pySunSpec Documentation

Release 1.0.3

SunSpec Alliance

Mar 29, 2021

Contents

Contents:

CHAPTER 1

Overview

The pySunSpec package provides objects and applications that support interaction with SunSpec compliant devices and documents. It can be run in most environments that support Python and is tested on Windows 7, macOS, and Ubuntu.

1.1 Features

- Provides access to SunSpec Modbus RTU and TCP devices
- High level object model allowing easy device scripting
- Minimal dependencies for core package allowing it to run in more constrained Python environments
- Runs on Windows, Mac, and Linux.

1.2 Requirements

- Python 2.7, 3.5-3.8
- pySerial

1.3 Installation

1.3.1 Windows

Python

Python is not included in the standard Windows 7 installation. To check the Python installation, open a Command Promt window and try to run Python. You should see the Python interactive prompt. Use ^Z <return> to exit Python:

```
C:\> python
Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

If your Windows environment does not have Python 2.x already installed, install the 32-bit verion of Python using the Python 2.7.x Windows installers at http://www.python.org/download/. Accept all the default settings during installation.

Setting Environment Variables

Set the PATH environment variable to include directories that contain Python packages and scripts. The environment variable settings are at: Computer > System properties > Advanced system settings > Environment Variables.

In the System variables section add the following to the 'Path' variable (if using a version different than 2.7.x, adjust the string):

```
C:\Python27;C:\Python27\Lib\site-packages\;C:\Python27\Scripts\;
```

If you would like to be able to execute Python scripts without specifying the '.py' in the script name, you can also add the '.PY' extension to the end of the 'PATHEXT' variable.

pySerial

To install the pySerial package, use the Window installer option (pyserial-x.y-win32.exe) at https://pypi.python.org/pypi/pyserial.

pySunSpec

Download the pysunspec archive (.zip) from .

Unpack the archive, enter the `pysunspec-x.y` directory and run:

```
C:\> python setup.py install
```

You can test the installation by opening a Command Prompt window and running the unittest discover command. You should see the results of the test execution with no test failures:

```
C:\> python -m unittest discover -v sunspec

test_client_device (core.test.test_client.TestClientDevice) ... ok
test_sunspec_client_device_1 (core.test.test_client.TestClientDevice) ... ok
test_sunspec_client_device_3 (core.test.test_client.TestClientDevice) ... ok
test_data (core.test.test_data.TestData) ... ok
test_device_blocktype_not_equal (core.test.test_device.TestDevice) ... ok
test_device_common_len_65 (core.test.test_device.TestDevice) ... ok
test_device_constant_sf (core.test.test_device.TestDevice) ... ok
test_device_from_pics (core.test.test_device.TestDevice) ... ok
test_device_models_smdx (core.test.test_device.TestDevice) ... ok
```

(continues on next page)

(continued from previous page)

```
test_device_modeltype (core.test.test_device.TestDevice) ... ok
test_device_modeltype_not_equal (core.test.test_device.TestDevice) ... ok
test_device_pointtype (core.test.test_device.TestDevice) ... ok
test_device_pointtype_not_equal (core.test.test_device.TestDevice) ... ok
test_device_to_pics (core.test.test_device.TestDevice) ... ok
test_device_value_get (core.test.test_device.TestDevice) ... ok
test_device_value_set (core.test.test_device.TestDevice) ... ok
test_modbus_client_device_rtu_read (core.test.test_modbus_client.TestModbusClient) ...
↪ ok
test_modbus_client_device_rtu_write (core.test.test_modbus_client.TestModbusClient) ..
↪. ok
test_modbus_client_device_tcp_read (core.test.test_modbus_client.TestModbusClient) ...
↪ ok
test_modbus_client_device_tcp_write (core.test.test_modbus_client.TestModbusClient) ..
↪. ok
test_modbus_mbmap_from_xml_element (core.test.test_modbus_mbmap.TestModbusMap) ... ok
test_modbus_mbmap_from_xml_file (core.test.test_modbus_mbmap.TestModbusMap) ... ok

