ModbusLib 0.4.2

Generated by Doxygen 1.12.0

1 ModbusLib	1
1.0.1 Overview	1
1.0.2 Using Library	2
1.0.2.1 Common usage (C++)	2
1.0.2.2 Client	2
1.0.2.3 Server	3
1.0.2.4 Using with C	4
1.0.2.5 Using with Qt	5
1.0.3 Examples	5
1.0.3.1 democlient	5
1.0.3.2 mbclient	5
1.0.3.3 demoserver	5
1.0.3.4 mbserver	6
1.0.4 Tests	6
1.0.5 Documenations	6
1.0.6 Building	6
1.0.6.1 Build using CMake	6
1.0.6.2 Build using qmake	7
2 Namespace Index	9
2.1 Namespace List	9
2.1 Namespace List	Э
3 Hierarchical Index	11
3.1 Class Hierarchy	11
4 Class Index	13
	13
4.1 Class List	13
5 File Index	15
5.1 File List	15
6 Namespace Documentation	17
6.1 Modbus Namespace Reference	17
6.1.1 Detailed Description	21
6.1.2 Enumeration Type Documentation	21
6.1.2.1 _MemoryType	21
6.1.2.2 Constants	21
6.1.2.3 FlowControl	21
6.1.2.4 Parity	22
6.1.2.5 ProtocolType	22
6.1.2.6 StatusCode	22
6.1.2.7 StopBits	23
6.1.3 Function Documentation	25
6.1.3.1 addressFromString()	25

6.1.3.2 asciiToBytes()
6.1.3.3 asciiToString() [1/2]
6.1.3.4 asciiToString() [2/2]
6.1.3.5 availableBaudRate()
6.1.3.6 availableDataBits()
6.1.3.7 availableFlowControl()
6.1.3.8 availableParity()
6.1.3.9 availableSerialPortList()
6.1.3.10 availableSerialPorts()
6.1.3.11 availableStopBits()
6.1.3.12 bytesToAscii()
6.1.3.13 bytesToString() [1/2]
6.1.3.14 bytesToString() [2/2]
6.1.3.15 crc16()
6.1.3.16 createClientPort() [1/2]
6.1.3.17 createClientPort() [2/2]
6.1.3.18 createPort() [1/2]
6.1.3.19 createPort() [2/2]
6.1.3.20 createServerPort() [1/2]
6.1.3.21 createServerPort() [2/2]
6.1.3.22 currentTimestamp()
6.1.3.23 enumKey() [1/2]
6.1.3.24 enumKey() [2/2]
6.1.3.25 enumValue() [1/4]
6.1.3.26 enumValue() [2/4]
6.1.3.27 enumValue() [3/4]
6.1.3.28 enumValue() [4/4]
6.1.3.29 getBit()
6.1.3.30 getBitS()
6.1.3.31 getBits()
6.1.3.32 getBitsS()
6.1.3.33 getLastErrorText()
6.1.3.34 getSettingBaudRate()
6.1.3.35 getSettingBroadcastEnabled()
6.1.3.36 getSettingDataBits()
6.1.3.37 getSettingFlowControl()
6.1.3.38 getSettingHost()
6.1.3.39 getSettingParity()
6.1.3.40 getSettingPort()
6.1.3.41 getSettingSerialPortName()
6.1.3.42 getSettingStopBits()
6.1.3.43 getSettingTimeout()

6.1.3.44 getSettingTimeoutFirstByte()	 33
6.1.3.45 getSettingTimeoutInterByte()	 33
6.1.3.46 getSettingTries()	 34
6.1.3.47 getSettingType()	 34
6.1.3.48 getSettingUnit()	 34
6.1.3.49 lrc()	 34
6.1.3.50 modbusLibVersion()	 34
6.1.3.51 modbusLibVersionStr()	 34
6.1.3.52 msleep()	 35
<b>6.1.3.53 readMemBits()</b> [1/2]	 35
<b>6.1.3.54 readMemBits()</b> [2/2]	 35
<b>6.1.3.55</b> readMemRegs() [1/2]	 35
<b>6.1.3.56 readMemRegs()</b> [2/2]	 36
6.1.3.57 sascii()	 36
6.1.3.58 sbytes()	 36
6.1.3.59 setBit()	 36
6.1.3.60 setBitS()	 37
6.1.3.61 setBits()	 . 37
6.1.3.62 setBitsS()	 37
6.1.3.63 setSettingBaudRate()	 37
6.1.3.64 setSettingBroadcastEnabled()	 38
6.1.3.65 setSettingDataBits()	 38
6.1.3.66 setSettingFlowControl()	 . 38
6.1.3.67 setSettingHost()	 . 38
6.1.3.68 setSettingParity()	 38
6.1.3.69 setSettingPort()	 38
6.1.3.70 setSettingSerialPortName()	 39
6.1.3.71 setSettingStopBits()	 . 39
6.1.3.72 setSettingTimeout()	 39
6.1.3.73 setSettingTimeoutFirstByte()	 39
6.1.3.74 setSettingTimeoutInterByte()	 39
6.1.3.75 setSettingTries()	 39
6.1.3.76 setSettingType()	 40
6.1.3.77 setSettingUnit()	 40
6.1.3.78 sflowControl()	 40
6.1.3.79 sparity()	 40
6.1.3.80 sprotocolType()	 40
6.1.3.81 sstopBits()	 40
6.1.3.82 StatusIsBad()	 41
6.1.3.83 StatusIsGood()	 41
6.1.3.84 StatusIsProcessing()	 41
6.1.3.85 StatusIsStandardError()	 41

6.1.3.86 StatusIsUncertain()	 	41
6.1.3.87 timer()	 	41
<b>6.1.3.88 toBaudRate()</b> [1/2]	 	41
<b>6.1.3.89 toBaudRate()</b> [2/2]	 	42
6.1.3.90 toDataBits() [1/2]	 	42
6.1.3.91 toDataBits() [2/2]	 	42
6.1.3.92 toFlowControl() [1/2]	 	42
6.1.3.93 toFlowControl() [2/2]	 	42
6.1.3.94 toModbusOffset()	 	42
6.1.3.95 toModbusString()	 	43
6.1.3.96 toParity() [1/2]	 	43
6.1.3.97 toParity() [2/2]	 	43
6.1.3.98 toProtocolType() [1/2]	 	43
6.1.3.99 toProtocolType() [2/2]	 	43
6.1.3.100 toStopBits() [1/2]	 	44
6.1.3.101 toStopBits() [2/2]	 	44
6.1.3.102 toString() [1/5]	 	44
<b>6.1.3.103 toString()</b> [2/5]	 	44
<b>6.1.3.104 toString()</b> [3/5]	 	44
6.1.3.105 toString() [4/5]	 	44
<b>6.1.3.106 toString()</b> [5/5]	 	45
<b>6.1.3.107 writeMemBits()</b> [1/2]	 	45
6.1.3.108 writeMemBits() [2/2]	 	45
6.1.3.109 writeMemRegs() [1/2]	 	45
<b>6.1.3.110 writeMemRegs()</b> [2/2]	 	46
7 Class Documentation		47
7.1 Modbus::Address Class Reference	 	47
7.1.1 Detailed Description	 	47
7.1.2 Constructor & Destructor Documentation	 	47
7.1.2.1 Address() [1/3]	 	47
7.1.2.2 Address() [2/3]	 	48
<b>7.1.2.3 Address()</b> [3/3]	 	48
7.1.3 Member Function Documentation	 	48
7.1.3.1 isValid()	 	48
7.1.3.2 number()	 	48
7.1.3.3 offset()	 	48
7.1.3.4 operator quint32()	 	48
7.1.3.5 operator=()	 	49
7.1.3.6 toString()	 	49
7.1.3.7 type()	 	49
7.2 Modbus::Defaults Class Reference	 	49

7.2.1 Detailed Description	50
7.2.2 Constructor & Destructor Documentation	50
7.2.2.1 Defaults()	50
7.2.3 Member Function Documentation	51
7.2.3.1 instance()	51
7.3 ModbusSerialPort::Defaults Struct Reference	51
7.3.1 Detailed Description	52
7.3.2 Constructor & Destructor Documentation	52
7.3.2.1 Defaults()	52
7.3.3 Member Function Documentation	52
7.3.3.1 instance()	52
7.4 ModbusTcpPort::Defaults Struct Reference	52
7.4.1 Detailed Description	53
7.4.2 Constructor & Destructor Documentation	53
7.4.2.1 Defaults()	53
7.4.3 Member Function Documentation	53
7.4.3.1 instance()	53
7.5 ModbusTcpServer::Defaults Struct Reference	53
7.5.1 Detailed Description	54
7.5.2 Constructor & Destructor Documentation	54
7.5.2.1 Defaults()	54
7.5.3 Member Function Documentation	54
7.5.3.1 instance()	54
7.6 ModbusAscPort Class Reference	54
7.6.1 Detailed Description	56
7.6.2 Constructor & Destructor Documentation	56
7.6.2.1 ModbusAscPort()	56
7.6.2.2 ~ModbusAscPort()	56
7.6.3 Member Function Documentation	56
7.6.3.1 readBuffer()	56
7.6.3.2 type()	56
7.6.3.3 writeBuffer()	57
7.7 ModbusClient Class Reference	57
7.7.1 Detailed Description	58
7.7.2 Constructor & Destructor Documentation	59
7.7.2.1 ModbusClient()	59
7.7.3 Member Function Documentation	59
7.7.3.1 diagnostics()	59
7.7.3.2 getCommEventCounter()	59
7.7.3.3 getCommEventLog()	59
7.7.3.4 isOpen()	60
7.7.3.5 lastPortErrorStatus()	60

7.7.3.6 lastPortErrorText()	. 60
7.7.3.7 lastPortStatus()	. 60
7.7.3.8 maskWriteRegister()	. 60
7.7.3.9 port()	. 60
7.7.3.10 readCoils()	. 60
7.7.3.11 readCoilsAsBoolArray()	. 61
7.7.3.12 readDiscreteInputs()	. 61
7.7.3.13 readDiscreteInputsAsBoolArray()	. 61
7.7.3.14 readExceptionStatus()	. 61
7.7.3.15 readFIFOQueue()	. 61
7.7.3.16 readHoldingRegisters()	. 62
7.7.3.17 readInputRegisters()	. 62
7.7.3.18 readWriteMultipleRegisters()	. 62
7.7.3.19 reportServerID()	. 62
7.7.3.20 setUnit()	. 62
7.7.3.21 type()	. 63
7.7.3.22 unit()	. 63
7.7.3.23 writeMultipleCoils()	. 63
7.7.3.24 writeMultipleCoilsAsBoolArray()	. 63
7.7.3.25 writeMultipleRegisters()	. 63
7.7.3.26 writeSingleCoil()	. 63
7.7.3.27 writeSingleRegister()	. 64
7.8 ModbusClientPort Class Reference	. 64
7.8.1 Detailed Description	. 67
7.8.2 Constructor & Destructor Documentation	. 67
7.8.2.1 ModbusClientPort()	. 67
7.8.3 Member Function Documentation	. 68
7.8.3.1 cancelRequest()	. 68
7.8.3.2 close()	. 68
7.8.3.3 currentClient()	. 68
7.8.3.4 diagnostics() [1/2]	. 68
7.8.3.5 diagnostics() [2/2]	. 68
7.8.3.6 getCommEventCounter() [1/2]	. 69
7.8.3.7 getCommEventCounter() [2/2]	. 69
7.8.3.8 getCommEventLog() [1/2]	. 70
7.8.3.9 getCommEventLog() [2/2]	. 70
7.8.3.10 getRequestStatus()	. 70
7.8.3.11 isBroadcastEnabled()	. 71
7.8.3.12 isOpen()	. 71
7.8.3.13 lastErrorStatus()	. 71
7.8.3.14 lastErrorText()	. 71
7.8.3.15 lastStatus()	. 71

7.8.3.16 lastStatusTimestamp()
7.8.3.17 maskWriteRegister() [1/2]
7.8.3.18 maskWriteRegister() [2/2]
7.8.3.19 port()
7.8.3.20 readCoils() [1/2] 72
7.8.3.21 readCoils() [2/2] 72
7.8.3.22 readCoilsAsBoolArray() [1/2]
7.8.3.23 readCoilsAsBoolArray() [2/2]
7.8.3.24 readDiscreteInputs() [1/2]
7.8.3.25 readDiscreteInputs() [2/2]
7.8.3.26 readDiscreteInputsAsBoolArray() [1/2]
7.8.3.27 readDiscreteInputsAsBoolArray() [2/2]
7.8.3.28 readExceptionStatus() [1/2]
7.8.3.29 readExceptionStatus() [2/2]
7.8.3.30 readFIFOQueue() [1/2]
7.8.3.31 readFIFOQueue() [2/2]
7.8.3.32 readHoldingRegisters() [1/2]
7.8.3.33 readHoldingRegisters() [2/2]
7.8.3.34 readInputRegisters() [1/2]
7.8.3.35 readInputRegisters() [2/2]
7.8.3.36 readWriteMultipleRegisters() [1/2]
7.8.3.37 readWriteMultipleRegisters() [2/2]
7.8.3.38 repeatCount()
7.8.3.39 reportServerID() [1/2]
7.8.3.40 reportServerID() [2/2] 79
7.8.3.41 setBroadcastEnabled()
7.8.3.42 setPort()
7.8.3.43 setRepeatCount()
7.8.3.44 setTries()
7.8.3.45 signalClosed()
7.8.3.46 signalError()
7.8.3.47 signalOpened()
7.8.3.48 signalRx()
7.8.3.49 signalTx()
7.8.3.50 tries()
7.8.3.51 type()
7.8.3.52 writeMultipleCoils() [1/2]
7.8.3.53 writeMultipleCoils() [2/2]
7.8.3.54 writeMultipleCoilsAsBoolArray() [1/2]
7.8.3.55 writeMultipleCoilsAsBoolArray() [2/2]
7.8.3.56 writeMultipleRegisters() [1/2]
7.8.3.57 writeMultipleRegisters() [2.72]

7.8.3.58 writeSingleCoil() [1/2]	. 83
7.8.3.59 writeSingleCoil() [2/2]	. 83
7.8.3.60 writeSingleRegister() [1/2]	. 84
7.8.3.61 writeSingleRegister() [2/2]	. 84
7.9 ModbusInterface Class Reference	. 85
7.9.1 Detailed Description	. 85
7.9.2 Member Function Documentation	. 86
7.9.2.1 diagnostics()	. 86
7.9.2.2 getCommEventCounter()	. 86
7.9.2.3 getCommEventLog()	. 87
7.9.2.4 maskWriteRegister()	. 87
7.9.2.5 readCoils()	. 88
7.9.2.6 readDiscreteInputs()	. 88
7.9.2.7 readExceptionStatus()	. 88
7.9.2.8 readFIFOQueue()	. 89
7.9.2.9 readHoldingRegisters()	. 89
7.9.2.10 readInputRegisters()	. 90
7.9.2.11 readWriteMultipleRegisters()	. 90
7.9.2.12 reportServerID()	. 91
7.9.2.13 writeMultipleCoils()	. 91
7.9.2.14 writeMultipleRegisters()	. 91
7.9.2.15 writeSingleCoil()	. 92
7.9.2.16 writeSingleRegister()	. 92
7.10 ModbusObject Class Reference	. 93
7.10.1 Detailed Description	. 94
7.10.2 Constructor & Destructor Documentation	. 94
7.10.2.1 ModbusObject()	. 94
7.10.2.2 ~ ModbusObject()	. 94
7.10.3 Member Function Documentation	. 94
7.10.3.1 connect() [1/2]	. 94
7.10.3.2 connect() [2/2]	. 95
7.10.3.3 disconnect() [1/3]	. 95
<b>7.10.3.4 disconnect()</b> [2/3]	. 95
<b>7.10.3.5 disconnect()</b> [3/3]	. 95
7.10.3.6 disconnectFunc()	. 95
7.10.3.7 emitSignal()	. 96
7.10.3.8 objectName()	. 96
7.10.3.9 sender()	. 96
7.10.3.10 setObjectName()	. 96
7.11 ModbusPort Class Reference	. 96
7.11.1 Detailed Description	. 97
7.11.2 Constructor & Destructor Documentation	. 97

7.11.2.1 $\sim$ ModbusPort()	97
7.11.3 Member Function Documentation	98
7.11.3.1 close()	98
7.11.3.2 handle()	98
7.11.3.3 isBlocking()	98
7.11.3.4 isChanged()	98
7.11.3.5 isNonBlocking()	98
7.11.3.6 isOpen()	98
7.11.3.7 isServerMode()	99
7.11.3.8 lastErrorStatus()	99
7.11.3.9 lastErrorText()	99
7.11.3.10 open()	99
7.11.3.11 read()	99
7.11.3.12 readBuffer()	99
7.11.3.13 readBufferData()	100
7.11.3.14 readBufferSize()	100
7.11.3.15 setError()	100
7.11.3.16 setNextRequestRepeated()	100
7.11.3.17 setServerMode()	100
7.11.3.18 setTimeout()	100
7.11.3.19 timeout()	101
7.11.3.20 type()	101
7.11.3.21 write()	101
7.11.3.22 writeBuffer()	101
7.11.3.23 writeBufferData()	101
7.11.3.24 writeBufferSize()	101
7.12 ModbusRtuPort Class Reference	102
7.12.1 Detailed Description	103
7.12.2 Constructor & Destructor Documentation	103
7.12.2.1 ModbusRtuPort()	103
$7.12.2.2 \sim ModbusRtuPort()$	104
7.12.3 Member Function Documentation	104
7.12.3.1 readBuffer()	104
7.12.3.2 type()	104
7.12.3.3 writeBuffer()	104
7.13 ModbusSerialPort Class Reference	105
7.13.1 Detailed Description	106
7.13.2 Constructor & Destructor Documentation	106
7.13.2.1 ~ModbusSerialPort()	106
7.13.3 Member Function Documentation	106
7.13.3.1 baudRate()	106
7.13.3.2 close()	107

7.13.3.3 dataBits()	07
7.13.3.4 flowControl()	07
7.13.3.5 handle()	07
7.13.3.6 isOpen()	07
7.13.3.7 open()	07
7.13.3.8 parity()	80
7.13.3.9 portName()	
7.13.3.10 read()	80
7.13.3.11 readBufferData()	80
7.13.3.12 readBufferSize()	80
7.13.3.13 setBaudRate()	80
7.13.3.14 setDataBits()	80
7.13.3.15 setFlowControl()	09
7.13.3.16 setParity()	09
7.13.3.17 setPortName()	09
7.13.3.18 setStopBits()	
7.13.3.19 setTimeoutFirstByte()	09
7.13.3.20 setTimeoutInterByte()	09
7.13.3.21 stopBits()	09
7.13.3.22 timeoutFirstByte()	10
7.13.3.23 timeoutInterByte()	10
7.13.3.24 write()	10
7.13.3.25 writeBufferData()	
7.13.3.26 writeBufferSize()	
7.14 ModbusServerPort Class Reference	
7.14.1 Detailed Description	12
7.14.2 Member Function Documentation	12
7.14.2.1 close()	
7.14.2.2 context()	
7.14.2.3 device()	
7.14.2.4 isBroadcastEnabled()	
7.14.2.5 isOpen()	
7.14.2.6 isStateClosed()	
7.14.2.7 isTcpServer()	
7.14.2.8 ModbusObject()	
7.14.2.9 open()	
7.14.2.10 process()	
7.14.2.11 setBroadcastEnabled()	
7.14.2.12 setContext()	
7.14.2.13 setDevice()	
7.14.2.14 setUnitMap()	
7.14.2.15 signalClosed()	15

7.14.2.16 signalError()	115
7.14.2.17 signalOpened()	115
7.14.2.18 signalRx()	115
7.14.2.19 signalTx()	115
7.14.2.20 type()	115
7.14.2.21 unitMap()	116
7.15 ModbusServerResource Class Reference	116
7.15.1 Detailed Description	117
7.15.2 Constructor & Destructor Documentation	118
7.15.2.1 ModbusServerResource()	118
7.15.3 Member Function Documentation	118
7.15.3.1 close()	118
7.15.3.2 isOpen()	118
7.15.3.3 open()	118
7.15.3.4 port()	118
7.15.3.5 process()	119
7.15.3.6 processDevice()	119
7.15.3.7 processInputData()	119
7.15.3.8 processOutputData()	119
7.15.3.9 type()	119
7.16 ModbusSlotBase< ReturnType, Args > Class Template Reference	119
7.16.1 Detailed Description	120
7.16.2 Constructor & Destructor Documentation	120
7.16.2.1 ∼ModbusSlotBase()	120
7.16.3 Member Function Documentation	120
7.16.3.1 exec()	120
7.16.3.2 methodOrFunction()	120
7.16.3.3 object()	121
$7.17\ ModbusSlotFunction < ReturnType, \ Args > Class\ Template\ Reference\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots$	121
7.17.1 Detailed Description	121
7.17.2 Constructor & Destructor Documentation	121
7.17.2.1 ModbusSlotFunction()	121
7.17.3 Member Function Documentation	122
7.17.3.1 exec()	122
7.17.3.2 methodOrFunction()	122
7.18 ModbusSlotMethod< T, ReturnType, Args > Class Template Reference	122
7.18.1 Detailed Description	123
7.18.2 Constructor & Destructor Documentation	123
7.18.2.1 ModbusSlotMethod()	123
7.18.3 Member Function Documentation	123
7.18.3.1 exec()	123
7.18.3.2 methodOrFunction()	124

7.18.3.3 object()	124
7.19 ModbusTcpPort Class Reference	124
7.19.1 Detailed Description	125
7.19.2 Constructor & Destructor Documentation	126
7.19.2.1 ModbusTcpPort() [1/2]	126
7.19.2.2 ModbusTcpPort() [2/2]	126
7.19.2.3 ~ ModbusTcpPort()	126
7.19.3 Member Function Documentation	126
7.19.3.1 autoIncrement()	126
7.19.3.2 close()	126
7.19.3.3 handle()	126
7.19.3.4 host()	127
7.19.3.5 isOpen()	127
7.19.3.6 open()	127
7.19.3.7 port()	127
7.19.3.8 read()	127
7.19.3.9 readBuffer()	127
7.19.3.10 readBufferData()	128
7.19.3.11 readBufferSize()	128
7.19.3.12 setHost()	128
7.19.3.13 setNextRequestRepeated()	128
7.19.3.14 setPort()	128
7.19.3.15 type()	128
7.19.3.16 write()	129
7.19.3.17 writeBuffer()	129
7.19.3.18 writeBufferData()	129
7.19.3.19 writeBufferSize()	129
7.20 ModbusTcpServer Class Reference	129
7.20.1 Detailed Description	131
7.20.2 Constructor & Destructor Documentation	131
7.20.2.1 ModbusTcpServer()	131
7.20.2.2 ~ModbusTcpServer()	132
7.20.3 Member Function Documentation	132
7.20.3.1 clearConnections()	132
7.20.3.2 close()	132
7.20.3.3 createTcpPort()	132
7.20.3.4 deleteTcpPort()	132
7.20.3.5 isOpen()	132
7.20.3.6 isTcpServer()	133
7.20.3.7 nextPendingConnection()	133
7.20.3.8 open()	133
7.20.3.9 port()	133

