

SNMP MIB User Guide

Raritan Inc.

October 19, 2017

Contents

Contents	ii
1 Quick Start	1
1.1 Determine the status of an outlet	1
1.2 Power cycle an outlet	2
1.3 Get the voltage reading of an inlet	2
1.4 Get the number of decimal digits of an inlet voltage reading	3
1.5 Receive a trap message on an external sensor alarm assertion	4
1.6 Get an asset management tag ID	5
1.7 Set the name of a rack unit in asset management	5
2 About this document	6
2.1 Scope	6
2.2 Audience	6
2.3 Prerequisites	6
3 Capabilities	7
4 Access	8
4.1 SNMP Interface Configuration	8
4.2 Protocol Versions	8
4.3 MIB Version	8
4.4 Authentication	9
5 Object Indexing	11
5.1 Scalar Objects	11
5.2 Columnar Objects – Single Index	11
5.3 Columnar Objects – Multiple Indexes	13
6 MIB tree	16
7 Interpreting Sensor Values	18
7.1 Decimal Digits	18
7.2 Signed vs. Unsigned	18

1 Quick Start

This section contains some brief examples of tasks that can be performed on a PDU's SNMP interface. All of them assume the following preconditions:

- SNMP was set up as shown in figure 1 and 2 in section 4.1
- the Net-SNMP tools are installed (see www.net-snmp.org)
- the MIB files (see section 4.3 how to get them) are located in the MIB search path, which can be adjusted by adding `-M+<path>` to the `snmp` command arguments

Each example will present the MIB object identifier to deal with and an SNMP command line with related execution output. The object identifier includes the applicable variables, e. g. “<outlet number>” which are replaced with actual values in the command and output example.

1.1 Determine the status of an outlet

Complete Object Identifier:

```
PDU2-MIB
::pdu2
 .control
  .outletControl
   .outletSwitchControlTable
    .outletSwitchControlEntry
     .outletSwitchingState
      .<pdu id>
       .<outlet number>
```

Command:

```
snmpget -v2c -c public -m+PDU2-MIB <pdu IP address> \
PDU2-MIB::outletSwitchingState.1.4
```

```
snmpget -v2c -c public <pdu IP address> \
.1.3.6.1.4.1.13742.6.4.1.2.1.3.1.4
```

Output:

```
PDU2-MIB::outletSwitchingState.1.4 = INTEGER: off(8)
```

1.2 Power cycle an outlet

Complete Object Identifier:

```
PDU2-MIB
::pdu2
 .control
  .outletControl
  .outletSwitchControlTable
  .outletSwitchControlEntry
  .switchingOperation
  .<pdu id>
  .<outlet number>
```

Command:

```
snmpset -v2c -c private -m+PDU2-MIB <pdu IP address> \
  PDU2-MIB::switchingOperation.1.4 = cycle

snmpset -v2c -c private <pdu IP address> \
  .1.3.6.1.4.1.13742.6.4.1.2.1.2.1.4 i 2
```

Output:

```
PDU2-MIB::switchingOperation.1.4 = INTEGER: cycle(2)
```

1.3 Get the voltage reading of an inlet

Complete Object Identifiers:

```
PDU2-MIB
::pdu2
 .measurements
  .measurementsInlet
  .inletSensorMeasurementsTable
  .inletSensorMeasurementsEntry
  .measurementsInletSensorValue
  .<pdu id>
  .<inlet number>
  .<sensor type>
```

Command:

```
snmpget -v2c -c public -m+PDU2-MIB <pdu IP address> \
  PDU2-MIB::measurementsInletSensorValue.1.1.rmsVoltage

snmpget -v2c -c public <pdu IP address> \
  .1.3.6.1.4.1.13742.6.5.2.3.1.4.1.1.4
```

Output:

```
PDU2-MIB::measurementsInletSensorValue.1.1.rmsVoltage = Gauge32: 397
```

1.4 Get the number of decimal digits of an inlet voltage reading

Complete Object Identifier:

```
PDU2-MIB
  ::pdu2
    .configuration
      .inlets
        .inletSensorConfigurationTable
          .inletSensorConfigurationEntry
            .inletSensorDecimalDigits
              .<pdu id>
                .<inlet number>
                  .<sensor type>
```

Command:

```
snmpget -v2c -c public -m+PDU2-MIB <pdu IP address> \
  PDU2-MIB::inletSensorDecimalDigits.1.1.rmsVoltage
```

```
snmpget -v2c -c public <pdu IP address> \
  .1.3.6.1.4.1.13742.6.3.3.4.1.7.1.1.4
```