----------------------------------------------------------------------
Ran 22 tests in 0.634s

OK

C:\>
```

You should now be ready to use the pySunSpec package.

1.3.2 Mac/Linux

Note: The following installation steps are performed from a terminal window. If a standard installation is performed, sudo (or the equivalent on the system) is necessary to allow update of the system directories.

Python

Verify Python 2.x is installed. Most current Mac/Linux systems come with Python already installed. To check the Python installation, open a terminal window and try to run Python. You should see the Python interactive prompt. Use ^D to exit Python:

```
$ python
Python 2.7.1 (r271:86832, Jul 31 2011, 19:30:53)
[GCC 4.2.1 (Based on Apple Inc. build 5658) (LLVM build 2335.15.00)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

For more detailed information about using Python in the Mac environment see http://docs.python.org/2/using/mac.html.

pySerial

To install the pySerial package, try using easy_install:

```
$ easy_install -U pyserial
```

If easy install is not present on the system download the archive (.tar.gz) from https://pypi.python.org/pypi/pyserial. Unpack the archive, enter the `pyserial-x.y` directory and run:

```
$ python setup.py install
```

pySunSpec

Download the pysunspec archive (.zip) from .

Unpack the archive, enter the `pysunspec-x.y` directory and run:

```
$ python setup.py install
```

You can test the installation by opening a Command Prompt window and running the unittest discover command. You should see the results of the test execution with no test failures:

```
$ python -m unittest discover -v sunspec

test_client_device (core.test.test_client.TestClientDevice) ... ok
test_sunspec_client_device_1 (core.test.test_client.TestClientDevice) ... ok
test_sunspec_client_device_3 (core.test.test_client.TestClientDevice) ... ok
test_data (core.test.test_data.TestData) ... ok
test_device_blocktype_not_equal (core.test.test_device.TestDevice) ... ok
test_device_common_len_65 (core.test.test_device.TestDevice) ... ok
test_device_constant_sf (core.test.test_device.TestDevice) ... ok
test_device_from_pics (core.test.test_device.TestDevice) ... ok
test_device_models_smdx (core.test.test_device.TestDevice) ... ok
test_device_modeltype (core.test.test_device.TestDevice) ... ok
test_device_modeltype_not_equal (core.test.test_device.TestDevice) ... ok
test_device_pointtype (core.test.test_device.TestDevice) ... ok
test_device_pointtype_not_equal (core.test.test_device.TestDevice) ... ok
test_device_to_pics (core.test.test_device.TestDevice) ... ok
test_device_value_get (core.test.test_device.TestDevice) ... ok
test_device_value_set (core.test.test_device.TestDevice) ... ok
test_modbus_client_device_rtu_read (core.test.test_modbus_client.TestModbusClient) ...
↪ ok
test_modbus_client_device_rtu_write (core.test.test_modbus_client.TestModbusClient) ..
↪. ok
test_modbus_client_device_tcp_read (core.test.test_modbus_client.TestModbusClient) ...
↪ ok
test_modbus_client_device_tcp_write (core.test.test_modbus_client.TestModbusClient) ..
↪. ok
test_modbus_mbmap_from_xml_element (core.test.test_modbus_mbmap.TestModbusMap) ... ok
test_modbus_mbmap_from_xml_file (core.test.test_modbus_mbmap.TestModbusMap) ... ok

----------------------------------------------------------------------
Ran 22 tests in 0.634s

OK

$
```

You should now be ready to use the pySunSpec package.

1.4 Interacting with a SunSpec Device

The SunSpecClientDevice object is used for high level access to a SunSpec device. It provides the ability to easily read and write all points within the models that comprise the device. The SunSpecClientDevice object is a wrapper around the ClientDevice object to provide the easiest syntax for basic operations. For complete access to the device instance and type information, the ClientDevice object can be referenced within the SunSpecClientDevice object.