	 133
7.20.3.11 setBroadcastEnabled()	 134
7.20.3.12 setPort()	 134
7.20.3.13 setTimeout()	 134
7.20.3.14 setUnitMap()	 134
7.20.3.15 signalCloseConnection()	 134
7.20.3.16 signalNewConnection()	 135
7.20.3.17 timeout()	 135
7.20.3.18 type()	 135
7.21 Modbus::SerialSettings Struct Reference	 135
7.21.1 Detailed Description	 136
7.22 Modbus::Strings Class Reference	 136
7.22.1 Detailed Description	 137
7.22.2 Constructor & Destructor Documentation	 137
7.22.2.1 Strings()	 137
7.22.3 Member Function Documentation	 137
7.22.3.1 instance()	 137
7.23 Modbus::TcpSettings Struct Reference	 138
7.23.1 Detailed Description	 138
O.E.L. Bernard Mark	139
8 File Documentation	
8 File Documentation 8.1 c:/Users/march/Dropbox/PRJ/ModbusLib/src/cModbus.h File Reference	
8.1 c:/Users/march/Dropbox/PRJ/ModbusLib/src/cModbus.h File Reference	 139
8.1 c:/Users/march/Dropbox/PRJ/ModbusLib/src/cModbus.h File Reference 8.1.1 Detailed Description	 139 143
8.1 c:/Users/march/Dropbox/PRJ/ModbusLib/src/cModbus.h File Reference 8.1.1 Detailed Description	 139 143 143
8.1 c:/Users/march/Dropbox/PRJ/ModbusLib/src/cModbus.h File Reference	 139 143 143 143
8.1 c:/Users/march/Dropbox/PRJ/ModbusLib/src/cModbus.h File Reference	139 143 143 143 143
8.1 c:/Users/march/Dropbox/PRJ/ModbusLib/src/cModbus.h File Reference	139 143 143 143 143
8.1 c:/Users/march/Dropbox/PRJ/ModbusLib/src/cModbus.h File Reference	139 143 143 143 143 144
8.1 c:/Users/march/Dropbox/PRJ/ModbusLib/src/cModbus.h File Reference	139 143 143 143 143 144 144
8.1 c:/Users/march/Dropbox/PRJ/ModbusLib/src/cModbus.h File Reference	139 143 143 143 143 144 144 144
8.1 c:/Users/march/Dropbox/PRJ/ModbusLib/src/cModbus.h File Reference	139 143 143 143 143 144 144 144
8.1 c:/Users/march/Dropbox/PRJ/ModbusLib/src/cModbus.h File Reference	139 143 143 143 143 144 144 144
8.1 c:/Users/march/Dropbox/PRJ/ModbusLib/src/cModbus.h File Reference	139 143 143 143 143 144 144 144 145 145
8.1 c:/Users/march/Dropbox/PRJ/ModbusLib/src/cModbus.h File Reference  8.1.1 Detailed Description  8.1.2 Typedef Documentation  8.1.2.1 pfDiagnostics  8.1.2.2 pfGetCommEventCounter  8.1.2.3 pfGetCommEventLog  8.1.2.4 pfMaskWriteRegister  8.1.2.5 pfReadCoils  8.1.2.6 pfReadDiscreteInputs  8.1.2.7 pfReadExceptionStatus  8.1.2.8 pfReadFIFOQueue  8.1.2.9 pfReadHoldingRegisters  8.1.2.10 pfReadInputRegisters	139 143 143 143 143 144 144 144 145 145
8.1 c:/Users/march/Dropbox/PRJ/ModbusLib/src/cModbus.h File Reference  8.1.1 Detailed Description  8.1.2 Typedef Documentation  8.1.2.1 pfDiagnostics  8.1.2.2 pfGetCommEventCounter  8.1.2.3 pfGetCommEventLog  8.1.2.4 pfMaskWriteRegister  8.1.2.5 pfReadCoils  8.1.2.6 pfReadDiscreteInputs  8.1.2.7 pfReadExceptionStatus  8.1.2.8 pfReadFIFOQueue  8.1.2.9 pfReadHoldingRegisters  8.1.2.10 pfReadInputRegisters  8.1.2.11 pfReadWriteMultipleRegisters	139 143 143 143 143 144 144 144 145 145
8.1 c:/Users/march/Dropbox/PRJ/ModbusLib/src/cModbus.h File Reference  8.1.1 Detailed Description  8.1.2 Typedef Documentation  8.1.2.1 pfDiagnostics  8.1.2.2 pfGetCommEventCounter  8.1.2.3 pfGetCommEventLog  8.1.2.4 pfMaskWriteRegister  8.1.2.5 pfReadCoils  8.1.2.6 pfReadDiscreteInputs  8.1.2.7 pfReadExceptionStatus  8.1.2.8 pfReadFIFOQueue  8.1.2.9 pfReadHoldingRegisters  8.1.2.10 pfReadInputRegisters	139 143 143 143 144 144 144 145 145 145 145
8.1 c:/Users/march/Dropbox/PRJ/ModbusLib/src/cModbus.h File Reference 8.1.1 Detailed Description 8.1.2 Typedef Documentation 8.1.2.1 pfDiagnostics 8.1.2.2 pfGetCommEventCounter 8.1.2.3 pfGetCommEventLog 8.1.2.4 pfMaskWriteRegister 8.1.2.5 pfReadCoils 8.1.2.6 pfReadDiscreteInputs 8.1.2.7 pfReadExceptionStatus 8.1.2.8 pfReadFIFOQueue 8.1.2.9 pfReadHoldingRegisters 8.1.2.10 pfReadInputRegisters 8.1.2.11 pfReadWriteMultipleRegisters 8.1.2.12 pfReportServerID	139 143 143 143 144 144 144 145 145 145 145
8.1 c:/Users/march/Dropbox/PRJ/ModbusLib/src/cModbus.h File Reference  8.1.1 Detailed Description  8.1.2 Typedef Documentation  8.1.2.1 pfDiagnostics  8.1.2.2 pfGetCommEventCounter  8.1.2.3 pfGetCommEventLog  8.1.2.4 pfMaskWriteRegister  8.1.2.5 pfReadCoils  8.1.2.6 pfReadDiscreteInputs  8.1.2.7 pfReadExceptionStatus  8.1.2.8 pfReadFIFOQueue  8.1.2.9 pfReadHoldingRegisters  8.1.2.10 pfReadInputRegisters  8.1.2.11 pfReadWriteMultipleRegisters  8.1.2.12 pfReportServerID  8.1.2.13 pfSlotCloseConnection	139 143 143 143 144 144 144 145 145 145 146 146
8.1 c:/Users/march/Dropbox/PRJ/ModbusLib/src/cModbus.h File Reference 8.1.1 Detailed Description 8.1.2 Typedef Documentation 8.1.2.1 pfDiagnostics 8.1.2.2 pfGetCommEventCounter 8.1.2.3 pfGetCommEventLog 8.1.2.4 pfMaskWriteRegister 8.1.2.5 pfReadCoils 8.1.2.6 pfReadDiscreteInputs 8.1.2.7 pfReadExceptionStatus 8.1.2.8 pfReadFIFOQueue 8.1.2.9 pfReadHoldingRegisters 8.1.2.10 pfReadInputRegisters 8.1.2.11 pfReadWriteMultipleRegisters 8.1.2.12 pfReportServerID 8.1.2.13 pfSlotCloseConnection 8.1.2.14 pfSlotCloseConnection	139 143 143 143 144 144 144 145 145 145 145 146 146
8.1 c://Users/march/Dropbox/PRJ/ModbusLib/src/cModbus.h File Reference 8.1.1 Detailed Description 8.1.2 Typedef Documentation 8.1.2.1 pfDiagnostics 8.1.2.2 pfGetCommEventCounter 8.1.2.3 pfGetCommEventLog 8.1.2.4 pfMaskWriteRegister 8.1.2.5 pfReadCoils 8.1.2.6 pfReadDiscreteInputs 8.1.2.7 pfReadExceptionStatus 8.1.2.8 pfReadFIFOQueue 8.1.2.9 pfReadHoldingRegisters 8.1.2.10 pfReadInputRegisters 8.1.2.11 pfReadWriteMultipleRegisters 8.1.2.12 pfReportServerID 8.1.2.13 pfSlotCloseConnection 8.1.2.14 pfSlotClosed 8.1.2.15 pfSlotError	139 143 143 143 144 144 144 145 145 145 146 146 146

8.1.2.19 pfSlotTx	147
8.1.2.20 pfWriteMultipleCoils	147
8.1.2.21 pfWriteMultipleRegisters	147
8.1.2.22 pfWriteSingleCoil	148
8.1.2.23 pfWriteSingleRegister	148
8.1.3 Function Documentation	148
8.1.3.1 cCliCreate()	148
8.1.3.2 cCliCreateForClientPort()	148
8.1.3.3 cCliDelete()	149
8.1.3.4 cCliGetLastPortErrorStatus()	149
8.1.3.5 cCliGetLastPortErrorText()	149
8.1.3.6 cCliGetLastPortStatus()	149
8.1.3.7 cCliGetObjectName()	149
8.1.3.8 cCliGetPort()	149
8.1.3.9 cCliGetType()	149
8.1.3.10 cCliGetUnit()	150
8.1.3.11 cClilsOpen()	150
8.1.3.12 cCliSetObjectName()	150
8.1.3.13 cCliSetUnit()	150
8.1.3.14 cCpoClose()	150
8.1.3.15 cCpoConnectClosed()	150
8.1.3.16 cCpoConnectError()	151
8.1.3.17 cCpoConnectOpened()	151
8.1.3.18 cCpoConnectRx()	151
8.1.3.19 cCpoConnectTx()	151
8.1.3.20 cCpoCreate()	151
8.1.3.21 cCpoCreateForPort()	151
8.1.3.22 cCpoDelete()	152
8.1.3.23 cCpoDiagnostics()	152
8.1.3.24 cCpoDisconnectFunc()	152
8.1.3.25 cCpoGetCommEventCounter()	152
8.1.3.26 cCpoGetCommEventLog()	152
8.1.3.27 cCpoGetLastErrorStatus()	153
8.1.3.28 cCpoGetLastErrorText()	153
8.1.3.29 cCpoGetLastStatus()	153
8.1.3.30 cCpoGetObjectName()	153
8.1.3.31 cCpoGetRepeatCount()	153
8.1.3.32 cCpoGetType()	153
8.1.3.33 cCpolsOpen()	153
8.1.3.34 cCpoMaskWriteRegister()	154
8.1.3.35 cCpoReadCoils()	154
8.1.3.36 cCpoReadCoilsAsBoolArray()	154

8.1.3.37 cCpoReadDiscreteInputs()
8.1.3.38 cCpoReadDiscreteInputsAsBoolArray()
8.1.3.39 cCpoReadExceptionStatus()
8.1.3.40 cCpoReadFIFOQueue()
8.1.3.41 cCpoReadHoldingRegisters()
8.1.3.42 cCpoReadInputRegisters()
8.1.3.43 cCpoReadWriteMultipleRegisters()
8.1.3.44 cCpoReportServerID()
8.1.3.45 cCpoSetObjectName()
8.1.3.46 cCpoSetRepeatCount()
8.1.3.47 cCpoWriteMultipleCoils()
8.1.3.48 cCpoWriteMultipleCoilsAsBoolArray()
8.1.3.49 cCpoWriteMultipleRegisters()
8.1.3.50 cCpoWriteSingleCoil()
8.1.3.51 cCpoWriteSingleRegister()
8.1.3.52 cCreateModbusDevice()
8.1.3.53 cDeleteModbusDevice()
8.1.3.54 cMaskWriteRegister()
8.1.3.55 cPortCreate()
8.1.3.56 cPortDelete()
8.1.3.57 cReadCoils()
8.1.3.58 cReadCoilsAsBoolArray()
8.1.3.59 cReadDiscreteInputs()
8.1.3.60 cReadDiscreteInputsAsBoolArray()
8.1.3.61 cReadExceptionStatus()
8.1.3.62 cReadHoldingRegisters()
8.1.3.63 cReadInputRegisters()
8.1.3.64 cReadWriteMultipleRegisters()
8.1.3.65 cSpoClose()
8.1.3.66 cSpoConnectCloseConnection()
8.1.3.67 cSpoConnectClosed()
8.1.3.68 cSpoConnectError()
8.1.3.69 cSpoConnectNewConnection()
8.1.3.70 cSpoConnectOpened()
8.1.3.71 cSpoConnectRx()
8.1.3.72 cSpoConnectTx()
8.1.3.73 cSpoCreate()
8.1.3.74 cSpoDelete()
8.1.3.75 cSpoDisconnectFunc()
8.1.3.76 cSpoGetDevice()
8.1.3.77 cSpoGetObjectName()
8.1.3.78 cSpoGetType()

8.1.3.79 cSpolsOpen()
8.1.3.80 cSpoIsTcpServer()
8.1.3.81 cSpoOpen()
8.1.3.82 cSpoProcess()
8.1.3.83 cSpoSetObjectName()
8.1.3.84 cWriteMultipleCoils()
8.1.3.85 cWriteMultipleCoilsAsBoolArray()
8.1.3.86 cWriteMultipleRegisters()
8.1.3.87 cWriteSingleCoil()
8.1.3.88 cWriteSingleRegister()
8.2 cModbus.h
8.3 c:/Users/march/Dropbox/PRJ/ModbusLib/src/Modbus.h File Reference
8.3.1 Detailed Description
8.4 Modbus.h
8.5 Modbus_config.h
8.6 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusAscPort.h File Reference
8.6.1 Detailed Description
8.7 ModbusAscPort.h
8.8 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusClient.h File Reference
8.8.1 Detailed Description
8.9 ModbusClient.h
8.10 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusClientPort.h File Reference
8.10.1 Detailed Description
8.11 ModbusClientPort.h
8.12 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusGlobal.h File Reference
8.12.1 Detailed Description
0.12.1 Detailed Description
8.12.2 Macro Definition Documentation
·
8.12.2 Macro Definition Documentation
8.12.2 Macro Definition Documentation       18         8.12.2.1 CharLiteral       18
8.12.2 Macro Definition Documentation       18         8.12.2.1 CharLiteral       18         8.12.2.2 GET_BIT       18
8.12.2 Macro Definition Documentation       18         8.12.2.1 CharLiteral       18         8.12.2.2 GET_BIT       18         8.12.2.3 GET_BITS       18
8.12.2 Macro Definition Documentation       18         8.12.2.1 CharLiteral       18         8.12.2.2 GET_BIT       18         8.12.2.3 GET_BITS       18         8.12.2.4 MB_RTU_IO_BUFF_SZ       18
8.12.2 Macro Definition Documentation       18         8.12.2.1 CharLiteral       18         8.12.2.2 GET_BIT       18         8.12.2.3 GET_BITS       18         8.12.2.4 MB_RTU_IO_BUFF_SZ       18         8.12.2.5 MB_UNITMAP_GET_BIT       18
8.12.2 Macro Definition Documentation       18         8.12.2.1 CharLiteral       18         8.12.2.2 GET_BIT       18         8.12.2.3 GET_BITS       18         8.12.2.4 MB_RTU_IO_BUFF_SZ       18         8.12.2.5 MB_UNITMAP_GET_BIT       18         8.12.2.6 MB_UNITMAP_SET_BIT       18
8.12.2 Macro Definition Documentation       18         8.12.2.1 CharLiteral       18         8.12.2.2 GET_BIT       18         8.12.2.3 GET_BITS       18         8.12.2.4 MB_RTU_IO_BUFF_SZ       18         8.12.2.5 MB_UNITMAP_GET_BIT       18         8.12.2.6 MB_UNITMAP_SET_BIT       18         8.12.2.7 SET_BIT       18
8.12.2 Macro Definition Documentation       18         8.12.2.1 CharLiteral       18         8.12.2.2 GET_BIT       18         8.12.2.3 GET_BITS       18         8.12.2.4 MB_RTU_IO_BUFF_SZ       18         8.12.2.5 MB_UNITMAP_GET_BIT       18         8.12.2.6 MB_UNITMAP_SET_BIT       18         8.12.2.7 SET_BIT       18         8.12.2.8 SET_BITS       18
8.12.2 Macro Definition Documentation       18         8.12.2.1 CharLiteral       18         8.12.2.2 GET_BIT       18         8.12.2.3 GET_BITS       18         8.12.2.4 MB_RTU_IO_BUFF_SZ       18         8.12.2.5 MB_UNITMAP_GET_BIT       18         8.12.2.6 MB_UNITMAP_SET_BIT       18         8.12.2.7 SET_BIT       18         8.12.2.8 SET_BITS       18         8.12.2.9 StringLiteral       18
8.12.2 Macro Definition Documentation       18         8.12.2.1 CharLiteral       18         8.12.2.2 GET_BIT       18         8.12.2.3 GET_BITS       18         8.12.2.4 MB_RTU_IO_BUFF_SZ       18         8.12.2.5 MB_UNITMAP_GET_BIT       18         8.12.2.6 MB_UNITMAP_SET_BIT       18         8.12.2.7 SET_BIT       18         8.12.2.8 SET_BITS       18         8.12.2.9 StringLiteral       18         8.13 ModbusGlobal.h       18
8.12.2 Macro Definition Documentation       18         8.12.2.1 CharLiteral       18         8.12.2.2 GET_BIT       18         8.12.2.3 GET_BITS       18         8.12.2.4 MB_RTU_IO_BUFF_SZ       18         8.12.2.5 MB_UNITMAP_GET_BIT       18         8.12.2.6 MB_UNITMAP_SET_BIT       18         8.12.2.7 SET_BIT       18         8.12.2.8 SET_BITS       18         8.12.2.9 StringLiteral       18         8.13 ModbusGlobal.h       18         8.14 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusObject.h File Reference       19
8.12.2 Macro Definition Documentation       18         8.12.2.1 CharLiteral       18         8.12.2.2 GET_BIT       18         8.12.2.3 GET_BITS       18         8.12.2.4 MB_RTU_IO_BUFF_SZ       18         8.12.2.5 MB_UNITMAP_GET_BIT       18         8.12.2.6 MB_UNITMAP_SET_BIT       18         8.12.2.7 SET_BIT       18         8.12.2.8 SET_BITS       18         8.12.2.9 StringLiteral       18         8.13 ModbusGlobal.h       18         8.14 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusObject.h File Reference       19         8.14.1 Detailed Description       19

	8.17 ModbusPlatform.h	196
	8.18 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusPort.h File Reference	196
	8.18.1 Detailed Description	196
	8.19 ModbusPort.h	197
	8.20 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusQt.h File Reference	198
	8.20.1 Detailed Description	200
	8.21 ModbusQt.h	200
	8.22 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusRtuPort.h File Reference	203
	8.22.1 Detailed Description	204
	8.23 ModbusRtuPort.h	204
	8.24 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusSerialPort.h File Reference	204
	8.24.1 Detailed Description	205
	8.25 ModbusSerialPort.h	205
	8.26 ModbusServerPort.h	206
	8.27 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusServerResource.h File Reference	207
	8.27.1 Detailed Description	207
	8.28 ModbusServerResource.h	207
	8.29 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusTcpPort.h File Reference	208
	8.29.1 Detailed Description	208
	8.30 ModbusTcpPort.h	208
	8.31 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusTcpServer.h File Reference	209
	8.31.1 Detailed Description	210
	8.32 ModbusTcpServer.h	210
In	dex	213

# **ModbusLib**

### 1.0.1 Overview

ModbusLib is a free, open-source Modbus library written in C++. It implements client and server functions for TCP, RTU and ASCII versions of Modbus Protocol. It has interface for C language (implements in cModbus.h header file). Also it has optional wrapper to use with Qt (implements in ModbusQt.h header file). Library can work in both blocking and non-blocking mode.

Library implements such Modbus functions as:

- 1 (0x01) READ\_COILS
- 2 (0x02) READ\_DISCRETE\_INPUTS
- 3 (0x03) READ\_HOLDING\_REGISTERS
- 4 (0x04) READ\_INPUT\_REGISTERS
- 5 (0x05) WRITE\_SINGLE\_COIL
- 6 (0x06) WRITE\_SINGLE\_REGISTER
- 7 (0x07) READ\_EXCEPTION\_STATUS
- 8 (0x08) DIAGNOSTICS
- 11 (0x0B) GET\_COMM\_EVENT\_COUNTER
- 12 (0x0C) GET\_COMM\_EVENT\_LOG
- 15 (0x0F) WRITE\_MULTIPLE\_COILS
- 16 (0x10) WRITE\_MULTIPLE\_REGISTERS
- 17 (0x11) REPORT\_SERVER\_ID
- 22 (0x16) MASK\_WRITE\_REGISTER
- 23 (0x17) WRITE\_MULTIPLE\_REGISTERS
- 24 (0x18) READ FIFO QUEUE

## 1.0.2 Using Library

#### 1.0.2.1 Common usage (C++)

Library was written in C++ and it is the main language to use it. To start using this library you must include ModbusClientPort.h (ModbusClient.h) or ModbusServerPort.h header files (of course after add include path to the compiler). This header directly or indirectly include Modbus.h main header file. Modbus.h header file contains declarations of main data types, functions and class interfaces to work with the library.

It contains definition of Modbus::StatusCode enumeration that defines result of library operations, ModbusInterface class interface that contains list of functions which the library implements, Modbus::createClientPort and Modbus::createServerPort functions, that creates corresponding ModbusClientPort and ModbusServerPort main working classes. Those classes that implements Modbus functions for the library for client and server version of protocol, respectively.

#### 1.0.2.2 Client

ModbusClientPort implements Modbus interface directly and can be used very simple:

```
#include <ModbusClientPort.h>
//...
void main()
{
    Modbus::TcpSettings settings;
    settings.host = "someadr.plc";
    settings.port = Modbus::STANDARD_TCP_PORT;
    settings.timeout = 3000;
    ModbusClientPort *port = Modbus::createClientPort(Modbus::TCP, &settings, true);
    const uint8_t unit = 1;
    const uint16_t offset = 0;
    const uint16_t count = 10;
    uint16_t values[count];
    Modbus::StatusCode status = port->readHoldingRegisters(unit, offset, count, values);
    if (Modbus::StatusCode status))
    {
            // process out array `values` ...
      }
      else
            std::cout « "Error: " « port->lastErrorText() « '\n';
      delete port;
}
```

User don't need to create any connection or open any port, library makes it automatically.

User can use ModbusClient class to simplify Modbus function's interface (don't need to use unit parameter):

```
#include <ModbusClientPort.h>
//...
void main()
{
    //...
    ModbusClient c1(1, port);
    ModbusClient c2(2, port);
    ModbusClient c3(3, port);
    Modbus::StatusCode s1, s2, s3;
    while(1)
    {
        s1 = c1.readHoldingRegisters(0, 10, values);
        s2 = c2.readHoldingRegisters(0, 10, values);
        s3 = c3.readHoldingRegisters(0, 10, values);
        Modbus::msleep(1);
    }
    //...
}
```

In this example 3 clients with unit address 1, 2, 3 are used. User don't need to manage its common resource port. Library make it automatically. First c1 client owns port, than when finished resource transferred to c2 and so on.

