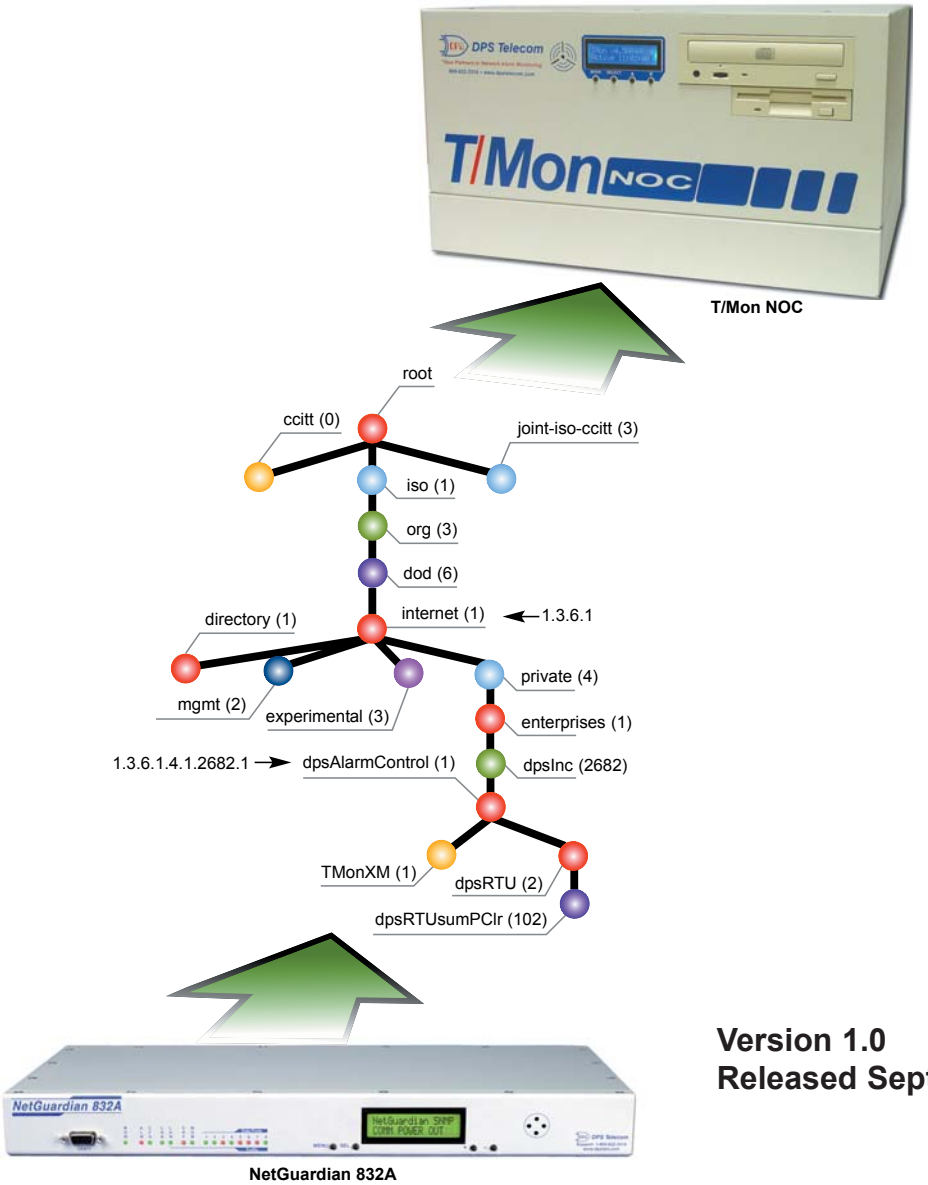
DPS Telecom

"Your Partners in Network Alarm Monitoring"

Demystifying the MIB
by Marshall DenHartog

A complete guide to the SNMP Management Information Base:

- Understanding the purpose and function of the MIB
- How to read the MIB
- Using the MIB to evaluate SNMP equipment



Version 1.0
Released September 27, 2004

www.dpstelecom.com · 1-800-622-3314

US $36.95

"We protect your network like your business depends on it"

Demystifying the MIB

What is the MIB?

The MIB, or Management Information Base, is an ASCII text file that describes SNMP network elements as a list of data objects. Think of it as a dictionary of the SNMP language — every object referred to in an SNMP message must be listed in the MIB.

What does the MIB do?

The fundamental purpose of the MIB is to translate numerical strings into human-readable text. When an SNMP device sends a Trap or other message, it identifies each data object in the message with a number string called an object identifier, or OID. (OIDs are defined more fully later in this paper.)