The SunSpecClientDevice is populated with dynamically created class objects based on the models found in the device. Point attibutes are added to the model and repeating blocks based on the points in the respective model definitions. The point attribute names are the same as the point ids in the model definitions. For points that have associated scale factors, the point value automatically incorporates the value of the scale factor and the scale factor points do not appear in the points list for the model.

The examples below are shown in interactive mode in the Python interpreter but would typically be performed in a Python script.

Create a device object to access Modbus RTU device at slave id 1 with serial settings of "9600,8,N,1" on serial port 'com6'. The physical device is scanned and the device object is created based on the SunSpec models found in the device.

```
>>> import sunspec.core.client as client
>>> d = client.SunSpecClientDevice(client.RTU, 1, 'com6')
>>>
```

Determine which models are present in the device:

```
>>> print d.models
['common', 'inverter', 'nameplate', 'settings', 'status', 'controls', 'volt_var']
>>>
```

Determine which points are present in a model. The points in the fixed block of a model appear as attributes of the model. Point in the repeating block appear as attributes of the repeating block instance as descriped in the section on repeating block naming below.

```
>>> print d.commmon.points
['Mn', 'Md', 'Opt', 'Vr', 'SN', 'DA']
>>>
```

View common model contents:

```
>>> print d.common

common (1):
Mn:  SunSpecTest
Md:  TestInverter-1
Opt:  opt_a_b_c
Vr:  1.2.3
SN:  sn-123456789
DA:  1

>>>
```

The device object mirrors the values in the actual physical device. When the device object is created all the values are read from the physical device.

To reacquire the values from the physical device, an explicit read operation must be done with a read() operation either on the device or a model within the device. The smallest granularity for a read operation is model to ensure all scale factor values are up to date.

To update the physical device with values that have been set in the device, an explict write() operation must be done on the device or a model within the device. The write operation is performed on the model. Only the fields that have changed in the model are actually written to the physical device. In general the updates to the device are made in Modbus offset order but this should not be assumed so if value update ordering is important, write() operations should be performed between object updates to achieve the desired order.

Perform read() to view latest inverter model contents:

```
>>> d.inverter.read()
>>> print d.inverter

inverter (103):
A:  12.4
AphA:  4.1
AphB:  4.2
AphC:  4.3
PhVphA:  240.1
PhVphB:  240.2
PhVphC:  240.3
W:  2970
Hz:  59.99
VA:  2978
VAr:  0.1
PF:  0.995
WH:  1234567
DCA:  10.0
DCV:  300.1
DCW:  3001
TmpCab:  40.1
TmpSnk:  40.2
TmpTrns:  40.3
TmpOt:  40.4
St:  1

>>>
```

If a model contains repeating blocks, the default block name within the model is 'repeating' which along with an index (starting at 1) can always be used to access the block. If the block has a name specified within the model definition, the name can be also used to access the block as well. The 'repeating_name' attribute of the model contains the alternate name value for the block if one exists. If there is no alternate name, the value of 'repeating_name' is also 'repeating'.

View repeating block name for volt_var model:

```
>>> print d.volt_var.repeating_name
curve
>>>
```

View volt_var model contents:

```
>>> d.volt_var

volt_var (126):
ActCrv:  1
ModEna:  0
WinTms:  0
RvrtTms:  600
NCrv:  2
NPt:  4
```

(continues on next page)

(continued from previous page)

```
curve[1]:
ActPt:  4
DeptRef:  2
V1:  95
VAr1:  100
V2:  98
VAr2:  0
V3:  102
VAr3:  0
V4:  105
VAr4:  -100
RmpDecTmm:  0
RmpIncTmm:  0
ReadOnly:  0

curve[2]:
ActPt:  4
DeptRef:  2
V1:  95
VAr1:  100
V2:  98
VAr2:  0
V3:  102
VAr3:  0
V4:  105
VAr4:  -100
RmpDecTmm:  0
RmpIncTmm:  0
ReadOnly:  0

>>>
```

