SNMP Management: SNMPv2

OBJECTIVES

- *Community-based security*
- *SNMPv2 enhancements*
  - *Additional messages*
  - *Formalization of SMI*
- *Get-bulk request and information-request*
- *SNMP MIB modifications*
- *Incompatibility with SNMPv1*
- *Proxy server*
- *Bilingual manager*

SNMPv1, which was originally called SNMP, was developed as an interim management protocol with OSI as the ultimate network management protocol. A placeholder, CMOT (CMIP over TCP/IP), was created in the Internet Management Information Base (MIB) for migrating from SNMP to CMIP. But the "best-laid plans…" never came about. SNMP caught on in the industry. Major vendors had incorporated SNMP modules in their network systems and components. SNMP now needed further enhancements.

Version 2 of Simple Network Management Protocol, SNMPv2, was developed when it became obvious that OSI network management standards were not going to be implemented in the near future. The working group that was commissioned by the IETF to define SNMPv2 released it in 1996. It is also a community-based administrative framework similar to SNMPv1 defined in STD 15 [RFC 1157], STD 16 [RFC 1155, 1212], and STD 17 [RFC 1213]. Although the original version was known as SNMP, it is now referred to as SNMPv1 to distinguish it from SNMPv2.

6.1 MAJOR CHANGES IN SNMPv2

Several significant changes were introduced in SNMPv2. One of the most significant changes was to improve the security function that SNMPv1 lacked. Unfortunately, after significant effort, due to lack of consensus, this was dropped from the final specifications, and SNMPv2 was released with the rest of the

changes. The security function continued to be implemented on an administrative framework based on the community name and the same administrative framework as in SNMPv1 was adopted for SNMPv2. SNMPv2 Working Group has presented a summary of the community-based Administrative Framework for the SNMPv2 framework, and referred to it as SNMPv2C in RFC 1901. RFC 1902 through RFC 1907 present the details on the framework. There are significant differences between the two versions of SNMP, and unfortunately version 2 is not backward compatible with version 1. RFC 1908 presents implementation schemes for the coexistence of the two versions.

The basic components of network management in SNMPv2 are the same as version 1. They are the agent and the manager, both performing the same functions. The manager-to-manager communication, shown in Figure 4.8, is formalized in version 2 by adding an additional message. Thus, the organizational model in version 2 remains essentially the same. In spite of the lack of security enhancements, major improvements to the architecture have been made in SNMPv2. We will list some of the highlights that would motivate the reader's interest in SNMPv2.

Bulk Data Transfer Message: Two significant messages were added. The first is the ability to request and receive bulk data using the get-bulk message. This speeds up the get-next-request process and is especially useful to retrieve data from tables.

Manager-to-Manager Message: The second additional message deals with interoperability between two network management systems. This extends the communication of management messages between management systems and thus makes network management systems interoperable.

Structure of Management Information (SMI): In SNMPv1, SMI is defined as STD 16, which is described in RFCs 1155 and 1212, along with RFC 1215, which describes traps. They have been consolidated and rewritten in RFCs 1902–1904 for SMI in SNMPv2. RFC 1902 deals with SMIv2, RFC 1903 with textual conventions, and RFC 1904 with conformances.

SMIv2 is divided into three parts: module definitions, object definitions, and trap definitions. An ASN.1 macro, MODULE-IDENTITY, is used to define an information module. It concisely conveys the semantics of the information module. The OBJECT-TYPE macro defines the syntax and semantics of a managed object. The trap is also termed notification and is defined by a NOTIFICATION-TYPE macro.

Textual Conventions are designed to help define new data types. They are also intended to make the semantics consistent and clear to the human reader. Although new data types could have been created using new ASN.1 class and tag, the decision was made to use the existing defined class types and apply restrictions to them.

Conformance Statements help the customer objectively compare features of various products. It also keeps vendors honest in claiming their product as being compatible with a given SNMP version. Compliance defines a minimum set of capabilities. Additional capabilities may be offered as options in the product by vendors.

Table Enhancements: Using a newly defined columnar object with a Syntax clause, *RowStatus*, conceptual rows could be added to or deleted from an aggregate object table. Further, a table can be expanded by augmenting another table to it, which is helpful in adding additional columnar objects to an existing aggregate object.

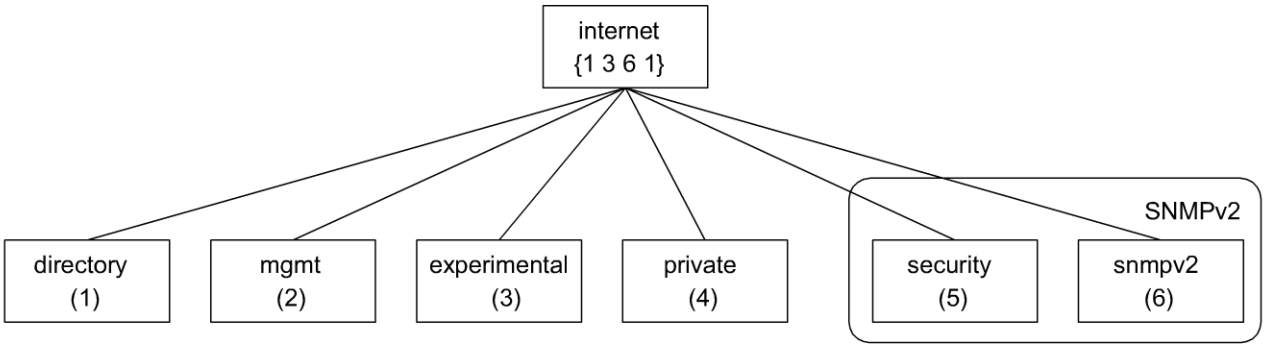

Figure 6.1 SNMPv2 Internet Group

MIB Enhancements: In SNMPv2, the Internet node in the MIB has two new subgroups: security and snmpV2, as shown in Figure 6.1. There are significant changes to System and SNMP groups of version 1. There are changes to the System group made under mib-2 node in the MIB. The SNMP entities in version 2 are a hybrid, with some entities from the SNMP group, and the rest from the groups under the newly created snmpV2 node.

Transport Mappings: There are several changes to the communication model in SNMPv2. Although use of UDP is the preferred transport protocol mechanism for SNMP management, other transport protocols could be used with SNMPv2. The mappings needed to define other protocols on to UDP are the subject of RFC 1906.

6.2 SNMPv2 SYSTEM ARCHITECTURE

SNMPv2 system architecture looks essentially the same as that of version 1, as shown in Figure 4.9. However, there are two significant enhancements in SNMPv2 architecture, which are shown in Figure 6.2. First, there are seven messages instead of five as in Figure 4.9. Second, two manager applications can communicate with each other at the peer level. Another message, report message, is missing from Figure 6.2. This is because even though it has been defined as a message, SNMPv2 Working Group did not specify its details. It is left for the implementers to generate the specifications. It is not currently being used and is hence omitted from the figure.

The messages *get-request*, *get-next request*, and *set-request* are the same as in version 1 and are generated by the manager application. The message, response, is also the same as get-response in version 1, and is now generated by both agent and manager applications. It is generated by the agent application in response to a get or set message from the manager application. It is also generated by the manager application in response to an *inform-request* message from another manager application.

An *inform-request* message is generated by a manager application and is transmitted to another manager application. As mentioned above, the receiving manager application responds with a response message. This set of communication messages is a powerful enhancement in SNMPv2, since it makes two network management systems interoperable.

The message *get-bulk-request* is generated by manager application. It is used to transfer large amounts of data from the agent to the manager, especially if it includes retrieval of table data. The retrieval is fast and efficient. The receiving entity generates and fills data for each entry in the request and transmits all the data as a response message back to the originator of the request.

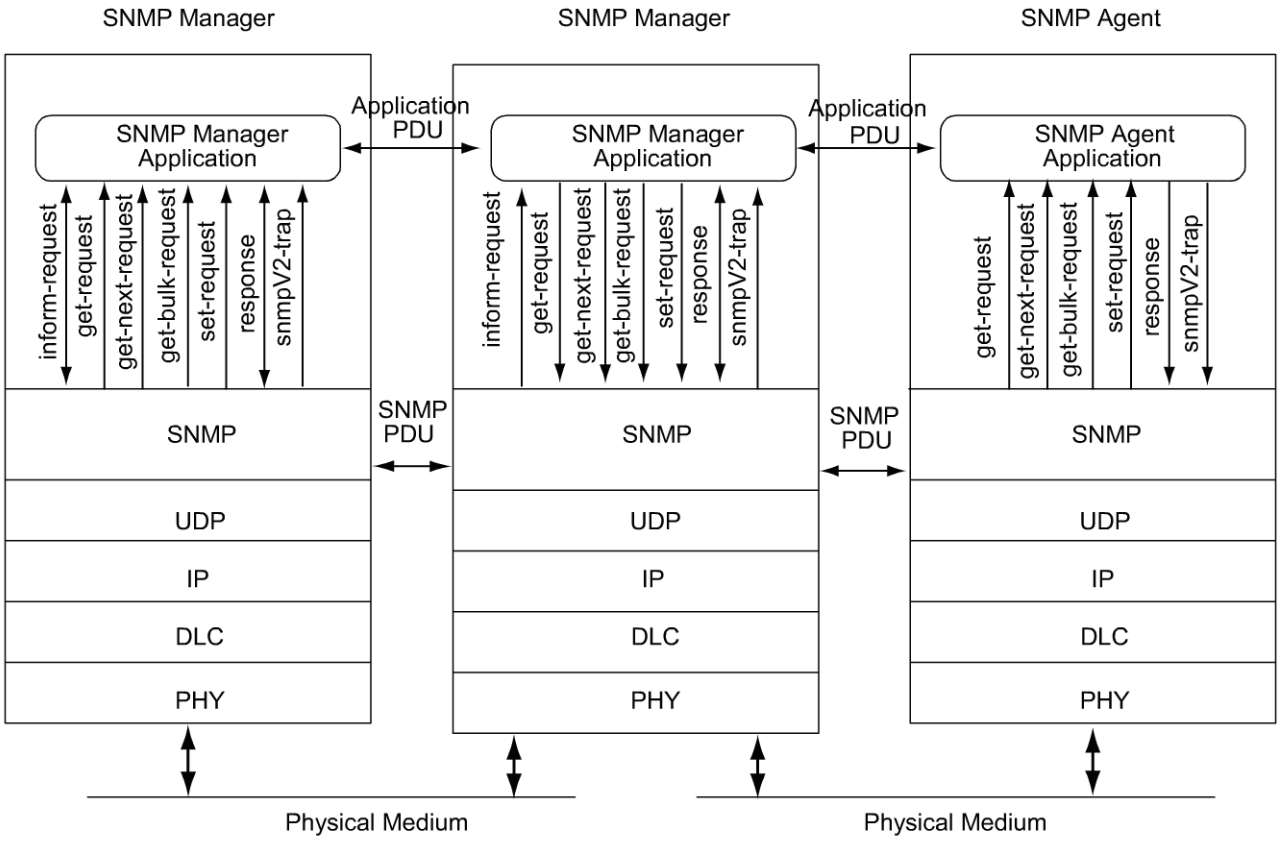

Figure 6.2 SNMPv2 Network Management Architecture

An *SNMPv2-trap event*, known as trap in version 1, is generated and transmitted by an agent process when an exceptional situation occurs. The destination to which it is sent is implementation-dependent. The PDU structure has been modified to be consistent with other PDUs.

Another enhancement in SNMPv2 over version 1 is the mapping of the SNMP layer over multiple transport domains. An example of this is shown in Figure 6.3, in which an SNMPv2 agent riding over a connectionless OSI transport layer protocol, Connectionless-Mode Network Service (CLNS), communicates with an SNMPv2 manager over the UDP transport layer. RFC 1906, which describes transport mappings, addresses a few well-known *transport layer mappings*; others can be added using a similar structure.

Details on the MIB relating to SNMPv2 are covered in Section 6.4 and communication protocol aspects of messages in Section 6.5. Although not a standard, RFC 1283 specifies SNMP over Connection-Oriented Transport Service (COTS), a connection-oriented OSI transport protocol. However, SNMP is not specified over connection-oriented Internet protocol, TCP.

6.3 SNMPv2 STRUCTURE OF MANAGEMENT INFORMATION

There are several changes to SMI in version 2, as well as enhancements to SMIv2 over that of SMIv1. As stated earlier, SMIv2 [RFC 1902] is divided into three parts: module definitions, object definitions, and notification definitions.

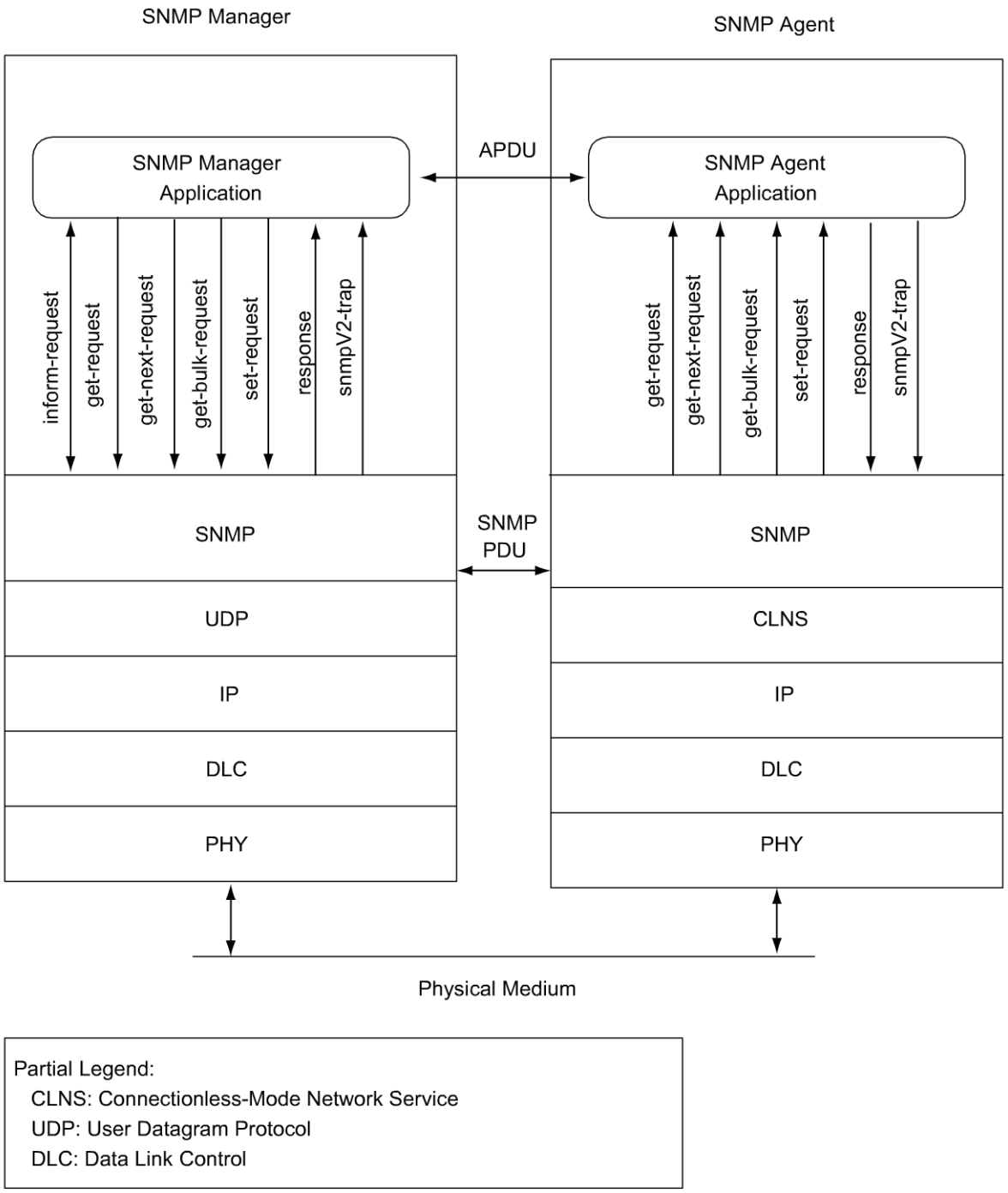

Figure 6.3 SNMPv2 Network Management Architecture on Multiple Transport Domains

We introduced the concept of a module in Section 3.6.1, which is a group of assignments that are related to each other. Module definitions describe the semantics of an information module and are formally defined by an ASN.1 macro, MODULE-IDENTITY.

Object definitions are used to describe managed objects. The OBJECT-TYPE macro that we discussed in Section 4.7.3 is used to define a managed object. OBJECT-TYPE conveys both syntax and semantics of the managed object.

Notification in SMIv2 is equivalent to trap in SMIv1. In SMIv1, trap is formally specified by an ASN.1 macro, TRAP-TYPE. In SMIv2, notification is specified by an ASN.1 macro, NOTIFICATION-TYPE, and conveys both its syntax and semantics.

In addition to the above three parts, there is an additional part defined in SMIv2, which formalizes the assignment of OBJECT IDENTIFIER. Even though we have two assignments in SMIv1, namely, object name and trap, they are not formally structured. In SMIv2, an ASN.1 macro, OBJECT-IDENTITY is introduced for the assignment of object name and notification to OBJECT IDENTIFIER, as shown in Figure 6.4.

6.3.1 SMI Definitions for SNMPv2

Figure 6.4 shows a skeleton of the SMIv2 and the reader is referred to RFC 1902 for a complete set of definitions. We have taken the liberty of presenting the definitions with some additional comments (marked by *) and structural indentations to bring out clearly the BEGIN and END of macros.

Definitions begin with the high-level nodes under the Internet MIB. Two additional nodes, security and SNMPv2, are introduced. The security node is just a placeholder and is reserved for the future. The *snmpV2* node has three subnodes: *snmpDomains*, *snmpProxys*, and *snmpModules*. The MIB tree showing all these nodes defined in SMIv2 is presented in Figure 6.5.

6.3.2 Information Modules

RFC 1902 defines *information module* as an ASN.1 module defining information relating to network management. SMI describes how to use a subset of ASN.1 to define an information module.

