Powercom3

Undercover Modbus Unit

User Manual

Revision 04

Camax UK Limited
Unit 8 Jubilee Court
Copgrove
Harrogate
North Yorkshire
Telephone: **01423 340 000**
Website: **www.camax.co.uk**
E-Mail: **sales@camax.co.uk**

Table of Contents

1. Introduction

The Modbus Unit has been specifically designed to work with Elster's A1140 and A1700 energy meters and fits under the terminal cover. The power supply of the unit is taken directly from a A1140 and from an additional power adapter when used with the A1700.

The unit can be configured in one of three modes; modbus mode, transparent mode with addressing and transparent mode without addressing.

2. Configuration

The configuration is done via the on-board DIP switch and terminal commands entered via the RS232 port.

2.1. DIP Switch

The 8-way switch is divided into 3 groups as follows:

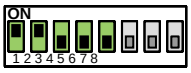
Address
Modbus disabled
Meter type

2.1.1 Address

The address field is bit mapped and has a range of 1 to 31. Switching a bit position to the on position will add the binary value to the address, e.g.

Address = 1

Address = 2



Address = 1 + 2 = 3

Address = 4

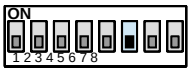
Calculate the address as follows:

Address = sum of value of bit positions set to on.

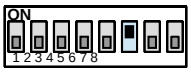
Each bit position has a value of $2^{((\text{bit position}) - 1)}$, so if position 1 and 3 is on the address will be:
Address = $2^0 + 2^2 = 5$

2.1.2 Modbus disable

Set this position to on to disable the modbus functionality. This will result in the unit to disable any meter protocol and enter the transparent mode.



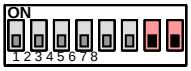
Modbus enabled



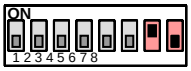
Modbus disabled

2.1.3 Meter type

This selects the meter type, valid settings are:



A1140



A1700

3. Device modes

3.1 Program mode

Program mode is entered when all dip switch positions are set to the on position.

The program mode command interpreter will listen on the RS232 port (RJ12) at 9600 baud, no parity and one stop bit.

A USB dongle available from **Camax UK** may be used to gain access to the RS232 port.

3.1.1 Command set

Typing *help* at the command line displays the available commands, these are: METER, RS232 and RS485.

3.1.1.1 METER command

Sets the outstation and password via parameters:
--addr=<outstation>
--pass=<oldpass,newpass>
e.g.
meter --addr=2
meter --pass=FEDC0003,12345678

3.1.1.2 RS232 command

Sets the parameters for the RS232 port (RJ12) via parameters:
--baud=<baudrate>
--parity=<0=none | 1=odd | 2=even>
--stopbits=<1 | 2>
--read-timeout=<milliseconds>

3.1.1.3 RS485 command

Sets the parameters for the RS485 port (RJ45) via parameters:
--baud=<baudrate>
--parity=<0=none | 1=odd | 2=even>
--stopbits=<1 | 2>
--read-timeout=<milliseconds>

3.1.1.4 SERNO command

Display the serial number

3.1.1.5 VERSION command

Display the firmware version

3.2 Non addressed transparent mode

This mode is entered then the address bits are all set to the off position and the modbus disable bit is set to on. The meter type field is ignored.

In this mode the unit acts as a true RS232 to RS485 converter and all data on both ports are forwarded to the other port. This mode is useful to daisy chain meters via RS485 when they are spaced further than the RS232 limit.

3.3 Addressed transparent mode

This mode is entered then the address bits are set to a non zero address and the modbus disable bit is set to on. The meter type field is ignored.

In this mode the unit acts as an addressable RS232 to RS485 converter. The unit uses a frame based protocol in the form of:

<frame-delimiter><destination-addr><source-addr><payload><crc16><frame-delimiter>
This mode will allow non addressable devices/meters to be daisy chained onto a RS485 bus.

The protocol definition is available on request.