#### 1.0.2.3 Server

Unlike client the server do not implement ModbusInterface directly. It accepts pointer to ModbusInterface in its constructor as parameter and transfer all requests to this interface. So user can define by itself how incoming Modbus-request will be processed:

```
#include <ModbusServerPort.h>
//...
class MyModbusDevice : public ModbusInterface
#define MEM SIZE 16
    uint16_t mem4x[MEM_SIZE];
public:
    \label{eq:myModbusDevice() { memset(mem4x, 0, sizeof(mem4x)); } }
    uint16_t getValue(uint16_t offset) { return mem4x[offset]; }
    void setValue(uint16_t offset, uint16_t value) { mem4x[offset] = value; }
    Modbus::StatusCode readHoldingRegisters(uint8_t unit,
                                               uint16_t offset,
                                               uint16_t count,
                                               uint16 t *values) override
        if (unit != 1)
             ceturn Modbus::Status_BadGatewayPathUnavailable;
        if ((offset + count) <= MEM_SIZE)</pre>
            memcpy(values, &mem4x[offset], count*sizeof(uint16_t));
return Modbus::Status_Good;
        return Modbus::Status_BadIllegalDataAddress;
};
void main()
    MyModbusDevice device;
    Modbus::TcpSettings settings;
    settings.port = Modbus::STANDARD_TCP_PORT;
    settings.timeout = 3000;
    ModbusServerPort *port = Modbus::createServerPort(&device, Modbus::TCP, &settings, false);
    int c = 0;
    while (1)
        port->process();
        Modbus::msleep(1);
        if (c % 1000 == 0) setValue(0, getValue(0)+1);
//...
```

In this example MyModbusDevice ModbusInterface class was created. It imlements only single function:  $read \leftarrow HoldingRegisters$  (0x03). All other functions will return Modbus::Status\_BadIllegalFunction by default.

This example creates Modbus TCP server that process connections and increment first 4x register by 1 every second. This example uses non blocking mode.

### 1.0.2.3.1 Non blocking mode

In non blocking mode Modbus function exits immediately even if remote connection processing is not finished. In this case function returns Modbus::Status\_Processing. This is 'Arduino'-style of programing, when function must not be blocked and return intermediate value that indicates that function is not finished. Then external code call this function again and again until Good or Bad status will not be returned.

#### Example of non blocking client:

```
#include <ModbusClientPort.h>
//...
void main()
{
    //...
    ModbusClientPort *port = Modbus::createClientPort(Modbus::TCP, &settings, false);
    //...
    while(1)
    {
        s1 = c1.readHoldingRegisters(0, 10, values);
        s2 = c2.readHoldingRegisters(0, 10, values);
    }
}
```

```
s3 = c3.readHoldingRegisters(0, 10, values);
    doSomeOtherStuffInCurrentThread();
    Modbus::msleep(1);
}
//...
}
```

So if user needs to check is function finished he can write:

```
//...
s1 = c1.readHoldingRegisters(0, 10, values);
if (!Modbus::StatusIsProcessing(s1)) {
    // ...
}
//...
```

### 1.0.2.3.2 Signal/slot mechanism

Library has simplified Qt-like signal/slot mechanism that can use callbacks when some signal is occured. User can connect function(s) or class method(s) to the predefined signal. Callbacks will be called in the order in which they were connected.

For example ModbusClientPort signal/slot mechanism:

```
#include <ModbusClientPort.h>

class Printable {
  public:
    void printTx(const Modbus::Char *source, const uint8_t* buff, uint16_t size)
    {
        std::cout « source « " Tx: " « Modbus::bytesToString(buff, size) « '\n';
     }
};

void printRx(const Modbus::Char *source, const uint8_t* buff, uint16_t size)
{
    std::cout « source « " Rx: " « Modbus::bytesToString(buff, size) « '\n';
}

void main()
{
    //...
    ModbusClientPort *port = Modbus::createClientPort(Modbus::TCP, &settings, false);
    Printable print;
    port->connect(&ModbusClientPort::signalTx, &print, &Printable::printTx);
    port->connect(&ModbusClientPort::signalRx, printRx);
    //...
}
```

## 1.0.2.4 Using with C

To use the library with pure C language user needs to include only one header: cModbus.h. This header includes functions that wraps Modbus interface classes and its methods.

```
#include <cModbus.h>
//...
void printTx(const Char *source, const uint8_t* buff, uint16_t size)
{
    Char s[1000];
    printf("%s Tx: %s\n", source, sbytes(buff, size, s, sizeof(s)));
}

void printRx(const Char *source, const uint8_t* buff, uint16_t size)
{
    Char s[1000];
    printf("%s Rx: %s\n", source, sbytes(buff, size, s, sizeof(s)));
}

void main()
{
    TcpSettings settings;
    settings.host = "someadr.plc";
    settings.port = STANDARD_TCP_PORT;
    settings.timeout = 3000;
    const uint8_t unit = 1;
```

```
cModbusClient client = cCliCreate(unit, TCP, &settings, true);
cModbusClientPort cpo = cCliGetPort(client);
StatusCode s;
cCpoConnectTx(cpo, printTx);
cCpoConnectRx(cpo, printRx);
while(1)
{
    s = cReadHoldingRegisters(client, 0, 10, values);
    //...
    msleep(1);
}
//...
```

## 1.0.2.5 Using with Qt

When including ModbusQt.h user can use ModbusLib in convinient way in Qt framework. It has wrapper functions for Qt library to use it together with Qt core objects:

#include <ModbusQt.h

## 1.0.3 Examples

Examples is located in examples folder or root directory.

#### 1.0.3.1 democlient

democlient example demonstrate all implemented functions for client one by one begining from function with lowest number and then increasing this number with predefined period and other parameters. To see list of available parameters you can print next commands:

```
$ ./democlient -?
$ ./democlient -help
```

#### 1.0.3.2 mbclient

mbclient is a simple example that can work like command-line Modbus Client Tester. It can use only single function at a time but user can change parameters of every supported function. To see list of available parameters you can print next commands:

```
$ ./mbclient -?
$ ./mbclient -help

Usage example:
$ ./mbclient -func 3 -offset 0 -count 10 -period 500 -n inf
```

## 1.0.3.3 demoserver

demoserver example demonstrate all implemented functions for server. It uses single block for every type of Modbus memory (0x, 1x, 3x and 4x) and emulates value change for the first 16 bit register by inceremting it by 1 every 1000 milliseconds. So user can run Modbus Client to check first 16 bit of 000001 (100001) or first register 400001 (300001) changing every 1 second. To see list of available parameters you can print next commands:

```
$ ./demoserver -?
$ ./demoserver -help
```

#### 1.0.3.4 mbserver

mbserver is a simple example that can work like command-line Modbus Server Tester. It implements all function of Modbus library. So remote client can work with server reading and writting values to it. To see list of available parameters you can print next commands:

```
$ ./mbserver -?
$ ./mbserver -help

Usage example:
$ ./mbserver -c0 256 -c1 256 -c3 16 -c4 16 -type RTU -serial /dev/ttyS0
```

### 1.0.4 Tests

Unit Tests using googletest library. Googletest source library must be located in external/googletest

### 1.0.5 Documenations

Documentation is located in docs directory. Documentation is automatically generated by doxygen.

## 1.0.6 Building

### 1.0.6.1 Build using CMake

1. Build Tools

Previously you need to install c++ compiler kit, git and cmake itself (qt tools if needed).

Then set PATH env variable to find compliler, cmake, git etc.

Don't forget to use appropriate version of compiler, linker (x86|x64).

2. Create project directory, move to it and clone repository:

```
$ cd ~
$ mkdir src
$ cd src
$ git clone https://github.com/serhmarch/ModbusLib.git
```

3. Create and/or move to directory for build output, e.g. ~/bin/ModbusLib:

```
$ cd ~
$ mkdir -p bin/ModbusLib
$ cd bin/ModbusLib
```

4. Run cmake to generate project (make) files.

```
$ cmake -S ~/src/ModbusLib -B .
```

To make Qt-compatibility (switch off by default for cmake build) you can use next command (e.g. for Windows 64):

```
$$ \sim -DMB_QT_ENABLED=ON -DCMAKE_PREFIX_PATH:PATH=C:/Qt/5.15.2/msvc2019_64 -S < path\to src\\ModbusLib>-B .
```

5. Make binaries (+ debug|release config):

```
$ cmake --build .
$ cmake --build . --config Debug
$ cmake --build . --config Release
```

6. Resulting bin files is located in ./bin directory.

### 1.0.6.2 Build using qmake

1. Update package list:

```
$ sudo apt-get update
```

2. Install main build tools like g++, make etc:

```
$ sudo apt-get install build-essential
```

3. Install Qt tools:

```
$ sudo apt-get install qtbase5-dev qttools5-dev
```

4. Check for correct instalation:

```
$ whereis qmake
qmake: /usr/bin/qmake
$ whereis libQt5Core*
libQt5Core.prl: /usr/lib/x86_64-linux-gnu/libQt5Core.prl
libQt5Core.so: /usr/lib/x86_64-linux-gnu/libQt5Core.so
libQt5Core.so.5: /usr/lib/x86_64-linux-gnu/libQt5Core.so.5
libQt5Core.so.5.15: /usr/lib/x86_64-linux-gnu/libQt5Core.so.5.15
libQt5Core.so.5.15: /usr/lib/x86_64-linux-gnu/libQt5Core.so.5.15.3
$ whereis libQt5Help*
libQt5Help.prl: /usr/lib/x86_64-linux-gnu/libQt5Help.prl
libQt5Help.so: /usr/lib/x86_64-linux-gnu/libQt5Help.so
libQt5Help.so.5: /usr/lib/x86_64-linux-gnu/libQt5Help.so.5
libQt5Help.so.5.15: /usr/lib/x86_64-linux-gnu/libQt5Help.so.5
libQt5Help.so.5.15: /usr/lib/x86_64-linux-gnu/libQt5Help.so.5.15
libQt5Help.so.5.15: 3: /usr/lib/x86_64-linux-gnu/libQt5Help.so.5.15.3
```

5. Install git:

```
$ sudo apt-get install git
```

6. Create project directory, move to it and clone repository:

```
$ cd ~
$ mkdir src
$ cd src
$ git clone https://github.com/serhmarch/ModbusLib.git
```

7. Create and/or move to directory for build output, e.g. ~/bin/ModbusLib:

```
$ cd ~
$ mkdir -p bin/ModbusLib
$ cd bin/ModbusLib
```

8. Run qmake to create Makefile for build:

```
$ qmake ~/src/ModbusLib/src/ModbusLib.pro -spec linux-g++
```

9. To ensure Makefile was created print:

```
$ 1s -1
total 36
-rw-r--r- 1 march march 35001 May 6 18:41 Makefile
```

10. Finaly to make current set of programs print:

\$ make

11. After build step move to <build\_folder>/bin to ensure everything is correct:

```
$ cd bin
$ pwd
~/bin/ModbusLib/bin
```

# **Namespace Index**

# 2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

## Modbus

Main Modbus namespace.	Contains classes,	functions and	d constants to work w	ith Modbus-
protocol				17

10 Namespace Index

# **Hierarchical Index**

# 3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Modbus::Address	47
Modbus::Defaults	49
ModbusSerialPort::Defaults	51
ModbusTcpPort::Defaults	52
ModbusTcpServer::Defaults	53
ModbusInterface	85
ModbusClientPort	64
ModbusObject	93
ModbusClient	57
ModbusClientPort	64
ModbusServerPort	111
ModbusServerResource	16
ModbusTcpServer	29
ModbusPort	96
ModbusSerialPort	05
ModbusAscPort	54
ModbusRtuPort	02
ModbusTcpPort	24
ModbusSlotBase < ReturnType, Args >	119
ModbusSlotBase < ReturnType, Args >	
ModbusSlotFunction < ReturnType, Args >	21
ModbusSlotMethod < T, ReturnType, Args >	22
Modbus::SerialSettings	35
Modbus::Strings	36
Modbus::TcpSettings	38

12 Hierarchical Index

# **Class Index**

# 4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Modbus::Address	
Class for convinient manipulation with Modbus Data Address	47
Modbus::Defaults	
Holds the default values of the settings	49
ModbusSerialPort::Defaults	
Holds the default values of the settings	51
ModbusTcpPort::Defaults	
Defaults class constain default settings values for ModbusTcpPort	52
ModbusTcpServer::Defaults	
Defaults class constain default settings values for ModbusTcpServer	53
ModbusAscPort	
Implements ASCII version of the Modbus communication protocol	54
ModbusClient	
The ModbusClient class implements the interface of the client part of the Modbus protocol	57
ModbusClientPort	
The ModbusClientPort class implements the algorithm of the client part of the Modbus	
communication protocol port	64
ModbusInterface	
Main interface of Modbus communication protocol	85
ModbusObject	
The ModbusObject class is the base class for objects that use signal/slot mechanism	93
ModbusPort	
The abstract class ModbusPort is the base class for a specific implementation of the Modbus	
communication protocol	96
ModbusRtuPort	
Implements RTU version of the Modbus communication protocol	102
ModbusSerialPort	
The abstract class ModbusSerialPort is the base class serial port Modbus communications	105
ModbusServerPort	
Abstract base class for direct control of ModbusPort derived classes (TCP or serial) for server	
side	111
ModbusServerResource	
Implements direct control for ModbusPort derived classes (TCP or serial) for server side	116
ModbusSlotBase < ReturnType, Args >	
ModbusSlotBase base template for slot (method or function)	119

14 Class Index

ModbusSlotFunction < ReturnType, Args >	
ModbusSlotFunction template class hold pointer to slot function	21
ModbusSlotMethod< T, ReturnType, Args >	
ModbusSlotMethod template class hold pointer to object and its method	22
ModbusTcpPort	
Class ModbusTcpPort implements TCP version of Modbus protocol	24
ModbusTcpServer	
The ModbusTcpServer class implements TCP server part of the Modbus protocol 1	29
Modbus::SerialSettings	
Struct to define settings for Serial Port	35
Modbus::Strings	
Sets constant key values for the map of settings	36
Modbus::TcpSettings	
Struct to define settings for TCP connection	38

# File Index

# 5.1 File List

Here is a list of all documented files with brief descriptions:

c:/Users/march/Dropbox/PRJ/ModbusLib/src/cModbus.h
Contains library interface for C language
c:/Users/march/Dropbox/PRJ/ModbusLib/src/Modbus.h
Contains general definitions of the Modbus protocol
c:/Users/march/Dropbox/PRJ/ModbusLib/src/Modbus_config.h
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusAscPort.h
Contains definition of ASCII serial port class
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusClient.h
Header file of Modbus client
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusClientPort.h
General file of the algorithm of the client part of the Modbus protocol port 176
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusGlobal.h
Contains general definitions of the Modbus libarary (for C++ and "pure" C)
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusObject.h
The header file defines the class templates used to create signal/slot-like mechanism 192
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusPlatform.h
Definition of platform specific macros
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusPort.h
Header file of abstract class ModbusPort
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusQt.h
Qt support file for ModbusLib
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusRtuPort.h
Contains definition of RTU serial port class
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusSerialPort.h
Contains definition of base serial port class
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusServerPort.h
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusServerResource.h
The header file defines the class that controls specific port
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusTcpPort.h
Header file of class ModbusTcpPort 208
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusTcpServer.h
Header file of Modbus TCP server

16 File Index

# **Chapter 6**

# **Namespace Documentation**

# 6.1 Modbus Namespace Reference

Main Modbus namespace. Contains classes, functions and constants to work with Modbus-protocol.

#### **Classes**

· class Address

Class for convinient manipulation with Modbus Data Address.

· class Defaults

Holds the default values of the settings.

struct SerialSettings

Struct to define settings for Serial Port.

class Strings

Sets constant key values for the map of settings.

struct TcpSettings

Struct to define settings for TCP connection.

## **Typedefs**

· typedef std::string String

Modbus::String class for strings.

• template<class T >

using **List** = std::list<T>

Modbus::List template class.

• typedef void \* Handle

Handle type for native OS values.

typedef char Char

Type for Modbus character.

typedef uint32\_t Timer

Type for Modbus timer.

• typedef int64\_t Timestamp

Type for Modbus timestamp (in UNIX millisec format)

typedef enum Modbus::\_MemoryType MemoryType

Defines type of memory used in Modbus protocol.

typedef QHash
 QString, QVariant
 Settings

Map for settings of Modbus protocol where key has type <code>QString</code> and value is <code>QVariant</code>.

#### **Enumerations**

```
    enum Constants { VALID_MODBUS_ADDRESS_BEGIN = 1 , VALID_MODBUS_ADDRESS_END = 247 ,

   STANDARD_TCP_PORT = 502 }
         Define list of contants of Modbus protocol.
enum MemoryType {
   Memory Unknown = 0xFFFF, Memory 0x = 0, Memory Coils = Memory 0x, Memory 1x = 1,
   Memory_DiscreteInputs = Memory_1x , Memory_3x = 3 , Memory_InputRegisters = Memory_3x ,
   Memory_4x = 4,
   Memory_HoldingRegisters = Memory_4x }
         Defines type of memory used in Modbus protocol.
enum StatusCode {
   Status Processing = 0x80000000 , Status Good = 0x000000000 , Status Bad = 0x01000000 ,
   Status Uncertain = 0x02000000,
   Status_BadlllegalFunction = Status_Bad | 0x01 , Status_BadlllegalDataAddress = Status_Bad | 0x02 ,
   Status_BadlllegalDataValue = Status_Bad | 0x03, Status_BadServerDeviceFailure = Status_Bad | 0x04,
   Status BadAcknowledge = Status Bad | 0x05 , Status BadServerDeviceBusy = Status Bad | 0x06 ,
   Status_BadNegativeAcknowledge = Status_Bad | 0x07 , Status_BadMemoryParityError = Status_Bad | 0x08
   Status BadGatewayPathUnavailable = Status Bad | 0x0A, Status BadGatewayTargetDeviceFailedToRespond
   = Status Bad | 0x0B , Status BadEmptyResponse = Status Bad | 0x101 , Status BadNotCorrectRequest ,
   Status\_BadNotCorrectResponse \ , \ Status\_BadWriteBufferOverflow \ , \ Status\_BadReadBufferOverflow \ , \ Status\_BadRea
   Status BadSerialOpen = Status Bad | 0x201,
   Status BadSerialWrite, Status BadSerialRead, Status BadSerialReadTimeout, Status BadSerialWriteTimeout
   Status_BadAscMissColon = Status_Bad | 0x301 , Status_BadAscMissCrLf , Status_BadAscChar ,
   Status BadLrc,
   Status_BadCrc = Status_Bad | 0x401 , Status_BadTcpCreate = Status_Bad | 0x501 , Status_BadTcpConnect
   , Status_BadTcpWrite ,
   Status BadTcpRead, Status BadTcpBind, Status BadTcpListen, Status BadTcpAccept,
   Status BadTcpDisconnect }
         Defines status of executed Modbus functions.

    enum ProtocolType { ASC , RTU , TCP }

         Defines type of Modbus protocol.
enum Parity {
   NoParity, EvenParity, OddParity, SpaceParity,
   MarkParity }
         Defines Parity for serial port.

    enum StopBits { OneStop , OneAndHalfStop , TwoStop }

         Defines Stop Bits for serial port.

    enum FlowControl { NoFlowControl , HardwareControl , SoftwareControl }

         FlowControl Parity for serial port.
```

#### **Functions**

- MODBUS\_EXPORT String getLastErrorText ()
- String toModbusString (int val)
- MODBUS EXPORT String bytesToString (const uint8 t \*buff, uint32 t count)
- MODBUS\_EXPORT String asciiToString (const uint8\_t \*buff, uint32\_t count)
- MODBUS EXPORT List< String > availableSerialPorts ()
- MODBUS EXPORT List< int32 t > availableBaudRate ()
- MODBUS EXPORT List< int8 t > availableDataBits ()
- MODBUS EXPORT List< Parity > availableParity ()
- MODBUS EXPORT List< StopBits > availableStopBits ()

- MODBUS\_EXPORT List< FlowControl > availableFlowControl ()
- MODBUS\_EXPORT ModbusPort \* createPort (ProtocolType type, const void \*settings, bool blocking)
- MODBUS\_EXPORT ModbusClientPort \* createClientPort (ProtocolType type, const void \*settings, bool blocking)
- MODBUS\_EXPORT ModbusServerPort \* createServerPort (ModbusInterface \*device, ProtocolType type, const void \*settings, bool blocking)
- StatusCode readMemRegs (uint32\_t offset, uint32\_t count, void \*values, const void \*memBuff, uint32\_←
  t memRegCount)
- StatusCode writeMemRegs (uint32\_t offset, uint32\_t count, const void \*values, void \*memBuff, uint32\_←
  t memRegCount)
- StatusCode readMemBits (uint32\_t offset, uint32\_t count, void \*values, const void \*memBuff, uint32\_←
  t memBitCount)
- StatusCode writeMemBits (uint32\_t offset, uint32\_t count, const void \*values, void \*memBuff, uint32\_←
  t memBitCount)
- bool StatusIsProcessing (StatusCode status)
- bool StatusIsGood (StatusCode status)
- bool StatusIsBad (StatusCode status)
- bool StatusIsUncertain (StatusCode status)
- bool StatusIsStandardError (StatusCode status)
- bool getBit (const void \*bitBuff, uint16\_t bitNum)
- bool getBitS (const void \*bitBuff, uint16 t bitNum, uint16 t maxBitCount)
- void setBit (void \*bitBuff, uint16\_t bitNum, bool value)
- void setBitS (void \*bitBuff, uint16 t bitNum, bool value, uint16 t maxBitCount)
- bool \* getBits (const void \*bitBuff, uint16 t bitNum, uint16 t bitCount, bool \*boolBuff)
- bool \* getBitsS (const void \*bitBuff, uint16 t bitNum, uint16 t bitCount, bool \*boolBuff, uint16 t maxBitCount)
- void \* setBits (void \*bitBuff, uint16 t bitNum, uint16 t bitCount, const bool \*boolBuff)
- void \* setBitsS (void \*bitBuff, uint16\_t bitNum, uint16\_t bitCount, const bool \*boolBuff, uint16\_t maxBitCount)
- MODBUS EXPORT uint32 t modbusLibVersion ()
- MODBUS EXPORT const Char \* modbusLibVersionStr ()
- uint16\_t toModbusOffset (uint32\_t adr)
- MODBUS\_EXPORT uint16\_t crc16 (const uint8\_t \*byteArr, uint32\_t count)
- MODBUS\_EXPORT uint8\_t lrc (const uint8\_t \*byteArr, uint32\_t count)
- MODBUS\_EXPORT StatusCode readMemRegs (uint32\_t offset, uint32\_t count, void \*values, const void \*memBuff, uint32\_t memRegCount, uint32\_t \*outCount)
- MODBUS\_EXPORT StatusCode writeMemRegs (uint32\_t offset, uint32\_t count, const void \*values, void \*memBuff, uint32\_t memRegCount, uint32\_t \*outCount)
- MODBUS\_EXPORT StatusCode readMemBits (uint32\_t offset, uint32\_t count, void \*values, const void \*memBuff, uint32\_t memBitCount, uint32\_t \*outCount)
- MODBUS\_EXPORT StatusCode writeMemBits (uint32\_t offset, uint32\_t count, const void \*values, void \*memBuff, uint32\_t memBitCount, uint32\_t \*outCount)
- MODBUS EXPORT uint32 t bytesToAscii (const uint8 t \*bytesBuff, uint8 t \*asciiBuff, uint32 t count)
- MODBUS\_EXPORT uint32\_t asciiToBytes (const uint8\_t \*asciiBuff, uint8\_t \*bytesBuff, uint32\_t count)
- MODBUS EXPORT Char \* sbytes (const uint8 t \*buff, uint32 t count, Char \*str, uint32 t strmaxlen)
- MODBUS\_EXPORT Char \* sascii (const uint8\_t \*buff, uint32\_t count, Char \*str, uint32\_t strmaxlen)
- MODBUS\_EXPORT const Char \* sprotocolType (ProtocolType type)
- MODBUS\_EXPORT const Char \* sparity (Parity parity)
- MODBUS\_EXPORT const Char \* sstopBits (StopBits stopBits)
- MODBUS\_EXPORT const Char \* sflowControl (FlowControl flowControl)
- MODBUS\_EXPORT Timer timer ()
- MODBUS\_EXPORT Timestamp currentTimestamp ()
- MODBUS\_EXPORT void msleep (uint32\_t msec)
- MODBUS EXPORT uint8 t getSettingUnit (const Settings &s, bool \*ok=nullptr)
- MODBUS EXPORT ProtocolType getSettingType (const Settings &s, bool \*ok=nullptr)
- MODBUS\_EXPORT uint32\_t getSettingTries (const Settings &s, bool \*ok=nullptr)
- MODBUS\_EXPORT QString getSettingHost (const Settings &s, bool \*ok=nullptr)