There are three kinds of information modules that are defined in SNMPv2. They are MIB modules, compliance statements for MIB modules, and capability statements for agent implementations. This classification scheme does not impose rigid taxonomy in the definition of managed objects. Figure 6.6 shows an example where *conformance information* and *compliance statements* are part of the SNMP group of SNMPv2 MIB. As we shall see later, the SNMP group in SNMPv2 contains some of the objects of version 1 and some new objects and object groups (to be defined later). It also has information on conformance requirements. In the example shown, the mandatory groups in implementing SNMPv2 are *snmpGroup*, *snmpSetGroup*, *systemGroup*, and *snmpBasicNotificationsGroup*. Thus, if a network component vendor claims that its management agent is SNMPv2 compliant, these groups as they are defined in SNMPv2 should be implemented.

MIB specifications contain only compliant statements in them. The *agent-capability statements* are part of implementation in the agent by the vendor. It might be included as part of an "enterprise-specific" module.

The information on SMIv2 has been split into three parts in the documentation. MIB modules for SMIv2 are covered in RFC 1902. The textual conventions to be used to describe MIB modules have been formalized in RFC 1903. The conformance information, which encompasses both compliance and agent capabilities, is covered in RFC 1904.

6.3.3 SNMP Keywords

Keywords used in the specifications of SMIv2 are a subset of ASN.1. But it is a different subset from that of SMIv1. Table 6.1 shows the comparison of keywords used in the two versions. We will address

```
SNMPv2-SMI DEFINITIONS ::=
BEGIN

-- the path to the root
        org             OBJECT IDENTIFIER ::= {iso 3}
                        …
        private         OBJECT IDENTIFIER ::= {internet 4}
        enterprises     OBJECT IDENTIFIER ::= {private 1}
        security        OBJECT IDENTIFIER ::= {internet 5}
        snmpV2          OBJECT IDENTIFIER ::= {internet 6}
       -- transport domains
        snmpDomains         OBJECT IDENTIFIER ::= {snmpV2 1}
              -- transport proxies
        snmpProxys          OBJECT IDENTIFIER ::= {snmpV2 2}
        --module identities
        snmpModules         OBJECT IDENTIFIER ::= {snmpV2 3}
       -- definitions for information modules
        MODULE-IDENTITY MACRO
        BEGIN
              <clauses> ::= <values>
        END
       -- definitions for OBJECT IDENTIFIER assignments*
        OBJECT-IDENTITY MACRO ::=
        BEGIN
                    <clauses> ::= <values>
        END

              --names of objects
              objectName ::=        OBJECT IDENTIFIER
              notificationName ::= OBJECT IDENTIFIER
       -- syntax of objects
                    <objectSyntax Productions>
                    <dataType Productions>
       -- definition of objects
        OBJECT-TYPE MACRO ::=
        BEGIN
              <clauses> ::= <values>
        END
       -- definition for notification
        NOTIFICATION-TYPE MACRO ::=
        BEGIN
              <clauses> ::= <values>
        END
      -- definition of administration identifiers
            zeroDotZero ::= { 0 0 } -- a value for null identifiers
END
```

Figure 6.4 Definitions of SMI for SNMPv2 (Skeleton)

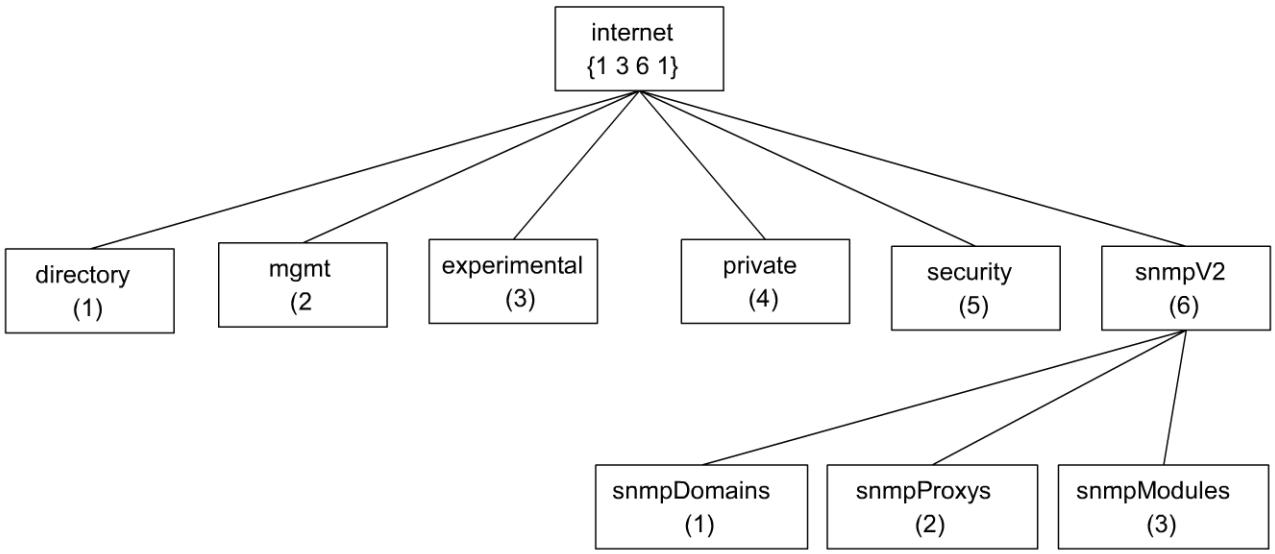

Figure 6.5 SNMPv2 Internet Nodes Defined in SMIv2

the new keywords for specific applications as we discuss them. It is worth noting here that some of the general keywords have been replaced with limited keywords. Thus, Counter is replaced by Counter32, Gauge by Gauge32, and INTEGER by Integer32. The NetworkAddress is deleted from use and only IpAddress is used.

It is also to be noted that reference in IMPORTS clause or in clauses of SNMPv2 macros to an informational module is not through "descriptor" as it was in version 1. It is referenced through specifying its module name, an enhancement in SNMPv2.

It should be observed that the expansion of the ASN.1 module macro occurs during the implementation phase of a product, and not at run-time.

6.3.4 Module Definitions

The MODULE-IDENTITY macro is added to SMIv2 specifying an informational module. It provides administrative information regarding the informational module as well as revision history. SMIv2 MODULE-IDENTITY macro is presented in Figure 6.7.

Figure 6.8 shows an example of a MODULE-IDENTITY macro (a real-world example of a non-existent module) for a network component vendor, InfoTech Services, Inc. (isi), which is updating their private-enterprises-isi MIB module {private.enterprises.isi}.

The last updated clause is mandatory and contains the date and time in UTC time format [RFC 1902]. "Z" refers to Greenwich Mean Time. The Text clause uses the NVT ASCII character set [RFC 854], which is a printable set. All clauses, except the Revision clause, must be present in the macro.

6.3.5 Object Definitions

The OBJECT-IDENTITY macro has been added in SMIv2 and is used to define information about an OBJECT-IDENTIFIER. It is presented in Figure 6.9. The STATUS clause has one of three values: current, deprecated, or obsolete. The value *mandatory* in SMIv1 is replaced with the value *current* in SMIv2. The value *optional* is not used in SMIv2. The new value, *deprecated*, has been added to define

```
SNMPv2-MIB DEFINITIONS ::=
BEGIN
        …               …               …               …
snmpMIB     MODULE IDENTITY ::= {snmpModules 1}
        …               …               …               …
snmpMIBObjects    OBJECT IDENTIFIER ::= {snmpMIB 1}

-- the SNMP group
snmp                OBJECT IDENTIFIER ::= {mib-2 11}
snmpInPkts                  OBJECT-TYPE ::= {snmp 1}
snmpOutPkts                 OBJECT-TYPE ::= { snmp 2}
        …               …               …               …
snmpSet             OBJECT IDENTIFIER ::= {snmpmibObjects 6}
snmpSetSerialNo             OBJECT-TYPE ::= { snmpSet 1}

-- conformance information
snmpMIBConformance
OBJECT IDENTIFIER ::=  {snmpMIB 2}
snmpMIBCompliances
                    OBJECT IDENTIFIER  ::=  {snmpMIBConformance 1}
snmpMIBGroups     OBJECT IDENTIFIER  ::=  {snmpMIBConformance 2}

-- compliance statements

snmpBasicCompliance MODULE-COMPLIANCE
        STATUS      current
        DESCRIPTION
                    "The compliance statement for SNMPv2 entities which
                    implement the SNMPv2 MIB."
        MODULE      -- this module
                MANDATORY-GROUPS  {snmpGroup, snmpSetGroup,
                                   systemGroup,
                                   snmpBasicNotificationsGroup}
                GROUP       snmpCommunityGroup
                DESCRIPTION
                    "This group is mandatory for SNMPv2 entities which support
                    community-based authentication."
        ::=  {snmpMIBCompliances  2 }

-- units of conformance
snmpGroup   OBJECT-GROUP    ::= {snmpMIBGroups 8}
snmpCommunityGroup      OBJECT-GROUP ::= {snmpMIBGroups 9}
snmpObsoleteGroup OBJECT-GROUP ::=  {snmpMIBGroups 10}
        …               …               …               …

END
```

Figure 6.6 Example of the SNMP Group including Conformance and Compliance in SNMPv2 MIB

Table 6.1 SNMP Keywords

KEYWORD	SNMPV1	SNMPV2
ACCESS	Y	Y
AGENT-CAPABILITIES	N	Y
AUGMENTS	N	Y
BEGIN	Y	Y
BITS	N	Y
CONTACT-INFO	N	Y
CREATION-REQUIRES	N	Y
Counter	Y	N
Counter32	N	Y
Counter64	N	Y
DEFINITIONS	Y	Y
DEFVAL	Y	Y
DESCRIPTION	Y	Y
DISPLAY-HINT	N	Y
END	Y	Y
ENTERPRISE	Y	N
FROM	Y	Y
GROUP	N	Y
Gauge	Y	N
Gauge32	N	Y
IDENTIFIER	Y	Y
IMPLIED	N	Y
IMPORTS	Y	Y
INCLUDES	N	Y
INDEX	Y	Y
INTEGER	Y	Y
Integer32	N	Y
IpAddress	Y	Y
LAST-UPDATED	N	Y
MANDATORY-GROUPS	N	Y
MAX-ACCESS	N	Y
MIN-ACCESS	N	Y
MODULE	N	Y
MODULE-COMPLIANCE	N	Y
MODULE-IDENTITY	N	Y
NOTIFICATION-GROUP	N	Y

Table 6.1 (continued)

KEYWORD	SNMPV1	SNMPV2
NOTIFICATION-TYPE	N	Y
NetworkAddress	Y	N
OBJECT	Y	Y
OBJECT-GROUP	N	Y
OBJECT-IDENTITY	N	Y
OBJECT-TYPE	Y	Y
OBJECTS	N	Y
OCTET	Y	Y
OF	Y	Y
ORGANIZATION	N	Y
Opaque	Y	Y
PRODUCT-RELEASE	N	Y
REFERENCE	Y	Y
REVISION	N	Y
SEQUENCE	Y	Y
SIZE	Y	Y
STATUS	Y	Y
STRING	Y	Y
SUPPORTS	N	Y
SYNTAX	Y	Y
TEXTUAL-CONVENTION	N	Y
TRAP-TYPE	Y	N
TimeTicks	Y	Y
UNITS	N	Y
Unsigned32	N	Y
VARIABLES	Y	N
VARIATION	N	Y
WRITE-SYNTAX	N	Y

objects that are required to be implemented in the current version, but may not exist in future versions of SNMP. This allows for backward compatibility during the transition between versions.

Although the REFERENCE clause was used only in an OBJECT-TYPE construct in SMIv1, it is used in many constructs in version 2.

Let us extend our hypothetical example of InfoTech Services and suppose that ISI makes a class of router products. It is given an OBJECT IDENTIFIER as isiRouter OBJECT IDENTIFIER ::= {private. enterprises.isi 1}. The class of router products can be specified at a high level using the OBJECT-IDEN-

```
MODULE-IDENTITY MACRO ::=
BEGIN
        TYPE NOTATION ::=
                                "LAST-UPDATED" value (Update UTCTime)
                                "ORGANIZATION" Text
                                "CONTACT-INFO" Text
                                "DESCRIPTION" Text
                                RevisionPart
        VALUE NOTATION ::=
                                value (VALUE OBJECT IDENTIFIER)
        RevisionPart ::=        Revisions | empty
        Revisions ::= Revision | Revisions Revision
        Revision ::=
                "REVISION" value (UTCTime)
                "DESCRIPTION" Text
        -- uses the NVT ASCII character set
        Text ::= """" string """"
END
```

Figure 6.7 MODULE-IDENTITY Macro

```
isiMIBModule        MODULE-IDENTITY
        LAST-UPDATED    "9802101100Z"
        ORGANIZATION    "InfoTech Services, Inc."
        CONTACT-INFO    "Mani Subramanian
                        Tele: 770-111-1111
                        Fax: 770-111-2222
                        email: manis@bellsouth.net"
        DESCRIPTION     "Version 1.1 of the InfoTech Services MIB module"
        Revision        "9709021500Z"
        DESCRIPTION     "Revision 1.0 on September 2, 1997 was a draft
                        version"
```

Figure 6.8 Example of MODULE-IDENTITY Macro

TITY macro as shown in Figure 6.10(a). The status of the *isiRouter* is current and is described as an 8-slot IP router. A reference is given for obtaining the details.

A specific implementation of the router in *isiRouter* class of products is *routerIsi123*. This is a managed object specified by the OBJECT-TYPE macro shown in Figure 6.10(b). We are already familiar with the OBJECT-TYPE macro by now.

Let us make sure that we clearly understand the terminology used with the term OBJECT. OBJECT IDENTIFIER defines the administrative identification of a node in the MIB. The OBJECT IDENTITY macro is used to assign an object identifier value to the object node in the MIB. The OBJECT-TYPE is a macro that defines the *type* of a managed object. It is also used to describe a new type of object. As we have learned in the previous chapters, an *object instance* is a specific instance of the *object (type)*. Thus, a specific instance of the *routerIsi123* could be identified by its IP address 10.1.2.3.

```
OBJECT-IDENTITY MACRO ::=
BEGIN
        TYPE NOTATION ::=
                        "STATUS"            Status
                        "DESCRIPTION"       Text
                        ReferPart

        VALUE NOTATION ::=
                        value (VALUE OBJECT IDENTIFIER)

        Status ::=          "current" | "deprecated" | "obsolete"
        ReferPart ::=       "REFERENCE" Text | empty
        Text ::=            """"string """"
END
```

Figure 6.9 OBJECT-IDENTITY Macro

```
isiRouter      OBJECT-IDENTITY
        STATUS              current
        DESCRIPTION         "An 8-slot IP router in the IP router family."
        REFERENCE           "ISI Memorandum No. ISI-R123 dated January 20,
                            1997"
        ::= {private.enterprises.isi 1}
```

(a) Example of an OBJECT-IDENTITY Macro

```
routerIsi123   OBJECT-TYPE
        SYNTAX              DisplayString
        MAX-ACCESS          read-only
        STATUS              current
        DESCRIPTION         "An 8-slot IP router that can switch up to
                            100 million packets per second."
        ::= {isiRouter 1}
```

(b) Example of an OBJECT-TYPE Macro

Figure 6.10 Example of OBJECT-IDENTITY and OBJECT-TYPE Macros

Comparing Figure 6.10(a) with Figure 6.10(b) we observe the difference between OBJECT-IDENTITY and OBJECT-TYPE. The status clause appears in both. The description clause that also appears in both describes different aspects of the object. The OBJECT-IDENTITY describes the high-level description; whereas the OBJECT-TYPE description focuses on the details needed for implementation.

Let us now visualize the router in Figure 6.10 with several slots for interface cards. We want to define the parameters associated with each interface. The parameters that are managed objects (or entities) are defined by an aggregate object, *IfTable*. For example, the *ifNumber* for our router example could be 32 if the router has eight slots and each card has four ports.

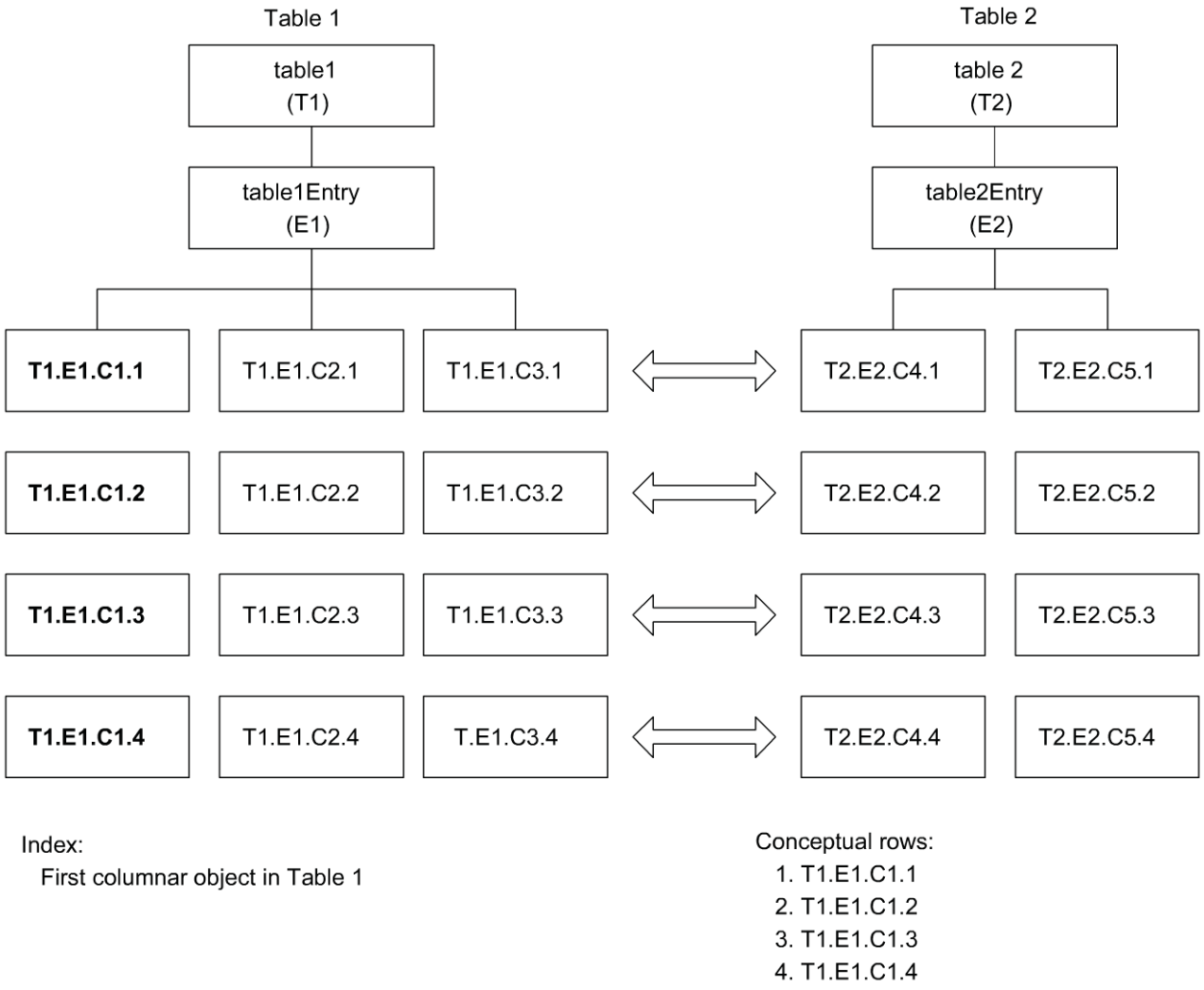

Figure 6.11 Augmentation of Tables

SMIv2 extends the concept table for an aggregate object from a single table to multiple tables. This allows for expansion of managed objects when the number of columnar objects needs to be increased, or when the objects are best organized by grouping them hierarchically. Let us first consider the case of adding columnar objects to an existing table with the following restrictions: (a) the number of conceptual rows is not affected by the addition; (b) there is one-to-one correspondence between the rows of the two tables; and (c) the INDEX of the second table is the same as that of the first table. This is shown in Figure 6.11.