Update a portion of volt_var curve 2 and make active curve:

```
>>> d.volt_var.curve[2].V1 = 96
>>> d.volt_var.curve[2].VAr1 = 100
>>> d.volt_var.curve[2].V2 = 97
>>> d.volt_var.curve[2].VAr2 = 0
>>> d.volt_var.curve.ActCrv = 2
>>> d.volt_var.write()
>>>
```

Enable volt_var curves:

```
>>> d.volt_var.ModEna = 1
>>> d.volt_var.write()
>>>
```

The close() method should be called for the device object when it is no longer needed:

```
>>> d.close()
```

CHAPTER 2

API Reference

2.1 sunspec.core — SunSpec Core Functionality

The core package contains modules that provide the ability to interact with SunSpec devices and encode/decode documents in SunSpec standard formats. The modules in the core package used to interact with SunSpec devices are organized into the following main functional areas: base SunSpec device/model support, Modbus protocol support, and client device support.

The device module provides objects that map to the SunSpec model definitions within the SunSpec standards. A device is a collection of SunSpec model definitions that are implemented by a SunSpec compliant device. The device module provides objects for device, model, block, and point instances as well as objects that support the model, block, and point type definitions.

The modbus package provides protocol support for Modbus TCP, Modbus RTU, and locally defined Modbus maps that can be accessed directly.

The client module extends the basic objects in the device module to provide objects that can be used to access devices using the Modbus protocol objects supplied in the modbus package.

2.2 sunspec.core.client — SunSpec Client classes

2.2.1 Classes

2.2.2 Exceptions

exception sunspec.core.client.**SunSpecClientError**

Derived from *sunspec.core.device.SunSpecError*. Raised for any sunspec module error in sunspec.core.client classes.

2.2.3 Constants

Device Types

sunspec.core.client.**RTU**

sunspec.core.client.**TCP**

sunspec.core.client.**MAPPED**

RTU Device Parity

sunspec.core.client.**PARITY_NONE**

sunspec.core.client.**PARITY_EVEN**

2.3 sunspec.core.device — SunSpec Device classes

The device module provides the base SunSpec device functionality. A SunSpec device consists of a set of SunSpec models which contain blocks and points as specified in the respective SunSpec model definitions.

2.3.1 Classes

class sunspec.core.device.**Device**(*addr=40000*)

Parameters **addr** – Modbus base address of device.

Raises *SunSpecError* – Any error encountered in device processing.

base_addr
Modbus base address of the device.

models_list
List of model objects present in the device in the order in which they appear in the device.

models
Dictionary of model object lists reperesenting model types present in the device indexed by model id. The elements are model lists to allow more than one model of the same model type to be present in the device.

add_model(*model*)
Add a model object to the device.

Parameters **model** – Model object to add to the device.

from_pics(*element=None*, *filename=None*, *pathlist=None*)
The PICS information for the device can be either an Element Tree element for a device from a document already being processed or the file name of document in the file system. Populates the device based on the elements within the device element.

Parameters

- **element** – Element Tree device element.
- **filename** – File name of the PICS document.
- **pathlist** – Pathlist object containing alternate paths to the PICS document.

to_pics(*parent*, *single_repeating=True*)
Adds the device and all elements within the device to the parent element. If *single_repeating* is True, only the first repeating block for each model is added to the document.

Parameters

- **parent** – Element Tree element on which to place the device element.
- **single_repeating** – Flag to indicate whether to include a single or all repeating blocks within each model in the PICS document.

not_equal(*device*)
Determines if the specified device instance is not equal based on all the device attribute values including models, blocks and points. If not equal, returns a string indicating why the device is not equal. Returns False if the device is equal.

Parameters **device** – Device to compare.

class sunspec.core.device.**Model**(*device=None*, *mid=None*, *addr=0*, *mlen=0*, *index=1*)

Parameters

- **device** – Device associated with the model.
- **mid** – Model id.
- **addr** – Modbus address of the first point in the model.
- **mlen** – Length of the model in Modbus registers.
- **index** – Model instance index for the model type within the device.