- MODBUS\_EXPORT uint16\_t getSettingPort (const Settings &s, bool \*ok=nullptr)
- MODBUS\_EXPORT uint32\_t getSettingTimeout (const Settings &s, bool \*ok=nullptr)
- MODBUS EXPORT QString getSettingSerialPortName (const Settings &s, bool \*ok=nullptr)
- MODBUS\_EXPORT int32\_t getSettingBaudRate (const Settings &s, bool \*ok=nullptr)
- MODBUS\_EXPORT int8\_t getSettingDataBits (const Settings &s, bool \*ok=nullptr)
- MODBUS\_EXPORT Parity getSettingParity (const Settings &s, bool \*ok=nullptr)
- MODBUS EXPORT StopBits getSettingStopBits (const Settings &s, bool \*ok=nullptr)
- MODBUS EXPORT FlowControl getSettingFlowControl (const Settings &s, bool \*ok=nullptr)
- MODBUS EXPORT uint32 t getSettingTimeoutFirstByte (const Settings &s, bool \*ok=nullptr)
- MODBUS EXPORT uint32 t getSettingTimeoutInterByte (const Settings &s, bool \*ok=nullptr)
- MODBUS\_EXPORT bool getSettingBroadcastEnabled (const Settings &s, bool \*ok=nullptr)
- MODBUS EXPORT void setSettingUnit (Settings &s, uint8 t v)
- MODBUS\_EXPORT void setSettingType (Settings &s, ProtocolType v)
- MODBUS\_EXPORT void setSettingTries (Settings &s, uint32\_t)
- MODBUS EXPORT void setSettingHost (Settings &s, const QString &v)
- MODBUS EXPORT void setSettingPort (Settings &s, uint16 t v)
- MODBUS EXPORT void setSettingTimeout (Settings &s, uint32 t v)
- MODBUS EXPORT void setSettingSerialPortName (Settings &s, const QString &v)
- MODBUS\_EXPORT void setSettingBaudRate (Settings &s, int32\_t v)
- MODBUS EXPORT void setSettingDataBits (Settings &s, int8 t v)
- MODBUS EXPORT void setSettingParity (Settings &s, Parity v)
- MODBUS\_EXPORT void setSettingStopBits (Settings &s, StopBits v)
- MODBUS EXPORT void setSettingFlowControl (Settings &s, FlowControl v)
- MODBUS EXPORT void setSettingTimeoutFirstByte (Settings &s, uint32 t v)
- MODBUS EXPORT void setSettingTimeoutInterByte (Settings &s, uint32 t v)
- MODBUS EXPORT void setSettingBroadcastEnabled (Settings &s, bool v)
- Address addressFromString (const QString &s)
- template<class EnumType >

QString enumKey (int value)

template < class EnumType >

QString enumKey (EnumType value, const QString &byDef=QString())

• template<class EnumType >

EnumType enumValue (const QString &key, bool \*ok=nullptr, EnumType defaultValue=static\_cast< Enum $\leftarrow$  Type >(-1))

template < class EnumType >

EnumType enumValue (const QVariant &value, bool \*ok=nullptr, EnumType defaultValue=static\_cast < EnumType >(-1))

template < class EnumType >

EnumType enumValue (const QVariant &value, EnumType defaultValue)

template < class EnumType >

EnumType enumValue (const QVariant &value)

- MODBUS\_EXPORT ProtocolType toProtocolType (const QString &s, bool \*ok=nullptr)
- MODBUS EXPORT ProtocolType toProtocolType (const QVariant &v, bool \*ok=nullptr)
- MODBUS EXPORT int32 t toBaudRate (const QString &s, bool \*ok=nullptr)
- MODBUS\_EXPORT int32\_t toBaudRate (const QVariant &v, bool \*ok=nullptr)
- MODBUS EXPORT int8 t toDataBits (const QString &s, bool \*ok=nullptr)
- MODBUS EXPORT int8 t toDataBits (const QVariant &v, bool \*ok=nullptr)
- MODBUS\_EXPORT Parity toParity (const QString &s, bool \*ok=nullptr)
- MODBUS\_EXPORT Parity to Parity (const QVariant &v, bool \*ok=nullptr)
- MODBUS\_EXPORT StopBits toStopBits (const QString &s, bool \*ok=nullptr)
- MODBUS\_EXPORT StopBits toStopBits (const QVariant &v, bool \*ok=nullptr)
- MODBUS EXPORT FlowControl toFlowControl (const QString &s, bool \*ok=nullptr)
- MODBUS EXPORT FlowControl toFlowControl (const QVariant &v, bool \*ok=nullptr)
- MODBUS EXPORT QString toString (StatusCode v)
- MODBUS\_EXPORT QString toString (ProtocolType v)

- MODBUS\_EXPORT QString toString (Parity v)
- MODBUS\_EXPORT QString toString (StopBits v)
- MODBUS\_EXPORT QString toString (FlowControl v)
- QString bytesToString (const QByteArray &v)
- QString asciiToString (const QByteArray &v)
- MODBUS EXPORT QStringList availableSerialPortList ()
- MODBUS\_EXPORT ModbusPort \* createPort (const Settings &settings, bool blocking=false)
- MODBUS\_EXPORT ModbusClientPort \* createClientPort (const Settings &settings, bool blocking=false)
- MODBUS\_EXPORT ModbusServerPort \* createServerPort (ModbusInterface \*device, const Settings &settings, bool blocking=false)

## 6.1.1 Detailed Description

Main Modbus namespace. Contains classes, functions and constants to work with Modbus-protocol.

# 6.1.2 Enumeration Type Documentation

## 6.1.2.1 \_MemoryType

enum Modbus::\_MemoryType

Defines type of memory used in Modbus protocol.

#### Enumerator

Memory_Unknown	Invalid memory type.
Memory_0x	Memory allocated for coils/discrete outputs.
Memory_Coils	Same as Memory_0x.
Memory_1x	Memory allocated for discrete inputs.
Memory_DiscreteInputs	Same as Memory_1x.
Memory_3x	Memory allocated for analog inputs.
Memory_InputRegisters	Same as Memory_3x.
Memory_4x	Memory allocated for holding registers/analog outputs.
Memory_HoldingRegisters	Same as Memory_4x.

#### 6.1.2.2 Constants

enum Modbus::Constants

Define list of contants of Modbus protocol.

#### Enumerator

VALID_MODBUS_ADDRESS_BEGIN	Start of Modbus device address range according to specification.
VALID_MODBUS_ADDRESS_END	End of the Modbus protocol device address range according to the
	specification.
STANDARD_TCP_PORT	Standard TCP port of the Modbus protocol.

# 6.1.2.3 FlowControl

enum Modbus::FlowControl

FlowControl Parity for serial port.

## Enumerator

NoFlowControl	No flow control.
HardwareControl	Hardware flow control (RTS/CTS).
SoftwareControl	Software flow control (XON/XOFF).

# 6.1.2.4 Parity

enum Modbus::Parity

Defines Parity for serial port.

## Enumerator

NoParity	No parity bit it sent. This is the most common parity setting.
EvenParity	The number of 1 bits in each character, including the parity bit, is always even.
OddParity	The number of 1 bits in each character, including the parity bit, is always odd. It ensures that at least one state transition occurs in each character.
SpaceParity	Space parity. The parity bit is sent in the space signal condition. It does not provide error detection information.
MarkParity	Mark parity. The parity bit is always set to the mark signal condition (logical 1). It does not provide error detection information.

# 6.1.2.5 ProtocolType

enum Modbus::ProtocolType

Defines type of Modbus protocol.

## Enumerator

ASC	ASCII version of Modbus communication protocol.
RTU	RTU version of Modbus communication protocol.
TCP	TCP version of Modbus communication protocol.

## 6.1.2.6 StatusCode

enum Modbus::StatusCode

Defines status of executed Modbus functions.

## Enumerator

Status_Processing	The operation is not complete. Further operation is required.
Status_Good	Successful result.
Status_Bad	Error. General.
Status_Uncertain	The status is undefined.

# Enumerator

Status_BadIllegalFunction	Standard error. The feature is not supported.
Status_BadIllegalDataAddress	Standard error. The leature is not supported.  Standard error. Invalid data address.
Status_BadIllegalDataValue	Standard error. Invalid data address.  Standard error. Invalid data value.
Status_BadServerDeviceFailure	Standard error. Failure during a specified operation.
Status_BadAcknowledge	Standard error. The server has accepted the request and is processing it, but it will take a long time.
Status_BadServerDeviceBusy	Standard error. The server is busy processing a long
Glatus_bauderverbevicebusy	command. The request must be repeated later.
Status_BadNegativeAcknowledge	Standard error. The programming function cannot be
	performed.
Status_BadMemoryParityError	Standard error. The server attempted to read a record
	file but detected a parity error in memory.
Status_BadGatewayPathUnavailable	Standard error. Indicates that the gateway was unable
	to allocate an internal communication path from the input port o the output port for processing the request.
	Usually means that the gateway is misconfigured or
	overloaded.
Status_BadGatewayTargetDeviceFailedToRespond	Standard error. Indicates that no response was
	obtained from the target device. Usually means that
	the device is not present on the network.
Status_BadEmptyResponse	Error. Empty request/response body.
Status_BadNotCorrectRequest	Error. Invalid request.
Status_BadNotCorrectResponse	Error. Invalid response.
Status_BadWriteBufferOverflow	Error. Write buffer overflow.
Status_BadReadBufferOverflow	Error. Request receive buffer overflow.
Status_BadSerialOpen	Error. Serial port cannot be opened.
Status_BadSerialWrite	Error. Cannot send a parcel to the serial port.
Status_BadSerialRead	Error. Reading the serial port (timeout)
Status_BadSerialReadTimeout	Error. Reading the serial port (timeout)
Status_BadSerialWriteTimeout	Error. Writing the serial port (timeout)
Status_BadAscMissColon	Error (ASC). Missing packet start character ':'.
Status_BadAscMissCrLf	Error (ASC). '\r\n' end of packet character missing.
Status_BadAscChar	Error (ASC). Invalid ASCII character.
Status_BadLrc	Error (ASC). Invalid checksum.
Status_BadCrc	Error (RTU). Wrong checksum.
Status_BadTcpCreate	Error. Unable to create a TCP socket.
Status_BadTcpConnect	Error. Unable to create a TCP connection.
Status_BadTcpWrite	Error. Unable to send a TCP packet.
Status_BadTcpRead	Error. Unable to receive a TCP packet.
Status_BadTcpBind	Error. Unable to bind a TCP socket (server side)
Status_BadTcpListen	Error. Unable to listen a TCP socket (server side)
Status_BadTcpAccept	Error. Unable accept bind a TCP socket (server side)
Status_BadTcpDisconnect	Error. Bad disconnection result.

# 6.1.2.7 StopBits

enum Modbus::StopBits

Defines Stop Bits for serial port.

#### Enumerator

OneStop	1 stop bit.
OneAndHalfStop	1.5 stop bit.
TwoStop	2 stop bits.

#### 6.1.3 Function Documentation

#### 6.1.3.1 addressFromString()

Convert String repr to Modbus::Address

## 6.1.3.2 asciiToBytes()

Function converts ASCII repr asciiBuff to binary byte array. Every byte of output bytesBuff are repr as two bytes in asciiBuff, where most signified tetrabits represented as leading byte in hex digit in ASCII encoding (upper) and less signified tetrabits represented as tailing byte in hex digit in ASCII encoding (upper). count is a size of input array asciiBuff.

Note

Output array bytesBuff must be at least twice smaller than input array asciiBuff.

## Returns

Returns size of bytesBuff in bytes which calc as {output = count / 2}

#### 6.1.3.3 asciiToString() [1/2]

```
QString Modbus::asciiToString ( {\tt const\ QByteArray\ \&\ v)} \quad [{\tt inline}]
```

Make string representation of ASCII array and separate bytes by space

## 6.1.3.4 asciiToString() [2/2]

Make string representation of ASCII array and separate bytes by space

## 6.1.3.5 availableBaudRate()

```
{\tt MODBUS\_EXPORT\ List<\ int32\_t\ >\ Modbus::} available {\tt BaudRate\ ()}
```

Return list of baud rates

## 6.1.3.6 availableDataBits()

```
MODBUS_EXPORT List< int8_t > Modbus::availableDataBits ()
```

Return list of data bits

## 6.1.3.7 availableFlowControl()

```
MODBUS_EXPORT List< FlowControl > Modbus::availableFlowControl ()
```

Return list of FlowControl values

## 6.1.3.8 availableParity()

```
MODBUS_EXPORT List< Parity > Modbus::availableParity ()
```

Return list of Parity values

## 6.1.3.9 availableSerialPortList()

```
MODBUS_EXPORT QStringList Modbus::availableSerialPortList ()
```

Returns list of string that represent names of serial ports

## 6.1.3.10 availableSerialPorts()

```
MODBUS_EXPORT List< String > Modbus::availableSerialPorts ()
```

Return list of names of available serial ports

## 6.1.3.11 availableStopBits()

```
MODBUS_EXPORT List< StopBits > Modbus::availableStopBits ()
```

Return list of StopBits values

#### 6.1.3.12 bytesToAscii()

Function converts byte array <code>bytesBuff</code> to ASCII repr of byte array. Every byte of <code>bytesBuff</code> are repr as two bytes in <code>asciiBuff</code>, where most signified tetrabits represented as leading byte in hex digit in ASCII encoding (upper) and less signified tetrabits represented as tailing byte in hex digit in ASCII encoding (upper). <code>count</code> is count bytes of <code>bytesBuff</code>.

Note

Output array asciiBuff must be at least twice bigger than input array bytesBuff.

#### Returns

Returns size of asciiBuff in bytes which calc as {output = count \* 2}

## 6.1.3.13 bytesToString() [1/2]

Make string representation of bytes array and separate bytes by space

# 6.1.3.14 bytesToString() [2/2]

Make string representation of bytes array and separate bytes by space

## 6.1.3.15 crc16()

CRC16 checksum hash function (for Modbus RTU).

#### Returns

Returns a 16-bit unsigned integer value of the checksum

#### 6.1.3.16 createClientPort() [1/2]

Same as Modbus::createClientPort(ProtocolType type, const void \*settings, bool blocking) but ProtocolType type and const void \*settings are defined by Modbus::Settings key-value map.

## 6.1.3.17 createClientPort() [2/2]

Function for creation ModbusClientPort with defined parameters:

#### **Parameters**

in	type	Protocol type: TCP, RTU, ASC.
in	settings	For TCP must be pointer: TcpSettings*, SerialSettings* otherwise.
in	blocking	If true blocking will be set, non blocking otherwise.

# 6.1.3.18 createPort() [1/2]

Same as Modbus::createPort(ProtocolType type, const void \*settings, bool blocking) but ProtocolType type and const void \*settings are defined by Modbus::Settings key-value map.

## 6.1.3.19 createPort() [2/2]

Function for creation ModbusPort with defined parameters:

#### **Parameters**

	in	type	Protocol type: TCP, RTU, ASC.
Ī	in	settings	For TCP must be pointer: TcpSettings*, SerialSettings* otherwise.
Ī	in	blocking	If true blocking will be set, non blocking otherwise.

#### 6.1.3.20 createServerPort() [1/2]

Same as Modbus::createServerPort(ProtocolType type, const void \*settings, bool blocking) butProtocolType type and const void \*settings are defined by Modbus::Settings key-value map.

#### 6.1.3.21 createServerPort() [2/2]

Function for creation ModbusServerPort with defined parameters:

#### **Parameters**

in	device	Pointer to the ModbusInterface implementation to which all requests for Modbus
		functions are forwarded.
in	type	Protocol type: TCP, RTU, ASC.
in	settings	For TCP must be pointer: TcpSettings*, SerialSettings* otherwise.
in	blocking	If true blocking will be set, non blocking otherwise.

## 6.1.3.22 currentTimestamp()

```
MODBUS_EXPORT Timestamp Modbus::currentTimestamp ()
```

Get current timestamp in UNIX format in milliseconds.

## 6.1.3.23 enumKey() [1/2]

Convert value to QString key for type

## 6.1.3.24 enumKey() [2/2]

Convert value to QString key for type

#### 6.1.3.25 enumValue() [1/4]

Convert key to value for enumeration by QString key

## 6.1.3.26 enumValue() [2/4]

Convert QVariant value to enumeration value (int - value, string - key).

#### 6.1.3.27 enumValue() [3/4]

Convert QVariant value to enumeration value (int - value, string - key). Stores result of convertion in output parameter ok. If value can't be converted, defaultValue is returned.

#### 6.1.3.28 enumValue() [4/4]

Convert QVariant value to enumeration value (int - value, string - key). If value can't be converted,  $default \leftarrow Value$  is returned.

## 6.1.3.29 getBit()

Returns the value of the bit with number 'bitNum' from the bit array 'bitBuff'.

#### 6.1.3.30 getBitS()

Returns the value of the bit with the number 'bitNum' from the bit array 'bitBuff', if the bit number is greater than or equal to 'maxBitCount', then 'false' is returned.

## 6.1.3.31 getBits()

Gets the values of bits with number bitNum and count bitCount from the bit array bitBuff and stores their values in the boolean array boolBuff, where the value of each bit is stored as a separate bool value.

#### Returns

A pointer to the boolBuff array.

## 6.1.3.32 getBitsS()

Similar to the Modbus::getBits(const void\*, uint16\_t, uint16\_t, bool\*) function, but it is controlled that the size does not exceed the maximum number of bits maxBitCount.

## Returns

A pointer to the boolBuff array.

# 6.1.3.33 getLastErrorText()

```
MODBUS_EXPORT String Modbus::getLastErrorText ()
```

Returns string representation of the last error

#### 6.1.3.34 getSettingBaudRate()

```
MODBUS_EXPORT int32_t Modbus::getSettingBaudRate ( const Settings & s, bool * ok = nullptr)
```

Get settings value for the serial port's baud rate. If value can't be retrieved that default value is returned and \*ok = false (if provided).

#### 6.1.3.35 getSettingBroadcastEnabled()

Get settings value for the serial port enables broadcast mode for 0 unit address. If value can't be retrieved that default value is returned and \*ok = false (if provided).

#### 6.1.3.36 getSettingDataBits()

Get settings value for the serial port's data bits. If value can't be retrieved that default value is returned and \*ok = false (if provided).

#### 6.1.3.37 getSettingFlowControl()

```
\label{eq:modbus} $$ MODBUS\_EXPORT FlowControl Modbus::getSettingFlowControl ( const Settings & s, $$ bool * ok = nullptr)$
```

Get settings value for the serial port's flow control. If value can't be retrieved that default value is returned and \*ok = false (if provided).

## 6.1.3.38 getSettingHost()

Get settings value for the IP address or DNS name of the remote device. If value can't be retrieved that default value is returned and \*ok = false (if provided).

# 6.1.3.39 getSettingParity()

Get settings value for the serial port's parity. If value can't be retrieved that default value is returned and \*ok = false (if provided).

#### 6.1.3.40 getSettingPort()

Get settings value for the TCP port of the remote device. If value can't be retrieved that default value is returned and \*ok = false (if provided).

#### 6.1.3.41 getSettingSerialPortName()

```
\begin{tabular}{ll} {\tt MODBUS\_EXPORT~QString~Modbus::getSettingSerialPortName~(}\\ &const~Settings~\&~s,\\ &bool~*ok~=~nullptr) \end{tabular}
```

Get settings value for the serial port name. If value can't be retrieved that default value is returned and \*ok = false (if provided).

#### 6.1.3.42 getSettingStopBits()

Get settings value for the serial port's stop bits. If value can't be retrieved that default value is returned and \*ok = false (if provided).

#### 6.1.3.43 getSettingTimeout()

```
MODBUS_EXPORT uint32_t Modbus::getSettingTimeout ( const Settings & s, bool * ok = nullptr)
```

Get settings value for connection timeout (milliseconds). If value can't be retrieved that default value is returned and \*ok = false (if provided).

## 6.1.3.44 getSettingTimeoutFirstByte()

```
MODBUS_EXPORT uint32_t Modbus::getSettingTimeoutFirstByte ( const Settings & s, bool * ok = nullptr)
```

Get settings value for the serial port's timeout waiting first byte of packet. If value can't be retrieved that default value is returned and \*ok = false (if provided).

# 6.1.3.45 getSettingTimeoutInterByte()

Get settings value for the serial port's timeout waiting next byte of packet. If value can't be retrieved that default value is returned and \*ok = false (if provided).

#### 6.1.3.46 getSettingTries()

Get settings value for number of tries a Modbus request is repeated if it fails. If value can't be retrieved that default value is returned and \*ok = false (if provided).

#### 6.1.3.47 getSettingType()

Get settings value for the type of Modbus protocol. If value can't be retrieved that default value is returned and \*ok = false (if provided).

