SimpleClient 05-Dec-2010

Table of Contents

Overview: **2**

Supported Modbus Functions: 2

Supported SBus Commands 2

SimpleClient Classes: 2

SimpleClient for Modbus/TCP **3**

Initialisation 3

SendRequest for Modbus 3

ReceiveResponse for Modbus 4

SimpleClient for SAIA Ether SBus **5**

Initialisation 5

SendRequest for SBus 5

ReceiveResponse for SBus 6

SimpleClient for MBLogic Web Service: **6**

Initialisation 7

SendRequest 7

Creating an Application: **8**

Exception Handling: **8**

Library Dependencies: **9**

ModbusDataLib Data Conversion Libraries **9**

List Oriented Functions 9

Miscellaneous 10

Word and Packed Binary String Conversions 10

SBus Conversions: 11

SimpleClient for Modbus/TCP (Obsolete Version) **11**

Initialisation 11

SendRequest for Modbus 11

ReceiveResponse for Modbus 12

ModbusDataStrLib (Obsolete): **13**

String oriented functions: 13

List oriented functions: 13

Coil value conversions: 13

Register value conversions: 13

Other conversions: 13

Overview:

SimpleClient is a simple Python library that may be used to help write custom client applications that communicate using industrial protocols. The following protocols are presently supported:

- Modbus/TCP
- A Modbus-like web service
- SAIA Ether SBus

Supported Modbus Functions:

The following Modbus functions are supported:

- 1- Read Coils.
- 2- Read Discrete Inputs.
- 3- Read Holding Registers.
- 4- Read Input Registers.
- 5- Write Single Coil.
- 6- Write Single Holding Register.
- 15 - Write Multiple Coils.
- 16 - Write Multiple Holding Registers.

Supported SBus Commands

The following SBus commands are supported:

- 2 - Read Flags
- 3 - Read Inputs
- 5 - Read Outputs
- 6 - Read Registers
- 11 - Write flags
- 13 - Write outputs
- 14 - Write Registers

SimpleClient Classes:

Each SimpleClient protocol is located in a separate module. The module names are:

Module Name	Protocol
ModbusTCPSimpleClient2.py	Modbus/TCP
ModbusRestSimpleClient.py	Modbus-like web service
SBusSimpleClient.py	Ether SBus

In addition, there is also an older module called ModbusTCPSimpleClient.py. This also provides a Modbus/TCP interface, but takes a different set of parameters and return values. It has been replaced by ModbusTCPSimpleClient2.py which provides an improved interface. You may use the older ModbusTCPSimpleClient.py for existing applications, but you should use the newer version for all new applications.

SimpleClient for Modbus/TCP

The methods (function calls) provided for Modbus/TCP are:

- SendRequest - This method sends a request.
- ReceiveResponse - This method receives the response.
- MakeRawRequest - This method constructs a message, but does not send it.
- SendRawRequest - This method sends a previously constructed message.
- GetRawResponse - This method recieves a response, but does not attempt to decode it.

SendRequest and ReceiveResponse are the methods normally used for creating an ordinary client program. MakeRawRequest, SendRawRequest, and GetRawResponse are not used in a normal client. The ability to manipulate a raw message however is sometimes useful when creating a program which is intended to test a server by sending malformed packets. As they are seldom used, they are not discussed further here.

Initialisation

The host parameters must be specified at initialisation:

Parameter Name	Type	Range	Description
host	string	Any valid host name	IP address of server
port	integer	Any valid port number	Port number of server
timeout	real	Any valid real number	Time-out in seconds.

A typical example is:

```
host = '192.168.10.1'
port = 502
timeout = 10.5
client = ModbusTCPSimpleClient2.ModbusTCPSimpleClient(host, port, timeout)
```

SendRequest for Modbus

The parameters are:

Parameter Name	Type	Range	Description
TransID	integer	0 - 65535	Modbus transaction ID
UnitID	integer	0 - 255	Modbus Unit ID
Function Code	integer	Any valid code	Modbus function code
Addr	integer	0 - 65535	Modbus memory address to read from server
Qty	integer	0 - 65535	Quantity of items to read from server
MsgData	string	Any valid data	A packed binary string containing the data to send

If the function code is not supported, a ParamError exception will be raised. Invalid data will cause a Python language exception to be raised.

Note: MsgData is optional for functions which do not send data.