Output:

```
PDU2-MIB::inletSensorDecimalDigits.1.1.rmsVoltage = Gauge32: 0
```

1.5 Receive a trap message on an external sensor alarm assertion

Complete Object Identifier:

```
PDU2-MIB
  ::pdu2
    .traps
      .externalSensorStateChange
```

snmptrapd Configuration File:

The host that should receive the SNMP notification messages needs to run **snmptrapd**. Its configuration file is usually located at `/etc/snmp/snmptrapd.conf` and specifies at least the community string and the object identifiers that should be handled. A minimalistic configuration for logging external sensor state changes will just contain two lines:

```
authCommunity log public
traphandle PDU2-MIB::externalSensorStateChange
```

Note that the string “public” must be specified in the SNMP notification settings of the PDU, see section 4.1.

Command:

```
snmptrapd -m+PDU2-MIB
```

Output:

On reception of an SNMP trap the daemon will print the trap contents to its console. The output will be printed without line breaks, the example has them to increase readability.

```
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (36334) 0:06:03.34
SNMPv2-MIB::snmpTrapOID.0 = OID: PDU2-MIB::externalSensorStateChange
PDU2-MIB::pduName = STRING: My PX
PDU2-MIB::pduNumber = INTEGER: 0
PDU2-MIB::pxInetAddressType = INTEGER: ipv4(1)
PDU2-MIB::pxInetIPAddress = Hex-STRING: C0 A8 02 93
PDU2-MIB::agentInetPortNumber = Gauge32: 161
PDU2-MIB::externalSensorNumber = INTEGER: 25
PDU2-MIB::typeOfSensor = INTEGER: humidity(11)
PDU2-MIB::measurementsExternalSensorTimeStamp = Gauge32: 1456130186
PDU2-MIB::measurementsExternalSensorValue = INTEGER: 42
PDU2-MIB::measurementsExternalSensorState = INTEGER: belowLowerWarning(3)
PDU2-MIB::oldSensorState = INTEGER: normal(4)
PDU2-MIB::externalSensorSerialNumber = STRING: AAR3Sh0005
PDU2-MIB::externalOnOffSensorSubtype = INTEGER: none(31)
PDU2-MIB::externalSensorChannelNumber = INTEGER: -1
SNMPv2-MIB::sysContact = STRING: n/a
SNMPv2-MIB::sysName = STRING: n/a
SNMPv2-MIB::sysLocation = STRING: n/a
```

1.6 Get an asset management tag ID

Complete Object Identifier:

```
ASSETMANAGEMENT-MIB
::assetManager
  .configuration
    .assetManagement
      .assetManagementTable
        .assetManagementEntry
          .tagID
            .<asset strip number>
              .<rack unit index>
```

Command:

```
snmpget -v2c -c public -m+ASSETMANAGEMENT-MIB <pdu IP address> \
  ASSETMANAGEMENT-MIB::tagID.1.5
```

```
snmpget -v2c -c public <pdu IP address> \
  .1.3.6.1.4.1.13742.7.1.7.1.1.6.1.5
```

Output:

```
ASSETMANAGEMENT-MIB::tagID.1.5 = STRING: B1A0C499D35F
```

1.7 Set the name of a rack unit in asset management

Complete Object Identifier:

```
ASSETMANAGEMENT-MIB
::assetManager
  .configuration
    .assetManagement
      .assetManagementTable
        .assetManagementEntry
          .rackUnitName
            .<asset strip number>
              .<rack unit index>
```

Command:

```
snmpset -v2c -c private -m+ASSETMANAGEMENT-MIB <pdu IP address> \
  ASSETMANAGEMENT-MIB::rackUnitName.1.5 = MyName
```

```
snmpset -v2c -c private <pdu IP address> \
  .1.3.6.1.4.1.13742.7.1.7.1.1.12.1.5 s MyName
```

Output:

```
ASSETMANAGEMENT-MIB::rackUnitName.1.5 = STRING: MyName
```

2 About this document

2.1 Scope

The Raritan PDU SNMP MIB User Guide provides information about the following:

- configuration, access and capabilities of the SNMP interface
- usage of tables
- structure of the PDU's Management Information Base

This document applies to the following Raritan PDU devices:

- PX2 / PX3 / PX3TS
- EMX
- BCM2 / PMC
- BCM
- PXE

The very basics of the SNMP protocol and the SMI concepts are not covered. Tools like MIB browsers or snmp utilities are not explained as well. This documents describes how the SNMP interface of a Raritan PDU can be utilized by the SNMP protocol by for example using such tools.

