snmp$_{o}rm$ Documentation

Release 0.1.0

Lipin Dmitriy

January 08, 2013

CONTENTS

CI status:

Source code is hosted on Github

CHAPTER

ONE

INTRODUCTION

snmp_orm is a Python-based tool providing a simple interface to work SNMP agents. Here is a very simplistic example that allows to display the system information of a given host:

```
from snmp_orm import get_device
d = get_device("127.0.0.1")
print dict(d.system)
```

CHAPTER

TWO

REFERENCE DOCS

2.1 Getting Started

CHAPTER

THREE

REFERENCE DOCS

3.1 API Documentation

The API documentation is organized alphabetically by module name.

3.1.1 snmp_orm Package

snmp_orm Package

Python-based tool providing a simple interface to work SNMP agents.

snmp_orm.__init__.**get_device**(*host*, ***kwargs*)

Return *Device* instance for specified host.

Arguments for device (for write access use *write_* prefix):

- **host** – IP or hostname if snmp agent;
- **port** – agent UDP port, default 161;
- **version** – SNMP version, 1, 2 or 3;
- **registry** – custom registry class for class lookup;
- **class_name** – adapter class;
- **community** – SNMP community;
- **sec_name** – security name;
- **sec_level** – security level;
- **auth_protocol** – auth protocol;
- **auth_passphrase** – auth passphrase;
- **priv_protocol** – priv protocol;
- **priv_passphrase** – priv passphrase.

Other arguments:

- **manager** custom instance of manager class;
- **device_cls** custom device class, will be used instead of autolookup.

adapter Module

Load and store all existed adapter's classes.

class snmp_orm.adapter.**AdapterRegistry**
Bases: dict

Storage for adapter's classes.

get_class(*module_name*)

snmp_orm.adapter.**get_adapter**(*host*, *version=None*, *class_name=None*, ***kwargs*)
Create adapter instance for given host with given settings.

snmp_orm.adapter.**registry** = {'snmp_orm.adapters.pysnmp': <class 'snmp_orm.adapters.pysnmp.Adapter'>}
Global storage of adapters.

config Module

Default configuration for snmp_orm.

snmp_orm.config.**BULK_ROW** = 50
How many rows should be read in bulk-mode at once.

snmp_orm.config.**DEBUG** = False
Should we print debug info?

snmp_orm.config.**DEFAULT_ADAPTER** = 'snmp_orm.adapters.pysnmp'
Which adapter we should use by default?

snmp_orm.config.**OID_OBJECT_ID** = '1.3.6.1.2.1.1.2.0'
Which OID should be used to detect device model.

snmp_orm.config.**SNMP_PORT** = 161
Default SNMP device's port to connect.

snmp_orm.config.**SNMP_TEST_AGENT_ADDRESS** = ('localhost', 60161)
Default SNMP device's address to connect, used in unit-tests.

defines Module

Some constants for snmp_orm.

snmp_orm.defines.**IANAifType** = {1: 'other', 2: 'regular1822', 3: 'hdh1822', 4: 'ddnX25', 5: 'rfc877x25', 6: 'ethernetCs
1.3.6.1.2.1.2.2.1.3

snmp_orm.defines.**ifStatus** = {1: 'up', 2: 'down', 3: 'testing', 4: 'unknown', 5: 'dormant', 6: 'notPresent', 7: 'lowerLay
1.3.6.1.2.1.2.2.1.8 (ifOperStatus)

snmp_orm.defines.**ipForwarding** = {1: 'forwarding', 2: 'not-forwarding'}
1.3.6.1.2.1.4.1 (ipForwarding)

snmp_orm.defines.**ipNetToMediaType** = {1: 'other', 2: 'invalid', 3: 'dynamic', 4: 'static'}
1.3.6.1.2.1.4.22.1.4 (ipNetToMediaType)

snmp_orm.defines.**ipRouteProto** = {1: 'other', 2: 'local', 3: 'netmgmt', 4: 'icmp', 5: 'egp', 6: 'ggp', 7: 'hello', 8: 'rip', 9
1.3.6.1.2.1.4.21.1.9 (ipRouteProto)

snmp_orm.defines.**ipRouteType** = {1: 'other', 2: 'invalid', 3: 'direct', 4: 'indirect'}
1.3.6.1.2.1.4.21.1.8 (ipRouteType)

device Module

Manager for devices.

class snmp_orm.device.**DeviceClassRegistry**(*packages=None*, *default=None*)
Bases: dict

Store mapping of OID to device class.

