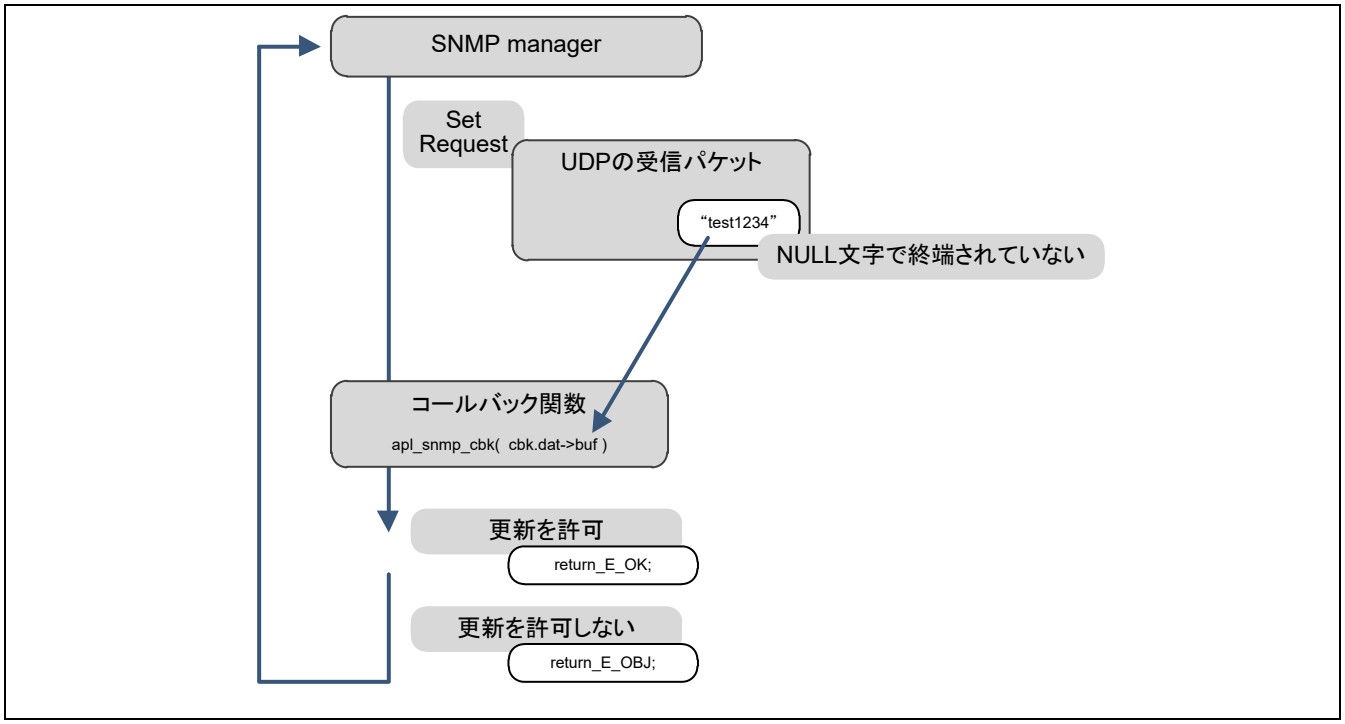

図7.3 SetRequestのコールバック関数

また、bufにはマネージャーがこれから更新するデータが格納されていますが、typが文字列の場合、bufの文字列はNULL文字で終端されていません。よって、文字列を参照して比較する場合は関数のstrcmpは使用できません。文字列を比較する場合は strncmp を使用してください。

GetRequestの場合の実装例は次のようになります。

```
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:
                /* データ型が文字列の場合 */
                len = strlen("New String");
                if (len < cbk_dat->buf_len) {
                    strcpy((char*)cbk_dat->buf, "New String");
                    cbk_dat->dat_len = len;
                }
                /* バッファサイズ(cbk_dat->buf_len)を超えて書き込まない */
                /* bufへ"New String"をstrcpyでコピー（NULL文字で終端しても良い）*/
                break;
            case 1:
                /* データ型が文字列の場合 */
                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;
                }
                /* bufへ文字列をforループでコピー（NULL文字で終端しなくても良い）*/
                break;
            case 3:
                /* データ型が数値の場合 */
                dat = (UW*)cbk_dat->buf;
                *dat += 1;      /* 数値を増加 */
                       break;
                       ....
return E_OK;      /* E_OKを返す */
```