Table 1 is called the aggregate object *table1* and has three columns and four rows; and Table 2 is called the aggregate object *table2* and has two columns and four rows. There is a one-to-one correspondence in rows between the two tables. The row object for table1 is *table1Entry*, and the row object for table 2 is *table2Entry*. The INDEX is defined in Table 1 for both tables and it is the columnar object T1.E1.C1. We are using the notations T1, E1, C1, etc., for easier visual conceptualization of the instance of an object in a table using the prefixes of table ID (e.g., T1) and entry (e.g., E1). The columnar object notation starts with C (e.g., C1). The value or values suffixed with the columnar object identifier uniquely identifies the row. Thus, the list of objects identified by the index T1.E1.C1.2 is the ones in the second rows of

```
table1Entry OBJECT-TYPE
        SYNTAX          TableT1Entry
        MAX-ACCESS      not-accessible
        STATUS          current
        DESCRIPTION     "An entry (conceptual row) in table T1"
        INDEX           {T1.E1.C1}
                ::= { table1 1}
table2Entry OBJECT-TYPE
        SYNTAX          TableT2Entry
        MAX-ACCESS      not-accessible
        STATUS          current
        DESCRIPTION     "An entry (conceptual row) in table T2"
        AUGMENTS        {table1Entry}
                ::= {table2 1}
```

Figure 6.12 ASN.1 Constructs for Augmentation of Tables

Tables 1 and 2. The value of the columnar object T2.E2.C4 in Table T2 corresponding to index T1.E1.C1.2 is T2.E2.C4.2. Table 1 is called the **base table**, and Table 2 is the **augmented table**. The indexing scheme comprises two clauses, the INDEX clause and the AUGMENTS clause. The constructs for the rows of the two tables in Figure 6.11 are shown in Figure 6.12. The object *table1Entry* has the INDEX clause and *table2Entry* has the AUGMENTS clause that refers to *table1Entry*. The combination of the two tables still provides four conceptual rows, T1.E1.C1.1 through T1.E1.C1.4 (identified by the index), the same number of rows as in the base table.

Figure 6.13 shows an example of augmentation of tables. We have augmented *ipAddrTable* in the standard MIB with a proprietary table, *IpAugAddrTable* that could add additional information to the rows of the table. *IpAddrTable* is the base table and *ipAugAddrTable* is the augmented table. In a practical case, the *ipAugAddrTable* could add two more columnar objects defining the board and port number associated with the *ipAdEntIfIndex*.

A table with a larger number of rows (**dense table**) can be augmented to the base table with combined indices of both, as shown in Figure 6.14. The INDEX clause for combining unequal-sized tables is the combined indices; i.e., combined columnar objects as the INDEX clause for the added aggregate object. In Figure 6.14, Table 1 consists of two rows and three columnar objects, T1.E1.C1, T1.E1.C2, and T1.E1.C3, with the first columnar object T1.E1.C1 being the index. Table 2 has four rows and two columnar objects, T2.E2.C4 and T2.E2.C5, with its first columnar object, T2.E2.C4, being the index. The combined index for specifying the aggregate object of Table 2 appended to Table 1 is the set of both first columnar objects, T1.E1.C1 and T2.E2.C4. Table 1 is called the **base table** and Table 2 is called the **dependent table**. As we see in Figure 6.14, the combined base table and the dependent table could have a maximum of 8 conceptual rows (multiplication of the rows of the two tables).

Figure 6.15 shows the constructs for augmenting a dense table to a base table. The two table objects, *table1* and *table2*, are nodes under the node table. The *table1Entry* defines a row in *table1* with the columnar object T1.E1.C1 as the index. The *table2Entry* is a row in *table2*. Its index is defined by the indices of both tables, namely T1.E1.C1 and T2.E2.C3.

We can visualize the application of augmentation of a dense table with an example of a router with multiple slots, each slot containing a particular type of board, for example, LEC and Ethernet shown in

```
ipAddrTable  OBJECT-TYPE
        SYNTAX          SEQUENCE OF IpAddrEntry
        MAX-ACCESS      not-accessible
        STATUS          current
        DESCRIPTION     "The table …"
        ::= {ip 20}

ipAddrEntry   OBJECT-TYPE
        SYNTAX          IpAddrEntry
        MAX-ACCESS      not-accessible
        STATUS          current
        DESCRIPTION     "The addressing information"
        INDEX           {ipAdEntAddr}
        ;;= {ipAddrTable 1}

ipAugAddrTable      OBJECT-TYPE
        SYNTAX          SEQUENCE OF IpAugAddrEntry
        MAX-ACCESS      not-accessible
        STATUS          current
        DESCRIPTION     "The augmented table to IP Address Table defining
                        board and port numbers"
        ::= {ipAug 1}

ipAugAddrEntry      OBJECT-TYPE
        SYNTAX          IpAugAddrEntry
        MAX-ACCESS      not-accessible
        STATUS          current
        DESCRIPTION     "The addressing information…"
        AUGMENTS        {ipAddrEntry}
        ::= {ipAugAddrTable 1}
```

Figure 6.13 Example of Augmentation of Tables

Figure 4.3(c). The slot and the board type will be defined in Table 1. Each board may have a different number of physical ports. The port configuration is defined by Table 2. By using the combination of the two tables, we can specify the details associated with a given port in a given slot.

The third possible scenario in appending an aggregate object to an existing aggregate object is the case where the augmented table has fewer rows than that of the base table. This is called a **sparse dependent table** case and is shown in Figure 6.16. In this example, the index for the second table is the same as that for the base table and the constructs are similar to the ones shown in Figure 6.12 except that the AUGMENTS clause is substituted with the INDEX clause for *table2Entry*. This is shown in Figure 6.17.

In SNMPv2, operational procedures were introduced for the creation and deletion of a row in a table. However, prior to discussing these procedures, let us first look at the textual convention that was specified to create a new object type in designing MIB modules. We will return to row creation and deletion in Section 6.3.7.

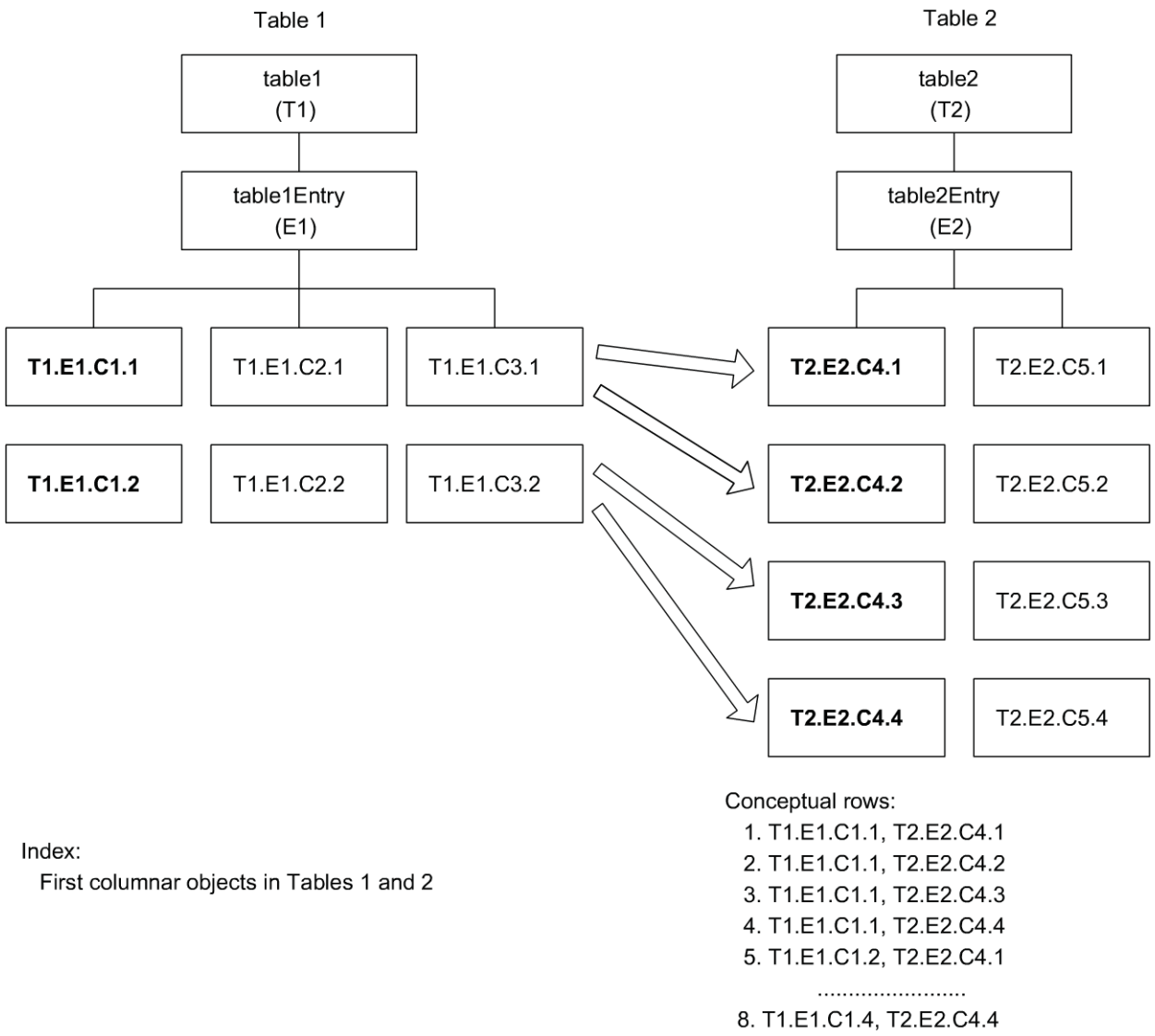

Figure 6.14 Combined Indexing of Tables

6.3.6 Textual Conventions

Textual conventions are designed to help definition of new data types following the structure defined in SMIv2. It is also intended to make the semantics consistent and clear to the human reader. Although new data types could have been created using new ASN.1 class and tag, the decision was made to use the existing defined class types and apply restrictions to them. This is accomplished by defining an ASN.1 macro, TEXTUAL-CONVENTION, in SMIv2.

The TEXTUAL-CONVENTION macro concisely conveys the syntax and semantics associated with a textual convention. SNMP-based management objects defined using a textual convention are encoded by the same Basic Encoding Rules that define their primitive types. However, they do have the special semantics as defined in the macro. For all textual conventions defined in an information module, the name shall be unique and mnemonic, similar to the data type and shall not exceed 64 characters. However, it is usually limited to 32 characters.

```
table1 OBJECT-TYPE
        SYNTAX          SEQUENCE OF table1Entry
        MAX-ACCESS      not-accessible
        STATUS          current
        DESCRIPTION     "Table 1 under T"
                ::= {  table 1}

table1Entry OBJECT-TYPE
        SYNTAX          Table1Entry
        MAX-ACCESS      not-accessible
        STATUS          current
        DESCRIPTION     "An entry (conceptual row) in Table 1"
        INDEX           {T1.E1.C1}
                ::= {table1 1}

table2 OBJECT-TYPE
        SYNTAX          SEQUENCE OF table2Entry
        MAX-ACCESS      not-accessible
        STATUS          current
        DESCRIPTION     "Table 2 under T"
                ::= {table 2}

table2Entry OBJECT-TYPE
        SYNTAX          Table2Entry
        MAX-ACCESS      not-accessible
        STATUS          current
        DESCRIPTION     "An entry (conceptual row) in Table 2"
        INDEX           {T1.E1.C1, T2.E2.C4}
                ::= {table2 1}
```

Figure 6.15 ASN.1 Constructs for Augmenting Dense Table

Let us now compare the definition of a type in SMIv1 with SMIv2. The textual convention was defined in SNMPv1 as an ASN.1 type assignment. For example, the textual convention for data type *DisplayString* in SNMPv1, from RFC 1213, is

```
DisplayString ::= OCTET STRING
-- This data type is used to model textual information taken from the NVT
-- ASCII character set. By convention, objects with this syntax are
-- declared as having
-- SIZE (0..255).
```

The same example of DisplayString in SNMPv2 is defined as:

```
DisplayString ::= TEXTUAL-CONVENTION
        DISPLAY-HINT  "255a"
        STATUS        current
        DESCRIPTION   "Represents textual information taken from the NVT ASCII character
                      set, as defined in pages 4, 10-11 of RFC 854. …."
        SYNTAX        OCTET STRING (SIZE (0..255) )
```

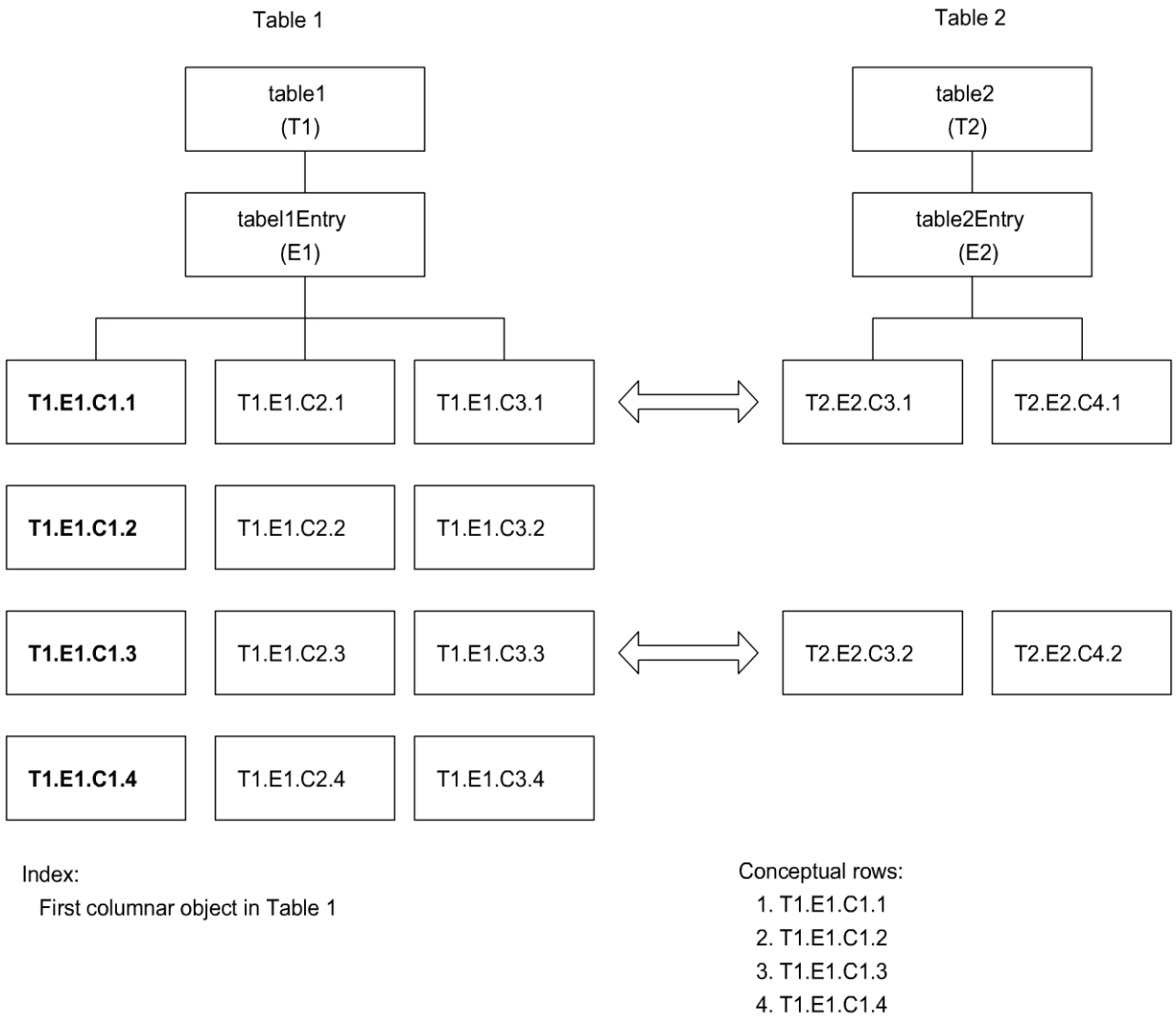

Figure 6.16 Addition of a Sparse Table to a Base Table

As we can see from the above example, the TEXTUAL-CONVENTION in SNMPv2 is defined as data type, and is used to convey the syntax and semantics of a textual convention. The macro for textual conventions is defined in RFC 1903, and a skeleton of it is presented in Figure 6.18. It has the definition of type and value notations with the formalized definition of data types.

All clauses except *DisplayPart* in the TEXTUAL-CONVENTION macro are self-explanatory and represent similar clauses as in SMIv1. The *DISPLAY-HINT* clause, which is optional, gives a hint as to how the value of an instance of an object, with the syntax defined using this textual convention, might be displayed. It is applicable to the situations where the underlying primitive type is either INTEGER or OCTET STRING.