The MIB provides a text label called for each OID. Your SNMP manager uses the MIB as a codebook for translating the OID numbers into a human-readable display.

Why do I need the MIB?

Your SNMP manager needs the MIB in order to process messages from your devices. Without the MIB, the message is just a meaningless string of numbers.

How do I get the MIB into my SNMP manager?

Your SNMP manager imports the MIB through a software function called compiling. Compiling converts the MIB from its raw ASCII format into a binary format the SNMP manager can use.

Why is the MIB important?

Because as far as SNMP managers and agents are concerned, if a component of a network device isn't described in the MIB, it doesn't exist.

For example, let's say you have an SNMP RTU with a built-in temperature sensor. You think you'll get temperature alarms from this device — but you never do, no matter how hot it gets. Why not? You read the RTU's MIB file and find out that it only lists discrete points, and not the temperature sensor. Since the sensor isn't described in the MIB, the RTU can't send Traps with temperature data.

Why do I need to understand the MIB?

As you can see, the MIB is your best guide to the real capabilities of an SNMP device. Just looking at the physical components of a device won't tell you what kind of Traps you can get from it. You might think it's strange that a manufacturer would add a component to a device and not describe it in the MIB. But the fact is, a lot of devices have sketchy MIBs that don't fully support all their functions.

When you're planning your SNMP monitoring, you need to be able to read the MIB so you can have a realistic idea of what capabilities you have. And if you're evaluating new SNMP equipment, ask the vendor for the MIB and examine it carefully before you purchase.

What Features Do I Need in an SNMP RTU?

Here are 5 essential features that your SNMP RTU must have:

1. **Discrete alarm inputs (also called digital inputs or contact closures):** These are typically used to monitor equipment failures, intrusion alarms, beacons, and flood and fire detectors.
2. **Analog alarm inputs:** Analog alarms measure continuously variable levels of voltage or current. Analog alarms monitor temperature, humidity and pressure, all of which can critically affect equipment performance.
3. **Ping alarms:** An RTU that supports ping alarms will ping devices on your network at regular intervals. If a device fails to respond, the RTU will send an alarm as an SNMP Trap.
4. **Control relays:** An RTU with control relay outputs will let you operate remote site equipment directly from your NOC.
5. **Terminal server function:** Your RTU can also serve as a terminal server to remote-site serial devices. Your devices connect to the RTU's serial ports, giving you immediate Telnet access via LAN from your NOC at any time.

DPS Telecom offers SNMP RTUs that meet all these requirements — and offer stand-alone local visibility through any web browser, expandable alarm capacity, LAN access via dial-up connection and more.

To learn more about DPS RTUs, request a live Web demo at www.dpstelecom.com/webdemo.

How do I look at a MIB?

A MIB file is just ASCII text, so you can view it in any word processor or text editor, such as Microsoft Notepad. Some manufacturers provide precompiled MIBs in binary format, but those aren't readable. You want the raw ASCII version of the MIB file.

Note: MIB files are sometimes provided as Unix text files. Unix text format is significantly different from DOS/Windows text format. DOS/Windows text files have a carriage return and a line feed at the end of each line; Unix files only have a line feed. If you view a Unix text file in a Windows text editor, all you'll see is one long line, because the editor will never hit a carriage return.

If you want to view MIB files on a Windows PC, ask your vendor for a DOS-formatted version. Or, if you're a do-it-yourself kind of person, you can get a conversion utility to convert between text formats.

Will I need to edit the MIB?

Generally speaking, no. MIB files aren't really designed to be edited by the end user. Theoretically, you could edit the text descriptions of managed objects to be more user-friendly, but it's better to use your SNMP manager's presentation software to create a useful display.

How do I read the MIB?

To read a MIB file, you have to understand just a little about how the MIB is structured. Don't worry — you don't have to master MIB notation in order to get useful information from the MIB. In this paper we're going to cover just the essentials you need to know to discover the telemetry capabilities of SNMP devices.

What does a MIB look like?