SetRequestの場合の実装例は次のようになります。

```
    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:
                /* データ型が文字列の場合 */
                len = strlen("root");
                res = strncmp((const char*)cbk_dat->buf, "root", len);
                if (res == 0) {
                    ercd = E_OBJ;            /* 先頭が"root"と同じ文字列の場合、更新を許可しない */
                }
                /* cbk_dat->bufの文字列はNULL文字で終端していないのでstrncmpで文字列を比較 */
                break;
            case 5:
                /* データ型がIPアドレスの場合 */
                dat = (UW*)cbk_dat->buf;
                if ((*dat & 0xffff0000) != 0xc0a80000) {
                    ercd = E_OBJ;            /* IPアドレスが192.168.*.*ではない場合、更新を許可しない */
                }
                break;
            default:
                break;
        }
    }
}
....
return ercd;
```

なお、オブジェクトの値の変更は実装例のようにcbk_dat->bufに対して行ってください。コールバック関数の中で関数のset_mib_objやset_val_mib_objを発行することはできません。

7.4 ベンダー可変拡張MIBの関数

この節では追加削除が可能なベンダー固有の拡張MIBに関する関数を説明します。この節の関数の引数のval_mib_idとval_obj_idはユーザーが宣言する配列変数のsnmp_mib_ven_valのデータのIDです。前節7.2のmib_idとobj_idとは異なります。

add_val_mib_nod（MIBのノードの追加）

【書式】

```
ER add_val_mib_nod(UH val_mib_id, UH val_obj_id, UH* mib_nod_cnt)
```

【パラメータ】

UH	val_mib_id	MIBのID
UH	val_obj_id	オブジェクトのID
UH*	mib_nod_cnt	追加時に消費したノード数

【戻り値】

ER	Ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_NOMEM	ノードの資源（メモリ）が不足
E_QOVR	すでにノードが追加されている
E_TMOUT	排他用セマフォがタイムアウトで取得できない
E_OBJ	その他のエラー

【解説】

この関数はval_mib_idとval_obj_idで指定されたベンダー固有の可変拡張MIBのノードをシステム内部のMIBのツリーへ追加します。引数のval_mib_idにはMIBのID、val_obj_idにはオブジェクトのIDを指定してください。

関数が成功した場合、引数のmib_nod_cntにはノードの追加で使用したノードの個数が代入されています。消費したノードの個数が不要な場合は引数のmib_nod_cntにNULL(0)を代入してください。

指定のMIBのノードがすでに追加されている場合はエラーのE_QOVRを返します。追加時にノードの資源が不足した場合はエラーのE_NOMEMを返します。

この関数でノードを追加した場合は、追加したノードを削除するまで、そのノードに関係する設定用データ（snmp_mib_ven_valの内容）を変更しないでください。

なお、テーブルのノードを追加する場合は図7.4のように最初に①Entryのノードのみを追加してください。次にEntryの下位のオブジェクトを追加してください。図7.4では最初にEntryの“*.5678.5.3.1”を追加しています。次に、下位の“*.5678.5.3.1.1.1”などを追加しています。