For INTEGER type, the display consists of two parts. The first part is a single character denoting the display format: "a" for ASCII, "b" for binary, "d" for decimal, "o" for octal, and "x" for hexadecimal. It is followed by a hyphen and an integer in the case of decimal display indicating the number of decimal points. For example, a hundredths value of 1234 with DISPLAY-HINT "d-2" is displayed as 12.34.

For OCTET-STRING type, the display hint consists of one or more octet-format specifications. A brief description of each part is shown in Table 6.2. For example, the DISPLAY-HINT "255a" indicates that the *DisplayString* is an ASCII string of up to a maximum of 255 characters.

```
table1 OBJECT-TYPE
        SYNTAX          SEQUENCE OF table1Entry
        MAX-ACCESS      not-accessible
        STATUS          current
        DESCRIPTION     "Table 1 under T"
                ::= { table 1}

table1Entry OBJECT-TYPE
        SYNTAX          Table1Entry
        MAX-ACCESS      not-accessible
        STATUS          current
        DESCRIPTION     "An entry (conceptual row) in Table 1"
        INDEX           {T1.E1.1}
                ::= {table1 1}

table2 OBJECT-TYPE
        SYNTAX          SEQUENCE OF table2Entry
        MAX-ACCESS      not-accessible
        STATUS          current
        DESCRIPTION     "Table 2 under T"
                ::= {table 2}

table2Entry OBJECT-TYPE
        SYNTAX          Table2Entry
        MAX-ACCESS      not-accessible
        STATUS          current
        DESCRIPTION     "An entry (conceptual row) in Table 2"
        INDEX           {table1Entry}
                ::= {table2 1}
```

Figure 6.17 ASN.1 Constructs for Augmenting Sparse Table

```
TEXTUAL-CONVENTION MACRO ::=
BEGIN
        TYPE NOTATION ::=
                                DisplayPart
                                "STATUS"  Status
                                "DESCRIPTION"  Text
                                ReferPart
                                "SYNTAX"  Syntax

        VALUE NOTATION ::=
                                value (VALUE Syntax)

         DisplayPart ::=        "DISPLAY-HINT" Text | empty
         Status ::=             "current" | "deprecated" | "obsolete"
                        ……
END
```

Figure 6.18 TEXTUAL-CONVENTION Macro [RFC 1903]

Table 6.2 DISPLAY-HINT for Octet-Format

1	(Optional) repeat indicator “*””	An integer, indicated by *, which specifies how many times the remainder of this octet-format should be repeated
2	Octet length	One or more decimal digits specifying the number of octets
3	Display format	“b” for binary, “x” for hexadecimal, “d” for decimal, “o” for octal, and “a” for ASCII for display
4	(Optional) display separator character	A single character other than a decimal digit or “*” produced after each application of the octet specification.
5	(Optional) repeat terminator character	A single character other than a decimal digit or “*” present if display character is present. Produced after the second and third part.

Table 6.3 shows the types for which textual conventions were specified in SMIv2. A brief description for each type is also given. They are applicable to all MIB modules. Only those textual conventions whose status is current are given in the table. One of the important textual conventions is *RowStatus*, which is used for the creation and deletion of conceptual rows, which we will discuss next.

Table 6.3 SMIv2 Textual Conventions for Initial Data Types

DisplayString	Textual information from NVT ASCII character set [RFC 854]
PhysAddress	Media- or physical-level address
MacAddress	IEEE 802 MAC address
TruthValue	Boolean value; INTEGER {true (1), false (2)}
TestAndIncr	Integer-valued information used for atomic operations
AutonomousType	An independently extensible type identification value
VariablePointer	Pointer to a specific object instance; e.g., syscontact.0, ifInOctets.3
RowPointer	Pointer to a conceptual row
RowStatus	Used to manage the creation and deletion of conceptual rows and is used as the value of the SYNTAX clause for the status column of a conceptual row
TimeStamp	Value of sysUpTime at which a specific occurrence happened
TimeInterval	Period of time, measured in units of 0.01 seconds
DateandTime	Date–time specifications
StorageType	Implementation information on the memory realization of a conceptual row as to the volatility and permanency
Tdomain	Kind of transport service
Taddress	Transport service address

6.3.7 Creation and Deletion of Rows in Tables

The creation of a row and deletion of a row are significant new features in SMIv2. This is patterned after a similar procedure that was developed for RMON, which we will cover in Chapter 8. There are two methods to create a row in a table. The first is to create a row and make it active, which is available immediately. The second method is to create the row and make it available at a later time. This means that we need to know the status of the row as to its availability.

The information on the status of the row is accomplished by introducing a new column, called the *status* column. In Table 6.3, we observe that for the textual convention, RowStatus is used as the value of the SYNTAX clause for the *status* column of a conceptual row. Table 6.4 shows the status with enumerated integer syntax for the six states associated with the row status. The last three states, along with the first one (1, 4, 5, and 6), are those that the manager uses to create or delete rows on the agent. The first three states (1, 2, and 3) are those that are used by the agent to send responses to the manager.

The MAX-ACCESS clause is extended to include "read-create" for the *status* object, which includes read, write, and create privileges. It is a superset of read-write. If a *status* columnar object is present, then no other columnar object of the same conceptual row may have a maximal access of "read-write." But it can have objects with maximum access of read-only and not-accessible. If an index object of a conceptual row is also a columnar object (it does not always have to be), it is called *auxiliary object* and its maximum access is made non-accessible. There could be more than one index object to define a conceptual row in a table.

Let us now analyze the create and delete operations using the conceptual table shown in Figure 6.19. The table, *table1*, originally has two rows and three columns. The first column, *status*, has the value of the status of the row as indicated by the enumerated integer syntax of RowStatus textual convention. The second columnar object, *index*, is the index for the conceptual row of *entry1*; and the third columnar object contains non-indexed data. We will illustrate the two types of row-creation and row-deletion operations by adding a third row and then deleting it.

As we notice from Table 6.4, there are two states for RowStatus, *createAndGo* and *createAndWait*, which are action operations. In the former, the manager sends a message to the agent to create a row and make the *status* active immediately. In the latter operation, the manager sends a message to create a

Table 6.4 RowStatus Textual Convention

STATE	ENUMERATION	DESCRIPTION
active	1	Row exists and is operational
notInService	2	Operation on the row is suspended
notReady	3	Row does not have all the columnar objects needed
createAndGo	4	This is a one-step process of creation of a row, immediately goes into the active state
createAndWait	5	Row is under creation and should not be commissioned into service
destroy	6	Same as Invalid in EntryStatus. Row should be deleted

row, but not to make it active immediately. Figure 6.20 shows the Create-and-Go operation. The manager process initiates a Set-Request-PDU to create a conceptual row with the values given for the three columnar instances of the row. The value for the index column is specified by the VarBind *index* = 3. This is suffixed to the other two columnar objects in the new row to be created. The value of *status* is specified as 4, which is the *createAndGo* state as seen in Table 6.4. The *set-request* message also specifies the default value *DefData* for *data.3*, and thus all the information needed to establish the row and

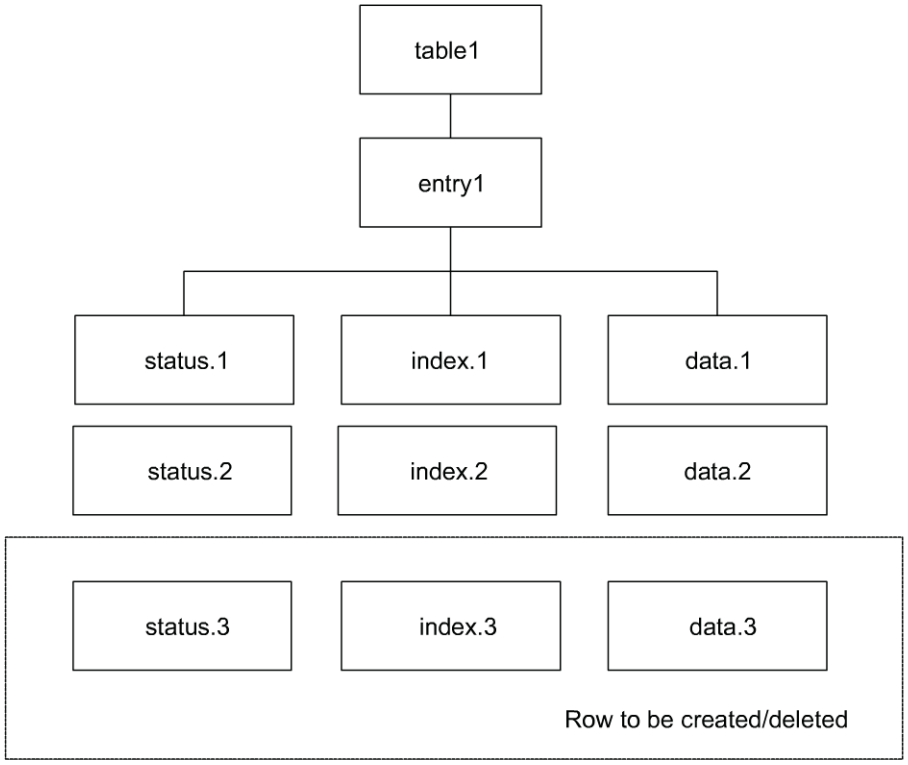

Figure 6.19 Conceptual Table for the Creation and Deletion of a Row

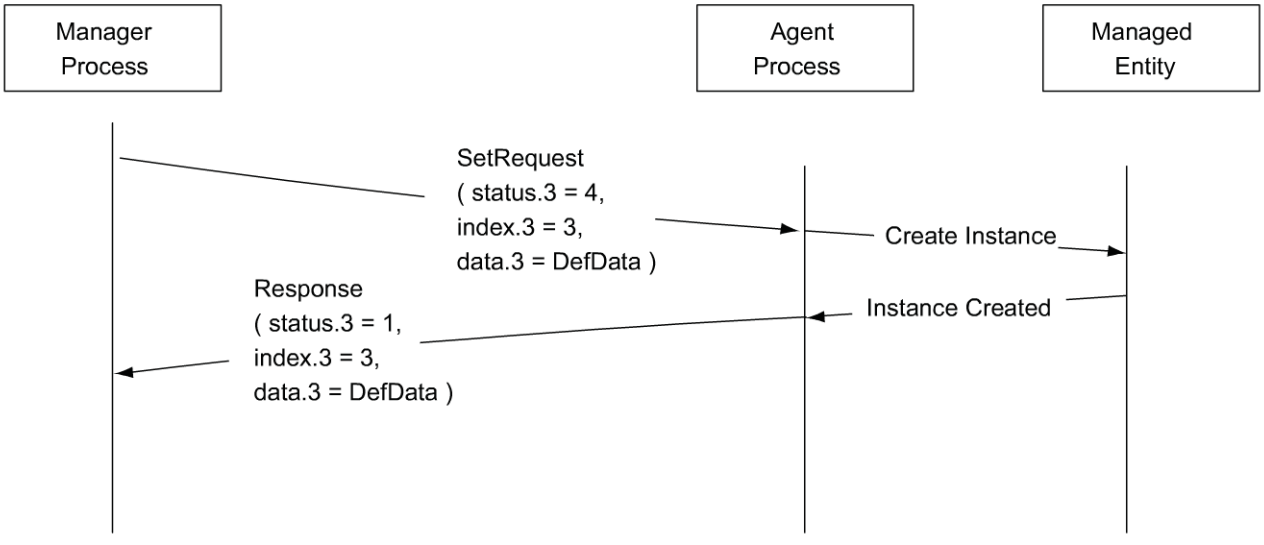

Figure 6.20 Create-and-Go Row Creation

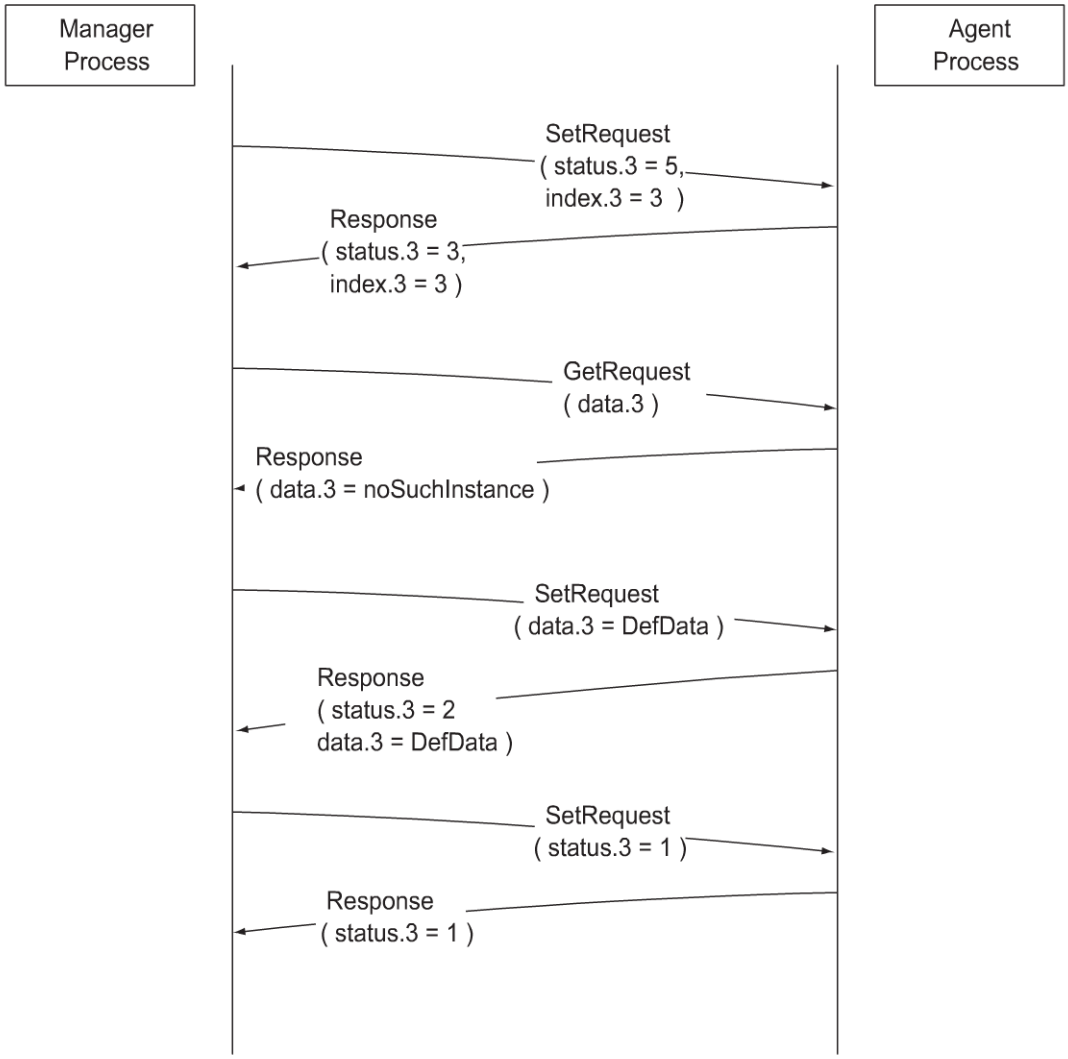

Figure 6.21 Create-and-Wait Row Creation

turn it into an active state is complete. The agent process interacts with the managed entity, creates the instance successfully, and then transmits a response to the manager process. The value of the *status* is 1, which denotes that the row is in an active state. The response also contains the values of the other columnar object instances.

Figure 6.21 presents a scenario for operational sequence in the creation of a row using the Create-and-Wait method. Again, this illustration takes the same scenario of adding the third row to the table shown in Figure 6.17. Only the manager and the agent are shown and not the managed entity in this figure. The manager process sends a Set-Request-PDU to the agent process. The value for *status* is 5, which is to create and wait. The third columnar object expects a default value, which is not in the set-request message. Hence, the agent process responds with a *status* value of 3, which is *notReady*. The manager sends a get-request to get the data for the row. The agent responds with *noSuchInstance* message, indicating that the data value is missing. The manager subsequently sends the value for *data* and receives a response of *notInService* (2) from the agent. The fourth and final exchange of messages in the figure is to activate the row with a *status* value of 1. With each message received from the manager, the agent either validates or sets the instance value on the managed entity.

Table 6.5 Table of States for Row Creation and Deletion

ACTION	A STATUS COLUMN DOES NOT EXIST	B STATUS COLUMN NOTREADY	C STATUS COLUMN NOTINSERVICE	D STATUS COLUMN ACTIVE
Set status column to createAndGo	noError -> D	inconsistent-Value	inconsistent-Value	inconsistent-Value
Set status column to createAndWait	noError, see 1 or wrongValue	inconsistent-Value	inconsistent-Value	inconsistent-Value
Set status column to active	inconsistent-Value	inconsistent-Value or see 2 ->D	noError ->D	noError ->D
Set status column to notInService	inconsistent-Value	inconsistent-Value or see 3 -> C	noError ->C	noError ->C or wrongValue
Set status column to destroy	noError ->A	noError ->A	noError ->A	noError ->A
Set any other column to some value	see 4	noError see 1	noError ->C	see 5 ->D

A summary of possible state transitions is given in Table 6.5. The first column lists the action; and the transitions based on the present state are listed in the next four columns.

1. goto B or C, depending on information available to the agent.
2. If other variable bindings included in the same PDU provide values for all columns, which are missing but are required, then return noError and goto D.
3. If other variable bindings included in the same PDU provide values for all columns, which are missing but are required, then return noError and goto C.
4. At the discretion of the agent, the return value may be either: *inconsistentName*: because the agent does not choose to create such an instance when the corresponding RowStatus instance does not exist, or *inconsistentValue*: if the supplied value is inconsistent with the state of some other MIB object's value, or *noError*; because the agent chooses to create the instance.

   If noError is returned, then the instance of the status column must also be created, and the new state is B or C, depending on the information available to the agent. If *inconsistentName* or *inconsistent Value* is returned, the row remains in state A.
5. Depending on the MIB definition for the column/table, either *noError* or *inconsistentValue* may be returned.

   NOTE: Other processing of the set request may result in a response other than *noError* being returned, e.g., wrongValue, *noCreation*, etc.

The operation of deletion of a row is simple. A *set-request* with a value of 6, which denotes *destroy,* for *status*, is sent by the manager process to the agent process. Independent of the current state of the row, the row is deleted and the response sent back by the agent. The instance in the managed entity is deleted in the process. This is shown in Figure 6.22.