Annexure A: Register Map

Instrumentation

Register Address	Description	Data type
100	Total Active Power	float32
102	Phase A Active Power	float32
104	Phase B Active Power	float32
106	Phase C Active Power	float32
108	Total Reactive Power	float32
110	Phase A Reactive Power	float32
112	Phase B Reactive Power	float32
114	Phase C Reactive Power	float32
116	Total Apparent Power	float32
118	Phase A Apparent Power	float32
120	Phase B Apparent Power	float32
122	Phase C Apparent Power	float32
124	Phase A Current Magnitude	float32
126	Phase B Current Magnitude	float32
128	Phase C Current Magnitude	float32
130	Frequency	float32
132	Phase A Voltage Magnitude	float32
134	Phase B Voltage Magnitude	float32
136	Phase C Voltage Magnitude	float32
138	System Power Factor	float32
140	Phase A Power Factor	float32
142	Phase B Power Factor	float32
144	Phase C Power Factor	float32
146	Phase A Current Angle	float32
148	Phase B Current Angle	float32
150	Phase C Current Angle	float32
152	Phase A Voltage Angle	float32
154	Phase B Voltage Angle	float32
156	Phase C Voltage Angle	float32

Cumulatives

Register Address	Description	Data type
200	Import Wh	float64
204	Export Wh	float64
208	Q1 varh	float64
212	Q2 varh	float64
216	Q3 varh	float64
220	Q4 varh	float64
224	VAh	float64
228	Customer Defined 1	float64
232	Customer Defined 2	float64
236	Customer Defined 3	float64
240	Hist Import Wh	float64
244	Hist Export Wh	float64
248	Hist Q1 varh	float64
252	Hist Q2 varh	float64
256	Hist Q3 varh	float64
260	Hist Q4 varh	float64
264	Hist VAh	float64
268	Hist Customer Defined 1	float64
272	Hist Customer Defined 2	float64
276	Hist Customer Defined 3	float64

Cumulatives (32bit Alternatives)

Register Address	Description	Data type
1200	Import Wh	float32
1202	Export Wh	float32
1204	Q1 varh	float32
1206	Q2 varh	float32
1208	Q3 varh	float32
1210	Q4 varh	float32
1212	VAh	float32
1214	Customer Defined 1	float32
1216	Customer Defined 2	float32
1218	Customer Defined 3	float32
1240	Hist Import Wh	float32
1242	Hist Export Wh	float32
1244	Hist Q1 varh	float32
1246	Hist Q2 varh	float32
1248	Hist Q3 varh	float32
1250	Hist Q4 varh	float32
1252	Hist VAh	float32
1254	Hist Customer Defined 1	float32
1256	Hist Customer Defined 2	float32
1258	Hist Customer Defined 3	float32

Time of use

Register Address	Description	Data type
300	TOU1	float64
304	TOU2	float64
308	TOU3	float64
312	TOU4	float64
316	HIST_TOU1	float64
320	HIST_TOU2	float64
324	HIST_TOU3	float64
328	HIST_TOU4	float64

Time of use (32bit Alternatives)

Register Address	Description	Data type
1300	TOU1	float32
1302	TOU2	float32
1304	TOU3	float32
1306	TOU4	float32
1316	HIST_TOU1	float32
1318	HIST_TOU2	float32
1320	HIST_TOU3	float32
1322	HIST_TOU4	float32

Maximum Demand

Register Address	Description	Data type
500	MD1	float64
504	MD2	float64
508	MD3	float64
512	MD4	float64
516	MD1 date	int32
518	MD2 date	int32
520	MD3 date	int32
522	MD4 date	int32
524	HIST_MD1	float64
528	HIST_MD2	float64
532	HIST_MD3	float64
536	HIST_MD4	float64
540	HIST_MD1 date	int32
542	HIST_MD2 date	int32
544	HIST_MD3 date	int32
546	HIST_MD4 date	int32

Maximum Demand (32bit Alternatives)

Register Address	Description	Data type
1500	MD1	float32
1502	MD2	float32
1504	MD3	float32
1506	MD4	float32
1524	HIST_MD1	float32
1526	HIST_MD2	float32
1528	HIST_MD3	float32
1530	HIST_MD4	float32

Multi Utility

Register Address	Description	Data type
600	MULTI_UTIL1	float64
604	MULTI_UTIL2	float64
608	MULTI_UTIL3	float64
612	MULTI_UTIL4	float64
616	HIST_MULTI_UTIL1	float64
620	HIST_MULTI_UTIL2	float64
624	HIST_MULTI_UTIL3	float64
628	HIST_MULTI_UTIL4	float64

Multi Utility (32bit Alternatives)