For an example, here are the first few lines of the standard DPS Telecom MIB file:

```
DPS-MIB-V38 DEFINITIONS ::= BEGIN
IMPORTS
       DisplayString
              FROM RFC1213-MIB
       OBJECT-TYPE
              FROM RFC-1212
       enterprises
              FROM RFC1155-SMI;
dpsInc  OBJECT  IDENTIFIER  ::=  {enterprises
2682}
dpsAlarmControl OBJECT IDENTIFIER ::= {dpsInc
1}
tmonXM OBJECT IDENTIFIER ::= {dpsAlarmControl
1}
tmonIdent OBJECT IDENTIFIER ::= {tmonXM 1}
tmonIdentManufacturer OBJECT-TYPE
       SYNTAX  DisplayString
       ACCESS  read-only
       STATUS  mandatory
       DESCRIPTION "The TMON/XM Unit manufac-
turer."
       ::= {tmonIdent  1}
tmonIdentModel OBJECT-TYPE
       SYNTAX  DisplayString
       ACCESS  read-only
       STATUS  mandatory
       DESCRIPTION "The TMON/XM model desig-
nation."
       ::= {tmonIdent  2}
```

Wow! What language is that?

The MIB is written in ASN.1 notation. (The initials stand for Abstract Syntax Notation 1.) ASN.1 is a standard notation maintained by the ISO (International Organization for Standardization) and used in everything from the World Wide Web to aviation control systems.

A full description of ASN.1 is completely beyond the scope of this white paper — standard references to ASN.1 run up to 600 pages. For our purposes, there are only a few things to understand about ASN.1:

1. It's human-readable.
2. It's specifically designed for communication between dissimilar computer systems, so it's the same for every machine.
3. It's extensible, so it can be used for describing almost anything.
4. Once a term is defined in ASN.1, it can be used as a building block for making other terms. This is very important for understanding MIB structure — you'll see why later on.

How ASN.1 builds new terms out of existing terms

ASN.1 defines each term as a sequence of components, some of which may be sequences themselves. To give a simplified example, here's how you might describe a letter in ASN.1:

```
Letter ::= SEQUENCE {
       opening      OCTET STRING,
       body         OCTET STRING,
       closing      OCTET STRING,
       address      AddressType
}
```

Note that while most of the elements in this sequence are defined using a primitive element (the "octet

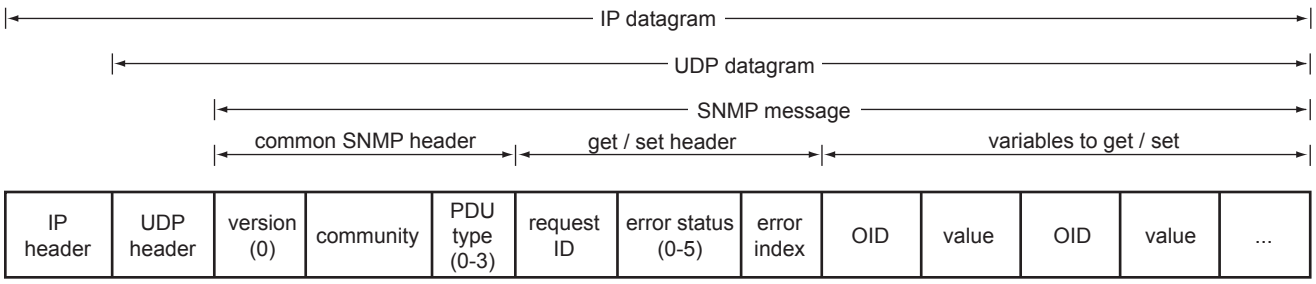


Figure 1. The OID identifies managed objects that can have assigned values

string," which is the equivalent of a byte), the address is simply defined as a text string, "AddressType." You can do this because AddressType is defined in another sequence, like so:

```
AddressType ::= SEQUENCE  {
      name          OCTET STRING,
      number        INTEGER,
      street        OCTET STRING,
      city          OCTET STRING,
      state         OCTET STRING,
      zipCode       INTEGER
}
```

For a computer parsing the sequence "Letter," AddressType will be read as an instruction to insert the octet string and integer structures listed in the sequence that defines AddressType

What terms are defined in the MIB?

The elements defined in the MIB can be extremely broad (for example, all objects created by private businesses) or they can be extremely specific (like a particular Trap message generated by a specific alarm point on an RTU.)

Each element in the MIB is given an object identifier, or OID. An OID is a number that uniquely identifies an element in the SNMP universe. Each OID is associated with a human-readable text label.

What is the function of an OID?

The OIDs identify the data objects that are the subjects of an SNMP message. When your SNMP device sends a Trap or a GetResponse, it transmits a series of OIDs, paired with their current values.

The location of the OID within the overall SNMP packet is shown in Figure 1.

What does an OID look like?

Here's an example: 1.3.6.1.4.1.2681.1.2.102

OK ... but what does it mean?