#### 6.1.3.48 getSettingUnit()

Get settings value for the unit number of remote device. If value can't be retrieved that default value is returned and \*ok = false (if provided).

# 6.1.3.49 Irc()

LRC checksum hash function (for Modbus ASCII).

Returns

Returns an 8-bit unsigned integer value of the checksum

#### 6.1.3.50 modbusLibVersion()

```
MODBUS_EXPORT uint32_t Modbus::modbusLibVersion ()
```

Returns version of current lib like (major << 16) + (minor << 8) + patch.

# 6.1.3.51 modbusLibVersionStr()

```
MODBUS_EXPORT const Char * Modbus::modbusLibVersionStr ()
```

Returns version of current lib as string constant pointer like "major.minor.patch".

#### 6.1.3.52 msleep()

Make current thread sleep with 'msec' milliseconds.

## 6.1.3.53 readMemBits() [1/2]

```
StatusCode Modbus::readMemBits (
          uint32_t offset,
          uint32_t count,
          void * values,
          const void * memBuff,
          uint32_t memBitCount) [inline]
```

Overloaded function

## 6.1.3.54 readMemBits() [2/2]

```
MODBUS_EXPORT StatusCode Modbus::readMemBits (
    uint32_t offset,
    uint32_t count,
    void * values,
    const void * memBuff,
    uint32_t memBitCount,
    uint32_t * outCount)
```

Function for copy (read) values from memory input memBuff and put it to the output buffer values for discretes (bits):

## Parameters

in	offset	Memory offset to read from memBuff in bit size.
in	count	Count of bits to read from memory memBuff.
out	values	Output buffer to store data.
in	memBuff	Pointer to the memory which holds data.
in	memBitCount	Size of memory buffer memBuff in bits.
out	outCount	Optional, can be NULL. If specified, then if the requested amount of memory exceeds the limits of this memory, the error is not returned, and the amount of memory read is reduced to the memory limits and this new amount is returned in outCount

## 6.1.3.55 readMemRegs() [1/2]

```
StatusCode Modbus::readMemRegs (
          uint32_t offset,
          uint32_t count,
          void * values,
          const void * memBuff,
          uint32_t memRegCount) [inline]
```

## Overloaded function

#### 6.1.3.56 readMemRegs() [2/2]

```
MODBUS_EXPORT StatusCode Modbus::readMemRegs (
    uint32_t offset,
    uint32_t count,
    void * values,
    const void * memBuff,
    uint32_t memRegCount,
    uint32_t * outCount)
```

Function for copy (read) values from memory input memBuff and put it to the output buffer values for 16 bit registers:

#### **Parameters**

in	offset	Memory offset to read from memBuff in 16-bit registers size.
in	count	Count of 16-bit registers to read from memory memBuff.
out	values	Output buffer to store data.
in	memBuff	Pointer to the memory which holds data.
in	memRegCount	Size of memory buffer memBuff in 16-bit registers.
out	outCount	Optional, can be NULL. If specified, then if the requested amount of memory exceeds the limits of this memory, the error is not returned, and the amount of memory read is reduced to the memory limits and this new amount is returned in outCount

## 6.1.3.57 sascii()

Make string representation of ASCII array and separate bytes by space

## 6.1.3.58 sbytes()

Make string representation of bytes array and separate bytes by space

# 6.1.3.59 setBit()

Sets the value of the bit with the number 'bitNum' to the bit array 'bitBuff'.

#### 6.1.3.60 setBitS()

Sets the value of the bit with the number 'bitNum' to the bit array 'bitBuff', controlling the size of the array 'maxBit ← Count' in bits.

## 6.1.3.61 setBits()

Sets the values of the bits in the bitBuff array starting with the number bitNum and the count bitCount from the boolBuff array, where the value of each bit is stored as a separate bool value.

#### Returns

A pointer to the bitBuff array.

# 6.1.3.62 setBitsS()

Similar to the Modbus::setBits(void\*,uint16\_t,uint16\_t,const bool\*) function, but it is controlled that the size does not exceed the maximum number of bits maxBitCount.

#### Returns

A pointer to the bitBuff array.

## 6.1.3.63 setSettingBaudRate()

Set settings value for the serial port's baud rate.

## 6.1.3.64 setSettingBroadcastEnabled()

```
MODBUS_EXPORT void Modbus::setSettingBroadcastEnabled ( Settings & s, bool v)
```

Set settings value for the serial port enables broadcast mode for 0 unit address.

## 6.1.3.65 setSettingDataBits()

```
MODBUS_EXPORT void Modbus::setSettingDataBits ( Settings & s, int8_t v)
```

Set settings value for the serial port's data bits.

## 6.1.3.66 setSettingFlowControl()

Set settings value for the serial port's flow control.

## 6.1.3.67 setSettingHost()

Set settings value for the IP address or DNS name of the remote device.

## 6.1.3.68 setSettingParity()

```
MODBUS_EXPORT void Modbus::setSettingParity ( Settings & s, Parity v)
```

Set settings value for the serial port's parity.

#### 6.1.3.69 setSettingPort()

Set settings value for the TCP port number of the remote device.

#### 6.1.3.70 setSettingSerialPortName()

```
MODBUS_EXPORT void Modbus::setSettingSerialPortName ( Settings & s, const QString & v)
```

Set settings value for the serial port name.

## 6.1.3.71 setSettingStopBits()

```
MODBUS_EXPORT void Modbus::setSettingStopBits ( Settings & s, StopBits v)
```

Set settings value for the serial port's stop bits.

## 6.1.3.72 setSettingTimeout()

```
MODBUS_EXPORT void Modbus::setSettingTimeout ( Settings & s, uint32_t v)
```

Set settings value for connection timeout (milliseconds).

## 6.1.3.73 setSettingTimeoutFirstByte()

```
MODBUS_EXPORT void Modbus::setSettingTimeoutFirstByte ( Settings & s, uint32_t v)
```

Set settings value for the serial port's timeout waiting first byte of packet.

## 6.1.3.74 setSettingTimeoutInterByte()

```
MODBUS_EXPORT void Modbus::setSettingTimeoutInterByte ( Settings & s, uint32_t v)
```

Set settings value for the serial port's timeout waiting next byte of packet.

#### 6.1.3.75 setSettingTries()

```
MODBUS_EXPORT void Modbus::setSettingTries ( Settings & s, uint32_t )
```

Set settings value for number of tries a Modbus request is repeated if it fails.

#### 6.1.3.76 setSettingType()

Set settings value the type of Modbus protocol.

## 6.1.3.77 setSettingUnit()

Set settings value for the unit number of remote device.

#### 6.1.3.78 sflowControl()

Returns pointer to constant string value that represent name of the FlowControl parameter or nullptr (NULL) if the value is invalid.

#### 6.1.3.79 sparity()

Returns pointer to constant string value that represent name of the Parity value or nullptr (NULL) if the value is invalid.

## 6.1.3.80 sprotocolType()

Returns pointer to constant string value that represent name of the ProtocolType value or nullptr (NULL) if the value is invalid.

#### 6.1.3.81 sstopBits()

Returns pointer to constant string value that represent name of the StopBits value or nullptr (NULL) if the value is invalid.

#### 6.1.3.82 StatusIsBad()

Returns a general indication that the operation result is unsuccessful.

## 6.1.3.83 StatusIsGood()

Returns a general indication that the operation result is successful.

## 6.1.3.84 StatusIsProcessing()

Returns a general indication that the result of the operation is incomplete.

#### 6.1.3.85 StatusIsStandardError()

Returns a general sign that the result is standard error.

## 6.1.3.86 StatusIsUncertain()

Returns a general sign that the result of the operation is undefined.

#### 6.1.3.87 timer()

```
MODBUS_EXPORT Timer Modbus::timer ()
```

Get timer value in milliseconds.

## 6.1.3.88 toBaudRate() [1/2]

Converts string representation to BaudRate value. If ok is not nullptr, failure is reported by setting \*ok to false, and success by setting \*ok to true.

#### 6.1.3.89 toBaudRate() [2/2]

Converts QVariant value to DataBits value. If ok is not nullptr, failure is reported by setting \*ok to false, and success by setting \*ok to true.

## 6.1.3.90 toDataBits() [1/2]

```
\label{eq:Modbus:toDataBits} \begin{tabular}{ll} Modbus::toDataBits ( \\ const QString & s, \\ bool * ok = nullptr) \end{tabular}
```

Converts string representation to DataBits value. If ok is not nullptr, failure is reported by setting \*ok to false, and success by setting \*ok to true.

#### 6.1.3.91 toDataBits() [2/2]

Converts QVariant value to DataBits value. If ok is not nullptr, failure is reported by setting \*ok to false, and success by setting \*ok to true.

#### 6.1.3.92 toFlowControl() [1/2]

Converts string representation to FlowControl enum value. If ok is not nullptr, failure is reported by setting \*ok to false, and success by setting \*ok to true.

#### 6.1.3.93 toFlowControl() [2/2]

Converts QVariant value to FlowControl enum value. If ok is not nullptr, failure is reported by setting \*ok to false, and success by setting \*ok to true.

#### 6.1.3.94 toModbusOffset()

Function extract only offset part from Modbus address and returns it.

#### 6.1.3.95 toModbusString()

Convert interger value to Modbus::String

Returns

Returns new Modbus::String value

#### 6.1.3.96 toParity() [1/2]

Converts string representation to Parity enum value. If ok is not nullptr, failure is reported by setting \*ok to false, and success by setting \*ok to true.

## 6.1.3.97 toParity() [2/2]

Converts QVariant value to Parity enum value. If ok is not nullptr, failure is reported by setting \*ok to false, and success by setting \*ok to true.

# 6.1.3.98 toProtocolType() [1/2]

```
MODBUS_EXPORT ProtocolType Modbus::toProtocolType ( const QString & s, bool * ok = nullptr)
```

Converts string representation to ProtocolType enum value. If ok is not nullptr, failure is reported by setting \*ok to false, and success by setting \*ok to true.

## 6.1.3.99 toProtocolType() [2/2]

```
MODBUS_EXPORT ProtocolType Modbus::toProtocolType ( const QVariant & v, bool * ok = nullptr)
```

Converts QVariant value to ProtocolType enum value. If ok is not nullptr, failure is reported by setting \*ok to false, and success by setting \*ok to true.

## 6.1.3.100 toStopBits() [1/2]

Converts string representation to StopBits enum value. If ok is not nullptr, failure is reported by setting \*ok to false, and success by setting \*ok to true.

#### 6.1.3.101 toStopBits() [2/2]

```
MODBUS_EXPORT StopBits Modbus::toStopBits ( const QVariant & v, bool * ok = nullptr)
```

Converts QVariant value to StopBits enum value. If ok is not nullptr, failure is reported by setting \*ok to false, and success by setting \*ok to true.

## 6.1.3.102 toString() [1/5]

```
\begin{tabular}{ll} {\tt MODBUS\_EXPORT} & {\tt QString} & {\tt Modbus::toString} & (\\ & & {\tt FlowControl} & v) \end{tabular}
```

Returns string representation of FlowControl enum value

## 6.1.3.103 toString() [2/5]

Returns string representation of Parity enum value

## 6.1.3.104 toString() [3/5]

```
\begin{tabular}{ll} {\tt MODBUS\_EXPORT~QString~Modbus::} {\tt toString~(} \\ & {\tt ProtocolType~v)} \end{tabular}
```

Returns string representation of ProtocolType enum value

# 6.1.3.105 toString() [4/5]

```
MODBUS_EXPORT QString Modbus::toString ( StatusCode \ v)
```

Returns string representation of StatusCode enum value

## 6.1.3.106 toString() [5/5]

```
MODBUS_EXPORT QString Modbus::toString ( StopBits v)
```

Returns string representation of StopBits enum value

## 6.1.3.107 writeMemBits() [1/2]

```
StatusCode Modbus::writeMemBits (
        uint32_t offset,
        uint32_t count,
        const void * values,
        void * memBuff,
        uint32_t memBitCount) [inline]
```

Overloaded function

## 6.1.3.108 writeMemBits() [2/2]

```
MODBUS_EXPORT StatusCode Modbus::writeMemBits (
    uint32_t offset,
    uint32_t count,
    const void * values,
    void * memBuff,
    uint32_t memBitCount,
    uint32_t * outCount)
```

Function for copy (write) values from input buffer values to memory memBuff for discretes (bits):

#### **Parameters**

in	offset	Memory offset to write to memBuff in bit size.
in	count	Count of bits to write into memory memBuff.
out	values	Input buffer that holds data to write.
in	memBuff	Pointer to the memory buffer.
in	memBitCount	Size of memory buffer memBuff in bits.
out	outCount	Optional, can be NULL. If specified, then if the requested amount of memory exceeds the limits of this memory, the error is not returned, and the amount of memory write is reduced to the memory limits and this new amount is returned in outCount

## 6.1.3.109 writeMemRegs() [1/2]

```
StatusCode Modbus::writeMemRegs (
          uint32_t offset,
          uint32_t count,
          const void * values,
          void * memBuff,
          uint32_t memRegCount) [inline]
```

## Overloaded function

# 6.1.3.110 writeMemRegs() [2/2]

```
MODBUS_EXPORT StatusCode Modbus::writeMemRegs (
            uint32_t offset,
            uint32_t count,
            const void * values,
            void * memBuff,
            uint32_t memRegCount,
            uint32_t * outCount)
```

Function for copy (write) values from input buffer values to memory memBuff for 16 bit registers:

## **Parameters**

in	offset	Memory offset to write to memBuff in 16-bit registers size.
in	count	Count of 16-bit registers to write into memory memBuff.
out	values	Input buffer that holds data to write.
in	memBuff	Pointer to the memory buffer.
in	memRegCount	Size of memory buffer memBuff in 16-bit registers.
out	outCount	Optional, can be NULL. If specified, then if the requested amount of memory exceeds the limits of this memory, the error is not returned, and the amount of memory write is reduced to the memory limits and this new amount is returned in outCount

# **Chapter 7**

# **Class Documentation**

# 7.1 Modbus::Address Class Reference

Class for convinient manipulation with Modbus Data Address.

```
#include <ModbusQt.h>
```

#### **Public Member Functions**

- Address ()
- Address (Modbus::MemoryType, quint16 offset)
- Address (quint32 adr)
- bool isValid () const
- MemoryType type () const
- quint16 offset () const
- quint32 number () const
- QString toString () const
- operator quint32 () const
- Address & operator= (quint32 v)

# 7.1.1 Detailed Description

Class for convinient manipulation with Modbus Data Address.

## 7.1.2 Constructor & Destructor Documentation

## 7.1.2.1 Address() [1/3]

```
Modbus::Address::Address ()
```

Defauilt constructor ot the class. Creates invalid Modbus Data Address

48 Class Documentation

## 7.1.2.2 Address() [2/3]

Constructor of the class. E.g. Address (Modbus::Memory\_4x, 0) creates 400001 standard address.

#### 7.1.2.3 Address() [3/3]

Constructor of the class. E.g. Address(400001) creates Address with type  $Modbus::Memory_4x$  and offset 0, and Address(1) creates Address with type  $Modbus::Memory_0x$  and offset 0.

## 7.1.3 Member Function Documentation

#### 7.1.3.1 isValid()

```
bool Modbus::Address::isValid () const [inline]
```

Returns true if memory type is Modbus:: Memory\_Unknown, false otherwise

## 7.1.3.2 number()

```
quint32 Modbus::Address::number () const [inline]
```

Returns memory number (offset+1) of Modbus Data Address

# 7.1.3.3 offset()

```
quint16 Modbus::Address::offset () const [inline]
```

Returns memory offset of Modbus Data Address

#### 7.1.3.4 operator quint32()

```
Modbus::Address::operator quint32 () const [inline]
```

Converts current Modbus Data Address to quint 32, e.g. Address (Modbus::Memory\_4x, 0) will be converted to 400001.

## 7.1.3.5 operator=()

```
Address & Modbus::Address::operator= ( quint32 v)
```

Assigment operator definition.

## 7.1.3.6 toString()

```
QString Modbus::Address::toString () const
```

Returns string repr of Modbus Data Address e.g. Address (Modbus::Memory\_4x, 0) will be converted to QString ("400001").

## 7.1.3.7 type()

```
MemoryType Modbus::Address::type () const [inline]
```

Returns memory type of Modbus Data Address

The documentation for this class was generated from the following file:

• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusQt.h

# 7.2 Modbus::Defaults Class Reference

Holds the default values of the settings.

```
#include <ModbusQt.h>
```

#### **Public Member Functions**

• Defaults ()

# **Static Public Member Functions**

• static const Defaults & instance ()

50 Class Documentation

#### **Public Attributes**

· const uint8\_t unit

Default value for the unit number of remote device.

const ProtocolType type

Default value for the type of Modbus protocol.

· const uint32\_t tries

Default value for number of tries a Modbus request is repeated if it fails.

· const QString host

Default value for the IP address or DNS name of the remote device.

const uint16\_t port

Default value for the TCP port number of the remote device.

• const uint32\_t timeout

Default value for connection timeout (milliseconds)

• const QString serialPortName

Default value for the serial port name.

const int32\_t baudRate

Default value for the serial port's baud rate.

· const int8 t dataBits

Default value for the serial port's data bits.

· const Parity parity

Default value for the serial port's parity.

• const StopBits stopBits

Default value for the serial port's stop bits.

const FlowControl flowControl

Default value for the serial port's flow control.

const uint32 t timeoutFirstByte

Default value for the serial port's timeout waiting first byte of packet.

const uint32\_t timeoutInterByte

Default value for the serial port's timeout waiting next byte of packet.

· const bool isBroadcastEnabled

Default value for the serial port enables broadcast mode for  ${\it 0}$  unit address.

## 7.2.1 Detailed Description

Holds the default values of the settings.

## 7.2.2 Constructor & Destructor Documentation

## 7.2.2.1 Defaults()

Modbus::Defaults::Defaults ()

Constructor ot the class.

#### 7.2.3 Member Function Documentation

#### 7.2.3.1 instance()

```
static const Defaults & Modbus::Defaults::instance () [static]
```

Returns a reference to the global Modbus::Defaults object.

The documentation for this class was generated from the following file:

• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusQt.h

## 7.3 ModbusSerialPort::Defaults Struct Reference

Holds the default values of the settings.

```
#include <ModbusSerialPort.h>
```

#### **Public Member Functions**

· Defaults ()

## **Static Public Member Functions**

• static const Defaults & instance ()

#### **Public Attributes**

const Modbus::Char \* portName

Default value for the serial port name.

const int32\_t baudRate

Default value for the serial port's baud rate.

const int8\_t dataBits

Default value for the serial port's data bits.

const Modbus::Parity parity

Default value for the serial port's patiry.

const Modbus::StopBits stopBits

Default value for the serial port's stop bits.

const Modbus::FlowControl flowControl

Default value for the serial port's flow control.

• const uint32\_t timeoutFirstByte

Default value for the serial port's timeout waiting first byte of packet.

• const uint32\_t timeoutInterByte

Default value for the serial port's timeout waiting next byte of packet.

52 Class Documentation

## 7.3.1 Detailed Description

Holds the default values of the settings.

## 7.3.2 Constructor & Destructor Documentation

#### 7.3.2.1 Defaults()

```
ModbusSerialPort::Defaults::Defaults ()
```

Constructor of the class.

#### 7.3.3 Member Function Documentation

#### 7.3.3.1 instance()

```
static const Defaults & ModbusSerialPort::Defaults::instance () [static]
```

Returns a reference to the global ModbusSerialPort::Defaults object.

The documentation for this struct was generated from the following file:

• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusSerialPort.h

# 7.4 ModbusTcpPort::Defaults Struct Reference

Defaults class constain default settings values for ModbusTcpPort.

```
#include <ModbusTcpPort.h>
```

## **Public Member Functions**

• Defaults ()

#### **Static Public Member Functions**

• static const Defaults & instance ()

#### **Public Attributes**

const Modbus::Char \* host

Default setting 'TCP host name (DNS or IP address)'.

const uint16\_t port

Default setting 'TCP port number' for the listening server.

• const uint32\_t timeout

Default setting for the read timeout of every single conncetion.

# 7.4.1 Detailed Description

Defaults class constain default settings values for ModbusTcpPort.

# 7.4.2 Constructor & Destructor Documentation

#### 7.4.2.1 Defaults()

```
ModbusTcpPort::Defaults::Defaults ()
```

Constructor of the class.

# 7.4.3 Member Function Documentation

# 7.4.3.1 instance()

```
static const Defaults & ModbusTcpPort::Defaults::instance () [static]
```

Returns a reference to the global default value object.

The documentation for this struct was generated from the following file:

• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusTcpPort.h

# 7.5 ModbusTcpServer::Defaults Struct Reference

Defaults class constain default settings values for ModbusTcpServer.

```
#include <ModbusTcpServer.h>
```

# **Public Member Functions**

• Defaults ()

### **Static Public Member Functions**

• static const Defaults & instance ()

# **Public Attributes**

const uint16\_t port

Default setting 'TCP port number' for the listening server.

• const uint32\_t timeout

Default setting for the read timeout of every single conncetion.

# 7.5.1 Detailed Description

Defaults class constain default settings values for ModbusTcpServer.

# 7.5.2 Constructor & Destructor Documentation

# 7.5.2.1 Defaults()

```
ModbusTcpServer::Defaults::Defaults ()
```

Constructor of the class.

# 7.5.3 Member Function Documentation

### 7.5.3.1 instance()

```
static const Defaults & ModbusTcpServer::Defaults::instance () [static]
```

Returns a reference to the global default value object.

The documentation for this struct was generated from the following file:

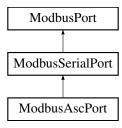
• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusTcpServer.h

# 7.6 ModbusAscPort Class Reference

Implements ASCII version of the Modbus communication protocol.

```
#include <ModbusAscPort.h>
```

Inheritance diagram for ModbusAscPort:



#### **Public Member Functions**

- ModbusAscPort (bool blocking=false)
- $\sim$ ModbusAscPort ()
- Modbus::ProtocolType type () const override

#### Public Member Functions inherited from ModbusSerialPort

- ∼ModbusSerialPort ()
- Modbus::Handle handle () const override
- Modbus::StatusCode open () override
- · Modbus::StatusCode close () override
- bool isOpen () const override
- const Modbus::Char \* portName () const
- void setPortName (const Modbus::Char \*portName)
- int32\_t baudRate () const
- void setBaudRate (int32 t baudRate)
- int8\_t dataBits () const
- void setDataBits (int8 t dataBits)
- Modbus::Parity parity () const
- void setParity (Modbus::Parity parity)
- · Modbus::StopBits stopBits () const
- void setStopBits (Modbus::StopBits stopBits)
- Modbus::FlowControl flowControl () const
- void setFlowControl (Modbus::FlowControl flowControl)
- uint32\_t timeoutFirstByte () const
- void setTimeoutFirstByte (uint32\_t timeout)
- uint32 t timeoutInterByte () const
- void setTimeoutInterByte (uint32\_t timeout)
- const uint8\_t \* readBufferData () const override
- uint16\_t readBufferSize () const override
- const uint8\_t \* writeBufferData () const override
- uint16\_t writeBufferSize () const override

### Public Member Functions inherited from ModbusPort

- virtual ∼ModbusPort ()
- virtual void setNextRequestRepeated (bool v)
- bool isChanged () const
- bool isServerMode () const
- virtual void setServerMode (bool mode)
- bool isBlocking () const
- bool isNonBlocking () const
- uint32 t timeout () const
- · void setTimeout (uint32\_t timeout)
- Modbus::StatusCode lastErrorStatus () const
- const Modbus::Char \* lastErrorText () const

#### **Protected Member Functions**

- Modbus::StatusCode writeBuffer (uint8\_t unit, uint8\_t func, uint8\_t \*buff, uint16\_t szInBuff) override
- Modbus::StatusCode readBuffer (uint8\_t &unit, uint8\_t &func, uint8\_t \*buff, uint16\_t maxSzBuff, uint16\_t \*szOutBuff) override

#### Protected Member Functions inherited from ModbusSerialPort

- Modbus::StatusCode write () override
- Modbus::StatusCode read () override

# Protected Member Functions inherited from ModbusPort

Modbus::StatusCode setError (Modbus::StatusCode status, const Modbus::Char \*text)

# 7.6.1 Detailed Description

Implements ASCII version of the Modbus communication protocol.