A typical example is:

```
TransID = 1
UnitID = 50
FunctionCode = 1
Addr = 567
Qty = 64
client.SendRequest(TransID, UnitID, FunctionCode, Addr, Qty)
```

Another typical example:

```
TransID = 1
UnitID = 50
FunctionCode = 5
Addr = 567
Qty = 64
MsgData = '\xFF\x00'
client.SendRequest(TransID, UnitID, FunctionCode, Addr, Qty, MsgData)
```

ReceiveResponse for Modbus

ReceiveResponse returns the following values:

Parameter	Type	Description
Transation ID	integer	This is an echo of the sent TID
Function	integer	This is an echo of the sent function code. If a Modbus error has occured, this will be the original function + 128.
Address	integer	For 5, 6, 15, 16, the starting address written to. For all others, this is 0.
Qty	integer	For 1, 2, 3, 4, this is the byte count. For 5, 6, this is 1, For 15, 16, the quantity written.
Message Data	string	The message data
Exception Code	integer	The Modbus exception code. This is 0 if there was no error.

Note: Message Data is a packed binary string containing the response data.

A typical example is:

```
TransID, Funct, Address, Qty, Data, ExCode = client.ReceiveResponse()
```

If a message cannot be decoded, a Python language exception will be raised. If the function is not supported, a ParamError exception will be raised. If the message is too short or too long to be a valid Modbus/TCP message, a MessageLengthError exception will be raised.

Software Exceptions:

SimpleClient2 raises exceptions for invalid parameters or messages. The exceptions are:

Exception	Description
MessageLengthError	The received message was of a length which was too short or too long to decode.
ParamError	The function code is not supported by SimpleClient.

SimpleClient for SAIA Ether SBus

The methods (function calls) provided for Modbus/TCP are:

- SendRequest - This method sends a request.
- ReceiveResponse - This method receives the response.
- MakeRawRequest - This method constructs a message, but does not send it.
- SendRawRequest - This method sends a previously constructed message.
- GetRawResponse - This method recieves a response, but does not attempt to decode it.

SendRequest and ReceiveResponse are the methods normally used for creating an ordinary client program. MakeRawRequest, SendRawRequest, and GetRawResponse are not used in a normal client. The ability to manipulate a raw message however is sometimes useful when creating a program which is intended to test a server by sending malformed packets. As they are seldom used, they are not discussed further here.

Initialisation

The host parameters must be specified at initialisation:

Parameter Name	Type	Range	Description
host	string	Any valid host name	IP address of server
port	integer	Any valid port number	Port number of server
timeout	real	Any valid real number	Time-out in seconds.

SendRequest for SBus

For SBusSimpleClient the parameters are:

Parameter Name	Type	Range	Description
msgsequence	integer	0 - 65535	SBus messge sequence
stnaddr	integer	0 - 255	SBus station address
cmdcode	integer	Any valid code	SBus command code
datacount	integer	0 - 65535	Quantity of items to read from server
dataaddr	integer	0 - 65535	SBus memory address to read from server
msgdata	string	Any valid data	A packed binary string containing the data to send

Note: MsgData is optional for functions which do not send data.

The SBus version of SimpleClient raises exceptions for invalid parameters or messages. A ParamError exception indicates that one or more of the parameters provided was invalid.

A typical example is:

```
msgsequence = 1
stnaddr = 50
cmdcode = 2
datacount = 10
dataaddr = 567
client.SendRequest(msgsequence, stnaddr, cmdcode, datacount, dataaddr)
```

ReceiveResponse for SBus

ReceiveResponse returns the following values:

For successful responses:

Parameter	Type	Description
telegramattr	integer	The telegram attribute
msgsequence	integer	This is an echo of the sent TID
msgdata	string/ integer	This is a binary string when it represents data, and a numeric code when it represents an ack or nak.

Note: Message Data is a packed binary string containing the response data.

Telegram attributes:

Value	Description
0	Request
1	response
2	ack/nak

Ack/Nak codes:

An Ack code of zero in the telegram attribute indicates no error. A Nak code of non-zero indicates an error has occured. You must check telegramattr before attempting to decode the message data.

Software Exceptions:

The SBus version of SimpleClient raises exceptions for invalid parameters or messages. The exceptions are:

Exception	Description
MessageLengthError	The received message was of a length which was too short or too long to decode.
CRCError	The received message had a bad CRC.

A typical example is:

```
telegramattr, msgsequence, msgdata = client.SBResponse()
```

SimpleClient for MBLogic Web Service:

SimpleClientWS is a client for a Modbus like web service. This web service was created for MBServer and is not part of a formal standard. The protocol itself is documented with MBServer, and so is not described in detail here. The protocol itself is a true web service protocol (as opposed to being an RPC tunnelled over http), and is based on REST principles.