Register Address	Description	Data type
1600	MULTI_UTIL1	float32
1602	MULTI_UTIL2	float32
1604	MULTI_UTIL3	float32
1606	MULTI_UTIL4	float32
1616	HIST_MULTI_UTIL1	float32
1618	HIST_MULTI_UTIL2	float32
1620	HIST_MULTI_UTIL3	float32
1622	HIST_MULTI_UTIL4	float32

General

Register Address	Description	Data type
400	Meter Date (seconds since 1 Jan 1970)	uint32
402	Uptime	uint32
404	Version	uint32
420	Meter Serial Number digit 1 and 2	uint16
421	Meter Serial Number digit 3 and 4	uint16
422	Meter Serial Number digit 5 and 6	uint16
423	Meter Serial Number digit 7 and 8	uint16

Annexure B: Status indication

Status LED (A)

Modbus enabled (Address > 0), meter found:
80ms on/80ms 0ff/80ms on/3s off, repeat

Modbus enabled (Address > 0), meter not found:
500ms on/500ms 0ff, repeat

Modbus enabled (Address = 0), invalid setting:
All LEDS 100ms on/100ms 0ff, repeat

Modbus disabled (Address = 0), non addressed transparent mode:
100ms on/1000ms 0ff, repeat

Modbus disabled (Address > 0), addressed transparent mode:
100ms on/100ms 0ff/100ms on/1000ms off, repeat

RS232 Receive LED (B)

Indicates incoming data on the RS232 interface.

RS232 Transmit LED (C)

Indicates outgoing data on the RS232 interface.

RS485 Receive LED (D)

Indicates incoming data on the RS485 interface.

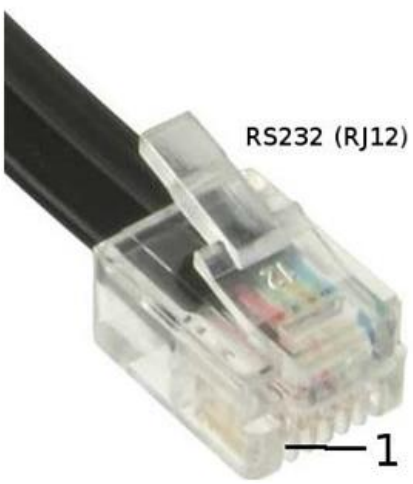
RS485 Transmit LED (E)

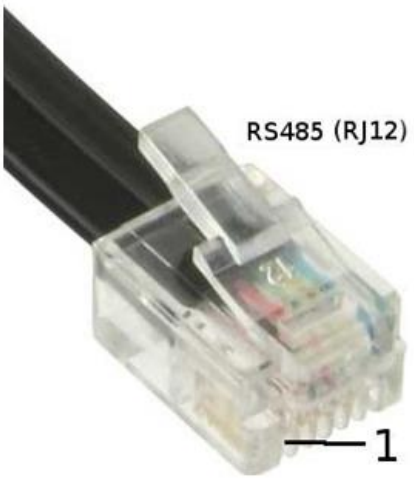
Indicates outgoing data on the RS485 interface.

Program mode

LEDS will show a walking 'on' from left to right.

Annexure C: Pinouts





RS232 – RJ12

Pin Number	Name	Direction	Description
1	Tx	Output	Transmit data output
2	Rx	Input	Receive data input
3	N/C		Not used
4	N/C		Not used
5	0V		Power supply common
6	V+		Power supply voltage (5..12VDC)

RS485 – RJ12

Pin Number	Name	Direction	Description
1	A	Bidir	Positive RS485 signal
2	B	Bidir	Negative RS485 signal
3	N/C		
4	N/C		
5	N/C		
6	N/C		
7	N/C		
8	N/C		

Annexure D: USB Adapter

Configuration of the modem on a PC is done via a USB adapter that provides the communications interface and PSU.