 ${\tt ModbusAscPort\ derived\ from\ ModbusSerialPort\ and\ implements\ write} Buffer\ and\ read Buffer\ for\ ASCII\ version\ of\ Modbus\ communication\ protocol.}$ 

# 7.6.2 Constructor & Destructor Documentation

### 7.6.2.1 ModbusAscPort()

```
ModbusAscPort::ModbusAscPort (
          bool blocking = false)
```

Constructor of the class. if blocking = true then defines blocking mode, non blocking otherwise.

### 7.6.2.2 ∼ModbusAscPort()

```
ModbusAscPort::~ModbusAscPort ()
```

Destructor of the class.

# 7.6.3 Member Function Documentation

## 7.6.3.1 readBuffer()

The function parses the packet that the read() function puts into the buffer, checks it for correctness, extracts its parameters, and returns the status of the operation.

Implements ModbusPort.

# 7.6.3.2 type()

```
Modbus::ProtocolType ModbusAscPort::type () const [inline], [override], [virtual]
```

Returns the Modbus protocol type. For ModbusAscPort returns Modbus::ASC.

Implements ModbusPort.

### 7.6.3.3 writeBuffer()

The function directly generates a packet and places it in the buffer for further sending. Returns the status of the operation.

Implements ModbusPort.

The documentation for this class was generated from the following file:

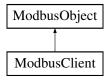
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusAscPort.h

# 7.7 ModbusClient Class Reference

The ModbusClient class implements the interface of the client part of the Modbus protocol.

```
#include <ModbusClient.h>
```

Inheritance diagram for ModbusClient:



### **Public Member Functions**

- ModbusClient (uint8 t unit, ModbusClientPort \*port)
- Modbus::ProtocolType type () const
- uint8\_t unit () const
- void setUnit (uint8\_t unit)
- bool isOpen () const
- ModbusClientPort \* port () const
- Modbus::StatusCode readCoils (uint16\_t offset, uint16\_t count, void \*values)
- Modbus::StatusCode readDiscreteInputs (uint16\_t offset, uint16\_t count, void \*values)
- Modbus::StatusCode readHoldingRegisters (uint16\_t offset, uint16\_t count, uint16\_t \*values)
- Modbus::StatusCode readInputRegisters (uint16\_t offset, uint16\_t count, uint16\_t \*values)
- Modbus::StatusCode writeSingleCoil (uint16\_t offset, bool value)
- Modbus::StatusCode writeSingleRegister (uint16\_t offset, uint16\_t value)
- Modbus::StatusCode readExceptionStatus (uint8 t \*value)
- Modbus::StatusCode diagnostics (uint16\_t subfunc, uint8\_t insize, const uint8\_t \*indata, uint8\_t \*outsize, uint8\_t \*outdata)
- Modbus::StatusCode getCommEventCounter (uint16 t \*status, uint16 t \*eventCount)
- Modbus::StatusCode getCommEventLog (uint16\_t \*status, uint16\_t \*eventCount, uint16\_t \*messageCount, uint8\_t \*eventBuffSize, uint8\_t \*eventBuff)
- Modbus::StatusCode writeMultipleCoils (uint16\_t offset, uint16\_t count, const void \*values)

- Modbus::StatusCode writeMultipleRegisters (uint16\_t offset, uint16\_t count, const uint16\_t \*values)
- Modbus::StatusCode reportServerID (uint8 t \*count, uint8 t \*data)
- Modbus::StatusCode maskWriteRegister (uint16\_t offset, uint16\_t andMask, uint16\_t orMask)
- Modbus::StatusCode readWriteMultipleRegisters (uint16\_t readOffset, uint16\_t readCount, uint16\_t \*read←
   Values, uint16 t writeOffset, uint16 t writeCount, const uint16 t \*writeValues)
- Modbus::StatusCode readFIFOQueue (uint16\_t fifoadr, uint16\_t \*count, uint16\_t \*values)
- Modbus::StatusCode readCoilsAsBoolArray (uint16 t offset, uint16 t count, bool \*values)
- Modbus::StatusCode readDiscreteInputsAsBoolArray (uint16 t offset, uint16 t count, bool \*values)
- Modbus::StatusCode writeMultipleCoilsAsBoolArray (uint16\_t offset, uint16\_t count, const bool \*values)
- Modbus::StatusCode lastPortStatus () const
- Modbus::StatusCode lastPortErrorStatus () const
- const Modbus::Char \* lastPortErrorText () const

# Public Member Functions inherited from ModbusObject

- ModbusObject ()
- virtual  $\sim$  ModbusObject ()
- const Modbus::Char \* objectName () const
- void setObjectName (const Modbus::Char \*name)
- template < class SignalClass , class T , class ReturnType , class ... Args > void connect (ModbusMethodPointer < SignalClass, ReturnType, Args ... > signalMethodPtr, T \*object, ModbusMethodPointer < T, ReturnType, Args ... > objectMethodPtr)
- template < class SignalClass , class ReturnType , class ... Args >
   void connect (ModbusMethodPointer < SignalClass, ReturnType, Args ... > signalMethodPtr, ModbusFunctionPointer <
   ReturnType, Args ... > funcPtr)
- template < class ReturnType , class ... Args > void disconnect (ModbusFunctionPointer < ReturnType, Args ... > funcPtr)
- void disconnectFunc (void \*funcPtr)
- template < class T , class ReturnType , class ... Args > void disconnect (T \*object, ModbusMethodPointer < T, ReturnType, Args ... > objectMethodPtr)
- template<class T >
   void disconnect (T \*object)

# Additional Inherited Members

# Static Public Member Functions inherited from ModbusObject

static ModbusObject \* sender ()

# Protected Member Functions inherited from ModbusObject

template < class T , class ... Args >
 void emitSignal (const char \*thisMethodId, ModbusMethodPointer < T, void, Args ... > thisMethod, Args ...
 args)

# 7.7.1 Detailed Description

The ModbusClient class implements the interface of the client part of the Modbus protocol.

ModbusClient contains a list of Modbus functions that are implemented by the Modbus client program. It implements functions for reading and writing different types of Modbus memory that are defined by the specification. The operations that return Modbus::StatusCode are asynchronous, that is, if the operation is not completed, it returns the intermediate status Modbus::Status\_Processing, and then it must be called until it is successfully completed or returns an error status.

# 7.7.2 Constructor & Destructor Documentation

### 7.7.2.1 ModbusClient()

Class constructor.

#### **Parameters**

in	unit	The address of the remote Modbus device to which this client is bound.
in	port	A pointer to the port object to which this client object belongs.

# 7.7.3 Member Function Documentation

# 7.7.3.1 diagnostics()

Same as ModbusClientPort::readInputRegisters(uint8\_t unit, uint16\_t offset, uint16\_t count but the unit address of the remote Modbus device is missing. It is preset in the constructor.

# 7.7.3.2 getCommEventCounter()

Same as ModbusClientPort::getCommEventCounter(uint8\_t unit, uint16\_t offset, uint16\_t count, uint16\_t \*values), but the unit address of the remote Modbus device is missing. It is preset in the constructor.

### 7.7.3.3 getCommEventLog()

Same as ModbusClientPort::getCommEventLog(uint8\_t unit, uint16\_t \*status, uint16\_t \*eventCount, uint16\_t \*messageCount, uint8\_t \*events), but the unit address of the remote Modbus device is missing. It is preset in the constructor.

# 7.7.3.4 isOpen()

```
bool ModbusClient::isOpen () const
```

Returns true if communication with the remote device is established, false otherwise.

#### 7.7.3.5 lastPortErrorStatus()

```
Modbus::StatusCode ModbusClient::lastPortErrorStatus () const
```

Returns the status of the last error of the performed operation.

#### 7.7.3.6 lastPortErrorText()

```
const Modbus::Char * ModbusClient::lastPortErrorText () const
```

Returns text repr of the last error of the performed operation.

### 7.7.3.7 lastPortStatus()

```
Modbus::StatusCode ModbusClient::lastPortStatus () const
```

Returns the status of the last operation performed.

### 7.7.3.8 maskWriteRegister()

Same as ModbusClientPort::writeMultipleRegisters(uint8\_t unit, uint16\_t offset, uint16\_t andMask, uint16\_t orMask), but the unit address of the remote Modbus device is missing. It is preset in the constructor.

# 7.7.3.9 port()

```
ModbusClientPort * ModbusClient::port () const
```

Returns a pointer to the port object to which this client object belongs.

# 7.7.3.10 readCoils()

Same as ModbusInterface::readCoils(uint8\_t unit, uint16\_t offset, uint16\_t count, void \*vabut the unit address of the remote Modbus device is missing. It is preset in the constructor.

### 7.7.3.11 readCoilsAsBoolArray()

Same as ModbusClientPort::readCoilsAsBoolArray(uint8\_t unit, uint16\_t offset, uint16\_t coubut the unit address of the remote Modbus device is missing. It is preset in the constructor.

#### 7.7.3.12 readDiscreteInputs()

Same as ModbusInterface::readDiscreteInputs(uint8\_t unit, uint16\_t offset, uint16\_t count, but the unit address of the remote Modbus device is missing. It is preset in the constructor.

#### 7.7.3.13 readDiscreteInputsAsBoolArray()

Same as ModbusClientPort::readWriteMultipleRegisters(uint8\_t unit, uint16 $\leftarrow$  \_t offset, readOffset, uint16\_t readCount, uint16\_t \*readValues, uint16 $\leftarrow$  t writeOffset, uint16\_t writeCount, const uint16\_t \*writeValues), but the unit address of the remote Modbus device is missing. It is preset in the constructor.

# 7.7.3.14 readExceptionStatus()

Same as ModbusInterface::readExceptionStatus(uint8\_t unit, uint8\_t \*status), but the unit address of the remote Modbus device is missing. It is preset in the constructor.

# 7.7.3.15 readFIFOQueue()

Same as ModbusClientPort::readFIFOQueue(uint8\_t unit, uint16\_t fifoadr, uint16\_t \*count, u but the unit address of the remote Modbus device is missing. It is preset in the constructor.

### 7.7.3.16 readHoldingRegisters()

Same as ModbusInterface::readHoldingRegisters(uint8\_t unit, uint16\_t offset, uint16\_t cour but the unit address of the remote Modbus device is missing. It is preset in the constructor.

#### 7.7.3.17 readInputRegisters()

Same as ModbusInterface::readInputRegisters(uint8\_t unit, uint16\_t offset, uint16\_t count, but the unit address of the remote Modbus device is missing. It is preset in the constructor.

### 7.7.3.18 readWriteMultipleRegisters()

Same as ModbusClientPort::readWriteMultipleRegisters(uint8\_t unit, uint16\_ $\leftarrow$  t offset, uint16\_t count, const uint16\_t \*values), but the unit address of the remote Modbus device is missing. It is preset in the constructor.

#### 7.7.3.19 reportServerID()

Same as ModbusClientPort::reportServerID (uint8\_t unit, uint8\_t \*count, uint8\_t \*data), but the unit address of the remote Modbus device is missing. It is preset in the constructor.

## 7.7.3.20 setUnit()

Sets the address of the remote Modbus device to which this client is bound.

#### 7.7.3.21 type()

```
Modbus::ProtocolType ModbusClient::type () const
```

Returns the type of the Modbus protocol.

# 7.7.3.22 unit()

```
uint8_t ModbusClient::unit () const
```

Returns the address of the remote Modbus device to which this client is bound.

#### 7.7.3.23 writeMultipleCoils()

Same as ModbusInterface::writeMultipleCoils (uint8\_t unit, uint16\_t offset, uint16\_t count, but the unit address of the remote Modbus device is missing. It is preset in the constructor.

#### 7.7.3.24 writeMultipleCoilsAsBoolArray()

Same as ModbusClientPort::writeMultipleCoilsAsBoolArray(uint8\_t unit, uint16\_t offset, uint but the unit address of the remote Modbus device is missing. It is preset in the constructor.

#### 7.7.3.25 writeMultipleRegisters()

Same as ModbusInterface::writeMultipleRegisters(uint8\_t unit, uint16\_t offset, uint16\_t cobut the unit address of the remote Modbus device is missing. It is preset in the constructor.

# 7.7.3.26 writeSingleCoil()

Same as ModbusInterface::writeSingleCoil(uint8\_t unit, uint16\_t offset, bool value), but the unit address of the remote Modbus device is missing. It is preset in the constructor.

# 7.7.3.27 writeSingleRegister()

Same as ModbusInterface::writeSingleRegister(uint8\_t unit, uint16\_t offset, uint16\_t value but the unit address of the remote Modbus device is missing. It is preset in the constructor.

The documentation for this class was generated from the following file:

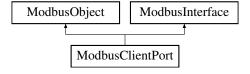
• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusClient.h

# 7.8 ModbusClientPort Class Reference

The ModbusClientPort class implements the algorithm of the client part of the Modbus communication protocol port.

```
#include <ModbusClientPort.h>
```

Inheritance diagram for ModbusClientPort:



#### **Public Types**

• enum RequestStatus { Enable , Disable , Process }

Sets the status of the request for the client.

#### **Public Member Functions**

- ModbusClientPort (ModbusPort \*port)
- · Modbus::ProtocolType type () const
- ModbusPort \* port () const
- void setPort (ModbusPort \*port)
- Modbus::StatusCode close ()
- bool isOpen () const
- uint32\_t tries () const
- void setTries (uint32\_t v)
- uint32\_t repeatCount () const
- void setRepeatCount (uint32\_t v)
- bool isBroadcastEnabled () const
- void setBroadcastEnabled (bool enable)
- Modbus::StatusCode readCoils (ModbusObject \*client, uint8\_t unit, uint16\_t offset, uint16\_t count, void \*values)

- Modbus::StatusCode readDiscreteInputs (ModbusObject \*client, uint8\_t unit, uint16\_t offset, uint16\_t count, void \*values)
- Modbus::StatusCode readHoldingRegisters (ModbusObject \*client, uint8\_t unit, uint16\_t offset, uint16\_
   t count, uint16\_t \*values)
- Modbus::StatusCode readInputRegisters (ModbusObject \*client, uint8\_t unit, uint16\_t offset, uint16\_t count, uint16\_t \*values)
- Modbus::StatusCode writeSingleCoil (ModbusObject \*client, uint8 t unit, uint16 t offset, bool value)
- Modbus::StatusCode writeSingleRegister (ModbusObject \*client, uint8\_t unit, uint16\_t offset, uint16\_t value)
- Modbus::StatusCode readExceptionStatus (ModbusObject \*client, uint8\_t unit, uint8\_t \*value)
- Modbus::StatusCode diagnostics (ModbusObject \*client, uint8\_t unit, uint16\_t subfunc, uint8\_t insize, const uint8\_t \*indata, uint8\_t \*outsize, uint8\_t \*outdata)
- Modbus::StatusCode getCommEventCounter (ModbusObject \*client, uint8\_t unit, uint16\_t \*status, uint16\_t \*eventCount)
- Modbus::StatusCode getCommEventLog (ModbusObject \*client, uint8\_t unit, uint16\_t \*status, uint16\_←
  t \*eventCount, uint16 t \*messageCount, uint8 t \*eventBuffSize, uint8 t \*eventBuff)
- Modbus::StatusCode writeMultipleCoils (ModbusObject \*client, uint8\_t unit, uint16\_t offset, uint16\_t count, const void \*values)
- Modbus::StatusCode writeMultipleRegisters (ModbusObject \*client, uint8\_t unit, uint16\_t offset, uint16\_←
  t count, const uint16 t \*values)
- Modbus::StatusCode reportServerID (ModbusObject \*client, uint8 t unit, uint8 t \*count, uint8 t \*data)
- Modbus::StatusCode maskWriteRegister (ModbusObject \*client, uint8\_t unit, uint16\_t offset, uint16\_t and
   — Mask, uint16\_t orMask)
- Modbus::StatusCode readWriteMultipleRegisters (ModbusObject \*client, uint8\_t unit, uint16\_t readOffset, uint16\_t readCount, uint16\_t \*readValues, uint16\_t writeOffset, uint16\_t writeCount, const uint16\_t \*write← Values)
- Modbus::StatusCode readFIFOQueue (ModbusObject \*client, uint8\_t unit, uint16\_t fifoadr, uint16\_t \*count, uint16\_t \*values)
- Modbus::StatusCode readCoilsAsBoolArray (ModbusObject \*client, uint8\_t unit, uint16\_t offset, uint16\_
   t count, bool \*values)
- Modbus::StatusCode readDiscreteInputsAsBoolArray (ModbusObject \*client, uint8\_t unit, uint16\_t offset, uint16\_t count, bool \*values)
- Modbus::StatusCode writeMultipleCoilsAsBoolArray (ModbusObject \*client, uint8\_t unit, uint16\_t offset, uint16 t count, const bool \*values)
- Modbus::StatusCode readCoils (uint8 t unit, uint16 t offset, uint16 t count, void \*values) override
- Modbus::StatusCode readDiscreteInputs (uint8 t unit, uint16 t offset, uint16 t count, void \*values) override
- Modbus::StatusCode readHoldingRegisters (uint8\_t unit, uint16\_t offset, uint16\_t count, uint16\_t \*values)
   override
- Modbus::StatusCode readInputRegisters (uint8\_t unit, uint16\_t offset, uint16\_t count, uint16\_t \*values) override
- Modbus::StatusCode writeSingleCoil (uint8\_t unit, uint16\_t offset, bool value) override
- Modbus::StatusCode writeSingleRegister (uint8 t unit, uint16 t offset, uint16 t value) override
- Modbus::StatusCode readExceptionStatus (uint8\_t unit, uint8\_t \*value) override
- Modbus::StatusCode diagnostics (uint8\_t unit, uint16\_t subfunc, uint8\_t insize, const uint8\_t \*indata, uint8\_t \*outsize, uint8\_t \*outdata) override
- $\bullet \ \ Modbus:: Status Code \ get CommEvent Counter \ (uint 8\_t \ unit, \ uint 16\_t \ *status, \ uint 16\_t \ *event Count) \ override$
- Modbus::StatusCode getCommEventLog (uint8\_t unit, uint16\_t \*status, uint16\_t \*eventCount, uint16\_←
  t \*messageCount, uint8\_t \*eventBuffSize, uint8\_t \*eventBuff) override
- Modbus::StatusCode writeMultipleCoils (uint8\_t unit, uint16\_t offset, uint16\_t count, const void \*values) override
- Modbus::StatusCode writeMultipleRegisters (uint8\_t unit, uint16\_t offset, uint16\_t count, const uint16\_←
  t \*values) override
- Modbus::StatusCode reportServerID (uint8 t unit, uint8 t \*count, uint8 t \*data) override
- Modbus::StatusCode maskWriteRegister (uint8\_t unit, uint16\_t offset, uint16\_t andMask, uint16\_t orMask)
  override
- Modbus::StatusCode readWriteMultipleRegisters (uint8\_t unit, uint16\_t readOffset, uint16\_t readCount, uint16\_t \*readValues, uint16\_t writeOffset, uint16\_t writeCount, const uint16\_t \*writeValues) override

 Modbus::StatusCode readFIFOQueue (uint8\_t unit, uint16\_t fifoadr, uint16\_t \*count, uint16\_t \*values) override

- Modbus::StatusCode readCoilsAsBoolArray (uint8\_t unit, uint16\_t offset, uint16\_t count, bool \*values)
- Modbus::StatusCode readDiscreteInputsAsBoolArray (uint8\_t unit, uint16\_t offset, uint16\_t count, bool \*values)
- Modbus::StatusCode writeMultipleCoilsAsBoolArray (uint8\_t unit, uint16\_t offset, uint16\_t count, const bool \*values)
- Modbus::StatusCode lastStatus () const
- Modbus::Timestamp lastStatusTimestamp () const
- Modbus::StatusCode lastErrorStatus () const
- const Modbus::Char \* lastErrorText () const
- const ModbusObject \* currentClient () const
- RequestStatus getRequestStatus (ModbusObject \*client)
- void cancelRequest (ModbusObject \*client)
- void signalOpened (const Modbus::Char \*source)
- void signalClosed (const Modbus::Char \*source)
- void signalTx (const Modbus::Char \*source, const uint8 t \*buff, uint16 t size)
- void signalRx (const Modbus::Char \*source, const uint8\_t \*buff, uint16\_t size)
- void signalError (const Modbus::Char \*source, Modbus::StatusCode status, const Modbus::Char \*text)

# Public Member Functions inherited from ModbusObject

- ModbusObject ()
- virtual ∼ModbusObject ()
- const Modbus::Char \* objectName () const
- void setObjectName (const Modbus::Char \*name)
- template < class SignalClass , class T , class ReturnType , class ... Args > void connect (ModbusMethodPointer < SignalClass, ReturnType, Args ... > signalMethodPtr, T \*object, ModbusMethodPointer < T, ReturnType, Args ... > objectMethodPtr)
- template < class SignalClass , class ReturnType , class ... Args >
   void connect (ModbusMethodPointer < SignalClass, ReturnType, Args ... > signalMethodPtr, ModbusFunctionPointer <
   ReturnType, Args ... > funcPtr)
- template < class ReturnType , class ... Args > void disconnect (ModbusFunctionPointer < ReturnType, Args ... > funcPtr)
- void disconnectFunc (void \*funcPtr)
- template < class T , class ReturnType , class ... Args > void disconnect (T \*object, ModbusMethodPointer < T, ReturnType, Args ... > objectMethodPtr)
- template < class T > void disconnect (T \*object)

# Public Member Functions inherited from ModbusInterface

#### **Friends**

· class ModbusClient

### **Additional Inherited Members**

### Static Public Member Functions inherited from ModbusObject

static ModbusObject \* sender ()

# Protected Member Functions inherited from ModbusObject

template < class T , class ... Args >
 void emitSignal (const char \*thisMethodId, ModbusMethodPointer < T, void, Args ... > thisMethod, Args ...
 args)

# 7.8.1 Detailed Description

The ModbusClientPort class implements the algorithm of the client part of the Modbus communication protocol port.