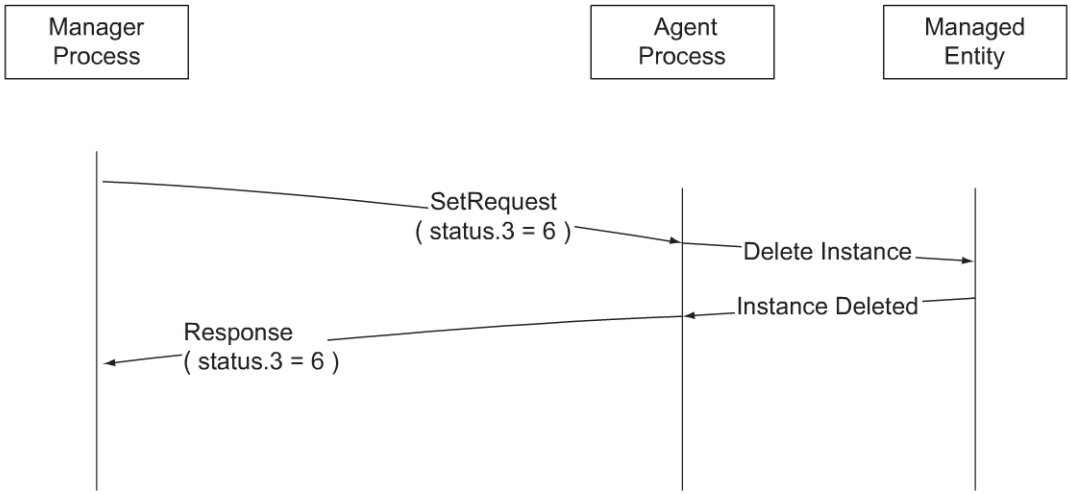

Figure 6.22 Row Deletion

6.3.8 Notification Definitions

The trap information in SMIv1 has been redefined using the NOTIFICATION-TYPE macro in SMIv2. As we will see in Section 6.5, the PDU associated with the trap information is made consistent with other PDUs. The NOTIFICATION-TYPE macro contains unsolicited information that is generated on an exception basis, for example, when set thresholds are crossed. It can be transmitted within either a SNMP-Trap-PDU from an agent or an InformRequest-PDU from a manager. Two examples of a NOTIFICATION-TYPE macro, drawn from RFC 1902 and RFC 1907 are shown in Figure 6.23. The first example, linkUp, is generated by an agent when a link that has been down comes up.

The OBJECTS clause defines the ordered sequence of MIB objects, which are included in the notification. It may or may not be present. The second example, coldStart, in Figure 6.23, has the OBJECTS clause missing and is not needed.

```
linkUp NOTIFICATION-TYPE
        OBJECTS     {ifIndex}
        STATUS      current
        DESCRIPTION
                "A linkUp trap signifies that the SNMPv2 entity, acting in an agent
                role, recognizes that one of the communication links represented in
                its configuration has come up."
        ::= {snmpTraps 4}

coldStart NOTIFICATION-TYPE
        STATUS      current
        DESCRIPTION
                "A coldStart trap signifies that the SNMPv2 entity, acting in
                an agent role, is reinitializing itself such that its configuration is
                unaltered."
        ::= {snmpTraps 1}
```

Figure 6.23 Examples of NOTIFICATION-TYPE Macro

The other two clauses, STATUS and DESCRIPTION, have the usual mappings.

We have not presented here discussions on refined syntax in some of the macros, as well as extension to informational modules. You are referred to RFC 1902 for a treatment of these, which also discusses the conversion of a managed object from the OSI to the SNMP version.

6.3.9 Conformance Statements

RFC 1904 defines SNMPv2 conformance statements for the implementation of network management standards. A product, generally, is considered to be in compliance with a particular standard when it meets the minimum set of features in its implementation. Minimum requirements for SNMPv2 compliance are called module compliance and are defined by an ASN.1 macro, MODULE-COMPLIANCE. It specifies the minimum MIB modules or a subset of modules that should be implemented. The actual MIB modules that are implemented in an agent are specified by another ASN.1 module, AGENT-CAPABILITIES. For the convenience of defining module compliance and agent capabilities, objects and traps have been combined into groups, which are subsets of MIB modules. Object grouping is defined by an ASN.1 macro, OBJECT-GROUP, and the group of traps is defined by the NOTIFICATION-GROUP macro.

Object Group. The OBJECT-GROUP macro defines a group of related objects in a MIB module and is used to define conformance specifications. It is compiled during implementation, not at run-time. The macro is shown in Figure 6.24. The implementation of an object in an agent implies that it executes the get and set operations from a manager. If an agent in SNMPv2 has not implemented an object, it returns a noSuchObject error message.

```
OBJECT-GROUP MACRO
BEGIN
        TYPE NOTATION ::=
                                ObjectsPart
                                "STATUS" Status
                                "DESCRIPTION" Text
                                ReferPart

        VALUE NOTATION ::=
                                value (VALUE OBJECT IDENTIFIER)

        ObjectsPart ::=         "OBJECTS" "{"objects"}"
        Objects ::=             Object | Objects "," Object
        Object ::=              value (Name  Object Name)
        Status ::=              "current" | "deprecated" | "obsolete"
        ReferPart ::=           "REFERENCE" Text  | empty

        -- uses the NVT ASCII character set
        Text ::=                """" string """"
END
```

Figure 6.24 OBJECT-GROUP Macro

```
systemGroup         OBJECT-GROUP
        OBJECTS     {sysDescr, sysObjectID, sysUpTime, sysContact, sysName,
                     sysLocation, sysServices, sysORLastChange, sysORID,
                     sysORUptime, sysORDesc}
        STATUS      current
        DESCRIPTION      "The system group defines objects that are common
                          to all managed systems."
        ::= {snmpMIBGroups 6}
```

Figure 6.25 Example of an OBJECT-GROUP Macro

The OBJECTS clause names each object contained in the conformance group. Each of the named objects is defined in the same informational module as the OBJECT-GROUP macro and has a MAX-ACCESS clause of "accessible-for-notify," "read-only," "read-write," or "read-create." Every object that is defined in an informational module with a MAX-ACCESS clause other than "not-accessible" is pres-ent in at least one object group. This prevents the mistake of adding an object to an information module, but forgetting to add it to a group.

The STATUS, DESCRIPTION, and REFERENCE clauses have the usual interpretations.

An example of an OBJECT-GROUP, systemGroup in SNMPv2, is shown in Figure 6.25. The system group defines the objects, which pertain to overall information about the system. Since it is so basic, it is implemented in all agent and management systems. All seven entities defined as values for OBJECTS should be implemented. There are some new entities, such as *sysORLastChange*, in the group that were not in SNMPv1. These will be addressed when we discuss SMPv2 MIB in the next section.

Notification Group. The notification group contains notification entities, or what was defined as traps in SMIv1. The NOTIFICATION-GROUP macro is shown in Figure 6.26. The macro is compiled during implementation, not during run-time. The value of an invocation of the NOTIFICATION-GROUP macro is the name of the group, which is an OBJECT IDENTIFIER.

An example of NOTIFICATION-GROUP, *snmpBasicNotificationsGroup*, is shown in Figure 6.27. According to this invocation, the conformance group, *snmpBasicNotificationsGroup*, has two notifications: *coldStart* and *authenticationFailure*.

Module Compliance. The MODULE-COMPLIANCE macro, shown in Figure 6.28, defines the minimum set of requirements for implementation of one or more MIB modules. The expansion of the MODULE-COMPLIANCE macro is done during the implementation and not during run-time. The MODULE-COMPLIANCE macro can be defined as a component of the information module or as a companion module.

The STATUS, DESCRIPTION, and REFERENCE clauses are self-explanatory.

The MODULE clause is used to name each module for which compliance requirements are specified. Modules are identified by the module name and its OBJECT IDENTIFIER. The latter can be dropped if the MODULE-COMPLIANCE is invoked within an MIB module and refers to the encompassing MIB module.

There are two CLAUSES of groups that are specified by the MODULE-COMPLIANCE macro. They are MANDATORY-GROUPS and GROUP. As the name implies, the MANDATORY-CLAUSE

```
NOTIFICATION-GROUP MACRO
BEGIN
        TYPE NOTATION ::=
                              NotificationsPart
                              "STATUS" Status
                              "DESCRIPTION" Text
                              ReferPart

        VALUE NOTATION ::=
                              value (VALUE OBJECT IDENTIFIER)



NotificationsPart ::=         "NOTIFICATIONS" "{"Notifications"}"
Notifications ::=             Notification | Notifications "," Notification
Notification ::=              value (Name NotificationName)

        Status  ::=                   "current" | "deprecated" | "obsolete"
        ReferPart ::=                 "REFERENCE" Text | empty

        -- uses the NVT ASCII character set
        Text ::= """" string """"

END
```

Figure 6.26 NOTIFICATION-GROUP Macro

```
snmpBasicNotificationsGr     oup NOTIFICATION-GROUP
        NOTIFICATIONS              {coldStart, authenticationFailure}
        STATUS                     current
        DESCRIPTION       "The two notifications which an SNMP-2 entity is
                            required to implement."
         ::= {snmpMIBGroups 7}
```

Figure 6.27 Example of a NOTIFICATION-GROUP Macro

modules have to be implemented for the system to be SNMPv2 compliant. The group specified by the GROUP clause is not mandatory for the MIB module, but helps vendors define specifications of the features that have been implemented.

When both WRITE-SYNTAX and SYNTAX clauses are present, restrictions are placed on the syntax for the object mentioned in the OBJECT clause. These restrictions are tabulated in Section 9 of RFC 1902.

The *snmpBasicCompliance* macro is an example of a MODULE-COMPLIANCE macro and is part of the SNMPv2 MIB presented in Figure 6.6. A system is defined as SNMPv2 compliant if and only if *snmpGroup*, *snmpSetGroup*, *systemGroup*, and *snmpBasicNotificationsGroup* are implemented. The GROUP, *snmpCommunityGroup*, is optional.

```
MODULE-COMPLIANCE MACRO
BEGIN
        TYPE NOTATION ::=
                        "STATUS" Status
                        "DESCRIPTION" text
                        ReferPart
                        ModulePart

        VALUE NOTATION ::=
                        value (VALUE OBJECT IDENTIFIER)

        Status  ::=             "current" | "deprecated" | "obsolete"
        ReferPart ::=           "REFERENCE" Text | empty
        ModulePart ::=          Modules | empty
        Modules ::=             Module | Modules Module
        Module ::=              -- name of module --
                                "MODULE" ModuleName
                                Mandatory Part
                                CompliancePart
        ModuleName ::=          moduleReference ModuleIdentifier | empty
-- must not be empty unless contained in MIB module
ModuleIdentifier ::=  value (ModuleID OBJECT IDENTIFIER) | empty
        MandatoryPart ::=       "MANDATORY-GROUPS" "{" Groups"}"
                                | empty
        Groups ::=              Group | Groups "," Group
        Group ::=               value (Group OBJECT IDENTIFIER)
        CompliancePart ::=      Compliances | empty
        Compliances ::=         Compliance | Compliances compliance
        Compliance ::=          ComplianceGroup | Object
        ComplianceGroup ::=     "GROUP" value (Name OBJECT IDENTIFIER)
                                "DESCRIPTION" Text
        Object ::=              "OBJECT" value (Name ObjectName)
                                SyntaxPart
                                WriteSyntaxPart
                                AccessPart
                                "DESCRIPTION" Text
        --must be a refinement for object's SYNTAX clause
        SyntaxPart ::=          "SYNTAX" type (SYNTAX) | empty
        --must be a refinement for object's SYNTAX clause
WriteSyntaxPart ::= "WRITE-SYNTAX" type (WriteSYNTAX) | empty
        AccessPart ::=          "MIN-ACCESS" Access | empty
        Access ::=              "not-accessible" | "accessible-for-notify" |
                                 "read-only" | "read-write" | "read-create"
                                -- uses the NVT ASCII character set
                                Text ::= "" string ""

END
```

Figure 6.28 MODULE-COMPLIANCE Macro

```
AGENT-CAPABILITIES
BEGIN
        TYPE NOTATION ::=
                                "PRODUCT-RELEASE" Text
                                "STATUS" Status
                                "DESCRIPTION" Test
                                ReferPart
                                ModulePart

        VALUE NOTATION  ::=
                                Value (VALUE OBJECT IDENTIFIER)

        Status ::=              "current" | "obsolete"
        ReferPart ::=           "REFERENCE" | empty
        ModulePart ::=          Modules | empty
        Modules ::=             Module | Modules Module
        Module ::=              -- name of module --
                                "SUPPORT" ModuleName
                                "INCLUDES" "{"Groups"}"
                                VariationsPart

        …               …               …               …

END
```

Figure 6.29 AGENT-CAPABILITIES Macro (Skeleton)

Agent Capabilities. The AGENT-CAPABILITIES macro is lengthy and the reader is referred to RFC 1904 for exact specifications. A skeleton of the macro and significant points of the macro are covered here and are shown in Figure 6.29.

The AGENT-CAPABILITIES macro for the router example given in Figure 6.10 is shown in Figure 6.30. Note that *snmpMIB* model, which is SNMPv2-MIB, includes *system* and *snmp* MIBs. Those MIBs and the associated groups are supported by the router. Other standard MIBs and groups supported by the router are indicated in Figure 6.30.

6.4 SNMv2 MANAGEMENT INFORMATION BASE

As mentioned in Section 6.2 and shown in Figure 6.5 two new MIB modules, security and SNMPv2, have been added to the Internet MIB. The SNMPv2 module has three submodules: snmpDomains, snmpProxys, and snmpModules. snmpDomains extends the SNMP standards to send management messages over transmission protocols other than UDP, which is the predominant and preferred way of transportation [RFC 1906]. Since UDP is the preferred protocol, systems that use another protocol need a proxy service to map on to UDP. Not much work has been done on snmpProxys, as of now.

There are changes made to the core MIB-II defined in SNMPv1. Figure 6.31 presents an overview of the changes to the Internet MIB and their relationship. The system module and the *snmp* module under

```
routerIsi123 AGENT-CAPABILITIES
        PRODUCT-RELEASE     "InfoTech Router isiRouter123 release 1.0"
        STATUS              current
        DESCRIPTION         "InfoTech High Speed Router"
        SUPPORTS            snmpMIB
                INCLUDES    {systemGroup, snmpGroup, snmpSetGroup,
                            snmpBasicNotificationsGroup}
                VARIATION   coldStart
                        DESCRIPTION     "A coldStart trap is generated on all
                                        reboots."
        SUPPORTS            IF-MIB
                INCLUDES    {ifGeneralGroup, ifPacketGroup}
        SUPPORTS            IP MIB
                INCLUDES    {ipGroup, icmpGroup}
        SUPPORTS            TCP-MIB
                INCLUDES    {tcpGroup}
        SUPPORTS            UDP-MIB
                INCLUDES    {udpGroup}
        SUPPORTS            EGP-MIB
                INCLUDES    {egpGroup}
        ::= {  isiRouter 1  }
```

Figure 6.30 Example of an AGENT-CAPABILITIES Macro

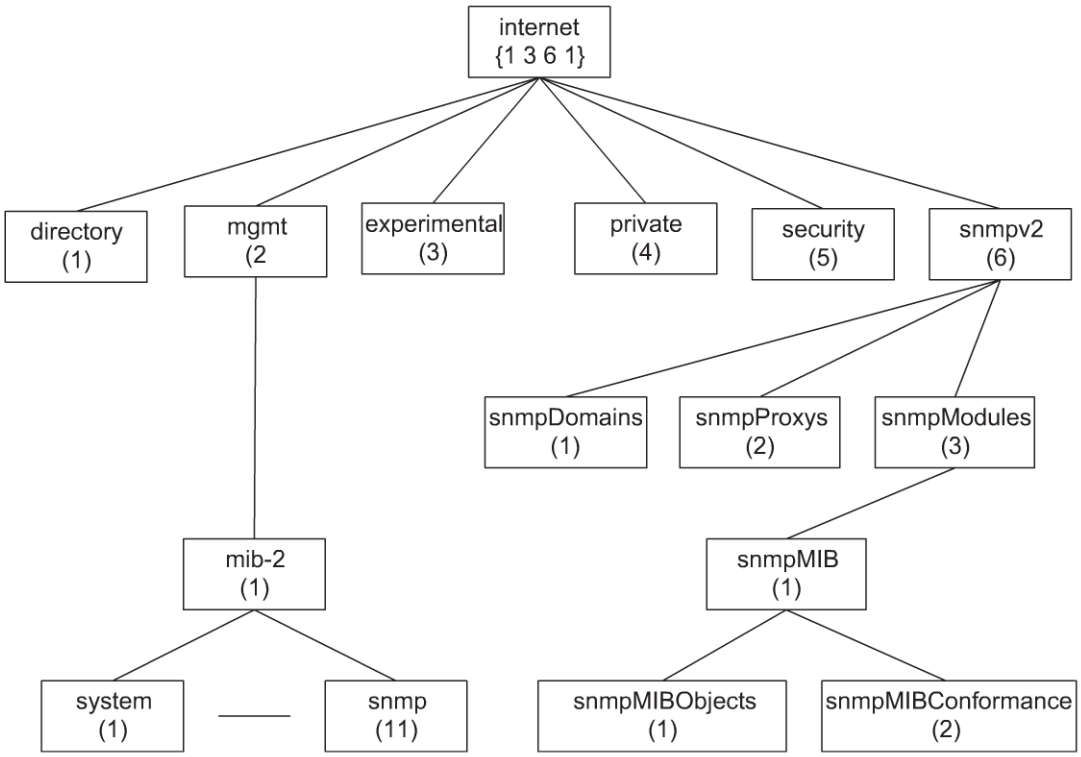

Figure 6.31 SNMPv2 Internet Group

mib-2 have significant changes as defined in RFC 1907. A new module *snmpMIB* has been defined, which is {snmpModules 1}. There are two modules under *snmpMIB*: *snmpMIBObjects* and *snmpMIB-Conformance*.

The MIB module *snmpMIBObjects* addresses the new objects introduced in SNMPv2, as well as those that are obsolete. This is primarily concerned with trap, which has been brought into the same format as other PDUs. Also, many of the unneeded objects in the SNMP group have been made obsolete.