Arguments:

- **packages, iterable, contains list of packages with defined** devices.Device;
- **default**, default class, that should returned if no class found;

class **AbstractDevice**(*host*, ***kwargs*)
Bases: snmp_orm.devices.base.NewBase

Abstract device class, used to find other devices in package.

classId = None

meta = <snmp_orm.devices.base.DeviceMeta instance at 0x2e5b680>

prepare_val_by_oid(*oid*, *var*)
Prepare value for given OID.

DeviceClassRegistry.**add**(*cls*)
Add class to registry.

DeviceClassRegistry.**find_devices**()
Find existed device classes.

DeviceClassRegistry.**get_class**(*key*)
Get class by it's OID or return default.

class snmp_orm.device.**DeviceManager**(*registry=None*)
Bases: object

Used to get class of given host.

Registry
alias of DeviceClassRegistry

get_class(*host*, ***kwargs*)

snmp_orm.device.**default_manager = <snmp_orm.device.DeviceManager object at 0x2e6ae90>**
Default device manager instance.

snmp_orm.device.**get_device**(*host*, ***kwargs*)
Return *Device* instance for specified host.

Arguments for device (for write access use *write_* prefix):

- **host** – IP or hostname if snmp agent;
- **port** – agent UDP port, default 161;
- **version** – SNMP version, 1, 2 or 3;
- **registry** – custom registry class for class lookup;
- **class_name** – adapter class;
- **community** – SNMP community;
- **sec_name** – security name;

- **sec_level** – security level;
- **auth_protocol** – auth protocol;
- **auth_passphrase** – auth passphrase;
- **priv_protocol** – priv protocol;
- **priv_passphrase** – priv passphrase.

Other arguments:

- **manager** custom instance of manager class;
- **device_cls** custom device class, will be used instead of autolookup.

snmp_orm.device.**maybe_oid_to_str**(*objectId*)
Convert given OID to string if needed.

`fields` Module

Implement fields here.

snmp_orm.fields.**Field**

snmp_orm.fields.**FromDictField**
Convert data to dict value

class snmp_orm.fields.**FromDictMapper**
Bases: snmp_orm.fields.Mapper

form(*var*)

toAsn1(*var*)

snmp_orm.fields.**FromDictTableField**
Convert data to dict value

class snmp_orm.fields.**Group**(***fields*)
Bases: object

snmp_orm.fields.**IPAddressField**
Convert data to IP

class snmp_orm.fields.**IPAddressMapper**
Bases: snmp_orm.fields.Mapper

form(*var*)

toAsn1(*var*)

snmp_orm.fields.**IPAddressTableField**
Convert data to IP

snmp_orm.fields.**IntegerField**
Convert data to int

class snmp_orm.fields.**IntegerMapper**
Bases: snmp_orm.fields.Mapper

form(*var*)

toAsn1(*var*)

snmp_orm.fields.**IntegerTableField**
Convert data to int

snmp_orm.fields.**LongIntegerField**
Convert data to long

class snmp_orm.fields.**LongIntegerMapper**
Bases: snmp_orm.fields.Mapper

form(*var*)

toAsn1(*var*)

snmp_orm.fields.**LongIntegerTableField**
Convert data to long

snmp_orm.fields.**MacField**
Convert data to MAC

class snmp_orm.fields.**MacMapper**
Bases: snmp_orm.fields.Mapper

form(*var*)

toAsn1(*var*)

snmp_orm.fields.**MacTableField**
Convert data to MAC

class snmp_orm.fields.**Mapper**
Bases: object

form(*var*)
Form method must convert pyasn1 format to base python objects

toAsn1(*var*)
toAsn1 method must convert base python objects to pyasn1 format

snmp_orm.fields.**OIDField**
Convert data to OID tuple

class snmp_orm.fields.**OIDMapper**
Bases: snmp_orm.fields.Mapper

form(*var*)

toAsn1(*var*)