Raises *SunSpecError* – Any error encountered in device processing.

device
Device instance that contains the model instance.

id
Model id. The model id maps to a SunSpec model type definition.

index
Model instance index for the model type within the device. Model instance indexes start at 1 for the first model type instance.

model_type
The *sunspec.core.device.ModelType* instance associated with the model.

addr
Modbus address of the first point in the model.

len
Length of the model in Modbus registers.

points_list
List of fixed block non-scale factor points ordered by offset.

points
Dictionary of fixed block non-scale factor points indexed by point id.

points_sf
Dictionary of fixed block scale factor points indexed by point id.

blocks
List of blocks contained in the model instance. Block 0 is the fixed block if present and blocks 1 to n are the repeating block instances.

load(*block_class=<class 'sunspec.core.device.Block'>*, *point_class=<class 'sunspec.core.device.Point'>*)
Loads the model instance with blocks and points based on the SunSpec model type definition.

Parameters

- **block_class** – Block class to use to create block instances.
- **point_class** – Point class to use to create point instances.

from_pics(*element*)
Sets the model contents based on an element tree model type element contained in a SunSpec PICS document.

Parameters **element** – Element Tree model element.

to_pics(*parent*, *single_repeating=True*)
Adds the model and all elements within the model to the parent element. If *single_repeating* is True, only the first repeating block is added to the document.

Parameters

- **parent** – Element Tree element on which to place the model element.
- **single_repeating** – Flag to indicate whether to include a single or all repeating blocks within the model in the PICS document.

not_equal(*model*)
Determines if the specified model instance is not equal based on all the model attribute values including blocks and points. If not equal, returns a string indicating why the model is not equal. Returns False if the model is equal.

Parameters **device** – Model to compare.

class sunspec.core.device.**Block**(*model*, *addr*, *blen*, *block_type*, *index=1*)

Parameters

- **model** – Model associated with the block.
- **addr** – Modbus address of the first point in the block.
- **blen** – Length of the block in Modbus registers.
- **block_type** – The *sunspec.core.device.BlockType* instance associated with the block.
- **index** – Block instance index for the block type within the model.

model
Model associated with the block.

block_type
The *sunspec.core.device.BlockType* instance associated with the block.

addr
Modbus address of the first point in the block.

len
Length of the block in Modbus registers.

type
Block type, either sunspec.core.suns.SUNS_BLOCK_FIXED or sunspec.core.suns.SUNS_BLOCK_REPEATING.

index
Block instance index for the block type within the model.

points_list
List of non-scale factor points in the block ordered by offset.

points
Dictionary of non-scale factor points in the block indexed by point id.

points_sf
Dictionary of scale factor points int the block indexed by point id.

from_pics(*element*)
Sets the block contents based on an element tree model type element contained in a SunSpec PICS document.

Parameters **element** – Element Tree model element.

to_pics(*parent*)
Adds the block and all elements within the block to the parent element.

Parameters **parent** – Element Tree element on which to place the block element.

not_equal(*block*)
Determines if the specified block instance is not equal based on all the block attribute values including points. If not equal, returns a string indicating why the block is not equal. Returns False if the block is equal.

Parameters **device** – Block to compare.

class sunspec.core.device.**Point**(*block=None*, *point_type=None*, *addr=None*, *sf_point=None*, *value=None*)

Parameters

- **block** – Block associated with the point.
- **point_type** – The *sunspec.core.device.PointType* instance associated with the point.
- **addr** – The Modbus address of the point.
- **sf_point** – Scale factor point associated with the point if present.
- **value** – Initial value for the *value_base* attribute of the point.

block
Block associated with the point.

point_type
The *sunspec.core.device.PointType* instance associated with the point.

addr
Modbus address of the point.

sf_point
Scale factor point associated with the point if present.

impl
Indication if the point is implemented. A value of True indicates the point is implmented. Intended to be used for cases when no initial value is given for the point but the implementation status is known as in SunSpec PICS documents.

value_base
Value of the point without the point scale factor applied.

value_sf
Scale factor point value.

dirty
Indication if the point has been written to the physical device since the last update of the point. A value of True indicates that the point has not been written since the last update.

value
Value of the point with the scale factor applied.

from_pics(*element*)
Sets the block contents based on an element tree model type element contained in a SunSpec PICS document.