We discussed conformance specifications and object groups in the previous section. These are specified under the *snmpMIBconformance* module. As SNMPv2 is currently defined, there is a strong coupling between system, *snmp*, *snmpMIBObjects*, and *snmpMIBconformance* modules. With this picture in mind, it will be a lot easier to follow RFC 1907, which discusses all these modules.

6.4.1 Changes to the System Group in SNMPv2

There are seven entities or objects in SNMPv2, which are common to a system. Additional information is added to the System group in SNMPv2, which contains a collection of objects that support various MIB modules. These are called object resources and are configurable both statically and dynamically. Figure 6.32 shows the MIB tree for the System group in SNMPv2. The *sysORLastChange* entity and *sysORTable* have been added to the set of objects in the System group. Table 6.6 presents the entity, OID, and a brief description of each entity for the System group.

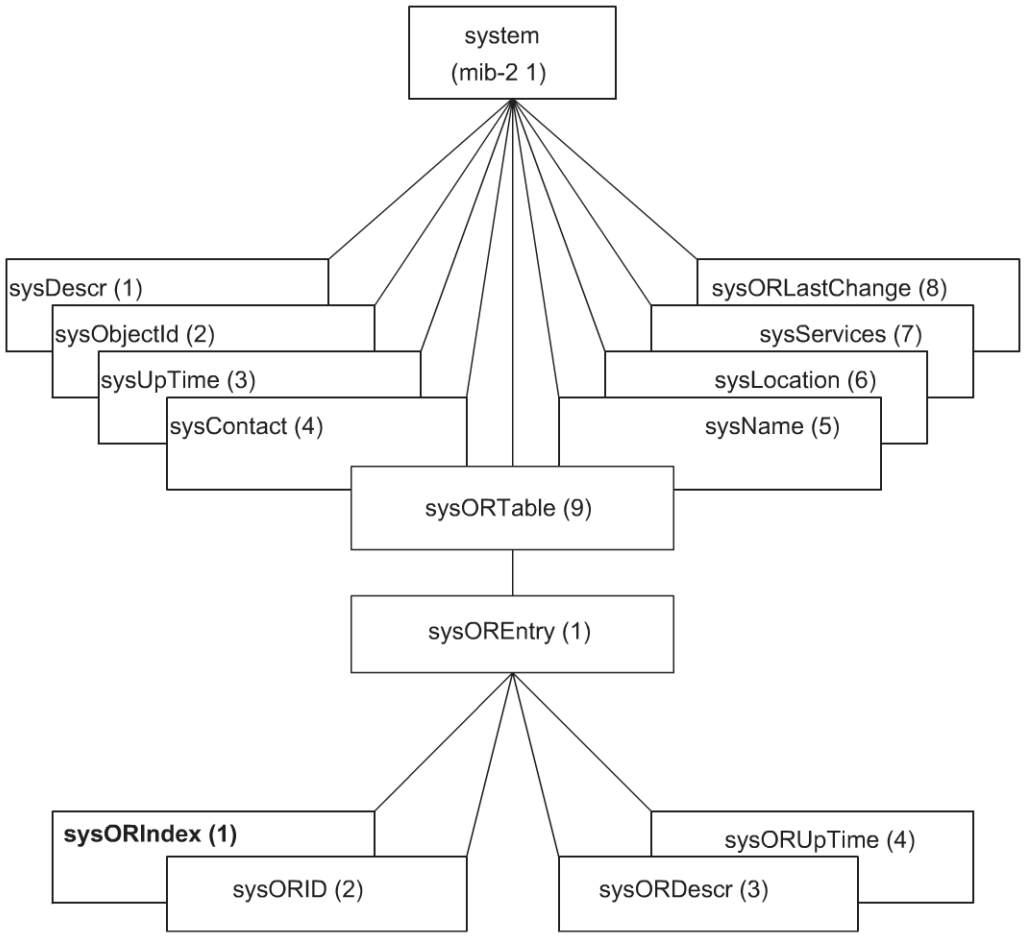

Figure 6.32 SNMPv2 System Group

Table 6.6 SNMPv2 System Group

ENTITY	OID	DESCRIPTION (BRIEF)
sysDescr	system 1	Textual description
sysObjectID	system 2	OBJECT IDENTIFIER of the entity
sysUpTime	system 3	Time (in hundredths of a second since last reset)
sysContact	system 4	Contact person for the node
sysName	system 5	Administrative name of the system
sysLocation	system 6	Physical location of the node
sysServices	system 7	Value designating the layer services provided by the entity
sysORLastChange	system 8	SysUpTime since last change in state or sysORID change
sysORTable	system 9	Table listing system resources that the agent controls; manager can configure these resources through the agent.
sysOREntry	sysORTable 1	An entry in the sysORTable
sysORIndex	sysOREntry 1	Row index, also index for the table
sysORID	sysOREntry 2	ID of the resource module
sysORDescr	sysOREntry 3	Textual description of the resource module
sysORUpTime	sysOREntry 4	System up-time since the object in this row was last instantiated

6.4.2 Changes to the SNMP Group in SNMPv2

The SNMP group in SNMPv2 has been considerably simplified from SNMPv1 by eliminating a large number of entities that were considered unnecessary. The simplified SNMP group is shown in Figure 6.33 (compare with Figure 5.21!). It has only eight entities, six old ones (1,3,4,5,6,30) and

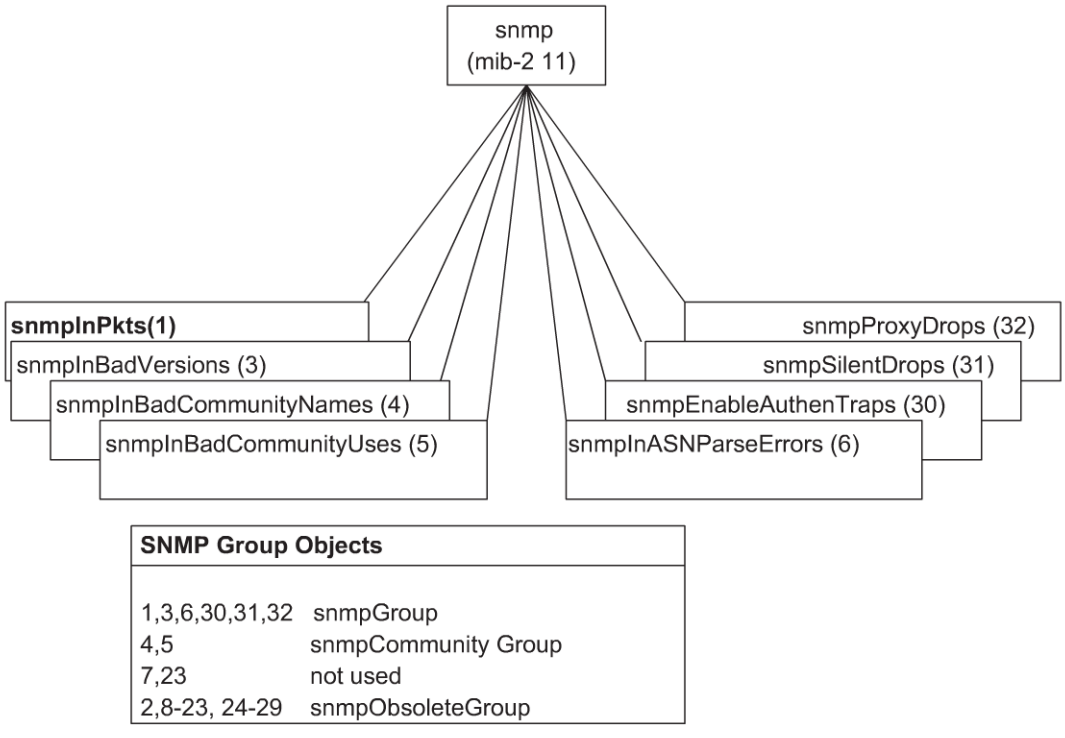

Figure 6.33 SNMPv2 SNMP Group

two new ones (31,32). Figure 6.33 also presents the four groups of all SNMP entities: *snmpGroup*, *snmpCommunityGroup*, *snmpObsoleteGroup*, and the group of two objects, 7 and 23, not used even in version 1. We will soon see that the *snmpGroup* is mandatory to implement for compliance of SNMPv2 and the *snmpCommunityGroup* is optional. The *snmpObsoleteGroup* is self-explanatory.

The SNMPv2 SNMP group table is shown in Table 6.7. All the unused and obsolete entities have been omitted for clarity.

6.4.3 Information for Notification in SNMPv2

Information on traps in SNMPv1 has been restructured in version 2 to conform to the rest of the PDUs. The macro TRAP-TYPE, used in version 1 and described in RFC 1215, has been made obsolete in SNMPv2. At the same time, enhancement to specifications has been made, and the terminology has been generalized to "notification," as the subheading indicates.

The information on notifications is defined under *snmpMIBObjects* and is shown in Figure 6.34. There are three modules under the *snmpMIBObjects* node: *snmpTrap* (4), *snmpTraps* (5), and *snmpSet* (6). The subnode designations 1, 2, and 3 under *snmpMIBObjects* have been made obsolete. A brief description of the subnodes and leaf objects under *snmpMIBObjects* is given in Table 6.8.

The *snmpTrap* group contains information on the OBJECT IDENTIFIERs of the trap and the enterprise responsible to send the trap. A new value, *accessible-for-notify*, has been added to the MAX-ACCESS clause to define objects under *snmpTrap*.

The entities under *snmpTraps* are the well-known traps that are currently in extensive use in SNMPv1. The *snmpSetSerialNo* is a single entity under *snmpSet* and is used by coordinating manager objects to

Table 6.7 SNMPv2 SNMP Group

ENTITY	OID	DESCRIPTION (BRIEF)
snmpInPkts	snmp (1)	Total number of messages delivered from transport service
snmpInBadVersions	snmp (3)	Total number of messages from transport service that are of unsupported version
snmpInBadCommunityNames	snmp (4)	Total number of messages from transport service that are of unknown community name
snmpInBadCommunityUses	snmp (5)	Total number of messages from transport service, of not allowed operation by the sending community
snmpInASNParseErrs	snmp (6)	Total number of ASN.1 and BER errors
snmpEnableAuthenTraps	snmp (30)	Override option to generate authentication failure traps
snmpSilentDrops	snmp (31)	Total number of the five types of received PDUs that were silently dropped due to exceptions in var-binds or max. message size
snmpProxyDrops	snmp (32)	Total number of the five types of received PDUs that were silently dropped due to inability to respond to a target proxy

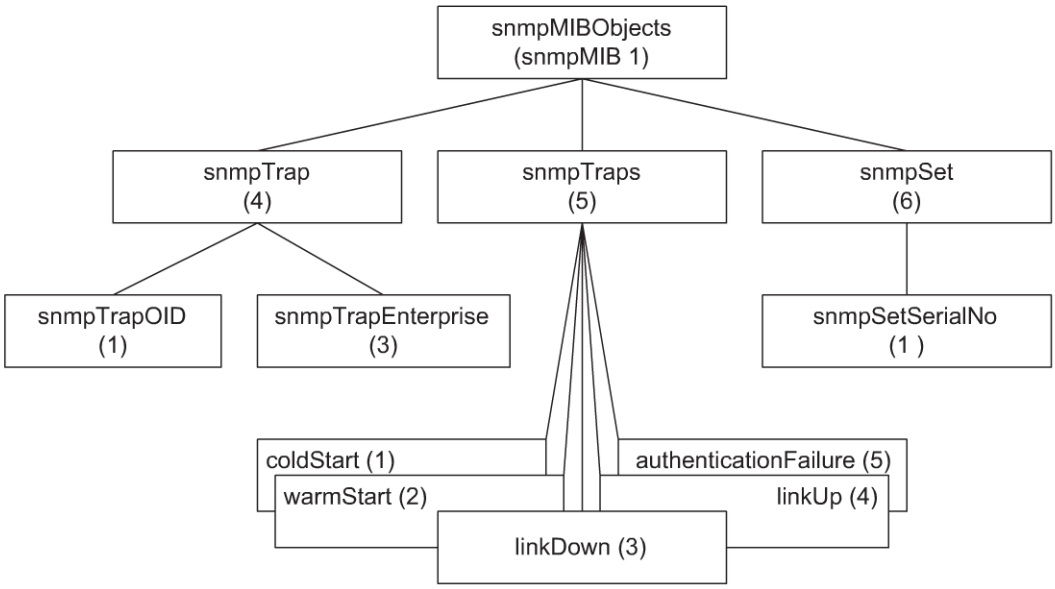

Figure 6.34 MIB Modules under snmpMIBObjects

Table 6.8 snmpMIBObjects MIB

ENTITY	OID	DESCRIPTION (BRIEF)
snmpTrap	snmpMIBObjects 4	Information group containing trap ID and enterprise ID
snmpTrapOID	snmpTrap 1	OBJECT IDENTIFIER of the notification
snmpTrapEnterprise	snmpTrap 2	OBJECT IDENTIFIER of the enterprise sending the notification
snmpTraps	snmpMIBObjects 5	Collection of well-known traps used in SNMPv1
coldStart	snmpTraps 1	Trap informing of a cold start of the object
warmStart	snmpTraps 2	Trap informing of a warm start of the object
linkDown	snmpTraps 3	Agent detecting a failure of a communication link
linkUp	snmpTraps 4	Agent detecting coming up of a communication link
authentificationFailure	snmpTraps 5	Agent reporting receipt of an unauthenticated protocol message
snmpSet	snmpMIBObjects 6	Manager-to-Manager notification messages
snmpSetSerialNo	snmpSet 1	Advisory lock between managers to coordinate set operation

perform the set operation. This is intended as coarse coordination only; fine-grain coordination may require more MIB objects in appropriate groups.

6.4.4 Conformance Information in SNMPv2

Conformance information is defined by the *snmpMIBConformance* module, as shown in Figure 6.35. It consists of two submodules, *snmpMIBcompliances* and *snmpMIBGroups*. The *snmpMIBCompliances* module has been extensively covered in Section 6.3.9. Units of conformance are defined in terms of OBJECT-GROUPS, mentioned in Section 6.3.9. Table 6.9 presents the various OBJECT-GROUPs defined in SNMPv2 and associated OBJECTS for all but *snmpObsoleteGroup*, which is shown in Figure 6.33.

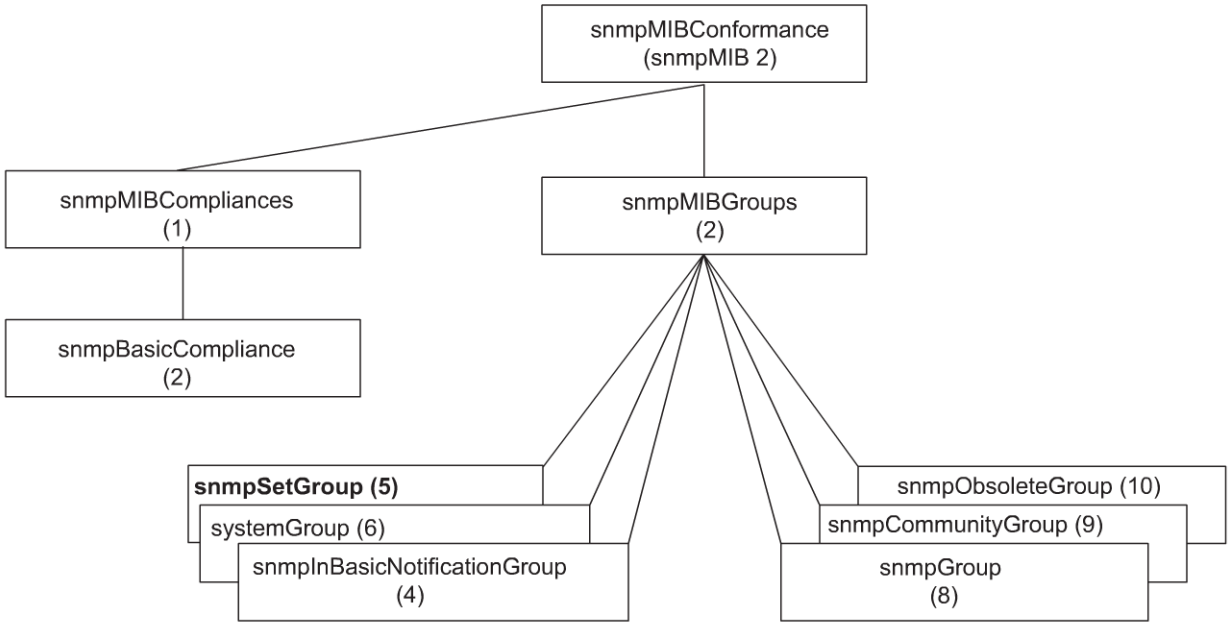

Figure 6.35 MIB Modules under snmpMIBConformance

6.4.5 Expanded Internet MIB-II

As SNMP network management expands covering legacy as well as new technology, MIB modules are continuously increasing. Figure 6.36 shows an expanded MIB-II when SNMPv2 was released and has more modules than those covered in RFC 1213. It is not intended to be an exhaustive list but includes RMON MIB module that we will be addressing in this textbook. Table 6.10 gives a description of each group in the MIB.

6.5 SNMPv2 PROTOCOL

SNMPv2 protocol operations are based on a community administrative model, which is the same as in SNMPv1. This was discussed in Section 5.2.2. We presented SNMPv2 protocol operations from a system architecture view in Section 6.2. In this section we will discuss details of PDU data structures and protocol operations.

6.5.1 Data Structure of SNMPv2 PDUs

The PDU data structure in SNMPv2 has been standardized to a common format for all messages. This improves the efficiency and performance of message exchange between systems. The significant improvement is bringing the trap data structure in the same format as the rest. The generic PDU message structure is shown in Figure 6.37 and is the same as Figure 5.8 of SNMPv1. The PDU type is indicated by an INTEGER. The error-status and error-index fields are either set to zero or ignored in the get-request, get-next-request, and set messages. The error-status is set to zero in the get-response message if there is no error; otherwise the type of error is indicated. The PDU and error-status are listed in Table 6.11. The error-index is set to zero if there is no error. If there is an error, it identifies the first variable binding in the variable-binding list that caused the error message. The first variable binding in a request's variable-binding list is index one, the second is index two, etc.