snmp_orm.fields.**OIDTableField**
Convert data to OID tuple

snmp_orm.fields.**SingleValueField**

class snmp_orm.fields.**TableField**
Bases: object

load_many(*adapter*)

load_one(*adapter*, *key*)

prepare_many(*vars*)

set_one(*adapter*, *key*, *value*)

snmp_orm.fields.**TableValueField**

snmp_orm.fields.**TimeTickField**
Convert data to timedelta

class snmp_orm.fields.**TimeTickMapper**
Bases: snmp_orm.fields.IntegerMapper

form(*var*)

toAsn1(*var*)

snmp_orm.fields.**TimeTickTableField**
Convert data to timedelta

snmp_orm.fields.**UnicodeField**
Convert data to unicode

class snmp_orm.fields.**UnicodeMapper**
Bases: snmp_orm.fields.Mapper

form(*var*)

toAsn1(*var*)

snmp_orm.fields.**UnicodeTableField**
Convert data to unicode

snmp_orm.fields.**format_key**(*key*)

settings Module

Implement device settings.

class snmp_orm.settings.**BaseSettings**(***kwargs*)
Bases: snmp_orm.settings.NewBase

allowed_keys = ('host', 'port', 'version', 'use_bulk', 'bulk_rows')

allowed_keys_set = set(['use_bulk', 'host', 'version', 'port', 'bulk_rows'])

default_values = {'use_bulk': True, 'version': <function <lambda> at 0x2bea2a8>, 'port': 161, 'bulk_rows': 50}

prepare_kwargs()

set_default(*key*, *default*)

class snmp_orm.settings.**SettingsMeta**
Bases: type

class snmp_orm.settings.**SnmpV2Settings**(***kwargs*)
Bases: snmp_orm.settings.BaseSettings

allowed_keys = ('community', 'host', 'port', 'version', 'use_bulk', 'bulk_rows')

allowed_keys_set = set(['community', 'bulk_rows', 'host', 'version', 'use_bulk', 'port'])

default_values = {'use_bulk': True, 'version': <function <lambda> at 0x2bea2a8>, 'port': 161, 'community': 'public

class snmp_orm.settings.**SnmpV3Settings**(***kwargs*)
Bases: snmp_orm.settings.BaseSettings

allowed_keys = ('sec_name', 'sec_level', 'auth_protocol', 'auth_passphrase', 'priv_protocol', 'priv_passphrase', 'host'

allowed_keys_set = set(['version', 'sec_name', 'sec_level', 'port', 'auth_protocol', 'host', 'priv_protocol', 'bulk_rows

default_values = {'bulk_rows': 50, 'version': <function <lambda> at 0x2bea2a8>, 'use_bulk': True, 'port': 161}

utils Module

Some useful tools.

snmp_orm.utils.**find_classes**(*cls*, *packages*)
Find all subclass of given class.

snmp_orm.utils.**get_all_parents**(*cls*)

snmp_orm.utils.**load_modules**(*packages*)
Find all attributes of given modules.

snmp_orm.utils.**oid_to_str**(*t*)

snmp_orm.utils.**str_to_oid**(*s*)

snmp_orm.utils.**symbol_by_name**(*name*, *aliases={}*, *imp=None*, *package=None*, *sep='.'*, *default=None*, ***kwargs*)
Get symbol by qualified name.

The name should be the full dot-separated path to the class:

```
modulename.ClassName
```

Example:

```
celery.concurrency.processes.TaskPool
                            ^- class name
```

or using ':' to separate module and symbol:

```
celery.concurrency.processes:TaskPool
```

If *aliases* is provided, a dict containing short name/long name mappings, the name is looked up in the aliases first.

Examples:

```
>>> symbol_by_name("celery.concurrency.processes.TaskPool")
<class 'celery.concurrency.processes.TaskPool'>

>>> symbol_by_name("default", {
...     "default": "celery.concurrency.processes.TaskPool"})
<class 'celery.concurrency.processes.TaskPool'>
```

Does not try to look up non-string names. >>> from celery.concurrency.processes import TaskPool >>> symbol_by_name(TaskPool) is TaskPool True

Subpackages

adapters Package

base Module Abstract adapter class.