SimpleClientWS is located in a library called "ModbusRestSimpleClient". The library contains a single class called "ModbusRestSimpleClient". This class in turn contains the following methods:

- SendRequest - This method constructs a web service message, sends it to the server, and returns the response.
- MakeRawRequest - This method constructs a web service message, but does not send it to the server.

- SendRawRequest - This method sends a previously constructed web service message to the server, and returns the response.

SendRequest is the methods normally used for creating an ordinary client program. MakeRawRequest and SendRawRequest are not used in a normal client. The ability to manipulate a raw web service message however is sometimes useful when creating a program which is intended to test a server by sending malformed packets. As they are seldom used, they are not discussed further here.

Initialisation

The host parameters must be specified at initialisation:

Parameter Name	Type	Range	Description
host	string	Any valid host name	IP address of server plus the URL to the web service
port	integer	Any valid port number	Port number of server

A typical example is:

```
host = 'localhost/modbus'
port = 8080
client = ModbusRestSimpleClient.ModbusRestSimpleClient(host, port)
```

SendRequest

SendRequest accepts the following parameters:

SendRequest (send)

Parameter Name	Type	Range	Description
TransID	integer	0 - 65535	Modbus Transacation ID
UnitID	integer	0 - 255	Modbus Unit ID
Function Code	integer	Any valid code	Modbus function code
Addr	integer	0 - 65535	Modbus memory address to read from server
Qty	integer	0 - 65535	Quantity of items to read from server
MsgData	string	Any valid data	An ASCII string containing the data to send

SendRequest returns the following values:

For Successful Responses

Parameter Name	Type	Description
TransID	integer	This is an echo of the sent TID
Function Code	integer	This is an echo of the sent function code
MsgData	string	An ASCII string containing the received data
httpstatus	integer	A standard http response code (200 = OK)
httpreason	string	A standard http reason

For Error Responses

Parameter Name	Type	Description
TransID	integer	This is an echo of the sent TID
Error Code	integer	The Modbus error code (function + 128)
Exception Code	integer	The Modbus exception code (1, 2, 3, or 4)
httpstatus	integer	A standard http response code
httpreason	string	A standard http reason

- For undecodable responses, the returned TransID, Function Code, and MsgData are (0, 0, '0') respectively. These are resposnses which do not fit a valid message pattern.
- httpstatus takes priorty in determining if a response is valid. If httpstatus indicates an error, the returned Modbus data is not meaningful.
- An httpstatus of 200 indicates success, while a 404 indicates failure (see a standard reference for other http codes).
- httpreason is simply a standard text representation of httpstatus. It is sufficient to examine httpstatus and ignore httpreason.

A typical example is:

```
host = 'localhost/modbus'
port = 8080
client = ModbusRestSimpleClient.ModbusRestSimpleClient(host, port)
SendTransID = 52
SendUnitID = 1
SendFunction = 5
SendAddr = 678
SendQty = 1
MsgData = 'FF00'
Recv_TransID, Recv_Function, Recv_Data, Recv_HttpStatus, Recv_HttpResult = \
mbclient.SendRequest(SendTransID, SendUnitID, SendFunction, SendAddr, SendQty, MsgData)
```

Creating an Application:

The following example shows a simple application:

```
import ModbusTCPSimpleClient2
client = ModbusTCPSimpleClient2.ModbusTCPSimpleClient('localhost', 8502, 10.0)
client.SendRequest(1, 2, 3, 0, 1)
TransID, Function, MsgData = client.ReceiveResponse()
```

Exception Handling:

"Exception handling" in this context refers to Python language exceptions, rather than Modbus exceptions. SimpleClient and SimpleClientWS allow most exceptions to "bubble up" to the user application level. Therefore, any use of either library in a serious application should enclose calls to library methods in exception handling constructs (try/except).

Some language exceptions are caught at a lower level. These result in the return error codes which are documented above.

The libraries also use Python language exceptions to indicate errors rather than passing coded parameters. The exception codes can be imported from the SBusMsg or ModbusTCPLib modules.

Library Dependencies:

SimpleClient and SimpleClientWS have the following library dependecies:

- ModbusTCPLib - This is used by ModbusTCPSimpleClient2 to construct Modbus messages. This library is included in the same source code package as ModbusTCPSimpleClient2. This must be included in any project using ModbusTCPSimpleClient2.
- SBusMsg - This is used by SBusSimpleClient to construct SBus messages. This library is included in the same source code package as SBusSimpleClient. This must be included in any project using SBusSimpleClient.
- As well as the above, there are dependencies on several standard Python language libraries which are not listed here. These standard language libraries should normally be automatically installed as part of a standard Python installation.