Table 6.9 SNMPv2 OBJECT-GROUPs

OBJECT-GROUPS	OID	OBJECTS
snmpSetGroup	snmpMIBGroups 5	snmpSetSerialNo
systemGroup	snmpMIBGroups 6	sysDescr
		sysObjectID
		sysUpTime
		sysContact
		sysName
		sysLocation
		sysServices
		sysORLastChange
		sysORID
		sysORUpTime
		sysORDescr
snmpBasicNotification Group	snmpMIBGroups 7	coldStart
		authenticationFailure
snmpGroup	snmpMIBGroups 8	snmpInPkts
		snmpInBadVersions
		snmpInASNParseErrs
		snmpSilentDrops
		snmpProxyDrops
		snmpEnableAuthenTraps
snmpCommunityGroup	snmpMIBGroups 9	snmpInBadCommunityNames
		snmpInBadCommunityUses
snmpObsoleteGroup	snmpMIBGroups 10	Please see Figure 6.33

There is a difference in usage of the error-status and error-index fields between SNMPv1 and SNMPv2. In version 1, any error encountered by the agent in responding to requests from the manager generates a non-zero value in either the error-status field or in both the error-status and error-index fields. Values in variable bindings are returned only under non-error conditions.

However, in SNMPv2, if only the error-status field of the Response-PDU is non-zero, the value fields of the variable binding in the variable-binding list are ignored. If both the error-status field and the error-index field of the Response-PDU are non-zero, then the value of the error-index field is the index of the variable binding (in the variable-binding list of the corresponding request) for which the request failed. Values in other variable bindings in the variable-binding list are returned with valid values and processed by the manager.

The generic PDU format is applicable to all SNMPv2 messages except the Get-Bulk-Request PDU, for which the format is shown in Figure 6.38. It can be seen that the format of the structure is the same in both cases, except that in the get-bulk-request message, the third and fourth fields are different. The third field, the error-status field, is replaced by non-repeaters; and the fourth field, the error-index

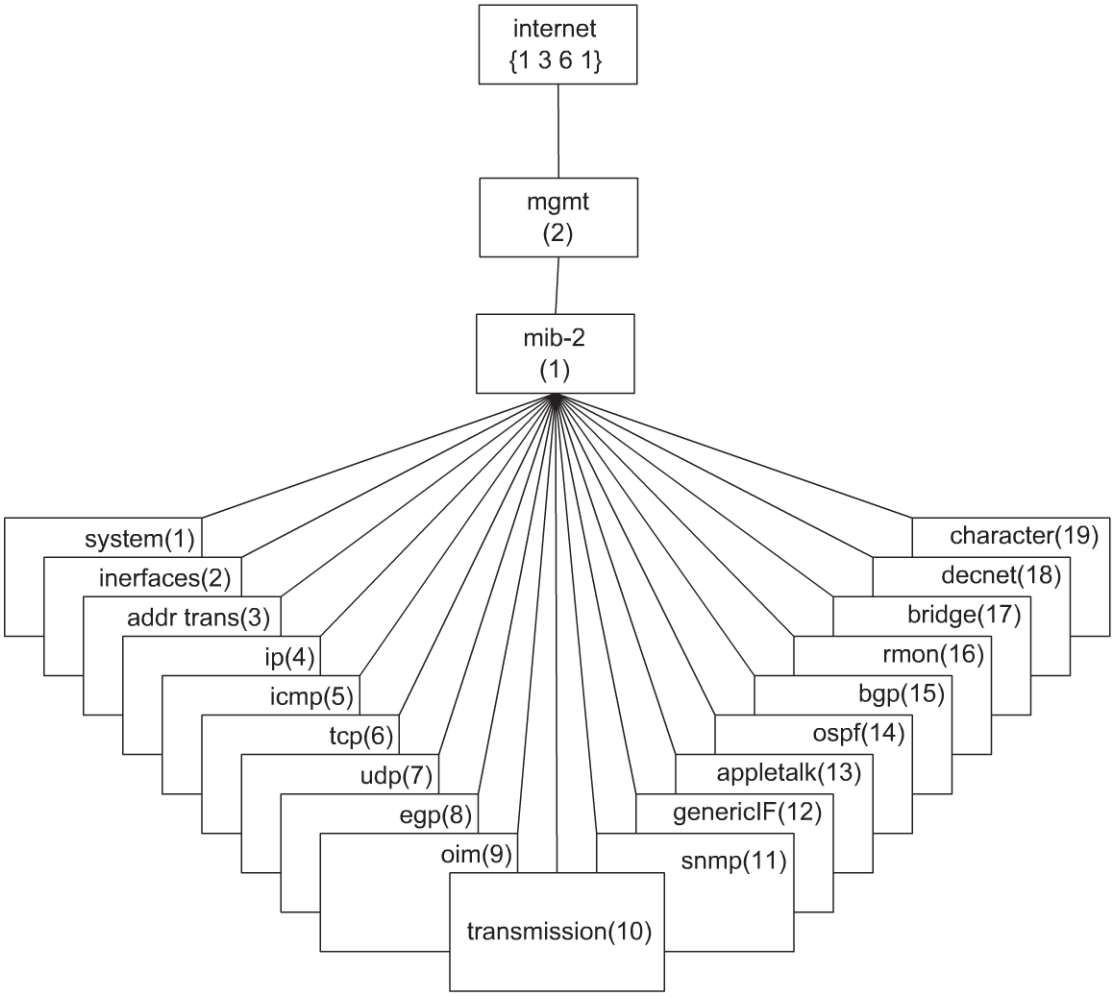

Figure 6.36 Expanded Internet MIB-II Group

Table 6.10 Expanded MIB-II Group

GROUP	OID	DESCRIPTION (BRIEF)
ifMIB	mib-2 31	Extension to interfaces group for new technologies
appletalk	mib-2 13	MIB for appletalk networks
ospf	mib-2 14	Open Shortest Path First routing protocol MIB
bgp	mib-2 15	MIB for Border Gateway Protocol for inter-autonomous network routing
rmon	mib-2 16	MIB for remote monitoring using RMON probe; there are MIBs under this for Ethernet and Token Ring networks
bridge	mib-2 17	MIB for bridges
decnet	mib-2 18	Digital Equipment Corporation DECnet MIB
character	mib-2 19	MIB for ports with character stream output for computer peripheral

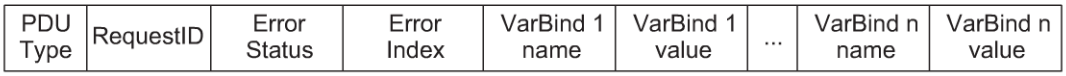

Figure 6.37 SNMPv2 PDU (all but bulk)

Table 6.11 Values for Types of PDU and Error-status Fields in SNMPv2 PDU

FIELD	TYPE	VALUE
PDU	0	Get-Request-PDU
	1	GetNextRequest-PDU
	2	Response-PDU
	3	Set-Request-PDU
	4	obsolete
	5	GetBulkRequest-PDU
	6	InformRequest-PDU
	7	SNMPv2-Trap-PDU
Error Status	0	noError
	1	tooBig
	2	noSuchName
	3	badValue
	4	readOnly
	5	genErr
	6	noAccess
	7	wrongType
	8	wrongLength
	9	wrongEncoding
	10	wrongValue
	11	noCreation
	12	inconsistentValue
	13	resourceUnavailable
	14	commitFailed
	15	undoFailed
	16	authorizationError
	17	notWritable
	18	inconsistentName

field, is replaced by max-repetitions. As we mentioned in Section 6.2, the get-bulk-request enables us to retrieve data in bulk. We can retrieve a number of both non-repetitive scalar values and repetitive tabular values with a single message. Non-repeaters indicate the number of non-repetitive field values

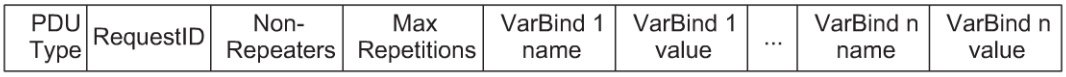

Figure 6.38 SNMPv2 GetBulkRequest PDU

requested; and the max-repetitions field designates the maximum number of table rows requested. We will next look at the SNMPv2 operations using PDUs.

6.5.2 SNMPv2 Protocol Operations

There are seven protocol operations in SNMPv2, as discussed in Section 6.2. We will ignore the report operation, which is not used. The messages, *get-request*, *get-next-request*, *set-request*, and *get-response*, are in both SNMPv1 and SNMPv2 versions and operate in a similar fashion. The two additional messages that are in SNMPv2, which are not in version 1, are the *GetBulkRequest* and *InformRequest*. The command, *get-bulk-request*, is an enhancement of *get-next request* and retrieves data in bulk efficiently. This is covered in the next subsection. The *InformRequest* is covered in a subsequent section along with SNMPv2-Trap, which has been modified in version 2.

GetBulkRequest PDU Operation. The *get-bulk-request* operation is added in SNMPv2 to retrieve bulk data from a remote entity. Its greatest benefit is in retrieving multiple rows of data from a table. The basic operation of get-bulk-request is the same as get-next-request. The third and fourth field positions are used in get-bulk-request message PDU as *non-repeaters* and *max-repetitions*, as shown in Figure 6.38. The non-repeaters field indicates the number of non-repetitive (scalar) objects to be retrieved. The max-repetitions field defines the maximum number of instances to be returned in the response message. This would correspond to the number of rows in an aggregate object. The value for the max-repetitions field is operation-dependent and is determined by such factors as the maximum size of the SNMP message, or the buffer size in implementation, or the expected size of the aggregate object table.

The data structure of the response for the get-bulk-request operation differs from other get and set operations. Successful processing of the get-bulk-request produces variable bindings (larger array of VarBindList) in the response PDU, which is larger than that contained in the corresponding request. Thus, there is no one-to-one relationship between the VarBindList of the request and response messages.

Figure 6.39 shows a conceptual MIB to illustrate the operation of get-next-request and get-bulk-request shown in Figure 6.40 and Figure 6.41. It is similar to Figure 5.12 with two additional rows added to the table. To notice the difference in improvement of get-bulk-request over get-next-request, let us look at Figure 6.40, which shows the sequence of operations for get-next-request for the MIB shown in Figure 6.39. The sequence starts with a get-request message from the manager process with a VarBindList array of two scalar variables A and B. It is subsequently followed by the get-next-request message with three columnar OBJECT IDENTIFIERS T.E.1, T.E.2, and T.E.3. The get-response returns the first instance values T.E.1.1, T.E.2.1, and T.E.3.1. The sequence of operation continues until the fourth instance is retrieved. The last get-next-request message with the OBJECT IDENTIFIERS T.E.1.4, T.E.2.4, and T.E.3.4 generates the values T.E.2.1, T.E.3.1, and Z. This is because there are no more instances of the table. It retrieves the three objects, which are logically the next lexicographically higher objects—namely T.E.2.1 (next to T.E.1.4), T.E.3.1 (next to T.E.2.4), and Z (next to T.E.3.4). The manager would stop the sequence at this message. However, if it continues the operation, it would receive a *noSuchName* error message.

Figure 6.41 shows the sequence of operations to retrieve the MIB shown in Figure 6.39 using the get-bulk message. The entire MIB data are retrieved in two requests. The first message GetBulkRequest (2, 3, A, B, T.E.1, T.E.2, T.E.3) is a request for receiving two non-repetitive objects (the first variable (2) in the request command) and three repetitive instances (the second operand (3) in the command) of the columnar objects (T.E.1, T.E.2, and T.E.3). The response returns values of A and B for the non-repetitive

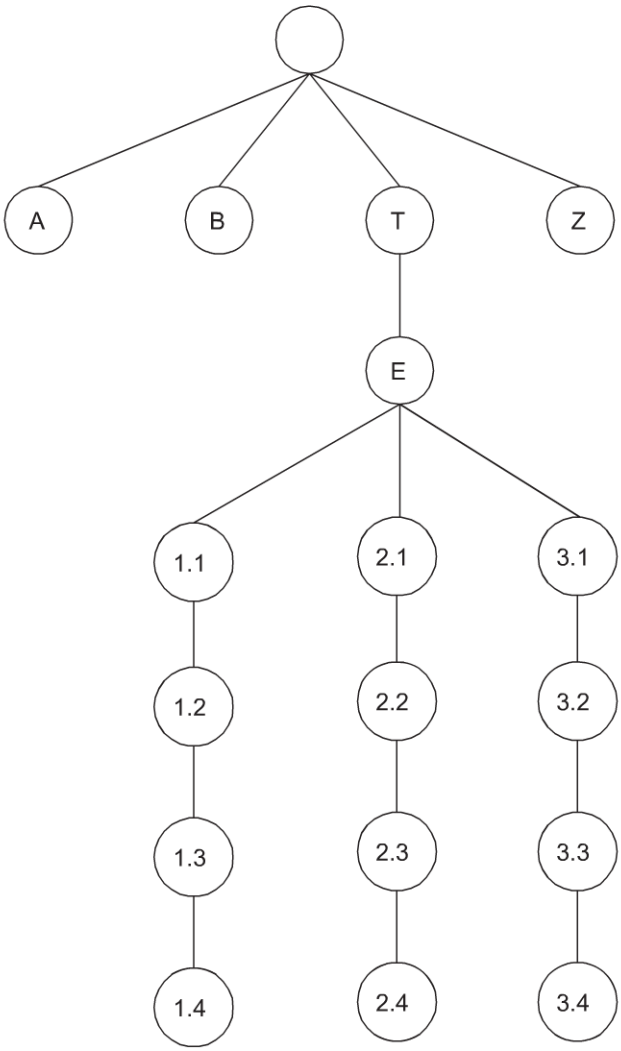

Figure 6.39 MIB for Operation Sequences in Figures 6.40 and 6.41

objects, and the first three rows of the aggregate object table. The second request is for three more rows of the table. Since there is only one more row left to send, the response message contains the information in the last row, the next lexicographic entity, Z, and the error message *endOfMibView*. The manager interprets this as end of the table.

Figure 6.42 shows the retrieval of the Address Translation table shown in Figure 5.16 using the get-bulk-request operation. Instead of four sets of get-next-request and get-response messages, only two get-bulk-request and response messages are needed in the get-bulk-request operation.

SNMPv2-Trap and InformRequest PDU Operations. The SNMPv2-Trap PDU performs the same function as in version 1. As we notice, the name has been changed, as well as its data structure to the generic format shown in Figure 6.37. The variable bindings in positions 1 and 2 are specified as *sysUpTime* and *snmpTrapOID*, as shown in Figure 6.43. The destination(s) to which a trap is sent is implementation-dependent.

A trap is defined by using a NOTIFICATION-TYPE macro. If the macro contains an OBJECTS clause, then the objects defined by the clause are in the variable bindings in the order defined in the

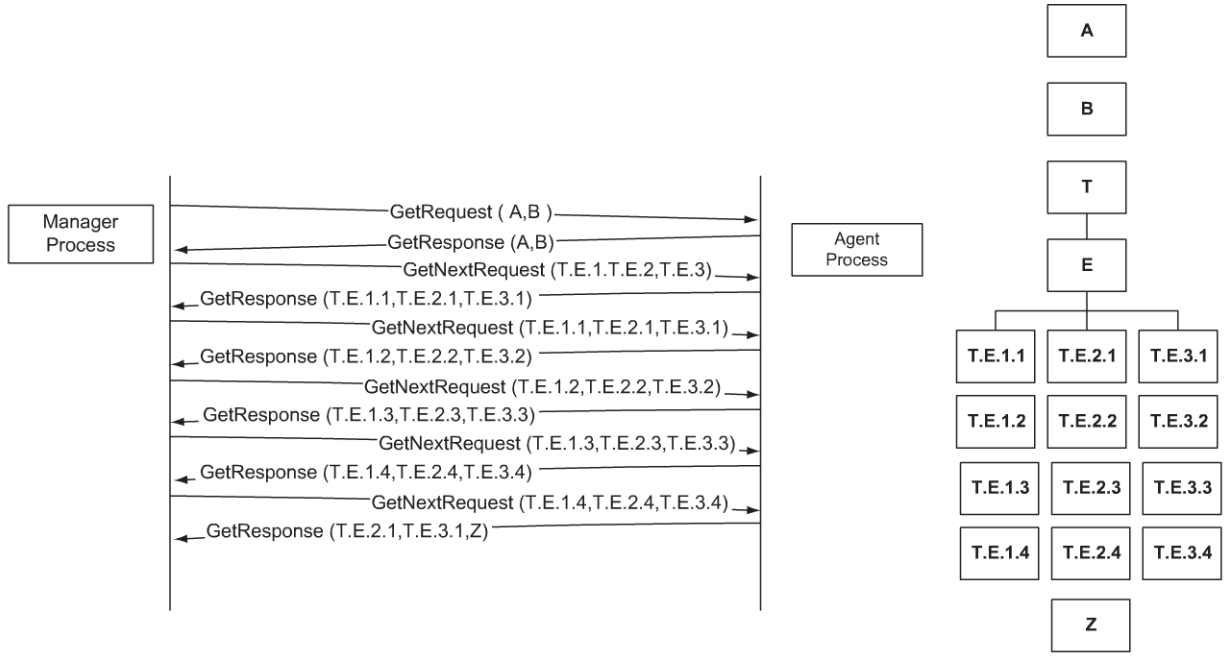

Figure 6.40 Get-Next-Request Operation for MIB in Figure 6.39

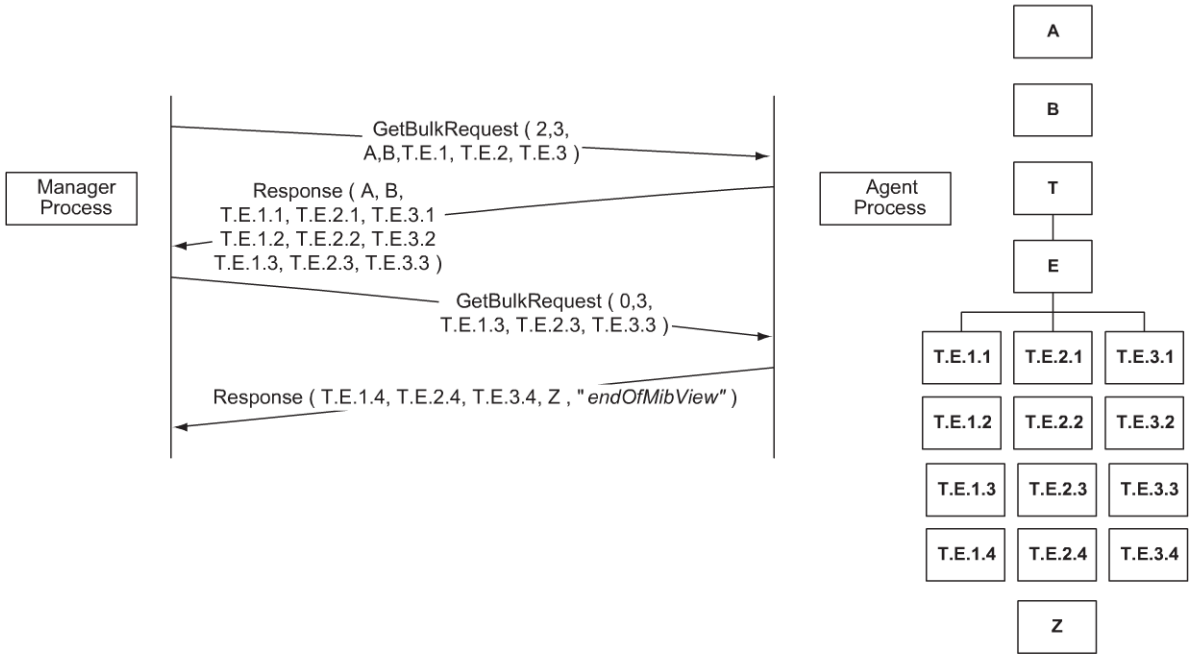

Figure 6.41 Get-Bulk-Request Operation for MIB in Figure 6.39

clause. For example, we may want to know what interface is associated with a *linkUp* trap. In this case the linkUp NOTIFICATION-TYPE would have *ifIndex* as an object in its OBJECTS clause, as shown in Figure 6.44.