class snmp_orm.adapters.base.**AbstractAdapter**(*settings_read*, *settings_write=None*)
Bases: object

get(**args*)
Return tuple of pairs:

```
((1, 3, 6, 1, 2, 1, 1, 1, 0),
    OctetString('DGS-3100-24 Gigabit stackable L2 Managed Switch'))
```

get_one(*oid*)
Return oid value.

get_snmp_v2_session(*host*, *port*, *version*, *community*, ***kwargs*)

get_snmp_v3_session(*host*, *port*, *version*, *sec_name=None*, *sec_level=None*, *auth_protocol=None*, *auth_passphrase=None*, *priv_protocol=None*, *priv_passphrase=None*, ***kwargs*)

getbulk(*rows=None*, **args*)
Return same as getnext method, but use rows number.

getnext(**args*)
Return table:

```
[((1, 3, 6, 1, 2, 1, 1, 1, 0),
  OctetString('DGS-3100-24 Gigabit stackable L2 Managed Switch')),
 ((1, 3, 6, 1, 2, 1, 1, 2, 0),
  ObjectIdentifier('1.3.6.1.4.1.171.10.94.1')),
 ((1, 3, 6, 1, 2, 1, 1, 3, 0),
  TimeTicks('512281800')),
 ((1, 3, 6, 1, 2, 1, 1, 4, 0),
  OctetString(''))]
```

set(**args*)

walk(*oid*)
Collect all rows in given OID.

exception snmp_orm.adapters.base.**AbstractException**
Bases: exceptions.Exception

class snmp_orm.adapters.base.**Walker**(*agent*, *baseoid*, *use_bulk=True*, *bulk_rows=None*)
Bases: six.Iterator

SNMP walker class

snmp_orm.adapters.base.**log**(*f*)

pysnmp Module Abstract adapter class.

class snmp_orm.adapters.pysnmp.**AbstractSession**(*host*, *port=None*)
Bases: object

format_varBindTable(*varBindTable*)

format_varBinds(*varBinds*)

get(**args*)

getbulk(*rows*, **args*)

getnext(**args*)

handle_error(*errorIndication*, *errorStatus*, *errorIndex*, *varBinds=None*, *varBindTable=None*)

set(**args*)

class snmp_orm.adapters.pysnmp.**Adapter**(*settings_read*, *settings_write=None*)
Bases: snmp_orm.adapters.base.AbstractAdapter

get_snmp_v2_session(*host*, *port*, *version*, *community*, ***kwargs*)

get_snmp_v3_session(*host*, *port*, *version*, *sec_name*, *sec_level*, *auth_protocol*, *auth_passphrase*, *priv_protocol*, *priv_passphrase*, ***kwargs*)

exception snmp_orm.adapters.pysnmp.**PySNMPError**
Bases: snmp_orm.adapters.base.AbstractException

class snmp_orm.adapters.pysnmp.**Session**(*host*, *port*, *version*, *community*)
Bases: snmp_orm.adapters.pysnmp.AbstractSession

class snmp_orm.adapters.pysnmp.**UsmSession**(*host*, *port=None*, *sec_name=None*, *sec_level=None*, *auth_protocol=None*, *auth_passphrase=None*, *priv_protocol=None*, *priv_passphrase=None*)
Bases: snmp_orm.adapters.pysnmp.AbstractSession

devices Package

devices Package

base Module Contains abstract device from which all other should be inherited.

class snmp_orm.devices.base.**AbstractContainer**(*adapter*, *meta*)
Bases: object

Container for group of fields. Created for each device and provide access to fields in given group.

group = None

items_list = None

keys()
Return all field's names.

prefix = None

class snmp_orm.devices.base.**AbstractDevice**(*host*, ***kwargs*)
Bases: snmp_orm.devices.base.NewBase

Abstract device class.

classId = None
Used to find device class by object OID.

meta = <snmp_orm.devices.base.DeviceMeta instance at 0x2e5b680>
Device meta-data.

prepare_val_by_oid(*oid*, *var*)
Prepare value for given OID.

class snmp_orm.devices.base.**DeviceBase**
Bases: type

class snmp_orm.devices.base.**DeviceMeta**
Meta-data for each device type.

fields = None
Device's fields mapping.

get_adapter(*host*, ***kwargs*)

groups = None
Device's group mapping.

lut = None
Device's lookup table. OID to field mapping.