The FTDI controller chip is used in the adapter. The newest driver is available at www.ftdichip.com

Annexure E: Python Code Examples

readInstrumentation.py

```
import sys
import getopt
import struct
from pymodbus.client.sync import ModbusSerialClient as ModbusClient

def getFloatBigEndian(client,unit, address):
	regs = client.read_input_registers(address=address, count=2, unit=unit).registers
	ii = regs[0] * 0x10000 + regs[1]
	f = struct.unpack('f', struct.pack('I', ii))
	return f[0]

def getFloatArray(client,unit,address,count):
	values = []
	startAddress = address
	for i in xrange(0, count):
		f = getFloatBigEndian(client=client,unit=unit, address=startAddress)
		values.append(f)
		startAddress = startAddress + 2;
	return values


if_name_== '_main_':

	labels = ['Ptot', 'Pa', 'Pb', 'Pc', 'Qtot', 'Qa', 'Qb', 'Qc', 'Stot', 'Sa', 'Sb', 'Sc', 'IaMag', 'IbMag', 'IcMag', 'Freq', 'Va', 'Vb', 'Vc', 'PFtot',
'PFa', 'PFb', 'PFc', 'IaAng', 'IbAng', 'IcAng', 'VaAng', 'VbAng', 'VcAng']

	optlist, arglist = getopt.getopt(sys.argv[1:], 'p:a:')

	serialPort = None
	address = None

	for option, value in optlist:
		if option == '-p':
			serialPort = str(value)
		elif option == '-a':
			address = int(value)

	if serialPort is None or address is None:
		print "Usage: python readInstrumentation.py -p <serial port> -a <modbus address>"
		sys.exit()

	client = ModbusClient(method='rtu', port=serialPort, parity='N', baudrate=9600, timeout=0.1)
	values = getFloatArray(client=client,unit=address, address=100, count=29)
	for ndx in range(len(values)):
		print labels[ndx], '\t', values[ndx]
```

readCumulatives.py

```
import sys
import getopt
import struct
from pymodbus.client.sync import ModbusSerialClient as ModbusClient

def getFloatBigEndian(client,unit, address):
	regs = client.read_input_registers(address=address, count=2, unit=unit).registers
	ii = regs[0] * 0x10000 + regs[1]
	f = struct.unpack('f', struct.pack('I', ii))
	return f[0]

def getDoubleBigEndian(client, unit, address):
		try:
			tmpaddress = int(address)
			tmpunit = int(unit)

		except:
			return None
		try:
			regs = client.read_input_registers(address=tmpaddress, count=4, unit=tmpunit).registers
			p = struct.pack('!HHHH', regs[0], regs[1], regs[2], regs[3])
			f = struct.unpack('!d', p)
			return f[0]
		except Exception, e:
			print 'Modbus error reading '+str(address)+' from unit '+str(unit)+':', str(e)
			return None


def getDoubleArray(client,unit,address,count):
	values = []
	startAddress = address
	for i in xrange(0, count):
		f = getDoubleBigEndian(client=client,unit=unit, address=startAddress)
		values.append(f)
		startAddress = startAddress + 4;
	return values

if_name_== '_main_':

	labels = ['ImportWh', 'ExportWh', 'Q1varh', 'Q2varh', 'Q3varh', 'Q4varh', 'VAh']

	optlist, arglist = getopt.getopt(sys.argv[1:], 'p:a:')

	serialPort = None
	address = None

	for option, value in optlist:
		if option == '-p':
			serialPort = str(value)
		elif option == '-a':
			address = int(value)

	if serialPort is None or address is None:
		print "Usage: python readCumulatives.py -p <serial port> -a <modbus address>"
		sys.exit()

	client = ModbusClient(method='rtu', port=serialPort, parity='N', baudrate=9600, timeout=0.1)
	values = getDoubleArray(client=client,unit=address, address=200, count=7)
	for ndx in range(len(values)):
		print labels[ndx], '\t', values[ndx]
```