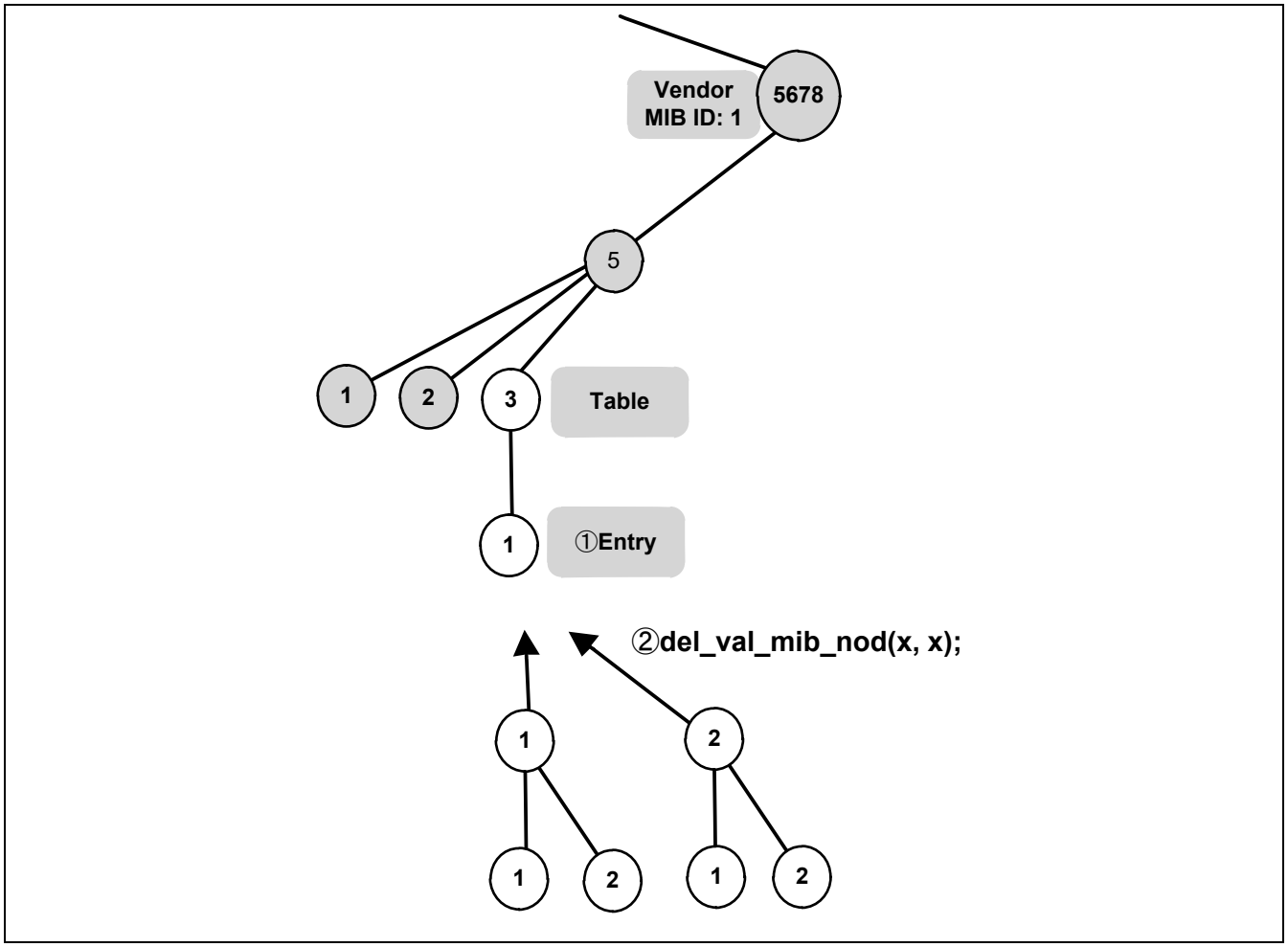
Vendor
MIB ID: 1
5678
5
1
2
3
Table
1
①Entry
②del_val_mib_nod(x, x);
1
2
1
2
1
2

図7.4　　テーブルの追加

del_val_mib_nod（MIBのノードの削除）

【書式】

```
ER del_val_mib_nod(UH val_mib_id, UH val_obj_id)
```

【パラメータ】

UH	val_mib_id	MIBのID
UH	val_obj_id	オブジェクトのID

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_NOID	削除するノードが存在しない
E_TMOUT	排他用セマフォがタイムアウトで取得できない
E_OBJ	その他のエラー

【解説】

この関数はadd_val_mib_nodで追加したベンダー固有の可変拡張MIBのノードを削除します。引数のval_mib_idにはMIBのID、val_obj_idにはオブジェクトのIDを指定してください。指定の削除するMIBのノードが存在しない場合はエラーのE_NOIDを返します。

なお、テーブルのノードを削除する場合は図7.4の追加とは逆に、最初にEntryの下位にあるオブジェクトを削除してください。Entryの下位のオブジェクトをすべて削除すると① Entryのノードも削除されます。よって、ユーザーがEntryのノードを削除する必要はありません。例えば、図7.4で最初に① “*.5678.5.3.1.1.1”、② “*.5678.5.3.1.1.2”と③ “*.5678.5.3.1.2.1”を削除し、最後に④ “*.5678.5.3.1.2.2”を削除した場合、④を削除すると同時にEntryの“*.5678.5.3.1”も削除されます。Entryの上位にあるTableのノードはユーザーで削除してください。

get_val_mib_obj（可変オブジェクトのデータの読み込み）

【書式】

```
ER get_val_mib_obj(VP buf, UH* len, UH val_mib_id, UH val_obj_id)
```

【パラメータ】

VP	buf	データを格納するバッファのポインタ
UH*	len	バッファとデータのサイズ（バイト）
UH	val_mib_id	MIBのID
UH	val_obj_id	オブジェクトのID