2.2 Audience

This document should primarily guide

Data Center Operators who want to configure PDUs, request values or control states sporadically,

DCIM software developers who want to integrate PDUs into higher level management frameworks and

Script Developers who want to write code that uses the PDU's SNMP interface.

2.3 Prerequisites

This document presumes understanding of

- Structured Management Information (SMI),
- the Simple Network Management Protocol (SNMP) and
- Management Information Base (MIB) documents.

3 Capabilities

The SNMP interface has almost the same capabilities as the web interface and the serial command line interface. It can be used to:

- retrieve logged and current sensor measurements
- get notifications on events
- control actuators
- configure names, thresholds etc.

The following features are not accessible via SNMP:

- User Management
- Network Configuration
- Services Configuration
- Event Rule Configuration

4 Access

4.1 SNMP Interface Configuration

The unit's SNMP interface can be enabled and set up in the web frontend or command line interface. The settings dialog can be found in the menu under Device Settings / Network Services / SNMP.

The screenshot shows the Raritan web user interface for a PX iPDU my PMC (10.0.42.2). The user is logged in as Administrator. The left sidebar shows the navigation menu with 'SNMP' selected under 'NETWORK SERVICES'. The main content area displays the 'SNMP' settings dialog, which is highlighted with a red border. The dialog is divided into several sections:

- SNMP Agent:**
 - Enable SNMP v1 / v2c:
 - Read Community String: public
 - Write Community String: private
 - Enable SNMP v3:
- MIB-II System Group:**
 - sysContact:
 - sysName:
 - sysLocation:
- SNMP Notifications:**
 - Enable SNMP Notifications:
 - Notification Type: SNMPv2c Trap
 - Timeout: 3 seconds
 - Number of Retries: 5
- Hosts Table:**

#	Host	Port	Community
1	<input type="text"/>	162	<input type="text"/>
2	<input type="text"/>	162	<input type="text"/>
3	<input type="text"/>	162	<input type="text"/>
- Download MIBs:**
 - PDU2-MIB: [download](#)
 - ASSETMANAGEMENT-MIB: [download](#)

A 'Save' button is located at the bottom right of the dialog.

Figure 1: SNMP settings dialog of web user interface

4.2 Protocol Versions

Both SNMP v1/v2c and v3 versions are supported. They can be enabled or disabled independently.

4.3 MIB Version

While changes of the MIB may happen between different PDU firmware versions, all changes are done in a downward compatible way. The used MIB should be newer than

Raritan. PX *iPDU my PMC* (10.0.42.2) Administrator Logout

< DEVICE SETTINGS

NETWORK SERVICES

HTTP

SNMP

SMTP SERVER

SSH

TELNET

MODBUS

SERVICE ADVERTISING

SNMP

SNMP Agent

Enable SNMP v1 / v2c

Read Community String

Write Community String

Enable SNMP v3

MIB-II System Group

sysContact

sysName

sysLocation

SNMP Notifications

Enable SNMP Notifications

Notification Type

Timeout seconds

Number of Retries

#	Host	Port	Community
1	<input type="text"/>	162	<input type="text"/>
2	<input type="text"/>	162	<input type="text"/>
3	<input type="text"/>	162	<input type="text"/>

Download MIBs

PDU2-MIB [download](#)

ASSETMANAGEMENT-MIB [download](#)

Save

Figure 2: SNMP notification settings dialog of the web user interface

or as current as the firmware of the PDU. A PDU's MIB can be downloaded from the web user interface of the unit – see the “Download” button in the left bottom corner of the SNMP settings dialog (figure 1). There are three different MIB files:

- PDU2-MIB
- ASSETMANAGEMENT-MIB
- LHX-MIB

4.4 Authentication

In **SNMP v1/v2c** authentication is accomplished with community strings. The read as well as the write community string can be configured in the SNMP settings (see figure 1).

SNMP v3 authentication is set up on a per user basis. The settings are accessible in user settings (see figure 3).