readTOU.py

```
import sys
import getopt
import struct
from pymodbus.client.sync import ModbusSerialClient as ModbusClient

def getFloatBigEndian(client,unit, address):
        regs = client.read_input_registers(address=address, count=2, unit=unit).registers
        ii = regs[0] * 0x10000 + regs[1]
        f = struct.unpack('f', struct.pack('I', ii))
        return f[0]

def getDoubleBigEndian(client, unit, address):
                try:
                        tmpaddress = int(address)
                        tmpunit = int(unit)

                except:
                        return None
                try:
                        regs = client.read_input_registers(address=tmpaddress, count=4, unit=tmpunit).registers
                        p = struct.pack('!HHHH', regs[0], regs[1], regs[2], regs[3])
                        f = struct.unpack('!d', p)
                        return f[0]
                except Exception, e:
                        print 'Modbus error reading '+str(address)+' from unit '+str(unit)+':', str(e)
                        return None


def getDoubleArray(client,unit,address,count):
        values = []
        startAddress = address
        for i in xrange(0, count):
                f = getDoubleBigEndian(client=client,unit=unit, address=startAddress)
                values.append(f)
                startAddress = startAddress + 4;
        return values

if_name_== '_main_':

        labels = ['TOU1', 'TOU2', 'TOU3', 'TOU4']

        optlist, arglist = getopt.getopt(sys.argv[1:], 'p:a:')

        serialPort = None
        address = None

        for option, value in optlist:
                if option == '-p':
                        serialPort = str(value)
                elif option == '-a':
                        address = int(value)

        if serialPort is None or address is None:
                print "Usage: python readTOU.py -p <serial port> -a <modbus address>"
                sys.exit()

        client = ModbusClient(method='rtu', port=serialPort, parity='N', baudrate=9600, timeout=0.1)
        values = getDoubleArray(client=client,unit=address, address=300, count=4)
        for ndx in range(len(values)):
                print labels[ndx], '\t', values[ndx]
```

readMD.py

```
import sys
import getopt
import struct
from pymodbus.client.sync import ModbusSerialClient as ModbusClient

def getIntBigEndian(client,unit, address):
        regs = client.read_input_registers(address=address, count=2, unit=unit).registers
        i = regs[0] * 0x10000 + regs[1]
        return i

def getDoubleBigEndian(client, unit, address):
                try:
                        tmpaddress = int(address)
                        tmpunit = int(unit)

                except:
                        return None
                try:
                        regs = client.read_input_registers(address=tmpaddress, count=4, unit=tmpunit).registers
                        p = struct.pack('!HHHH', regs[0], regs[1], regs[2], regs[3])
                        f = struct.unpack('!d', p)
                        return f[0]
                except Exception, e:
                        print 'Modbus error reading '+str(address)+' from unit '+str(unit)+':', str(e)
                        return None

def getIntArray(client,unit,address,count):
        values = []
        startAddress = address
        for i in xrange(0, count):
                f = getIntBigEndian(client=client,unit=unit, address=startAddress)
                values.append(f)
                startAddress = startAddress + 2;
        return values

def getDoubleArray(client,unit,address,count):
        values = []
        startAddress = address
        for i in xrange(0, count):
                f = getDoubleBigEndian(client=client,unit=unit, address=startAddress)
                values.append(f)
                startAddress = startAddress + 4;
        return values

if_name_== '_main_':

        labels = ['MD1', 'MD2', 'MD3', 'MD4']

        optlist, arglist = getopt.getopt(sys.argv[1:], 'p:a:')

        serialPort = None
        address = None

        for option, value in optlist:
                if option == '-p':
                        serialPort = str(value)
                elif option == '-a':
                        address = int(value)

        if serialPort is None or address is None:
                print "Usage: python readMD.py -p <serial port> -a <modbus address>"
                sys.exit()

        client = ModbusClient(method='rtu', port=serialPort, parity='N', baudrate=9600, timeout=0.1)
        values = getDoubleArray(client=client,unit=address, address=500, count=4)
        ts = getIntArray(client=client,unit=address, address=516, count=4)
        for ndx in range(len(values)):
                print labels[ndx], '\t', values[ndx], ts[ndx]
```

readMeterSerial.py

```
import sys
import getopt
import struct
from pymodbus.client.sync import ModbusSerialClient as ModbusClient

def regArrayToString(regs):
	s = ''
	for reg in regs:
		s += chr((reg >> 8) & 0x00FF)
		s += chr(reg & 0x00FF)
	s = s.strip('\x00')
	return s


if __name__== '__main__':

	optlist, arglist = getopt.getopt(sys.argv[1:], 'p:a:')

	serialPort = None
	address = None

	for option, value in optlist:
		if option == '-p':
			serialPort = str(value)
		elif option == '-a':
			address = int(value)

	if serialPort is None or address is None:
		print "Usage: python readMeterSerial.py -p <serial port> -a <modbus address>"
		sys.exit()

	client = ModbusClient(method='rtu', port=serialPort, parity='N', baudrate=9600, timeout=0.1)
	regs = client.read_input_registers(address=420, count=4, unit=address).registers
	print "Serial Number\t", regArrayToString(regs)
```