An InformRequest PDU is generated by a manager (in contrast to a trap generated by an agent) to inform another manager of information in its MIB view. While a trap is received passively by a manager, an InformRequest generates a response in the receiving manager to send to the sending manager.

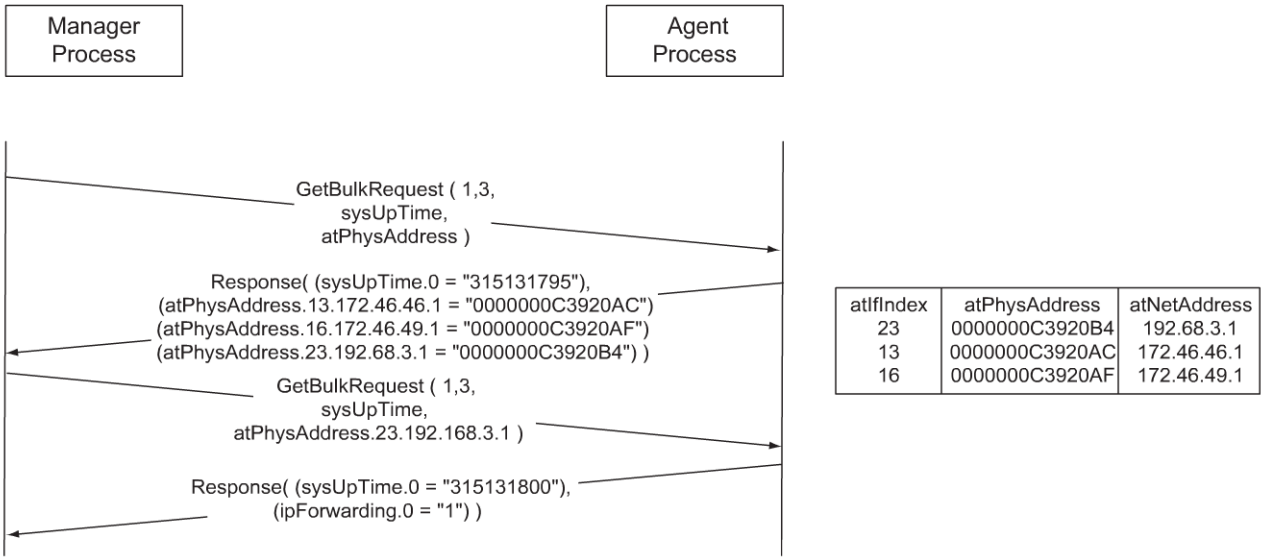

Figure 6.42 Get-Bulk-Request Example

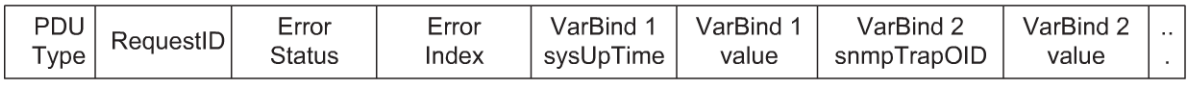

Figure 6.43 SNMPv2 Trap PDU

```
linkUp NOTIFICATION-TYPE
        OBJECTS         { ifIndex }
        STATUS          current
        DESCRIPTION     "A linkUp trap signifies that the SNSMPv2 entity,
                        acting in an agent role, recognizes that one of the
                        communication links represented in its configuration
                        has come up."
```

Figure 6.44 Example of an OBJECTS Clause in a NOTIFICATION-TYPE Macro

6.6 COMPATIBILITY WITH SNMPv1

An SNMP proxy server, in general, converts a set of non-SNMP entities into a set of SNMP-defined MIB entities. Unfortunately, SNMPv2 MIB is not backward compatible with SNMPv1 and hence requires conversion of messages. SNMPv2 IETF Working Group has proposed [RFC 1908] two schemes for migration from SNMPv1 to SNMPv2: bilingual manager and SNMP proxy server.

6.6.1 Bilingual Manager

One of the migration paths to transition to SNMPv2 from version 1 is to implement both SNMPv1 and SNMPv2 interpreter modules in the manager with a database that has profiles of the agents' version. The interpreter modules do all the conversions of MIB variables and SNMP protocol operations in both directions. The bilingual manager does the common functions needed for a management system. The SNMP

PDU contains the version number field to identify the version (see Figure 5.5). This arrangement is shown in Figure 6.45. This is expensive to implement and maintain. The alternative scheme is to use a proxy server.

6.6.2 SNMP Proxy Server

The SNMPv2 proxy server configuration is shown in Figure 6.46. The requests to and responses from, as well as traps from, SNMPv2 agents are processed by the SNMPv2 manager with no changes. A proxy server is implemented as a front-end module to the SNMPv2 manager for communication with SNMPv1 agents.

Figure 6.47 details the conversions that are done by an SNMP v2–v1 proxy server. The get-Request, GetNextRequest-PDU, and Set-Request-PDU from the SNMPv2 manager are passed through unaltered by the proxy server. There are two modifications done to the GetBulkRequest PDU. The values for the two fields, non-repeaters and max-repetitions, are set to zero and transmitted as GetNextRequest PDU. The GetResponse from SNMPv1 is passed through unaltered by the proxy server to the SNMPv2 manager, unless a response has a *tooBigError* value. In the exception case, the contents of the variable-binding field are removed before propagating the response. The trap from the SNMPv1 agent is prepended with two VarBind fields, *sysUpTime*.0 and *snmpTrapOID*.0, with their associated values and then passed on to the SNMPv2 manager as SNMPv2-Trap PDU.

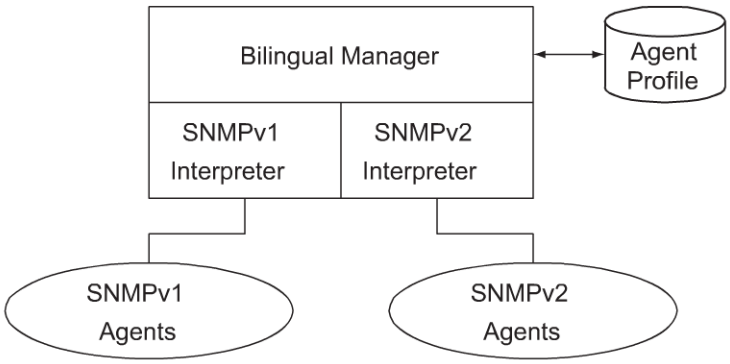

Figure 6.45 SNMP Bilingual Manager

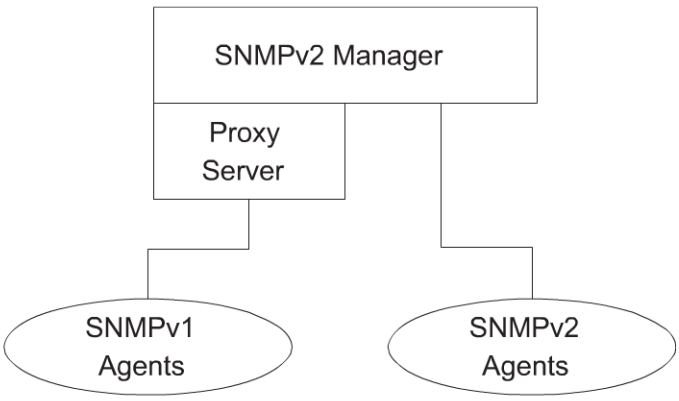

Figure 6.46 SNMPv2 Proxy Server Configuration

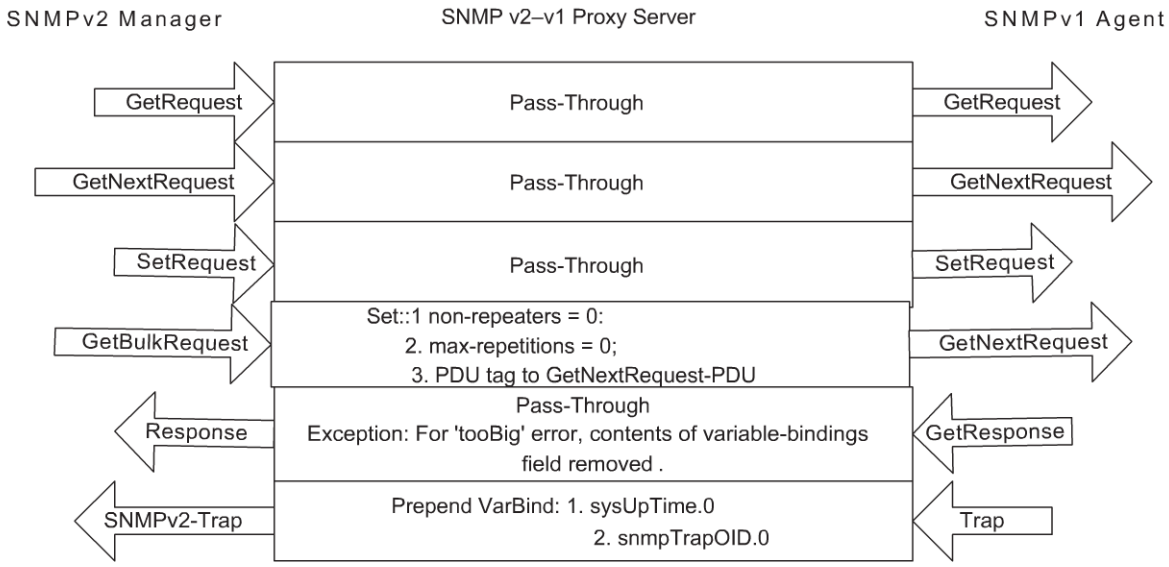

Figure 6.47 SNMP v2–v1 Proxy Server

Summary

A significant number of network management systems and agents that are on the market today use SNMP version 1, referred to as SNMPv1. However, some of the features that have been added to SNMPv1 have been formally defined in SNMPv2. We have learned enhancements in SNMPv2 over that of SNMPv1 in this chapter.

The enhancements to SNMP architecture are the formalization of manager-to-manager communication and the inclusion of traps as part of the SMI and messages, instead of as an appendix to SMI as in SNMPv1. Three additional messages have been added. They are get-bulk-request, inform-request, and report. Only get-bulk-request and inform-request details have been defined and the report is left to the implementers of a system. The report is not used in practice at present.

There are several changes to SMI in SMIv2. Modules are formally introduced using the MODULE-IDENTITY macro. An OBJECT-IDENTITY macro defines the MIB objects; and a NOTIFICATION-TYPE macro defines traps and notifications. SMIv2 has been split into three parts, each being defined in a separate RFC. They are module definitions, textual conventions, and conformance specifications. Module definitions specify the rules for defining new modules. Textual conventions help define precise descriptions of modules for human understanding. Conformance specifications are intended to interpret what the vendor is specifying in the network component with regard to compliance with SNMP management. Object groups are introduced to group a number of related entities. Conformance specifications detail the mandatory groups that should be implemented to be SNMP conformant. The object groups also help vendors define the capabilities of the system when they implement additional groups beyond that of mandatory ones.

Two new modules have been added to the Internet module. They are security and snmpV2. The security module is, as of now, a placeholder in the MIB tree as no consensus could be reached within the working group in defining it. It is specified in SNMPv3, which is covered in Chapter 7. The System and SNMP groups have been modified in the Internet MIB. Additional objects have been added to the System group that supports various MIB modules. A large number of entities have been made obsolete in the SNMP module. Obsolete entities are defined as an obsolete group in the SNMPv2 module. The SNMPv2 module also defines the MIB definition for compliance groups. Object groups defining a collection of related entities are defined to specify vendor compliance and capabilities.

All protocol PDUs, including trap, have been unified into a common data format. The newly introduced get-bulk-request is intended to improve the efficiency of get-next request in SNMPv1 by retrieving data in large quantities. The get-next-request is still maintained in this version. Interoperability between management systems has been facilitated by a new message, inform-request. We have given a conceptual presentation of table management, as this has become important when multiple management systems try to set configuration on an agent at the same time.

The unfortunate part of SNMPv2 is that it is not backward compatible with SNMPv1. Two schemes have been recommended for migrating from version 1 to version 2. Proxy server is the preferred approach over that of a bilingual manager. Proxy server can also be developed for managing non-SNMP agents with an SNMP manager.

Exercises

1. Define the OBJECT-IDENTITY module for the following objects mentioned in Exercise 4.11:
   (a) hats
   (b) jacketQuantity
2. Write the OBJECT TYPE modules for ipAddrTable, ipAddrEntry, and ipAdEntIfIndex in an IP address translation table shown in Figure 4.20 in SMIv2.
3. Add two columnar objects, cardNumber (of interface card) and portNumber (port in the interface card), to an IP address table in a router. The index values for the IP address table rows are 150.50.51.1, 150.50.52.1, 150.50.53.1, and 150.50.54.1. The packets to the first two addresses are directed to ports 1 and 2 of interface card 1. The last two addresses refer to ports 1 and 2 of interface card 2.
   (a) Draw a conceptual base table and an augmented table (ipAug 1).
   (b) Present the ASN.1 constructs for both down to the leaf level of the MIB tree. Limit your leaf for *ipTable* to *ipAdEntAddr* object.
4. Table 6.12 shows the output of a network management system detailing the addresses of a router in a network. Three columnar objects (Index, IP Address, and Physical Address) belong to the Address Translation table, atTable. Treat the other three columns as belonging to an augmented table, atAugTable (atAug 1). Repeat Exercises 3(a) and (b) for this case. Use SMIv2 textual conventions.

Table 6.12 Table for Exercise 4

atIfIndex	intType	intNumber	PortNumber	IP Address atNetAddress	Physical Address atPhysAddress
3	6	0	2	172.46.41.1	00:00:0c:35:C1:D2
4	6	0	3	172.46.42.1	00:00:0c:35:C1:D3
5	6	0	4	172.46.43.1	00:00:0c:35:C1:D4
6	6	0	5	172.46.44.1	00:00:0c:35:C1:D5
2	6	0	1	172.46.63.1	00:00:0c:35:C1:D1
7	15	1	0	172.46.165.1	00:00:0c:35:C1:D8
1	6	0	0	172.46.252.1	00:00:0c:35:C1:D0

5. In Exercise 3, the router interfaces with subnets are reconfigured as virtual LANs. There is only one interface card with two ports handling two subnets each. The packets to the two subnets, 150.50.51.1 and 150.50.52.1, are directed to port 1 of the interface card; and the packets to 150.50.53.1 and 150.50.54.1 are connected to port 2. The second table is the dependent table, ipDepTable (ipDep 1).

(a) Draw a conceptual base table and a dependent table.
(b) Present the ASN.1 constructs for both down to the leaf level of the MIB tree. Limit your leaf for *ipTable* to *ipAdEntAddr* object.

6. A table is used in a corporation for each branch to maintain an inventory of their equipment in the agent system located at the branch. The inventory table is maintained remotely from the central location. Items can be added, deleted, or changed. The objects that make up the table are:

Branch ID	{corp 100}
Table name	invTable
Row name	invEntry
Columnar object 1	invStatus
Columnar object 2	invNumber (index)
Columnar object 3	make
Columnar object 4	model
Columnar object 5	serNumber

(a) Draw the inventory conceptual table.
(b) Write the detailed ASN.1 constructs for the table.

7. In Exercise 6, the following equipment is to be added as the 100th inventory number:

make	Sun
model	Ultra5
serNumber	S12345

(a) Add the conceptual row to the table in Exercise 6(a).
(b) Draw the operational sequence diagram for create-and-go operation to create the new row.

8. In Exercise 6, equipment with the inventory number 50 is no longer in use and is hence to be deleted. Draw the operational sequence to delete the conceptual row.
9. Generate an ASN.1 OBJECT-GROUP macro for Address Translation group in SNMPv2 implementation.
10. Draw request-response messages, as shown in Figure 6.40 and Figure 6.41, to retrieve all columnar objects of the Address Translation group shown in Table 6.13. Assume that you know the number of rows in the table in making requests.
(a) get-next-request and response
(b) get-bulk-request and response
(c) Compare the results of (a) and (b)

Table 6.13 Table for Exercises 10

INDEX	IP ADDRESS	PHYSICAL ADDRESS
3	172.46.41.1	00:00:0c:35:C1:D2
4	172.46.42.1	00:00:0c:35:C1:D3
5	172.46.43.1	00:00:0c:35:C1:D4
6	172.46.44.1	00:00:0c:35:C1:D5
2	172.46.63.1	00:00:0c:35:C1:D1
7	172.46.165.1	00:00:0c:35:C1:D8
1	172.46.252.1	00:00:0c:35:C1:D0

11. Fill in the values for the SNMPv2 Trap PDU shown in Figure 6.43 for a message sent by the hub shown in Figure 4.2(a) one second after it is reset following a failure. (You may want to compare the result with that of Exercise 3 in Chapter 5 for SNMPv1.)