An example command that uses SNMP v3 access to display the whole object tree below the **PDU2-MIB::pdu2** object is:

The screenshot shows the Raritan web interface for configuring SNMPv3. The left sidebar contains navigation links: HOME, USER MANAGEMENT, USERS (selected), ROLES, CHANGE PASSWORD, PREFERENCES, and DEFAULT PREFERENCES. The main panel is titled 'SNMPv3' and includes the following settings:

- Enable SNMPv3:**
- Security Level:** Authentication & Privacy
- Authentication Password:**
 - Same as User Password:
 - Password: [input field]
 - Confirm Password: [input field]
- Privacy Password:**
 - Same as Authentication Password:
 - Password: [input field]
 - Confirm Password: [input field]
- Protocol:**
 - Authentication: SHA-1
 - Privacy: DES

Figure 3: SNMP v3 user setup dialog of web user interface

```
snmpwalk -v3 -l authPriv -u <username> \
  -a SHA -A <authentication passphrase> \
  -x DES -X <privacy passphrase> \
  <pdu IP address> PDU2-MIB::pdu2
```

Arguments:

- l security level – “noAuthNoPriv”, “authNoPriv”, “authPriv”
- a authentication protocol – “MD5”, “SHA”
- x privacy protocol – “DES”, “AES”

5 Object Indexing

5.1 Scalar Objects

Information unique to an SNMP agent is represented by “scalar” objects. An example is `pduCount` which is located at

```
PDU2-MIB::pdu2.configuration.pduCount
```

Scalar objects are marked out as such by appending a zero to their object identifier.

```
PDU2-MIB::pdu2.configuration.pduCount.0 = INTEGER: 1
```

In numerical representation:

```
.1.3.6.1.4.1.13742.6.3.1.0 = INTEGER: 1
```

Since one SNMP agent running on a PDU is not limited to represent only a single PDU instance, most information entities are indexed by the PDU identifier – **pduId** – which is fixed to **1** for PX products but may vary on BCM2 products.

```
pduId OBJECT-TYPE
    SYNTAX Integer32(0..256)
    DESCRIPTION
        "A unique value for each PDU/Power meter.
        PX2/3 and transfer switch: pduId = 1
        BCM2:
            main controller: pduId = 0
            power meter: pduId = rotary switch setting
                        for the power meter
        It is the same as the MeterID in the GUI
        Example:
            a power meter, rotary switch setting = 5,
              pduId = 5
            a panel (power meter + branch metering),
              rotary switch setting = 23, pduId = 23
        "
```

This “indexed” information is structured in tables as described in the next section.

5.2 Columnar Objects – Single Index

This section explains the use of SNMP tables in the PDU2-MIB that are indexed by a single value. Consider the following example:

```

nameplateTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF NameplateEntryStruct
    ...
    DESCRIPTION
        "A list of PDU nameplate entries. The number of
        entries is given by the value of pduCount."
    ::= { unit 1 }

nameplateEntry OBJECT-TYPE
    SYNTAX      NameplateEntryStruct
    ...
    DESCRIPTION
        "An entry providing PDU nameplate information."
    INDEX       { pduId }
    ::= { nameplateTable 1 }

NameplateEntryStruct ::= SEQUENCE {
    pduId          Integer32,
    pduManufacturer DisplayString,
    pduModel       DisplayString,
    pduSerialNumber DisplayString,
    pduRatedVoltage DisplayString,
    pduRatedCurrent DisplayString,
    pduRatedFrequency DisplayString,
    pduRatedVA     DisplayString,
    pduImage       DisplayString
}

```

This part of the MIB defines a one-dimensional table of entries that hold nameplate information. It is one-dimensional because the INDEX clause contains a *single* identifier.

```

nameplateEntry OBJECT-TYPE
    ...
    INDEX { pduId }
    ...

```

One entry in the table corresponds to one PDU and a **pduId** value is sufficient to specify an entry in the table. See table 1 and 2 as examples. An SNMP agent running on a PX has a single PDU identifier with the value 1 for that PX per definition, as in the first example. The second example shows the nameplate table of a BCM2 device that may have multiple PDU identifiers.

pduId	pduManufacturer	pduModel	pduSerialNumber	...
1	"Raritan"	"PX3-5526X2-M5"	"da119317649bb578"	
pduId.1	pduManufacturer.1	pduModel.1	pduSerialNumber.1	

Table 1: Nameplate table of a PDU

The table structure is built as follows. Each cell of the table header corresponds to one member of the **NameplateEntryStruct** sequence. The cells of the tables body are the objects. Their object identifiers are noted below.