The OID is a kind of address. It locates this particular element within the entire SNMP universe. The OID describes a tree structure, as shown in Figure 2, and each number separated by a decimal point represents a branch on that tree.

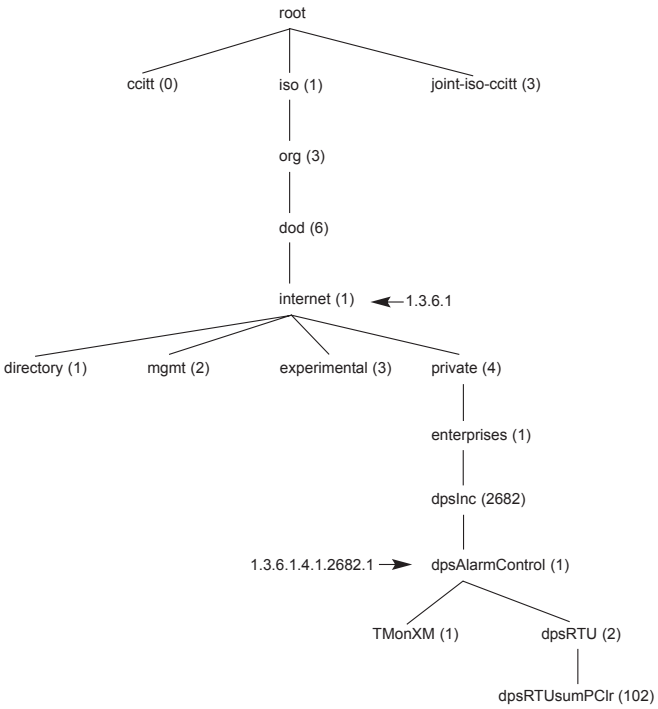


Figure 2. The branch of the MIB object identifier tree that represents managed elements used by DPS Telecom equipment.

The first few numbers identify the domain of the organization that issued the OID, followed by numbers that identify objects within the domain. Imagine if your home address started "Universe, Milky Way Galaxy ..." and ended with your house number. In a similar way, each OID begins at the root level of the OID domain and gradually becomes more specific.

Each element of the OID also has a human-readable text designation. From left to right, our sample OID reads:

1 (iso): The International Organization for Standardization, one of the two organizations that assign OID domains.

3 (org): An ISO-recognized organization.

6 (dod): U.S. Department of Defense, the agency originally responsible for the Internet.

1 (internet): Internet OID.

4 (private): Private organizations.

1 (enterprises): Business enterprises.

2682 (dpsInc): DPS Telecom.

1 (dpsAlarmControl): DPS alarm and control devices.

2 (dpsRTU): DPS remote telemetry unit.

102 (dpsRTUsumPClr): A Trap generated when all the alarm points on an RTU are clear.

When I look at my MIB files, I don't see long strings of numbers like that

That's because each element of an OID only needs to be defined once. Remember, in ASN.1 notation, once a term is defined, it can be used as a building block to define other terms. The last number of an OID — its most specific element — refers back to the more general elements defined earlier in the MIB.

Here's how the last four elements in our sample OID are defined in the DPS Telecom MIB:

```
dpsInc OBJECT IDENTIFIER ::= {enterprises
2682}
dpsAlarmControl OBJECT IDENTIFIER ::= {dpsInc
1}
dpsRTU OBJECT IDENTIFIER ::= {dpsAlarmControl
2}
dpsRTUsumPClr TRAP-TYPE
      ENTERPRISE  dpsRTU
      VARIABLES {  sysDescr,   sysLocation,
dpsRTUDateTime }
      DESCRIPTION "Generated when all points
clear."
      ::= 102
```

Look at how each term is defined as the term that came immediately before it in the OID, plus one more element. For example, dpsInc is enterprises (1.3.6.1.4) plus one more element, called 2682. The next term, dpsAlarmControl, is dpsInc (1.3.6.1.4.2682), plus one more element, called 1. And so on. Each term in the OID is defined as an extension of earlier terms, going all the way back to the root term, iso.

An OID is meaningless unless every element it refers to is specifically called out and identified in the MIB. So when you're compiling your MIB files on your SNMP manager, you need to supply not only the OIDs defined by your equipment vendor, but also OIDs for public entities: iso, org, dod, internet, and so on.

So every MIB file needs to describe the entire OID tree?

Fortunately, no. The upper levels of the OID tree — the parts that define the general OID structure — are defined in a series of standard reference MIB files called RFCs.

(The initials stand for Request for Comment. The RFCs that define SNMP OIDs are part of a larger group of RFC documents that define the Internet as a whole.)