ModbusDataLib Data Conversion Libraries

ModbusDataLib is used to convert data between the normal formats used in the Python language and the "packed binary strings" required for Modbus and SBus messages.

List Oriented Functions

The packed binary strings in all list oriented functions are compatible with Modbus messages.

1. bin2boollist(binval): Accepts a packed binary and outputs a list of boolean values.

Example:

```
result = ModbusDataLib.bin2boollist(x)
'\x2F' --> [True, True, True, True, False, True, False, False]
```

2. boollist2bin(boollist): Accepts a list of boolean values and outputs a packed binary string. If the length of the input list is not an even multiple of 8, it is padded out with False values to fit.

Example:

```
result = ModbusDataLib.boollist2bin(x)
[True, True, True, True, False, True, False, False] --> '\x2F'
```

3. **bin2intlist(binval): Accepts a packed binary string and outputs a list of** *unsigned* 16 bit integers.

Example:

```
result = ModbusDataLib.bin2intlist(x)
'\xF1\x23\x12\xD9' --> [61731, 4825]
```

4. intlist2bin(intlist): Accepts a list of *unsigned* 16 bit integers and outputs a packed binary string.

Example:

```
result = ModbusDataLib.intlist2bin(x)
[61731, 4825] --> '\xF1\x23\x12\xD9'
```

5. signedbin2intlist(binval): Same as bin2intlist but outputs a list of *signed* integers.

Example:

```
result = ModbusDataLib.signedbin2intlist(x)
'\xF1\x23\x12\xD9' --> [-3805, 4825]
```

6. signedintlist2bin(intlist): Same as intlist2bin but accepts a list of *signed* integers.

Example:

```
result = ModbusDataLib.signedintlist2bin(x)
[-3805, 4825] --> '\xF1\x23\x12\xD9'
```

Miscellaneous

1. coilvalue(state) - If state = 0, it returns 'x00x00', else it returns 'xFFx00'. This is used for providing the correct parameter values required by Modbus function 5 (write single coil).

Example:

```
result = ModbusDataLib.coilvalue(0)
0 --> '\x00\x00'
1 --> '\xFF\x00'
```

Word and Packed Binary String Conversions

These differ from the list oriented functions in that they operate on single integers rather than on lists of integers.

1. Int2BinStr(intdata): Pack a 16 bit integer into a binary string. This may be used where a binary string is expected, but the data is in integer format. Parameters: intdata (integer). Returns: binary string.

Example:

```
result = ModbusDataLib.Int2BinStr(x)
```

2. BinStr2Int(strdata): Convert a packed binary string to a 16 bit integer. Parameters: intdata (binary string). Returns: integer.

Example:

```
result = ModbusDataLib.BinStr2Int(x)
```

3. SignedInt2BinStr(intdata): Same as Int2BinStr but accepts a signed integer instead of unsigned.

Example:

```
result = ModbusDataLib.SignedInt2BinStr(x)
```

4. BinStr2SignedInt(strdata): Same as BinStr2Int but returns a signed integer instead of unsigned.

Example:

```
result = ModbusDataLib.BinStr2SignedInt(x)
```

SBus Conversions:

SBusMsg also includes two conversion functions which provides convertions for 32 bit integer registers.

- signedbin2int32list - Equivalent to bin2intlist, but for 32 bit integers.
- signedint32list2bin - Equivalent to intlist2bin, but for 32 bit integers.

SimpleClient for Modbus/TCP (Obsolete Version)

The obsolete version of ModbusSimpleClient is documented here. For new applications do not use this version. Use ModbusSimpleClient2 for all new applications.

The methods (function calls) provided for Modbus/TCP are:

- SendRequest - This method sends a request.
- ReceiveResponse - This method receives the response.
- MakeRawRequest - This method constructs a message, but does not send it.
- SendRawRequest - This method sends a previously constructed message.
- GetRawResponse - This method recieves a response, but does not attempt to decode it.

SendRequest and ReceiveResponse are the methods normally used for creating an ordinary client program. MakeRawRequest, SendRawRequest, and GetRawResponse are not used in a normal client. The ability to manipulate a raw message however is sometimes useful when creating a program which is intended to test a server by sending malformed packets. As they are seldom used, they are not discussed further here.