As a detailed example, the full SNMP object identifier of the PDU model string in table 1 "PX3-5526X2-M5" is broken down in table 3.

To address objects in the table, the shortened symbolic object identifier consists of the column name and the index. For example the **pduModel** string (from **NameplateEntryStruct**) of PDU 1 is:

pduId	pduManufacturer	pduModel	pduSerialNumber	...
0	“Raritan”	“PMC-1001”	“da119317649bb579”	
pduId.0	pduManufacturer.0	pduModel.0	pduSerialNumber.0	
4	“Raritan”	“PMM”	“00000000”	
pduId.4	pduManufacturer.4	pduModel.4	pduSerialNumber.4	
5	“Raritan”	“PMM”	“00000000”	
pduId.5	pduManufacturer.5	pduModel.5	pduSerialNumber.5	
11	“Raritan”	“PMM”	“00000000”	
pduId.11	pduManufacturer.11	pduModel.11	pduSerialNumber.11	

Table 2: Nameplate table of a PMC with three PMMs

Numerical	Symbolic	Note
.1.3.6.1.4.1	.iso.org.dod.internet.private.enterprises	
.13742	.raritan	
.6	.pdu2	
.3	.configuration	
.2	.unit	
.1	.nameplateTable	table object
.1	.nameplateEntry	defines columns and index
.3	.pduModel	selected column
.1	.<pdu id>	index / selected row

Table 3: Object identifier of PDU model string in the unit’s nameplate table

PDU2-MIB::pduModel.1

Note: Most object identifiers that are used as indexes or parts of indexes like “outletId”, “inletPoleIndex” or “circuitId” are **1-based** integer values.

5.3 Columnar Objects – Multiple Indexes

In cases where a single value is not sufficient as an index to a table of objects, the index may consist of multiple instances of identifiers. Consider the table that holds outlet sensor measurement variables **outletSensorMeasurementsTable**.

```

outletSensorMeasurementsTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF OutletSensorMeasurementsEntryStruct
    ...
    DESCRIPTION
        "A list of outlet sensor entries. The number of
         entries is given by the value of outletCount for the PDU."
 ::= { measurementsOutlet 3 }

outletSensorMeasurementsEntry OBJECT-TYPE
    SYNTAX      OutletSensorMeasurementsEntryStruct
    ...
    DESCRIPTION
        "An entry containing measurement objects for an outlet sensor."
    INDEX       { pduId, outletId, sensorType }
 ::= { outletSensorMeasurementsTable 1 }

OutletSensorMeasurementsEntryStruct ::= SEQUENCE {
    measurementsOutletSensorIsAvailable TruthValue,
    measurementsOutletSensorState      SensorStateEnumeration,
    measurementsOutletSensorValue      Unsigned32,
    measurementsOutletSensorTimeStamp  Unsigned32,
    measurementsOutletSensorSignedValue Integer32
}

```

The structure is the same as in section 5.2 – table object, entry object and the table columns. The difference here is the INDEX clause.

```

outletSensorMeasurementsEntry OBJECT-TYPE
    ...
    INDEX       { pduId, outletId, sensorType }
    ...

```

The definition states that the index consists of three parts. Each outlet sensor within one SNMP agent is identified by its PDU's identifier, its outlet number and the type of the sensor. The index that is appended to the object identifier of **outletSensorMeasurementsEntry** is formed by these three parts. For example, an object identifier to the **measurementsOutletSensorValue** of the active power of the fifth outlet is formed as shown in table 4.

Numerical	Symbolic	Note
.1.3.6.1.4.1	.iso.org.dod.internet.private.enterprises	
.13742	.raritan	
.6	.pdu2	
.5	.measurements	
.4	.measurementsOutlet	
.3	.outletSensorMeasurementsTable	table object
.1	.outletSensorMeasurementsEntry	defines columns and index
.4	.measurementsOutletSensorValue	selected column
.1	.<pdu id>	} indexes / selected row
.5	.<outlet number>	
.5	.<sensor type>	

Table 4: Object identifier of active power sensor value of the fifth outlet

The symbolic object identifier can be shortened to:

`PDU2-MIB::measurementsOutletSensorValue.1.5.activePower`

As another example, table 5 shows a part of the whole outlet sensor measurements table. The first column contains the index of the entries starting with the **rmsCurrent** sensor of the fourth outlet.