The RFC MIB defines a basic dictionary of terms that vendors use to write their own equipment-specific MIBs. So a vendor-created MIB doesn't have to define the entire OID tree. The vendor's MIB file only has to define the unique OIDs that describe the vendor's equipment.

At the beginning of every MIB file is an IMPORTS line that calls out the terms used in the MIB and the RFC MIB that defines those terms.

Let's take another look at the very beginning of the DPS Telecom MIB:

```
DPS-MIB-V38 DEFINITIONS ::= BEGIN
IMPORTS
       DisplayString
              FROM RFC1213-MIB
       OBJECT-TYPE
              FROM RFC-1212
       enterprises
              FROM RFC1155-SMI;
```

From this IMPORTS line we can read that the DPS MIB is written using three terms defined in other MIBs — DisplayString, OBJECT-TYPE and enterprises — and these terms are defined in the RFC MIBs listed.

How to avoid the most common cause of compile errors

Your SNMP manager can't compile your MIB files correctly unless it also compiles the RFC MIBs that your MIB files refer to. For example, to compile the DPS Telecom MIB, you need to compile RFC1213-MIB, RFC 1212 and RFC1155-SMI. Compile errors are often caused by missing RFC MIBs.

RFC MIBs are publicly available on the Internet, or your vendor can supply the RFC MIBs you need.

All MIB files are written as extensions of the master RFCs. For this reason, you'll sometimes hear people say that there's only one MIB for all SNMP devices, and that individual MIB files are merely subsections of the unified Management Information Base.

That may be true in theory, but in real life, you only need to worry about is the equipment you use, the MIBs that support your equipment, and the RFCs that support those MIBs.

So I'm reading the MIB. What information am I looking for?

You don't need to carefully read over every last line of the MIB file. For your purposes, you're only looking for particular items that will tell you what elements of the device you can monitor and control.

A well-written MIB will be divided into sections. Sections will be identified by comment lines. (In MIB notation, comments lines are identified by two hyphens.) So if you find a line that reads something like:

```
-- TRAP definitions
```

You know you've found what you're looking for.

There are also text labels that identify the MIB objects you're interested in. For example, in SNMP v1 MIBs, Traps are identified by the text label "TRAP-TYPE." If you know the text labels for the kinds of objects you're looking for, you can scan the MIB in a series of Ctrl-F searches.

The MIB objects you need to know

From the perspective of a telemetry manager, what you need to know from the MIB is:

1. What other RFC MIBs you need to support this device
2. What event reports (Traps) the device can send to the SNMP manager
3. What information you can request from the device (the SNMP equivalent of an alarm poll)
4. What characteristics of the device you can control via SNMP

RFC MIBs

The first thing you should look for in the MIB is what RFC MIBs are required to support this device. The necessary RFCs will be called out in the IMPORTS line at the beginning of the MIB.

Traps: Event Reports

For telemetry purposes, the MIB elements you're most interested in are what Traps the device can send. Traps are often described as alarms, but it's better to think of them as event reports.

When a Trap is called out in the MIB, it means that the device is configured to generate a report whenever the element listed changes state. This doesn't mean that the event is necessarily important. Many Traps are merely status messages.

In SNMP v1 MIBs, Traps are always designated with the text label TRAP-TYPE. Here's an example from the MIB for the DPS Telecom NetGuardian RTU::

```
dpsRTUp8005Set TRAP-TYPE
       ENTERPRISE  dpsRTU
       VARIABLES { sysDescr,  sysLocation,
dpsRTUDateTime,
dpsRTUAPort, dpsRTUCAddress, dpsRTUADisplay,
dpsRTUAPoint,dpsRTUAPntDesc, dpsRTUAState }
       DESCRIPTION "Generated when discrete
point 5 is set."
       ::=  8005
```

Fortunately, you can ignore a lot of this gobbledygook. Here are the elements that you're interested in:

TRAP-TYPE: This tells you it's a Trap.

DESCRIPTION: This is a human-readable description of the Trap. It should give you a good basic indication of what the Trap signifies.

VARIABLES: This tells you actual information will be included in the Trap. When an actual Trap is sent, each of these variables will be paired with a numerical value that indicates its current state. A variable-and-value pair is called a variable binding.