ModbusClient contains a list of Modbus functions that are implemented by the Modbus client program. It implements functions for reading and writing various types of Modbus memory defined by the specification. In the non blocking mode if the operation is not completed it returns the intermediate status Modbus::Status\_Processing, and then it must be called until it is successfully completed or returns an error status.

ModbusClientPort has number of Modbus functions with interface like readCoils (ModbusObject \*client, ...). Several clients can automatically share a current ModbusClientPort resource. The first one to access the port seizes the resource until the operation with the remote device is completed. Then the first client will release the resource and the next client in the gueue will capture it, and so on in a circle.

```
#include <ModbusClient.h>
//...
void main()
{
    //...
    ModbusClientPort *port = Modbus::createClientPort(Modbus::TCP, &settings, false);
    ModbusClient c1(1, port);
    ModbusClient c2(2, port);
    ModbusClient c3(3, port);
    Modbus::StatusCode s1, s2, s3;
    //...
    while(1)
    {
        s1 = c1.readHoldingRegisters(0, 10, values);
        s2 = c2.readHoldingRegisters(0, 10, values);
        s3 = c3.readHoldingRegisters(0, 10, values);
        doSomeOtherStuffInCurrentThread();
        Modbus::msleep(1);
    }
    //...
}
//...
```

# 7.8.2 Constructor & Destructor Documentation

# 7.8.2.1 ModbusClientPort()

Constructor of the class.

#### **Parameters**

in	port	A pointer to the port object which belongs to this client object. Lifecycle of the port object is
		managed by this ModbusClientPort-object

#### 7.8.3 Member Function Documentation

#### 7.8.3.1 cancelRequest()

Cancels the previous request specified by the \*rp pointer for the client.

### 7.8.3.2 close()

```
Modbus::StatusCode ModbusClientPort::close ()
```

Closes connection and returns status of the operation.

# 7.8.3.3 currentClient()

```
const ModbusObject * ModbusClientPort::currentClient () const
```

Returns a pointer to the client object whose request is currently being processed by the current port.

#### 7.8.3.4 diagnostics() [1/2]

Same as ModbusClientPort::readInputRegisters(uint8\_t unit, uint16\_t offset, uint16\_t count but has client as first parameter to seize current ModbusClientPort resource.

# 7.8.3.5 diagnostics() [2/2]

Function provides a series of tests for checking the communication system between a client device and a server, or for checking various internal error conditions within a server.

#### **Parameters**

in	unit	Address of the remote Modbus device.
in	subfunc	Address of the remote Modbus device.
in	insize	Size of the input buffer (in bytes).
in	indata	Pointer to the buffer where the input (request) data is stored.
ou	t <i>outsize</i>	Size of the buffer (in bytes) where the output data is stored.
ou	t outdata	Pointer to the buffer where the output data is stored.

#### Returns

The result Modbus::StatusCode of the operation. Default implementation returns  $Status\_Bad \leftarrow IllegalFunction$ .

Reimplemented from ModbusInterface.

# 7.8.3.6 getCommEventCounter() [1/2]

Same as ModbusClientPort::getCommEventCounter(uint8\_t unit, uint16\_t offset, uint16\_t count, uint16\_t \*values) but has client as first parameter to seize current ModbusClientPort resource.

# 7.8.3.7 getCommEventCounter() [2/2]

Function is used to get a status word and an event count from the remote device's communication event counter.

#### **Parameters**

in	unit	Address of the remote Modbus device.
out	status	Returned status word.
out	eventCount	Returned event counter.

# Returns

The result Modbus::StatusCode of the operation. Default implementation returns  $\texttt{Status\_Bad} \leftarrow \texttt{IllegalFunction}$ .

Reimplemented from ModbusInterface.

# 7.8.3.8 getCommEventLog() [1/2]

Same as ModbusClientPort::getCommEventLog(uint8\_t unit, uint16\_t \*status, uint16\_t \*eventCount, uint16\_t \*messageCount, uint8\_t \*events) but has client as first parameter to seize current ModbusClientPort resource.

# 7.8.3.9 getCommEventLog() [2/2]

Function is used to get a status word and an event count from the remote device's communication event counter.

# **Parameters**

in	unit	Address of the remote Modbus device.
out	status	Returned status word.
out	eventCount	Returned event counter.
out	messageCount	Returned message counter.
out <i>eventBuffSize</i>		Size of the buffer where the output events (bytes) is stored.
out <i>eventBuff</i>		Pointer to the buffer where the output events (bytes) is stored.

## Returns

The result Modbus::StatusCode of the operation. Default implementation returns  $Status\_Bad \leftarrow IllegalFunction$ .

Reimplemented from ModbusInterface.

# 7.8.3.10 getRequestStatus()

Returns status the current request for client.

The client usually calls this function to determine whether its request is pending/finished/blocked. If function returns <code>Enable</code>, <code>client</code> has just became current and can make request to the port, <code>Process</code> - current <code>client</code> is already processing, <code>Disable</code> - other client owns the port.

### 7.8.3.11 isBroadcastEnabled()

```
bool ModbusClientPort::isBroadcastEnabled () const
```

Returns true if broadcast mode for 0 unit address is enabled, false otherwise. Broadcast mode for 0 unit address is required by Modbus protocol so it is enabled by default

# 7.8.3.12 isOpen()

```
bool ModbusClientPort::isOpen () const
```

Returns true if the connection with the remote device is established, false otherwise.

# 7.8.3.13 lastErrorStatus()

```
Modbus::StatusCode ModbusClientPort::lastErrorStatus () const
```

Returns the status of the last error of the performed operation.

### 7.8.3.14 lastErrorText()

```
const Modbus::Char * ModbusClientPort::lastErrorText () const
```

Returns the text of the last error of the performed operation.

### 7.8.3.15 lastStatus()

```
Modbus::StatusCode ModbusClientPort::lastStatus () const
```

Returns the status of the last operation performed.

# 7.8.3.16 lastStatusTimestamp()

```
Modbus::Timestamp ModbusClientPort::lastStatusTimestamp () const
```

Returns the timestamp of the last operation performed.

### 7.8.3.17 maskWriteRegister() [1/2]

Same as ModbusClientPort::writeMultipleRegisters(uint8\_t unit, uint16\_t offset, uint16\_t andMask, uint16\_t orMask) but has client as first parameter to seize current ModbusClientPort resource.

#### 7.8.3.18 maskWriteRegister() [2/2]

Function is used to modify the contents of a specified holding register using a combination of an AND mask, an OR mask, and the register's current contents. The function's algorithm is: Result = (Current Contents AND And\_Mask) OR (Or\_Mask AND (NOT And\_Mask))

#### **Parameters**

in	unit	Address of the remote Modbus device.	
in	offset	Starting offset (0-based).	
in	andMask	16-bit unsigned integer value AND mask.	
in	orMask	16-bit unsigned integer value OR mask.	

#### Returns

The result  ${\tt Modbus::StatusCode}$  of the operation. Default implementation returns  ${\tt Status\_Bad} {\hookleftarrow}$   ${\tt IllegalFunction.}$ 

Reimplemented from ModbusInterface.

### 7.8.3.19 port()

```
ModbusPort * ModbusClientPort::port () const
```

Returns a pointer to the port object that is used by this algorithm.

# 7.8.3.20 readCoils() [1/2]

Same as ModbusClientPort::readCoils(uint8\_t unit, uint16\_t offset, uint16\_t count, void \*v but has client as first parameter to seize current ModbusClientPort resource.

# 7.8.3.21 readCoils() [2/2]

Function for read discrete outputs (coils, 0x bits).

#### **Parameters**

in	unit	Address of the remote Modbus device.
in	offset	Starting offset (0-based).
in	count	Count of coils (bits).
out	values	Pointer to the output buffer (bit array) where the read values are stored.

#### Returns

The result Modbus::StatusCode of the operation. Default implementation returns Status\_Bad← IllegalFunction.

Reimplemented from ModbusInterface.

#### 7.8.3.22 readCoilsAsBoolArray() [1/2]

Same as ModbusClientPort::readCoilsAsBoolArray(uint8\_t unit, uint16\_t offset, uint16\_t coubut has client as first parameter to seize current ModbusClientPort resource.

# 7.8.3.23 readCoilsAsBoolArray() [2/2]

Same as ModbusClientPort::readCoils(uint8\_t unit, uint16\_t offset, uint16\_t count, void \*v but the output buffer of values values is an array, where each discrete value is located in a separate element of the array of type bool.

# 7.8.3.24 readDiscreteInputs() [1/2]

Same as ModbusClientPort::readDiscreteInputs(uint8\_t unit, uint16\_t offset, uint16\_t count but has client as first parameter to seize current ModbusClientPort resource.

# 7.8.3.25 readDiscreteInputs() [2/2]

Function for read digital inputs (1x bits).

#### **Parameters**

in	unit	Address of the remote Modbus device.
in	offset	Starting offset (0-based).
in	count	Count of inputs (bits).
out	values	Pointer to the output buffer (bit array) where the read values are stored.

#### Returns

The result Modbus::StatusCode of the operation. Default implementation returns Status\_Bad← IllegalFunction.

Reimplemented from ModbusInterface.

## 7.8.3.26 readDiscreteInputsAsBoolArray() [1/2]

Same as ModbusClientPort::readDiscreteInputsAsBoolArray(uint8\_t unit, uint16\_t offset, uinbut has client as first parameter to seize current ModbusClientPort resource.

#### 7.8.3.27 readDiscreteInputsAsBoolArray() [2/2]

Same as ModbusClientPort::readDiscreteInputs(uint8\_t unit, uint16\_t offset, uint16\_t count but the output buffer of values values is an array, where each discrete value is located in a separate element of the array of type bool.

# 7.8.3.28 readExceptionStatus() [1/2]

Same as ModbusClientPort::readExceptionStatus(uint8\_t unit, uint8\_t \*status) but has client as first parameter to seize current ModbusClientPort resource.

# 7.8.3.29 readExceptionStatus() [2/2]

Function to read ExceptionStatus.

#### **Parameters**

in	unit	Address of the remote Modbus device.
out	status	Pointer to the byte (bit array) where the exception status is stored.

#### Returns

The result Modbus::StatusCode of the operation. Default implementation returns  $Status\_Bad \leftarrow IllegalFunction$ .

Reimplemented from ModbusInterface.

#### 7.8.3.30 readFIFOQueue() [1/2]

Same as ModbusClientPort::readFIFOQueue (uint8\_t unit, uint16\_t fifoadr, uint16\_t \*count, u but has client as first parameter to seize current ModbusClientPort resource.

#### 7.8.3.31 readFIFOQueue() [2/2]

Function for read the contents of a First-In-First-Out (FIFO) queue of register in a remote device.

# **Parameters**

in	unit	Address of the remote Modbus device.
in	fifoadr	Address of FIFO (0-based).
in	count	Count of registers.
out	values	Pointer to the output buffer where the read values are stored.

### Returns

The result Modbus::StatusCode of the operation. Default implementation returns Status\_Bad← IllegalFunction.

Reimplemented from ModbusInterface.

#### 7.8.3.32 readHoldingRegisters() [1/2]

Same as ModbusClientPort::readHoldingRegisters(uint8\_t unit, uint16\_t offset, uint16\_t coubut has client as first parameter to seize current ModbusClientPort resource.

# 7.8.3.33 readHoldingRegisters() [2/2]

Function for read holding (output) 16-bit registers (4x regs).

#### **Parameters**

	in	unit	Address of the remote Modbus device.
	in	offset	Starting offset (0-based).
Ī	in	count	Count of registers.
	out	values	Pointer to the output buffer (bit array) where the read values are stored.

### Returns

The result Modbus::StatusCode of the operation. Default implementation returns  $Status\_Bad \leftarrow IllegalFunction$ .

Reimplemented from ModbusInterface.

#### 7.8.3.34 readInputRegisters() [1/2]

Same as ModbusClientPort::readInputRegisters(uint8\_t unit, uint16\_t offset, uint16\_t count but has client as first parameter to seize current ModbusClientPort resource.

# 7.8.3.35 readInputRegisters() [2/2]

Function for read input 16-bit registers (3x regs).

#### **Parameters**

in	unit	Address of the remote Modbus device.
in	offset	Starting offset (0-based).
in	count	Count of registers.
out	values	Pointer to the output buffer (bit array) where the read values are stored.

#### Returns

The result Modbus::StatusCode of the operation. Default implementation returns Status\_Bad← IllegalFunction.

Reimplemented from ModbusInterface.

# 7.8.3.36 readWriteMultipleRegisters() [1/2]

Same as ModbusClientPort::readWriteMultipleRegisters(uint8\_t unit, uint16← \_t offset, readOffset, uint16\_t readCount, uint16\_t \*readValues, uint16\_← t writeOffset, uint16\_t writeCount, const uint16\_t \*writeValues) but has client as first parameter to seize current ModbusClientPort resource.

# 7.8.3.37 readWriteMultipleRegisters() [2/2]

This function code performs a combination of one read operation and one write operation in a single MODBUS transaction.

#### **Parameters**

in	unit	Address of the remote Modbus device.
in <i>readOffset</i>		Starting offset for read(0-based).
in	readCount	Count of registers to read.
out	readValues	Pointer to the output buffer which values must be read.
in writeOffse		Starting offset for write(0-based).
in writeCount		Count of registers to write.
in	writeValues	Pointer to the input buffer which values must be written.

#### Returns

The result Modbus::StatusCode of the operation. Default implementation returns  $Status\_Bad \leftarrow IllegalFunction$ .

Reimplemented from ModbusInterface.

### 7.8.3.38 repeatCount()

```
uint32_t ModbusClientPort::repeatCount () const [inline]
```

Same as tries (). Used for backward compatibility.

### 7.8.3.39 reportServerID() [1/2]

Same as ModbusClientPort::reportServerID(uint8\_t unit, uint8\_t \*count, uint8\_t \*data) but has client as first parameter to seize current ModbusClientPort resource.

# 7.8.3.40 reportServerID() [2/2]

Function to read the description of the type, the current status, and other information specific to a remote device.

#### **Parameters**

	in	unit	Address of the remote Modbus device.
ſ	in	count	Count of bytes returned.
Ī	in	data	Pointer to the output buffer where the read data are stored.

### Returns

The result Modbus::StatusCode of the operation. Default implementation returns  $Status\_Bad \leftarrow IllegalFunction$ .

Reimplemented from ModbusInterface.

# 7.8.3.41 setBroadcastEnabled()

Enables broadcast mode for 0 unit address. It is enabled by default.

See also

```
isBroadcastEnabled()
```

# 7.8.3.42 setPort()

Set new port object for current client port control. Previous port object is deleted.

# 7.8.3.43 setRepeatCount()

Same as setTries(). Used for backward compatibility.

# 7.8.3.44 setTries()

Sets the number of tries a Modbus request is repeated if it fails.

# 7.8.3.45 signalClosed()

Calls each callback of the port when the port is closed.  $\verb"source"$  - current port's name

# 7.8.3.46 signalError()

Calls each callback of the port when error is occured with error's status and text.

### 7.8.3.47 signalOpened()

Calls each callback of the port when the port is opened. source - current port's name

#### 7.8.3.48 signalRx()

Calls each callback of the incoming packet 'Rx' from the internal list of callbacks, passing them the input array 'buff' and its size 'size'.

# 7.8.3.49 signalTx()

Calls each callback of the original packet 'Tx' from the internal list of callbacks, passing them the original array 'buff' and its size 'size'.

### 7.8.3.50 tries()

```
uint32_t ModbusClientPort::tries () const
```

Returns the setting of the number of tries of the Modbus request if it fails.

# 7.8.3.51 type()

```
Modbus::ProtocolType ModbusClientPort::type () const
```

Returns type of Modbus protocol.

# 7.8.3.52 writeMultipleCoils() [1/2]

Same as ModbusClientPort::writeMultipleCoils(uint8\_t unit, uint16\_t offset, uint16\_t count but has client as first parameter to seize current ModbusClientPort resource.

### 7.8.3.53 writeMultipleCoils() [2/2]

Function is used to modify the contents of a specified holding register using a combination of an AND mask, an OR mask, and the register's current contents.

#### **Parameters**

in	unit	Address of the remote Modbus device.
in	offset	Starting offset (0-based).
in	count	Count of coils.
in	values	Pointer to the input buffer (bit array) which values must be written.

#### Returns

The result Modbus::StatusCode of the operation. Default implementation returns  $Status\_Bad \leftarrow IllegalFunction$ .

Reimplemented from ModbusInterface.

#### 7.8.3.54 writeMultipleCoilsAsBoolArray() [1/2]

Same as ModbusClientPort::writeMultipleCoilsAsBoolArray(uint8\_t unit, uint16\_t offset, uinbut has client as first parameter to seize current ModbusClientPort resource.

# 7.8.3.55 writeMultipleCoilsAsBoolArray() [2/2]

Same as ModbusClientPort::writeMultipleCoils(uint8\_t unit, uint16\_t offset, uint16\_t count but the input buffer of values is an array, where each discrete value is located in a separate element of the array of type bool.

### 7.8.3.56 writeMultipleRegisters() [1/2]

Same as ModbusClientPort::writeMultipleRegisters(uint8\_t unit, uint16\_t offset, uint16\_t obut has client as first parameter to seize current ModbusClientPort resource.

# 7.8.3.57 writeMultipleRegisters() [2/2]

Function for write holding (output) 16-bit registers (4x regs).

#### **Parameters**

in	unit	Address of the remote Modbus device.
in	offset	Starting offset (0-based).
in	count	Count of registers.
in	values	Pointer to the input buffer which values must be written.

#### Returns

The result Modbus::StatusCode of the operation. Default implementation returns  $Status\_Bad \leftarrow IllegalFunction$ .

Reimplemented from ModbusInterface.

### 7.8.3.58 writeSingleCoil() [1/2]

Same as ModbusClientPort::writeSingleCoil(uint8\_t unit, uint16\_t offset, bool value) but has client as first parameter to seize current ModbusClientPort resource.

### 7.8.3.59 writeSingleCoil() [2/2]

Function for write one separate discrete output (0x coil).

#### **Parameters**

in	unit	Address of the remote Modbus device.	
in	offset	Starting offset (0-based).	
in	value	Boolean value to be set.	

#### Returns

The result Modbus::StatusCode of the operation. Default implementation returns  $Status\_Bad \leftarrow IllegalFunction$ .

Reimplemented from ModbusInterface.

# 7.8.3.60 writeSingleRegister() [1/2]

Same as ModbusClientPort::writeSingleRegister(uint8\_t unit, uint16\_t offset, uint16\_t value but has client as first parameter to seize current ModbusClientPort resource.

# 7.8.3.61 writeSingleRegister() [2/2]

Function for write one separate 16-bit holding register (4x).

## **Parameters**

in	unit	Address of the remote Modbus device.
in	offset	Starting offset (0-based).
in	value	16-bit unsigned integer value to be set.

### Returns

The result Modbus::StatusCode of the operation. Default implementation returns  $Status\_Bad \leftarrow IllegalFunction$ .

Reimplemented from ModbusInterface.

The documentation for this class was generated from the following file:

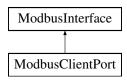
• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusClientPort.h

# 7.9 ModbusInterface Class Reference

Main interface of Modbus communication protocol.

#include <Modbus.h>

Inheritance diagram for ModbusInterface:



#### **Public Member Functions**

- virtual Modbus::StatusCode readCoils (uint8\_t unit, uint16\_t offset, uint16\_t count, void \*values)
- virtual Modbus::StatusCode readDiscreteInputs (uint8\_t unit, uint16\_t offset, uint16\_t count, void \*values)
- virtual Modbus::StatusCode readHoldingRegisters (uint8\_t unit, uint16\_t offset, uint16\_t count, uint16\_
   t \*values)
- virtual Modbus::StatusCode readInputRegisters (uint8\_t unit, uint16\_t offset, uint16\_t count, uint16\_t \*values)
- virtual Modbus::StatusCode writeSingleCoil (uint8\_t unit, uint16\_t offset, bool value)
- virtual Modbus::StatusCode writeSingleRegister (uint8 t unit, uint16 t offset, uint16 t value)
- virtual Modbus::StatusCode readExceptionStatus (uint8\_t unit, uint8\_t \*status)
- virtual Modbus::StatusCode diagnostics (uint8\_t unit, uint16\_t subfunc, uint8\_t insize, const uint8\_t \*indata, uint8\_t \*outsize, uint8\_t \*outdata)
- virtual Modbus::StatusCode getCommEventCounter (uint8\_t unit, uint16\_t \*status, uint16\_t \*eventCount)
- virtual Modbus::StatusCode getCommEventLog (uint8\_t unit, uint16\_t \*status, uint16\_t \*eventCount, uint16\_t \*messageCount, uint8\_t \*eventBuffSize, uint8\_t \*eventBuff)
- virtual Modbus::StatusCode writeMultipleCoils (uint8\_t unit, uint16\_t offset, uint16\_t count, const void \*values)
- virtual Modbus::StatusCode writeMultipleRegisters (uint8\_t unit, uint16\_t offset, uint16\_t count, const uint16\_t \*values)
- virtual Modbus::StatusCode reportServerID (uint8\_t unit, uint8\_t \*count, uint8\_t \*data)
- virtual Modbus::StatusCode maskWriteRegister (uint8\_t unit, uint16\_t offset, uint16\_t andMask, uint16\_t or
   — Mask)
- virtual Modbus::StatusCode readWriteMultipleRegisters (uint8\_t unit, uint16\_t readOffset, uint16\_t read Count, uint16\_t \*readValues, uint16\_t writeOffset, uint16\_t writeCount, const uint16\_t \*writeValues)
- virtual Modbus::StatusCode readFIFOQueue (uint8\_t unit, uint16\_t fifoadr, uint16\_t \*count, uint16\_t \*values)

# 7.9.1 Detailed Description

Main interface of Modbus communication protocol.

ModbusInterface constains list of functions that ModbusLib is supported. There are such functions as ←: 1 (0x01) - READ\_COILS 2 (0x02) - READ\_DISCRETE\_INPUTS 3 (0x03) - READ\_HOLDING\_REGISTERS 4 (0x04) - READ\_INPUT\_REGISTERS 5 (0x05) - WRITE\_SINGLE\_COIL 6 (0x06) - WRITE\_SINGLE\_← REGISTER 7 (0x07) - READ\_EXCEPTION\_STATUS 8 (0x08) - DIAGNOSTICS 11 (0x0B) - GET\_COMM\_← EVENT\_COUNTER 12 (0x0C) - GET\_COMM\_EVENT\_LOG 15 (0x0F) - WRITE\_MULTIPLE\_COILS 16 (0x10) - WRITE\_MULTIPLE\_REGISTERS 17 (0x11) - REPORT\_SERVER\_ID 22 (0x16) - MASK\_WRITE\_REGISTER 23 (0x17) - READ\_WRITE\_MULTIPLE\_REGISTERS 24 (0x18) - READ\_FIFO\_QUEUE

Default implementation of every Modbus function returns Modbus::Status\_BadIllegalFunction.