Parameters **element** – Element Tree model element.

to_pics(*parent*)
Adds the point to the parent element.

Parameters **parent** – Element Tree element on which to place the point element.

not_equal(*point*)
Determines if the specified point instance is not equal based on all the point attribute values. If not equal, returns a string indicating why the point is not equal. Returns False if the point is equal.

Parameters **device** – Point to compare.

class sunspec.core.device.**ModelType**(*mid=None*)

Parameters **mid** – Model id that identifies a specific SunSpec model type definition.

id
Model id that identifies a specific SunSpec model type definition.

len
Length in Modbus registers of the model type as specified in the model definition.

label
Label string as specified in the model definition.

description
Description string as specified in the model definition.

notes
Notes string as specified in the model definition.

fixed_block
Fixed block type as specified in the model definition if present.

repeating_block
Repeating block type as specified in the model definition if present.

from_smdx(*element*)
Sets the model type attributes based on an element tree model type element contained in an SMDX model definition.

Parameters **element** – Element Tree model type element.

not_equal(*model_type*)
Determines if the specified model type instance is not equal based on all the model type attribute values including blocks and points. If not equal, returns a string indicating why the model type is not equal. Returns False if the model type is equal.

Parameters **model_type** – Model type to compare.

class sunspec.core.device.**BlockType**(*btype=None*, *blen=0*, *name=None*, *model_type=None*)

Parameters

- **btype** – Block type as specified in the model definition. Valid values are sunspec.core.suns.SUNS_BLOCK_FIXED or sunspec.core.suns.SUNS_BLOCK_REPEATING.
- **blen** – Block length in Modbus registers.

type
Block type as specified in the model definition. Valid values are sunspec.core.suns.SUNS_BLOCK_FIXED or sunspec.core.suns.SUNS_BLOCK_REPEATING.

len
Block length in Modbus registers.

points_list
List containing the points in the block in offset order.

points
Dictionary containg the points in the block indexed by the point id.

from_smdx(*element*)
Sets the block type attributes based on an element tree block type element contained in an SMDX model definition.

Parameters **element** – Element Tree block type element.

not_equal(*block_type*)
Determines if the specified block type instance is not equal based on all the block type attribute values including points. If not equal, returns a string indicating why the block type is not equal. Returns False if the block type is equal.

Parameters **block_type** – Block type to compare.

class sunspec.core.device.**PointType**(*pid=None*, *offset=None*, *ptype=None*, *plen=None*, *mandatory=None*, *access=None*, *sf=None*, *block_type=None*)

Parameters

- **pid** – Point id as specified in the model definition.
- **offset** – Point offset within the block as specified in the model definition.
- **ptype** – Point type as specified in the model definition. Valid values are defined in sunspec.core.suns.SUNS_TYPE_*.
- **plen** – Point length in Modbus registers for points that have a type of 'string'.
- **mandatory** – Mandatory indication as specified in the model definition. Valid values are sunspec.core.suns.SUNS_MANDATORY_TRUE or sunspec.core.suns.SUNS_MANDATORY_FALSE.
- **access** – Point access setting as specfied in the model definition. Valid values are sunspec.core.suns.SUNS_ACCESS_R or sunspec.core.suns.SUNS_ACCESS_RW.
- **sf** – Id of the scale factor point associated with the point or None if the point does not have a scale factor.

id
Point id as specified in the model definition.

offset
Point offset within the block as specified in the model definition.

type
Point type as specified in the model definition. Valid values are defined in sunspec.core.suns.SUNS_TYPE_*.

len
Point length in Modbus registers for points that have a type of 'string'.

mandatory
Mandatory indication as specified in the model definition. Valid values are sunspec.core.suns.SUNS_MANDATORY_TRUE or sunspec.core.suns.SUNS_MANDATORY_FALSE.