Initialisation

The host parameters must be specified at initialisation:

Parameter Name	Type	Range	Description
host	string	Any valid host name	IP address of server
port	integer	Any valid port number	Port number of server
timeout	real	Any valid real number	Time-out in seconds.

A typical example is:

```
host = '192.168.10.1'
port = 502
timeout = 10.5
client = ModbusTCPSimpleClient.ModbusTCPSimpleClient(host, port, timeout)
```

SendRequest for Modbus

The parameters for SendRequest depend on the protocol. For ModbusTCPSimpleClient and ModbusRestSimpleClient the parameters are:

Parameter Name	Type	Range	Description
TransID	integer	0 - 65535	Modbus transaction ID
UnitID	integer	0 - 255	Modbus Unit ID
Function Code	integer	Any valid code	Modbus function code
Addr	integer	0 - 65535	Modbus memory address to read from server
Qty	integer	0 - 65535	Quantity of items to read from server
MsgData	string	Any valid data	A packed binary string containing the data to send

Note: MsgData is optional for functions which do not send data.

A typical example is:

```
TransID = 1
UnitID = 50
FunctionCode = 1
Addr = 567
Qty = 64
client.SendRequest(TransID, UnitID, FunctionCode, Addr, Qty)
```

Another typical example:

```
TransID = 1
UnitID = 50
FunctionCode = 5
Addr = 567
Qty = 64
MsgData = '\xFF\x00'
client.SendRequest(TransID, UnitID, FunctionCode, Addr, Qty, MsgData)
```

ReceiveResponse for Modbus

ReceiveResponse returns the following values:

For successful responses:

Parameter	Type	Description
Transation ID	integer	This is an echo of the sent TID
Function	integer	This is an echo of the sent function code
Message Data	string	The message data

Note: Message Data is a packed binary string containing the response data.

For error responses:

Parameter	Type	Description
Transation ID	integer	This is an echo of the sent TID
Error code	integer	The Modbus error code (function + 128)
Exception code	integer	The Modbus exception code (1, 2, 3, or 4)

For undecodable responses: (0, 0, '0'). These are resposnses which do not fit a valid message pattern.

A typical example is:

```
TransID, Function, MsgData = client.ReceiveResponse()
```

ModbusDataStrLib (Obsolete):

ModbusDataStrLib is an obsolete library. Do not use this in new applications.

ModbusDataStrLib implements a number of functions which are useful when encoding and decoding data in Modbus and SBus format. These functions are:

String oriented functions:

- 1. inversorbin(data) - Accepts a string in raw binary format (e.g. 'x2F'), and returns an ASCII string of 0 and/or 1 characters. E.g. '11110100'
- 2. bininversor(data) - The inverse of inversorbin.
- 3. bin2hex(bin) - Accepts a string in raw binary format. (e.g. 'x2Fx91') and returns a string in ASCII hexadecimal. (e.g. '2F91')
- 4. hex2bin(hexa) - The inverse of bin2hex.

List oriented functions:

- 1. bin2boollist(binval) - Same as inversorbin, but outputs a list of boolean values instead of an ASCIII string.
- 2. boollist2bin(boollist) - Same as bininversor, but accepts a list of boolean values instead of an ASCII string.
- 3. bin2intlist(binval) - Same as bin2hex, but outputs a list of integers instead of an ASCII hex string.
- 4. intlist2bin(intlist) - Same as hex2bin, but accepts a list instead of integers of a string of hexadecimal ASCII characters.

Coil value conversions:

- 1. coilvalue(state) - If state = 0, it returns 'x00x00', else it returns 'xFFx00'. This is used for providing the correct parameter values required by Modbus function 5 (write single coil).

Register value conversions:

- 1. Int2BinStr(intdata) - Pack a 16 bit integer into a binary string. This may be used where a binary string is expected, but the data is in integer format. Parameters - intdata (integer). Returns - binary string.
- 2. BinStr2Int(strdata) - Convert a packed binary string to a 16 bit integer. Parameters - intdata (binary string). Returns - integer.
- 3. SignedInt2BinStrintdata) - Same as Int2BinStr but accepts a signed integer instead of unsigned.
- 4. BinStr2SignedInt(strdata) - Same as BinStr2Int but returns a signed integer instead of unsigned.

Other conversions:

- 1. bitreversebin(data) - Reverses the bit order in each byte of data in a binary string.

- 2. bin2bitstr(data) - Equivalent to inversorbin, but does not reverse the bit order.
- 3. bit2binstr(data) - Equivalent to bininversor, but does not reverse the bit order.