The variables look pretty cryptic, but it's easy to find out what they mean. Each variable is a text label for an OID defined elsewhere in the MIB. You can do a Ctrl-F search for any variable term and find its definition. For example, "dpsRTUAPort" is defined in the DPS MIB like this:

```
dpsRTUAPort OBJECT-TYPE
       SYNTAX  INTEGER
       ACCESS  read-only
       STATUS  mandatory
       DESCRIPTION "RTU port number."
       ::= {dpsRTUAlarmEntry 1}
```

Trap variables are your best guide to what alarms you'll get from an SNMP device. Depending on the device, the variables can be highly detailed or they can be vague summary alarms.

Object-Types: Data you can read and sometimes write

When reading the MIB, you'll also want to know what information you can directly request from the device, and what information you can send to the device. These functions are controlled by the SNMP commands GetRequest and SetRequest.

If you want to translate these commands into classic telemetry terms, you can roughly think of a GetRequest as an alarm poll and a SetRequest as a control command.

GetRequests and SetRequests operate on a type of element called an object-type. Object-types are called out in the MIB like this:

```
tmonAState OBJECT-TYPE
       SYNTAX  DisplayString (SIZE (8))
       ACCESS  read-only
       STATUS  mandatory
       DESCRIPTION "The current alarm state."
       ::= {tmonAlarmEntry 4}
```

There are many different kinds of object-types. The specific object-types you might find in a MIB depend on the type of device, what kind of components it has, what the functions of those components, are, etc.

Quick Primer on SNMP Messages

In SNMP v1, there are only 5 basic PDUs (program datagram units):

GetRequest: a manager-to-agent message requesting the current value of a managed object.

GetNext:a manager-to-agent message requesting the current value of the managed object one number after the one named in the request. (This is a way of walking down a table of values.)

SetRequest: a manager-to-agent message that writes a new value to a managed object

GetResponse: an agent-to-manager message in response to a GetRequest or a SetRequest. In either case, the message reports the current value of the managed object named in the manager's request

Trap: an agent-to-manager message reporting a change in the value of a managed object

You're probably not going to be interested in every object-type listed in the MIB, because you're not going to be interested in everything about the device's functions.

When searching for object-types, it's helpful to start with a plan of what functions of the device you want to manage. What information do you want to retrieve? What controls do you want to set? Knowing the device's functions and how you want to use them will help you narrow down what object-types you should look for in the MIB.

Access

The most important entry in an object-type description is the ACCESS line. This controls whether you can read and write the data described in the object-type.

There are three access settings: not-accessible, read-only and read-write.

Not-accessible means the object-type is there, but you can't request the data in a GetRequest.

Read-only means you can request the data in a GetRequest, but you can't write new data for the object-type in a SetRequest.

Read-write means you're free to retrieve the data in a GetRequest and write new data for the object-type in a SetRequest.

In the example of the alarm state object-type:

```
tmonAState OBJECT-TYPE
       SYNTAX  DisplayString (SIZE (8))
       ACCESS  read-only
       STATUS  mandatory
       DESCRIPTION "The current alarm state."
       ::= {tmonAlarmEntry 4}
```

The access here is read-only, because the alarm state is set by the alarm input on that alarm point.

Here's an example of an object-type with read-write access:

```
dpsRTUDateTime OBJECT-TYPE
       SYNTAX  DisplayString (SIZE (23))
       ACCESS  read-write
       STATUS  mandatory
       DESCRIPTION "The RTU system date and
time."
       ::= {dpsRTUIdent  4}
```

Here the access is read-write, because this is a value that you can set from your SNMP manager. You can retrieve the current settings from the RTU's internal clock through a GetRequest. And if the clock needs to be reset, you can write new data in a SetRequest.

I want to use a device feature that isn't described in the MIB. What can I do?

You can ask the vendor to extend the MIB to include this feature. DPS Telecom has extended its MIB to support client needs and will continue to do so if a client needs it.

But you need to understand that extending a MIB is actually a software development project. The MIB is not just a text file. It's also a software interface document to the embedded firmware of your SNMP device. Making additions to the MIB requires rewriting the device firmware.

This is a serious project, involving writing code, debugging it, and undergoing a thorough quality assurance process. Extending a MIB is a large commitment of time and resources.

7 Reasons Why a Basic SNMP Manager Is a Lousy Telemetry Master

SNMP is a standard protocol that has wide acceptance in the industry and is flexible enough to describe almost anything. Because of these advantages, many network managers have come to believe that SNMP should be used for all telemetry monitoring applications.

SNMP certainly has its place in an effective telemetry monitoring solution, but this doesn't mean that any off-the-shelf SNMP manager can provide adequate visibility and control of your network.