class snmp_orm.devices.base.**FieldInfo**
Bases: tuple

Contains information about field.

cls
Alias for field number 1

name
Alias for field number 0

class snmp_orm.devices.base.**TableListProxy**(*adapter*, *field*)
Bases: dict

Proxy that support TableField.

clear(**args*, ***kwargs*)

copy(**args*, ***kwargs*)

fromkeys(**args*, ***kwargs*)

get(**args*, ***kwargs*)

get_by_index(*key*)

has_key(**args*, ***kwargs*)

items(**args*, ***kwargs*)

iteritems(**args*, ***kwargs*)

iterkeys(**args*, ***kwargs*)

itervalues(**args*, ***kwargs*)

keys(**args*, ***kwargs*)

load()

method_name = '__ge__'

pop(**args*, ***kwargs*)

popitem(**args*, ***kwargs*)

setdefault(**args*, ***kwargs*)

update(**args*, ***kwargs*)

values(**args*, ***kwargs*)

viewitems(**args*, ***kwargs*)

viewkeys(**args*, ***kwargs*)

viewvalues(**args*, ***kwargs*)

snmp_orm.devices.base.**get**(*adapter*, *field*, *index=None*)

snmp_orm.devices.base.**load**(*fn*)
Ensure that class will be initialized before method call.

snmp_orm.devices.base.**set_one**(*adapter*, *field*, *value*)

default Module Device that contains default OIDs that implemented by many devices.

class snmp_orm.devices.default.**Device**(*host*, ***kwargs*)
Bases: snmp_orm.devices.base.AbstractDevice

Device with default OID's defined.

ifNumber
SNMP MIB-2 Interfaces (1.3.6.1.2.1.2)

ifTable
alias of Container

ip
ip (1.3.6.1.2.1.4)

alias of Container

meta = <snmp_orm.devices.base.DeviceMeta instance at 0x2e6bea8>

system
SNMP MIB-2 System (1.3.6.1.2.1.1)

alias of Container

Subpackages

dlink Package

sw3100 Module Implement custom OID for D-link SW-3100 switch.

class snmp_orm.devices.dlink.sw3100.**Device**(*host*, ***kwargs*)
Bases: snmp_orm.devices.default.Device

classId = '1.3.6.1.4.1.171.10.94.1'

meta = <snmp_orm.devices.base.DeviceMeta instance at 0x2e70950>

tests Package

agent Module
class snmp_orm.tests.agent.**Agent**(*host*, *port*)
Bases: object

cbFun(*transportDispatcher*, *transportDomain*, *transportAddress*, *wholeMsg*)

prepare()

registerInstr(*instr*)

start()

stop()
class snmp_orm.tests.agent.**BackgroundAgent**(*host*, *port*)
Bases: snmp_orm.tests.agent.Agent

Container
alias of Thread

start()

stop()

class snmp_orm.tests.agent.**Instr**
Bases: object

Abstract MIB instruction.

execute(*module*, **args*, ***kwargs*)

name

class snmp_orm.tests.agent.**SysDescr**
Bases: snmp_orm.tests.agent.Instr

execute(*module*)

name = (1, 3, 6, 1, 2, 1, 1, 1, 0)

class snmp_orm.tests.agent.**Uptime**
Bases: snmp_orm.tests.agent.Instr

birthday = 1357649732.682215

execute(*module*)

name = (1, 3, 6, 1, 2, 1, 1, 3, 0)

test_adapter Module Load and store all existed adapter's classes.

class snmp_orm.tests.test_adapter.**TestSequenceFunctions**(*methodName='runTest'*)
Bases: snmp_orm.tests.utils.TestCase

setUp()

test_adapter_class()

test_adapter_get()

test_adapter_result_type()

utils Module
class snmp_orm.tests.utils.**TestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

instructions = None

setUp()

tearDown()

3.2 Changelog

3.2.1 0.1.0 (initial release)

- Base prototype;

CHAPTER

FOUR

THANKS

A number of people have contributed to snmp_orm by reporting problems, suggesting improvements or submitting changes. Some of these people are:

- Xiongfei(Alex) GUO <xfguo@credosemi.com>
- Lei(Carmack) Jia <jialer.2007@gmail.com>

CHAPTER

FIVE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

S