Index	measurementsOutlet-			...
	SensorIsAvailable	SensorState	SensorValue	
		...		
1.4.rmsCurrent	true	normal	151	
1.4.rmsVoltage	true	normal	229	
		...		
1.4.onOff	true	on	0	
1.4.frequency	true	normal	500	
1.5.rmsCurrent	true	normal	0	
1.5.rmsVoltage	true	normal	0	
		...		
1.5.onOff	true	off	0	
		...		

Table 5: Parts of the outlet sensor measurements table that show sensors of outlets number 4 and 5

To inspect SNMP tables the **snmptable** command might be useful, for example

```
snmptable -m+PDU2-MIB -v2c -cpublic -Cilb <pdu IP address> \
PDU2-MIB::inletSensorConfigurationTable | cut -c1-57
```

prints a nicely readable left hand part of the inlet sensor configuration table:

index	LogAvailable	Units	DecimalDigits
1.1.rmsCurrent	false	amp	3
1.1.unbalancedCurrent	false	percent	0
1.1.rmsVoltage	false	volt	0
1.1.activePower	false	watt	0
1.1.apparentPower	false	voltamp	0
1.1.powerFactor	false	none	2
1.1.activeEnergy	false	wattHour	0
1.1.frequency	false	hertz	1
1.1.residualCurrent	false	amp	3
1.1.rcmState	false	none	0
1.2.rmsCurrent	false	amp	3
1.2.unbalancedCurrent	false	percent	0
1.2.rmsVoltage	false	volt	0
1.2.activePower	false	watt	0
1.2.apparentPower	false	voltamp	0
1.2.powerFactor	false	none	2
1.2.activeEnergy	false	wattHour	0
1.2.frequency	false	hertz	1
1.2.residualCurrent	false	amp	3
1.2.rcmState	false	none	0

6 MIB tree

This section describes the basic structure of the PDU2-MIB tree. The following clip of the PDU2-MIB file shows the various subtrees that reside below the **PDU2-MIB::pdu2** object:

```

raritan MODULE-IDENTITY
    ...
    ::= { enterprises 13742 }

pdu2 OBJECT IDENTIFIER ::= { raritan 6 }

traps          OBJECT IDENTIFIER ::= { pdu2 0 }
board          OBJECT IDENTIFIER ::= { pdu2 1 }
environmental OBJECT IDENTIFIER ::= { pdu2 2 }
configuration  OBJECT IDENTIFIER ::= { pdu2 3 }
control        OBJECT IDENTIFIER ::= { pdu2 4 }
measurements  OBJECT IDENTIFIER ::= { pdu2 5 }
log            OBJECT IDENTIFIER ::= { pdu2 6 }
conformance   OBJECT IDENTIFIER ::= { pdu2 9 }
reliability    OBJECT IDENTIFIER ::= { pdu2 10 }

```

The subtree **configuration** contains most of the information that is writable / configurable. The **measurements** and **log** subtrees contain sensor values and state information. The **control** subtree allows manipulation of the PDU such as switching outlets or resetting energy counters.

Table 6 shows the most commonly used tables of the **PDU2-MIB**. Each cell represents a name that is built by its column name and its row name appended. An “x” mark states that a table with this name can actually be found in the MIB. For example: **unitConfigurationTable** does exist while **unitPoleConfigurationTable** does not.

Tables	Prefixes						
	unit	inlet	outlet	over- Current- Protector	power- Meter	transfer- circuit	Switch
ConfigurationTable	x	x	x	x	x	x	x
PoleConfigurationTable		x	x	x		x	x
SensorConfigurationTable	x	x	x	x		x	x
PoleSensorConfigurationTable		x	x			x	
SwitchControlTable			x				x ¹
SensorControlTable	x	x	x			x	
SensorMeasurementsTable	x	x	x	x		x	x
PoleSensorMeasurementsTable		x	x			x	
SensorLogTable	x	x	x	x		x	x
PoleSensorLogTable		x	x			x	

Table 6: Table names present in the MIB

The particular contents of these tables are quite common:

-ConfigurationTable

Contains metadata, capabilities, configuration and state information of the PDU’s parts; for example the current rating of an outlet or the unit’s IP address.

¹The actual name is “transferSwitchControlTable” rather than transferSwitchSwitchControlTable.

-SensorConfigurationTable

Contains sensor metadata such as accuracy or tolerance and the sensors' thresholds.

-SwitchControlTable

Contains objects that allow switching of outlets or between inlets of a transfer switch.