The typical off-the-shelf SNMP manager is not designed for displaying and processing telemetry data, especially not for the kind of real-world monitoring tasks network managers most need performed. These capabilities can be added to an SNMP manager, but it may require substantial custom software development.

Using an off-the-shelf SNMP systems for mission-critical telemetry is disappointing at best and disastrous at worst. If you're used to the standards of classic telecom telemetry, an off-the-shelf SNMP manager will not provide the detailed alarm data you expect. Before you commit to an SNMP monitoring solution, you need to make sure it supports essential telemetry functions.

Before you buy … check for these 7 essential telemetry features:

1. **Basic SNMP managers don't provide complete, precise alarm descriptions**
   A basic SNMP manager doesn't record location, time, severity or descriptions of alarm events. To adapt an off-the-shelf SNMP manager to monitor these factors, you must create and maintain a master alarm list representing all the monitored points in your network — and then also create and maintain a database associating all the Traps that may be sent to the SNMP manager with the alarms on that list.
2. **Basic SNMP managers can't identify cleared alarms**
   Even more work is required to identify whether a Trap represents an alarm or a clear condition. Creating this addition to the Trap association database often requires analyzing multiple variable bindings within the Trap packet.

7 Features That SNMP Managers Can't Match

1. **Detailed alarm notifications in plain English that your staff will immediately understand and take action on.** Every notification includes full information about the alarm, including its severity, location, date/time stamp, and a user-defined description.
2. **Immediate notification of changes of state (COSs)**, including new alarms and alarms that have cleared. You don't have to hunt to find out what's changed in your network — T/Mon lists it for you.
3. **A continuously updated list of all current standing alarms.** Even if the system operator acknowledges the alarm, it remains in the Standing Alarms screen until it is cleared.
4. **Text message windows displaying specific instructions for the appropriate action for an alarm.** System operators, even without extra training, will know precisely what to do and who to call in case of an alarm.
5. **Nuisance alarm filtering.** Unimportant alarms that generate meaningless status notices or oscillate between alarm and clear conditions subconsciously train your staff to ignore the alarm monitoring system. T/Mon filters out nuisance alarms, allowing your staff to focus its attention on serious threats.
6. **Pager and e-mail notifications.** Send alarm notifications directly to maintenance personnel, even if they're away from the NOC.
7. **Derived alarms and controls** that combine and correlate data from multiple alarm inputs and automatically control remote site equipment to correct complex threats.

The T/Mon NOC Remote Alarm Monitoring System provides total visibility of your network status and automatically notifies the right people to keep your network running.

Sign up for a Web demo of T/Mon NOC at www.dpstelecom.com/webdemo

3. **Basic SNMP managers don't maintain a history of standing alarms**
   Relying on a basic SNMP manager for alarm management can potentially result in completely losing visibility of threats to your network. A basic SNMP manager doesn't maintain a list of standing alarms. Instead, the typical SNMP manager maintains an event log of newly reported Traps and a history log of acknowledged Traps. As soon as a Trap is acknowledged, it is considered cleared. Imagine what might happen to your network if a system operator acknowledges an alarm, and then, for whatever reason, fails to correct the alarm condition. Who would know the alarm is still standing?
4. **Basic SNMP managers don't identify system operators**
   Basic SNMP managers do not record the identity of the system operator who acknowledges an alarm. In the example of the negligent system operator, it would be impossible to determine who had made the mistake or to assign responsibility for the resulting problems.
5. **Basic SNMP managers are insufficiently secure for multiple users**
   Out of the box, the typical SNMP manager is not designed for multi-user security. All Traps are posted to one alarm list; all users may view all alarms, and all users may acknowledge all alarms.
6. **Basic SNMP managers don't sort or filter alarms**
   Basic SNMP managers have no built-in functions for organizing alarms by logical category, posting the same alarm to multiple logical categories, or sorting which alarms the user wants to see. If Jones is in charge of all equipment for the Western region, and Smith is in charge of power plants, both need to know about a generator failure in Tucson, but neither one needs to know about all the alarms in the network. And if one manager corrects the alarm condition and acknowledges the alarm, the other manager needs to know it was acknowledged and by whom. Unfortunately, standard SNMP managers will not support these functions.
7. **Basic SNMP managers don't provide the alarm notification you need**
   No SNMP manager supports the advanced features necessary for best quality telemetry monitoring, such as notifications escalation, legacy protocol mediation, nuisance alarm silencing, automatic control relay operation, and automatic notifications by pager and e-mail.