access
Point access setting as specfied in the model definition. Valid values are sunspec.core.suns.SUNS_ACCESS_R or sunspec.core.suns.SUNS_ACCESS_RW.

sf
Id of the scale factor point associated with the point or None if the point does not have a scale factor.

label
Label string as specified in the model definition.

description
Description string as specified in the model definition.

notes
Notes string as specified in the model definition.

value_default
Default value for a point instance if no value specified.

is_impl
Contains the function to call with the point value as an argument to determine if the point is implemented.

data_to
Contains the function to call to transform a binary data string to the point value.

to_data
Contains the function to call to transform the point value to a binary data string.

to_value
Contains the function to call to transform a point value string into a point value of the type associated with the point.

from_smdx(*element*, *strings=False*)
Sets the point attributes based on an element tree point element contained in an SMDX model definition.

Parameters

- **element** – Element Tree point type element.
- **strings** – Indicates if *element* is a subelement of the 'strings' definintion within the model definition.

not_equal(*point_type*)
Determines if the specified point type instance is not equal based on all the point type attribute values. If not equal, returns string indicating why the point type is not equal. Returns False if the point type is equal.

Parameters **point_type** – Point type to compare.

2.3.2 Exceptions

exception sunspec.core.device.**SunSpecError**
Raised for sunspec.core.device errors.

2.4 sunspec.core.modbus — SunSpec Modbus Package

The Modbus package provides standard Modbus support for Modbus RTU, Modbus TCP, and Modbus file mapped client devices.

2.5 sunspec.core.modbus.client — Modbus Client classes

2.5.1 Classes

2.5.2 Exceptions

exception sunspec.core.modbus.client.**ModbusClientError**
Raised for general errors in sunspec.core.modbus.client modules.

exception sunspec.core.modbus.client.**ModbusClientTimeout**
Raised for Modbus timeout errors. Derived from *ModbusClientError*.

exception sunspec.core.modbus.client.**ModbusClientException**
Raised for Modbus protocol exceptions. Derived from *ModbusClientError*.

2.5.3 Constants

Parity

sunspec.core.modbus.client.**PARITY_NONE**

sunspec.core.modbus.client.**PARITY_EVEN**

Read Modbus Functions

sunspec.core.modbus.client.**FUNC_READ_HOLDING**

sunspec.core.modbus.client.**FUNC_READ_INPUT**

2.6 sunspec.core.modbus.mbmap — Modbus Map classes

The mbmap module implements a local Modbus map image. The map supports read and write operations and can be used by Modbus clients in place of an actual modbus device. The module supports an xml encoding for representing modbus maps as a document.

The xml representation has a root element of *mbmap* that contains a set of *regs* elements representing one or more registers in the map. Currently only big endian is supported.

Attributes for *mbmap* element:

Attribute	Description	Valid values	Default value
addr	Base Modbus address	Valid modbus address	40000
func	Modbus function associated with the map	holding, input	holding

Attributes for *regs* element:

Attribute	Description	Valid values	Default value
offset	Register offset	Value with map length	Next offset
type	Register(s) type	s16, u16, s32, u32, s64, u64, f32, f64, string, hexstr	hexstr
len	String length for string type	Value within map length	Length of the *regs* element value

If the registers are contiguous, offset is optional.

The *hexstr* type is a series of ascii hex characters. Spaces are removed from the string before processing and can be used to increase readability.

Example:

```
<mbmap>
  <!-- common model -->
  <regs type="string" len="2">SunS</regs>
  <regs type="u16">1</regs>
  <regs type="u16">66</regs>
  <regs type="string" len="16">SunSpecTest</regs>
  <regs type="string" len="16">TestInverter-1</regs>
  <regs type="string" len="8">opt_a_b_c</regs>
  <regs type="string" len="8">1.2.3</regs>
  <regs type="string" len="16">sn-123456789</regs>
  <regs type="u16">1</regs>
  <regs type="u16">0</regs>

  <!-- short test model -->
  <regs type="u16">63010</regs>
  <regs type="u16">4</regs>

  <!-- example of hexstr -->
  <regs>00 79 00 1E</regs>
  <regs>0079 001E</regs>

  <!-- end of models -->
  <regs type="u16">0xffff</regs>
  <regs type="u16">0</regs>
</mbmap>
```