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	引数の誤り
E_NOSPT	データ型が対応していない
E_BOVR	バッファの長さの不足
E_OBJ	その他のエラー

【解説】

この関数はval_mib_idとval_obj_idで指定されたベンダー固有の可変拡張MIBのオブジェクトのデータを読み出してbufに格納します。

関数の仕様はget_mib_objと同じです。

set_val_mib_obj（可変オブジェクトのデータの書き出し）

【書式】

```
ER set_val_mib_obj(VP buf, UH len, UH val_mib_id, UH val_obj_id)
```

【パラメータ】

VP	buf	データを格納したバッファのポインタ
UH	len	データのサイズ（バイト）
UH	val_mib_id	MIBのID
UH	val_obj_id	オブジェクトのID

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード

【エラーコード】

E_PAR	引数の誤り
E_NOSPT	データ型が対応していない
E_OBJ	データのサイズの過不足、その他のエラー

【解説】

この関数はval_mib_idとval_obj_idで指定したベンダー固有の可変拡張MIBのオブジェクトのデータを書き出します。

関数の仕様はset_mib_objと同じです。

改訂記録	RZ/T1グループ μNet3/SNMP ユーザーズマニュアル

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00		—	初版発行
2.00	2020.11.1	2. 仕様概要	
		10	「2.2 MIB-II 一覧」の表2.2 Interfaces Groupの説明で、登録可能なデバイスが従来の仕様の2つを 16に変更
		4. 資源の設定	
		25, 26	「4.1.1 資源一覧」の表4.1 を修正
		5. SNMP の設定	
		31	「5.1 基本設定」 を修正
		35	「5.1.1 SNMP の設定」に CFG_SNMP_MAX_OID_DEP を追加
		6. ベンダー MIB の設定	
		46 ～ 47	「6.2.1 MIB とオブジェクトの ID」の説明で MIB テーブルを RAM 領域に配置するように変更、 オブジェクトのテーブルの要素の最大数の誤り (65536 個) を正しい個数の 65534 個に修正 MIBテーブルの要素の最大数の誤り(255/256個)を正しい個数の254個に修正
		48	「6.2.2 MIB テーブル」に MIB テーブルの予約変数の説明を追記、MIBテーブルの要素の最大数の誤り(255/256個)を正しい個数の254個に修正
		51	「6.2.4 オブジェクトのテーブル」にテーブルを設定する場合は先に Entry を代入する説明を追記
		50 ～ 53	「6.2.4 オブジェクトのテーブル」、「6.2.5 データのテーブル」にデータ型に TYP_OBJ_ID を追加
		56 ～ 58	「6.3 ベンダーの可変プライベートMIB の設定」のベンダー固有の拡張MIBの追加削除に関する説明を追加
		7. インタフェース	
		60 ～ 73	「7.2 関数」の snmp_ini、snmp_ena、snmp_dis に E_SYS のエラーを追加、snmp_extにsnmp_disを発行してからsnmp_extを発行する順序の説明を追加、E_OBJのエラーを追加、snmp_enaの説明にポートの接続完了後にトラップを送信する説明を追加、get_mib_obj、set_mib_objにobj_idの型の誤りを修正、エラーのE_NOSPTの記述不足を修正、set_mib_objに引数の型の誤りを修正 snd_trpの説明にインフォームのタイムアウトの検出が 1 秒間隔の説明を追記、インフォームのタイムアウトの説明を追加、E_QOVRのエラーを追加
		74 ～ 76	「7.3 コールバック関数」の構造体のreqに、SNMP_REQ_GET_VALとSNMP_REQ_SET_VALを追加
		81 ～ 85	「7.4 ベンダー可変拡張MIBの関数」に、add_val_mib_nod、del_val_mib_nod、get_val_mib_obj、set_val_mib_objを追加del_val_mib_nodの説明を変更、Entryのノードが自動で削除されることを追記、set_val_mib_objの引数の型の誤りを修正

RZ/T1 グループ　ユーザーズ・マニュアル
μNet3/SNMP編

発行年月日　2013年06月27日　Rev.1.00
　　2020年11月01日　Rev.2.00

発行　　ルネサス エレクトロニクス株式会社
　　〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

RZ/T1 グループ ユーザーズ・マニュアル

μNet3/SNMP 編

ルネサス エレクトロニクス株式会社

R01US0202JJ0200