It is true that many, but not all, of these functions can be added to standard SNMP managers, but implementing telemetry monitoring in a basic SNMP manager usually involves a substantial amount of custom software module development. Even when pre-built software modules are available, they usually require custom tweaking to perform exactly as you want them to.

The need for extensive customization eliminates the advantage of using a simple open standard, and it is difficult to justify significant development costs after purchasing an already expensive SNMP manager. Why take the time, trouble, and expense to recreate capabilities that are already present in a high-quality, SNMP-capable network alarm management system?

And in fact, it is much easier to adapt a traditional telemetry master to process SNMP Traps than to adapt an SNMP manager to perform telemetry functions. There is no question that SNMP is right for many applications, and it is clear that SNMP will be increasingly used in the future.

SNMP is an effective tool, but it's only one item in your telemetry monitoring toolkit, and it can be used more effectively when it is part of a total alarm management solution.

For details about an alarm management system that overcomes these 7 barriers , send an email to:

solutions@dpstelecom.com

Why You Need Help With Your SNMP Implementation

IMPLEMENTING AN SNMP network alarm monitoring system can seem deceptively easy — you just look on the Web, find a few vendors, compare a few features, add some configuration and you're done, right?

The truth is, developing a network monitoring system on your own is one of the riskiest things you can do. Here are some of the typical problems you might face if you don't get expert advice when you're designing your system:

1. **Implementation time is drawn out:** It's going to take longer than you think. Network monitoring is a highly technical subject, and you have a lot to learn if you want a successful implementation. And anytime you are trying to do something you've never done before, you are bound to make mistakes — mistakes that extend your time and your budget beyond their limits.
2. **Resources are misused:** If you're not fully informed about your options for mediating legacy protocols to SNMP, you may replace equipment that could have been integrated into your new system. Rushing into a systemwide replacement when you could have integrated can cost you hundreds of thousands of dollars.
3. **Opportunities are missed:** If you install a new network monitoring system today, you're committing your company to that system for as long as 8 to10 years. Many telecoms design what they think is a state-of-the-art monitoring system — and then find that their technology is actually a generation behind.

DPS Telecom Guarantees Your Success — or Your Money Back

When you're choosing a network monitoring vendor, don't take chances. Be skeptical. Ask the hard questions. Above all, look for experience. Don't take a sales rep's word that his company can do custom development. Ask how many systems they've worked with, how many protocols they can integrate to SNMP, and check for client testimonials.

DPS Telecom has created hundreds of successful SNMP monitoring implementations for telecoms, utility telecoms, and transportation companies. (Check out www.dpstelecom.com/case-studies for some examples.) DPS Telecom monitoring solutions are proven performers under real-world conditions.

You're never taking any risk when you work with DPS Telecom. Your SNMP monitoring solution is backed by a 30-day, no-risk, money-back guarantee. Test your DPS monitoring solution at your site for 30 days. If you're dissatisfied for any reason, just send it back for a full refund.

What to Do Next

Before you make a decision about your SNMP monitoring, there's a lot more you need to know. There's dangers you want to avoid — and there's also opportunities to improve your remote site maintenance that you don't want to miss.

Get the information you need — register now for a free, live Web demonstration of SNMP monitoring solutions with the T/Mon Remote Alarm Monitoring System. There's no obligation to buy — no high-pressure salesmen — just straightforward information to help you make the best decision about your network monitoring. You'll get complete information on hardware, software, specific applications, specifications, features and benefits . . . plus you'll be able to ask questions and get straight answers.

Call 1-800-622-3314 today to schedule your free Web demo of SNMP monitoring solutions — or register on the Web at www.dpstelecom.com/tmon-webdemo.

"I would personally like to let you know how beneficial the installation of the SNMP responder was to the mission of our department. We were looking for a way to integrate our local ILEC region in HP OpenView without a major network change. The SNMP responder was the answer. This migration will allow us not only to monitor all alarms in one spot but also build extensive collection reports of our whole network."

—**Todd Matherne**, EATEL

"It is hard to find companies with the intelligence and aptitude to meet the customer's exact needs, and I believe that is what DPS is all about."

—**Lee Wells**, Pathnet

About the Author

Marshall DenHartog has seven years' experience working with SNMP, including designing private MIB extensions, creating SNMP systems for multiple platforms, and developing SNMP-based monitoring for several nationwide networks.

DenHartog's experience with both the theoretical and practical sides of SNMP have equipped him to write a straightforward guide to the SNMP Management Information Base.

www.dpstelecom.com
1-800-622-3314

DPS Telecom
"Your Partners in Network Alarm Monitoring"

US $36.95