-SensorControlTable

Contains objects that allow resetting of cumulating sensor values such as energy counters.

-SensorMeasurementsTable

Contains sensor readings and state information.

-SensorLogTable

Contains objects to access logged measurements data such as minimum, maximum and average values.

To inspect parts of the tree, the tool **snmptranslate** can be used. Using the command line option **-Tp** the whole subtree below the object identifier submitted to the command will be displayed in human readable format. For example:

```
snmptranslate -m+PDU2-MIB -Tp PDU2-MIB::pdu2

...

| | |
| | +---transferSwitchPoleConfigurationTable(5)
| | |
| | | +---transferSwitchPoleConfigurationEntry(1)
| | | | Index: pduId, transferSwitchId, transferSwitchPoleIndex
| | | |
| | | | +--- ---- Integer32 transferSwitchPoleIndex(1)
| | | | | Range: 1..256
| | | | +--- -R-- EnumVal transferSwitchPoleLine(2)
| | | | | Textual Convention: LineEnumeration
| | | | | Values: lineL1(1), lineL2(2), lineL3(3), lineNeutral(4)
| | | | +--- -R-- Integer32 transferSwitchPoleIn1Node(3)
| | | | +--- -R-- Integer32 transferSwitchPoleIn2Node(4)
| | | | +--- -R-- Integer32 transferSwitchPoleOutNode(5)
| | |
| | +---powerMeter(10)
| | |
| | +---powerMeterConfigurationTable(2)
| | |
| | | +---powerMeterConfigurationEntry(1)
| | | | Index: pduId
| | | |
| | | +--- -RW- Unsigned powerMeterPhaseCTRating(2)

...
```

7 Interpreting Sensor Values

7.1 Decimal Digits

All sensor readings are reported as integer values. Since the actual values may have fractional parts, it could be necessary to adjust to the number of decimal digits. Therefore each **SensorConfigurationTable** has a column called **DecimalDigits**. See section 1.4 for an example. To calculate the actual value the integer reading value must be divided by ten to the power of the decimal digits. Additionally, the column **SensorUnits** of the **SensorConfigurationTable** can be read to determine the unit of the reading.

For example:

```
PDU2-MIB::inletSensorDecimalDigits.1.1.rmsCurrent = Gauge32: 3
PDU2-MIB::measurementsInletSensorValue.1.1.rmsCurrent = Gauge32: 1424
PDU2-MIB::inletSensorUnits.1.1.rmsCurrent = INTEGER: amp(2)
```

These values would mean that the inlet current is $\frac{1424}{10^3} = 1.424$ A.

7.2 Signed vs. Unsigned

Sensor readings are reported as integer values. Since some types of sensors can have negative values it is important to know if the integer value must be interpreted as a signed or an unsigned number. To determine the signedness of a reading, the according **SensorSignedMinimum** in the **SensorConfigurationTable** must be checked for a negative value. If a sensor can have negative values, the **SensorSignedValue** in the **MeasurementsTable** must be used to determine the sensor reading and the unsigned **SensorValue** object will not exist for this sensor. Otherwise the unsigned **SensorValue** can be used. If a sensor reading can exceed the value of *INT32_MAX* = 2147483647, that is if **SensorMaximum** in the **SensorConfigurationTable** is above *INT32_MAX*, the unsigned **SensorValue** in the **MeasurementsTable** must be used and the **SensorSignedValue** object will not exist. See table 7.

Table	SensorConfigurationTable		MeasurementsTable	
Column	SensorSignedMinimum (INTEGER)	SensorMaximum (Gauge32)	SensorSignedValue (INTEGER)	SensorValue (Gauge32)
Cases	≥ 0	$\leq INT32_MAX$	yes	yes
	≥ 0	$> INT32_MAX$	no	yes
	< 0	$\leq INT32_MAX$	yes	no
	< 0	$> INT32_MAX$	does not apply	

Table 7: Determination which sensor reading values to use.

For example:

```
PDU2-MIB::circuitSensorSignedMinimum.1.1.phaseAngle = INTEGER: -1800
PDU2-MIB::measurementsInletPoleSensorSignedValue.1.1.3.phaseAngle = INTEGER: -1200
```

The first line states that the phase angle sensor value can have negative values. That indicates that the signed integer has to be used. In the above example the phase angle is -120 degrees.

The same procedure applies to determining whether the signed or unsigned values must be used when reading sensor thresholds or sensor log entries.