2.6.1 Classes

class sunspec.core.modbus.mbmap.**ModbusMap**(*slave_id=None*, *func='holding'*, *base_addr=40000*, *ns=None*, *lid=None*, *mapid=None*, *time=None*)

Parameters

- **slave_id** – Modbus slave id.
- **func** –

  Modbus function string associated with the map. Valid values are: *sunspec.core.modbus.mbmap.MBMAP_FUNC_HOLDING* or *sunspec.core.modbus.mbmap.MBMAP_FUNC_INPUT*.
- **base_addr** – Base address of the Modbus map.

Raises *ModbusMapError* – Raised for any modbus map error.

slave_id
Modbus slave id.

func
Actual Modbus function associated with the map.

base_addr
Base address of the Modbus map.

regs
List of *sunspec.core.modbus.mbmap.ModbusMapRegs* blocks that comprise the Modbus register map.

from_xml(*filename=None*, *pathlist=None*, *element=None*)
Load Modbus map from a Modbus map (mbmap) formatted file.

Parameters

- **filename** – File name of the Modbus map file
- **pathlist** – Pathlist object containing alternate paths to the Modbus map file.

read(*addr*, *count*, *op=None*)
Read Modbus map registers.

Parameters

- **addr** – Starting Modbus address.
- **count** – Read length in Modbus registers.

Returns Byte string containing register contents.

write(*addr*, *data*)
Write Modbus map registers.

Parameters

- **addr** – Starting Modbus address.
- **count** – Byte string containing register contents.

not_equal(*mbmap*)
Determines if the specified modbus map instance is not equal based on the content of the map. If not equal, returns a string indicating why the map is not equal. Returns False if the map is equal.

class sunspec.core.modbus.mbmap.**ModbusMapRegs**(*offset*, *count*, *data*, *access='r'*)

Parameters

- **offset** – Register offset into Modbus map.
- **count** – Register count.
- **data** – Byte string containing register data.

- **access** –

  Access for the register block. Valid values are: `sunspec.core.modbus.mbmap.MBMAP_REGS_ACCESS_R` and `sunspec.core.modbus.mbmap.MBMAP_REGS_ACCESS_RW`.

Raises exception ModbusMapError: Raised for any modbus map error.

offset
Start register offset of the register block.

count
Register count in the block.

data
Byte string containing data in the register block.

access
Access setting for the block. The access setting is currently not enforced.

read(*offset*, *count*)
Read Modbus map registers in register block.

Parameters

- **offset** – Register offset into Modbus map.
- **count** – Register count.

Returns Byte string containing register contents.

write(*offset*, *data*)
Write Modbus map registers tp register block.

Parameters

- **addr** – Register offset into Modbus map.
- **count** – Byte string containing register contents.

append(*offset*, *count*, *data*, *access='r'*)
Append registers to end of register block.

Parameters

- **offset** – Register offset into Modbus map.
- **count** – Register count.
- **data** – Byte string containing register data.
- **access** –

  Access for the register block. Valid values are: `sunspec.core.modbus.mbmap.MBMAP_REGS_ACCESS_R` and `sunspec.core.modbus.mbmap.MBMAP_REGS_ACCESS_RW`.

not_equal(*regs*)
Determines if the specified modbus map block instance is not equal based on the content of the map block. If not equal, returns a string indicating why the map block is not equal. Returns False if the map block is equal.

2.6.2 Exceptions

exception sunspec.core.modbus.mbmap.**ModbusMapError**
Raised for errors in sunspec.core.modbus.mbmap modules.

2.6.3 Constants

Modbus Map Functions

sunspec.core.modbus.mbmap.**MBMAP_FUNC_INPUT**

sunspec.core.modbus.mbmap.**MBMAP_FUNC_HOLDING**

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

M

N

O

P

R

S

T

V

W