# 7.9.2 Member Function Documentation

# 7.9.2.1 diagnostics()

Function provides a series of tests for checking the communication system between a client device and a server, or for checking various internal error conditions within a server.

# **Parameters**

in	unit	Address of the remote Modbus device.
in	subfunc	Address of the remote Modbus device.
in	insize	Size of the input buffer (in bytes).
in	indata	Pointer to the buffer where the input (request) data is stored.
out	outsize	Size of the buffer (in bytes) where the output data is stored.
out	outdata	Pointer to the buffer where the output data is stored.

#### Returns

The result Modbus::StatusCode of the operation. Default implementation returns  $Status\_Bad \leftarrow IllegalFunction$ .

Reimplemented in ModbusClientPort.

# 7.9.2.2 getCommEventCounter()

Function is used to get a status word and an event count from the remote device's communication event counter.

# **Parameters**

in	unit	Address of the remote Modbus device.
out	status	Returned status word.
out	eventCount	Returned event counter.

# Returns

The result Modbus::StatusCode of the operation. Default implementation returns  $Status\_Bad \leftarrow IllegalFunction$ .

Reimplemented in ModbusClientPort.

# 7.9.2.3 getCommEventLog()

Function is used to get a status word and an event count from the remote device's communication event counter.

#### **Parameters**

in	unit	Address of the remote Modbus device.
out	status	Returned status word.
out <b>eventCount</b>		Returned event counter.
out	messageCount	Returned message counter.
out	eventBuffSize	Size of the buffer where the output events (bytes) is stored.
out	eventBuff	Pointer to the buffer where the output events (bytes) is stored.

#### Returns

The result Modbus::StatusCode of the operation. Default implementation returns Status\_Bad← IllegalFunction.

Reimplemented in ModbusClientPort.

#### 7.9.2.4 maskWriteRegister()

Function is used to modify the contents of a specified holding register using a combination of an AND mask, an OR mask, and the register's current contents. The function's algorithm is: Result = (Current Contents AND And\_Mask) OR (Or\_Mask AND (NOT And\_Mask))

#### **Parameters**

	in <i>unit</i> in <i>offset</i>		Address of the remote Modbus device.
			Starting offset (0-based).
	in	andMask	16-bit unsigned integer value AND mask.
	in	orMask	16-bit unsigned integer value OR mask.

### Returns

The result Modbus::StatusCode of the operation. Default implementation returns  $\texttt{Status\_Bad} \leftarrow \texttt{IllegalFunction}$ .

Reimplemented in ModbusClientPort.

### 7.9.2.5 readCoils()

Function for read discrete outputs (coils, 0x bits).

#### **Parameters**

	in	unit	Address of the remote Modbus device.
	in	offset	Starting offset (0-based).
Ī	in	count	Count of coils (bits).
Ī	out	values	Pointer to the output buffer (bit array) where the read values are stored.

#### Returns

The result Modbus::StatusCode of the operation. Default implementation returns Status\_Bad← IllegalFunction.

Reimplemented in ModbusClientPort.

# 7.9.2.6 readDiscreteInputs()

Function for read digital inputs (1x bits).

# Parameters

in	unit	Address of the remote Modbus device.
in	offset	Starting offset (0-based).
in	count	Count of inputs (bits).
out	values	Pointer to the output buffer (bit array) where the read values are stored.

# Returns

The result Modbus::StatusCode of the operation. Default implementation returns  $Status\_Bad \leftarrow IllegalFunction$ .

Reimplemented in ModbusClientPort.

#### 7.9.2.7 readExceptionStatus()

Function to read ExceptionStatus.

#### **Parameters**

	in	unit	Address of the remote Modbus device.
ſ	out	status	Pointer to the byte (bit array) where the exception status is stored.

#### Returns

The result Modbus::StatusCode of the operation. Default implementation returns  $Status\_Bad \leftarrow IllegalFunction$ .

Reimplemented in ModbusClientPort.

## 7.9.2.8 readFIFOQueue()

Function for read the contents of a First-In-First-Out (FIFO) queue of register in a remote device.

#### **Parameters**

in	unit	Address of the remote Modbus device.
in	fifoadr	Address of FIFO (0-based).
in	count	Count of registers.
out	values	Pointer to the output buffer where the read values are stored.

## Returns

The result Modbus::StatusCode of the operation. Default implementation returns  $Status\_Bad \leftarrow IllegalFunction$ .

Reimplemented in ModbusClientPort.

#### 7.9.2.9 readHoldingRegisters()

Function for read holding (output) 16-bit registers (4x regs).

#### **Parameters**

	in	unit	Address of the remote Modbus device.	
	in	offset	Starting offset (0-based).	
	in	count	Count of registers.	
Ī	out	values	Pointer to the output buffer (bit array) where the read values are stored.	

# Returns

The result Modbus::StatusCode of the operation. Default implementation returns Status\_Bad← IllegalFunction.

Reimplemented in ModbusClientPort.

# 7.9.2.10 readInputRegisters()

Function for read input 16-bit registers (3x regs).

#### **Parameters**

in	unit	Address of the remote Modbus device.
in	offset	Starting offset (0-based).
in	count	Count of registers.
out	values	Pointer to the output buffer (bit array) where the read values are stored.

# Returns

The result Modbus::StatusCode of the operation. Default implementation returns Status\_Bad← IllegalFunction.

Reimplemented in ModbusClientPort.

## 7.9.2.11 readWriteMultipleRegisters()

This function code performs a combination of one read operation and one write operation in a single MODBUS transaction.

#### **Parameters**

in	unit	Address of the remote Modbus device.	
in	readOffset	Starting offset for read(0-based).	
in	readCount	Count of registers to read.	
out	readValues	Pointer to the output buffer which values must be read.	
in	writeOffset	Starting offset for write(0-based).	
in	writeCount	Count of registers to write.	
in	writeValues	Pointer to the input buffer which values must be written.	

# Returns

The result Modbus::StatusCode of the operation. Default implementation returns  $Status\_Bad \leftarrow IllegalFunction$ .

Reimplemented in ModbusClientPort.

# 7.9.2.12 reportServerID()

Function to read the description of the type, the current status, and other information specific to a remote device.

#### **Parameters**

	in	unit	Address of the remote Modbus device.	
	in	count	Count of bytes returned.	
ĺ	in	data	Pointer to the output buffer where the read data are stored.	

#### Returns

The result Modbus::StatusCode of the operation. Default implementation returns Status\_Bad← IllegalFunction.

Reimplemented in ModbusClientPort.

# 7.9.2.13 writeMultipleCoils()

Function is used to modify the contents of a specified holding register using a combination of an AND mask, an OR mask, and the register's current contents.

# Parameters

in	unit	Address of the remote Modbus device.	
in	offset	Starting offset (0-based).	
in	count	Count of coils.	
in	values	Pointer to the input buffer (bit array) which values must be written.	

#### Returns

The result Modbus::StatusCode of the operation. Default implementation returns  $Status\_Bad \leftarrow IllegalFunction$ .

Reimplemented in ModbusClientPort.

#### 7.9.2.14 writeMultipleRegisters()

Function for write holding (output) 16-bit registers (4x regs).

#### **Parameters**

in	unit	Address of the remote Modbus device.	
in	offset	offset Starting offset (0-based).	
in	count	Count of registers.	
in	values	Pointer to the input buffer which values must be written.	

#### Returns

The result Modbus::StatusCode of the operation. Default implementation returns  $Status\_Bad \leftarrow IllegalFunction$ .

Reimplemented in ModbusClientPort.

# 7.9.2.15 writeSingleCoil()

Function for write one separate discrete output (0x coil).

#### **Parameters**

in	unit	Address of the remote Modbus device.
in	offset	Starting offset (0-based).
in	value	Boolean value to be set.

## Returns

The result Modbus::StatusCode of the operation. Default implementation returns  $Status\_Bad \leftarrow IllegalFunction$ .

Reimplemented in ModbusClientPort.

# 7.9.2.16 writeSingleRegister()

Function for write one separate 16-bit holding register (4x).

# Parameters

in	unit	Address of the remote Modbus device.
in	offset	Starting offset (0-based).
in	value	16-bit unsigned integer value to be set.

#### Returns

The result Modbus::StatusCode of the operation. Default implementation returns Status\_Bad← IllegalFunction.

Reimplemented in ModbusClientPort.

The documentation for this class was generated from the following file:

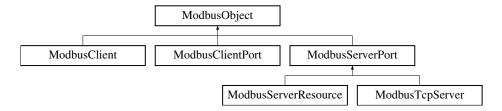
• c:/Users/march/Dropbox/PRJ/ModbusLib/src/Modbus.h

# 7.10 ModbusObject Class Reference

The ModbusObject class is the base class for objects that use signal/slot mechanism.

```
#include <ModbusObject.h>
```

Inheritance diagram for ModbusObject:



#### **Public Member Functions**

- ModbusObject ()
- virtual ∼ModbusObject ()
- const Modbus::Char \* objectName () const
- void setObjectName (const Modbus::Char \*name)
- template < class SignalClass , class T , class ReturnType , class ... Args > void connect (ModbusMethodPointer < SignalClass, ReturnType, Args ... > signalMethodPtr, T \*object, ModbusMethodPointer < T, ReturnType, Args ... > objectMethodPtr)
- template < class SignalClass , class ReturnType , class ... Args > void connect (ModbusMethodPointer < SignalClass, ReturnType, Args ... > signalMethodPtr, ModbusFunctionPointer < ReturnType, Args ... > funcPtr)
- template < class ReturnType , class ... Args > void disconnect (ModbusFunctionPointer < ReturnType, Args ... > funcPtr)
- void disconnectFunc (void \*funcPtr)
- template < class T , class ReturnType , class ... Args > void disconnect (T \*object, ModbusMethodPointer < T, ReturnType, Args ... > objectMethodPtr)
- template < class T >
   void disconnect (T \*object)

#### **Static Public Member Functions**

• static ModbusObject \* sender ()

#### **Protected Member Functions**

template < class T, class ... Args >
 void emitSignal (const char \*thisMethodId, ModbusMethodPointer < T, void, Args ... > thisMethod, Args ...
 args)

# 7.10.1 Detailed Description

The ModbusObject class is the base class for objects that use signal/slot mechanism.

ModbusObject is designed to be a base class for objects that need to use simplified Qt-like signal/slot mechanism. User can connect signal of the object he want to listen to his own function or method of his own class and then it can be disconnected if he is not interesting of this signal anymore. Callbacks will be called in order which it were connected.

ModbusObject has a map which key means signal identifier (pointer to signal) and value is a list of callbacks functions/methods connected to this signal.

ModbusObject has objectName() and setObjectName methods. This methods can be used to simply identify object which is signal's source (e.g. to print info in console).

Note

ModbusObject class is not thread safe

#### 7.10.2 Constructor & Destructor Documentation

## 7.10.2.1 ModbusObject()

```
ModbusObject::ModbusObject ()
```

Constructor of the class.

# 7.10.2.2 ~ModbusObject()

```
virtual ModbusObject::~ModbusObject () [virtual]
```

Virtual destructor of the class.

## 7.10.3 Member Function Documentation

# 7.10.3.1 connect() [1/2]

Same as ModbusObject::connect (ModbusMethodPointer, T\*, ModbusMethodPointer) but connects ModbusFunctionPointer to current object's signal signalMethodPtr.

#### 7.10.3.2 connect() [2/2]

# Note

MyClass c; MyReceiver r;

SignalClass template type refers to any class but it must be this or derived class. It makes separate SignalClass to easely refers signal of the derived class.

#### 7.10.3.3 disconnect() [1/3]

Disconnects function funcPtr from all signals of current object.

c.connect(&MyClass::signalSomething, r, &MyReceiver::slotSomething);

#### 7.10.3.4 disconnect() [2/3]

Disconnect all slots of T \*object from all signals of current object.

#### 7.10.3.5 disconnect() [3/3]

Disconnects slot represented by pair (object, objectMethodPtr) from all signals of current object.

#### 7.10.3.6 disconnectFunc()

Disconnects function funcPtr from all signals of current object, but funcPtr is a void pointer.

# 7.10.3.7 emitSignal()

Template method for emit signal. Must be called from within of the signal method.

#### 7.10.3.8 objectName()

```
const Modbus::Char * ModbusObject::objectName () const
```

Returns a pointer to current object's name string.

#### 7.10.3.9 sender()

```
static ModbusObject * ModbusObject::sender () [static]
```

Returns a pointer to the object that sent the signal. This pointer is valid in thread where signal was occured only. So this function must be called only within the slot that is a callback of signal occured.

#### 7.10.3.10 setObjectName()

Set name of current object.

The documentation for this class was generated from the following file:

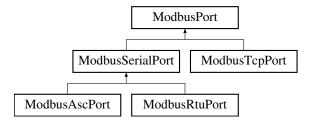
• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusObject.h

# 7.11 ModbusPort Class Reference

The abstract class  ${\tt ModbusPort}$  is the base class for a specific implementation of the  ${\tt ModbusPort}$  is the base class for a specific implementation of the  ${\tt ModbusPort}$  is the base class for a specific implementation of the  ${\tt ModbusPort}$  is the base class for a specific implementation of the  ${\tt ModbusPort}$  is the base class for a specific implementation of the  ${\tt ModbusPort}$  is the base class for a specific implementation of the  ${\tt ModbusPort}$  is the base class for a specific implementation of the  ${\tt ModbusPort}$  is the base class for a specific implementation of the  ${\tt ModbusPort}$  is the base class for a specific implementation of the  ${\tt Modbus}$  communication protocol.

```
#include <ModbusPort.h>
```

Inheritance diagram for ModbusPort:



#### **Public Member Functions**

- virtual ∼ModbusPort ()
- virtual Modbus::ProtocolType type () const =0
- virtual Modbus::Handle handle () const =0
- virtual Modbus::StatusCode open ()=0
- virtual Modbus::StatusCode close ()=0
- virtual bool isOpen () const =0
- virtual void setNextRequestRepeated (bool v)
- bool isChanged () const
- bool isServerMode () const
- virtual void setServerMode (bool mode)
- bool isBlocking () const
- bool isNonBlocking () const
- uint32\_t timeout () const
- · void setTimeout (uint32 t timeout)
- Modbus::StatusCode lastErrorStatus () const
- const Modbus::Char \* lastErrorText () const
- virtual Modbus::StatusCode writeBuffer (uint8\_t unit, uint8\_t func, uint8\_t \*buff, uint16\_t szInBuff)=0
- virtual Modbus::StatusCode readBuffer (uint8\_t &unit, uint8\_t &func, uint8\_t \*buff, uint16\_t maxSzBuff, uint16 t \*szOutBuff)=0
- virtual Modbus::StatusCode write ()=0
- virtual Modbus::StatusCode read ()=0
- virtual const uint8 t \* readBufferData () const =0
- virtual uint16\_t readBufferSize () const =0
- virtual const uint8\_t \* writeBufferData () const =0
- virtual uint16\_t writeBufferSize () const =0

#### **Protected Member Functions**

Modbus::StatusCode setError (Modbus::StatusCode status, const Modbus::Char \*text)

# 7.11.1 Detailed Description

The abstract class ModbusPort is the base class for a specific implementation of the Modbus communication protocol.

ModbusPort contains general functions for working with a specific port, implementing a specific version of the Modbus communication protocol. For example, versions for working with a TCP port or a serial port.

#### 7.11.2 Constructor & Destructor Documentation

#### 7.11.2.1 ~ ModbusPort()

virtual ModbusPort::~ModbusPort () [virtual]

Virtual destructor.

#### 7.11.3 Member Function Documentation

#### 7.11.3.1 close()

```
virtual Modbus::StatusCode ModbusPort::close () [pure virtual]
```

Closes the port (breaks the connection) and returns the status the result status.

Implemented in ModbusSerialPort, and ModbusTcpPort.

# 7.11.3.2 handle()

```
virtual Modbus::Handle ModbusPort::handle () const [pure virtual]
```

Returns the native handle value that depenp on OS used. For TCP it socket handle, for serial port - file handle.

Implemented in ModbusSerialPort, and ModbusTcpPort.

## 7.11.3.3 isBlocking()

```
bool ModbusPort::isBlocking () const
```

Returns true if the port works in synch (blocking) mode, false otherwise.

# 7.11.3.4 isChanged()

```
bool ModbusPort::isChanged () const
```

Returns true if the port settings have been changed and the port needs to be reopened/reestablished communication with the remote device, false otherwise.

# 7.11.3.5 isNonBlocking()

```
bool ModbusPort::isNonBlocking () const
```

Returns true if the port works in asynch (nonblocking) mode, false otherwise.

#### 7.11.3.6 isOpen()

```
virtual bool ModbusPort::isOpen () const [pure virtual]
```

Returns true if the port is open/communication with the remote device is established, false otherwise.

Implemented in ModbusSerialPort, and ModbusTcpPort.

## 7.11.3.7 isServerMode()

```
bool ModbusPort::isServerMode () const
```

Returns true if the port works in server mode, false otherwise.

# 7.11.3.8 lastErrorStatus()

```
Modbus::StatusCode ModbusPort::lastErrorStatus () const
```

Returns the status of the last error of the performed operation.

#### 7.11.3.9 lastErrorText()

```
const Modbus::Char * ModbusPort::lastErrorText () const
```

Returns the pointer to const Char text buffer of the last error of the performed operation.

#### 7.11.3.10 open()

```
virtual Modbus::StatusCode ModbusPort::open () [pure virtual]
```

Opens port (create connection) for further operations and returns the result status.

Implemented in ModbusSerialPort, and ModbusTcpPort.

# 7.11.3.11 read()

```
virtual Modbus::StatusCode ModbusPort::read () [pure virtual]
```

Implements the algorithm for reading from the port and returns the status of the operation.

Implemented in ModbusSerialPort, and ModbusTcpPort.

# 7.11.3.12 readBuffer()

The function parses the packet that the read() function puts into the buffer, checks it for correctness, extracts its parameters, and returns the status of the operation.

Implemented in ModbusAscPort, ModbusRtuPort, and ModbusTcpPort.

## 7.11.3.13 readBufferData()

```
virtual const uint8_t * ModbusPort::readBufferData () const [pure virtual]
```

Returns pointer to data of read buffer.

Implemented in ModbusSerialPort, and ModbusTcpPort.

# 7.11.3.14 readBufferSize()

```
virtual uint16_t ModbusPort::readBufferSize () const [pure virtual]
```

Returns size of data of read buffer.

Implemented in ModbusSerialPort, and ModbusTcpPort.

#### 7.11.3.15 setError()

Sets the error parameters of the last operation performed.

### 7.11.3.16 setNextRequestRepeated()

```
\label{eq:continuous} \begin{tabular}{ll} virtual void ModbusPort::setNextRequestRepeated ( \\ bool v) & [virtual] \end{tabular}
```

For the TCP version of the Modbus protocol. The identifier of each subsequent parcel is automatically increased by 1. If you set <code>setNextRequestRepeated(true)</code> then the next ID will not be increased by 1 but for only one next parcel.

Reimplemented in ModbusTcpPort.

#### 7.11.3.17 setServerMode()

Sets server mode if true, false for client mode.

# 7.11.3.18 setTimeout()

Sets the setting for the connection timeout of the remote device.

#### 7.11.3.19 timeout()

```
uint32_t ModbusPort::timeout () const
```

Returns the setting for the connection timeout of the remote device.

#### 7.11.3.20 type()

```
virtual Modbus::ProtocolType ModbusPort::type () const [pure virtual]
```

Returns the Modbus protocol type.

Implemented in ModbusAscPort, ModbusRtuPort, and ModbusTcpPort.

#### 7.11.3.21 write()

```
virtual Modbus::StatusCode ModbusPort::write () [pure virtual]
```

Implements the algorithm for writing to the port and returns the status of the operation.

Implemented in ModbusSerialPort, and ModbusTcpPort.

# 7.11.3.22 writeBuffer()

The function directly generates a packet and places it in the buffer for further sending. Returns the status of the operation.

Implemented in ModbusAscPort, ModbusRtuPort, and ModbusTcpPort.

## 7.11.3.23 writeBufferData()

```
virtual const uint8_t * ModbusPort::writeBufferData () const [pure virtual]
```

Returns pointer to data of write buffer.

Implemented in ModbusSerialPort, and ModbusTcpPort.

## 7.11.3.24 writeBufferSize()

```
virtual uint16_t ModbusPort::writeBufferSize () const [pure virtual]
```

Returns size of data of write buffer.

Implemented in ModbusSerialPort, and ModbusTcpPort.

The documentation for this class was generated from the following file:

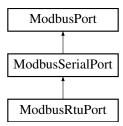
• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusPort.h

# 7.12 ModbusRtuPort Class Reference

Implements RTU version of the Modbus communication protocol.

#include <ModbusRtuPort.h>

Inheritance diagram for ModbusRtuPort:



#### **Public Member Functions**

- ModbusRtuPort (bool blocking=false)
- ∼ModbusRtuPort ()
- Modbus::ProtocolType type () const override

#### Public Member Functions inherited from ModbusSerialPort

- ∼ModbusSerialPort ()
- Modbus::Handle handle () const override
- Modbus::StatusCode open () override
- Modbus::StatusCode close () override
- bool isOpen () const override
- const Modbus::Char \* portName () const
- void setPortName (const Modbus::Char \*portName)
- int32\_t baudRate () const
- void setBaudRate (int32\_t baudRate)
- int8\_t dataBits () const
- void setDataBits (int8\_t dataBits)
- Modbus::Parity parity () const
- void setParity (Modbus::Parity parity)
- · Modbus::StopBits stopBits () const
- void setStopBits (Modbus::StopBits stopBits)
- Modbus::FlowControl flowControl () const
- void setFlowControl (Modbus::FlowControl flowControl)
- uint32\_t timeoutFirstByte () const
- void setTimeoutFirstByte (uint32\_t timeout)
- uint32\_t timeoutInterByte () const
- void setTimeoutInterByte (uint32\_t timeout)
- const uint8\_t \* readBufferData () const override
- uint16\_t readBufferSize () const override
- const uint8\_t \* writeBufferData () const override
- uint16\_t writeBufferSize () const override

#### Public Member Functions inherited from ModbusPort

- virtual ∼ModbusPort ()
- virtual void setNextRequestRepeated (bool v)
- bool isChanged () const
- bool isServerMode () const
- virtual void setServerMode (bool mode)
- bool isBlocking () const
- bool isNonBlocking () const
- uint32 t timeout () const
- void setTimeout (uint32\_t timeout)
- · Modbus::StatusCode lastErrorStatus () const
- const Modbus::Char \* lastErrorText () const

#### **Protected Member Functions**

- Modbus::StatusCode writeBuffer (uint8\_t unit, uint8\_t func, uint8\_t \*buff, uint16\_t szInBuff) override
- Modbus::StatusCode readBuffer (uint8\_t &unit, uint8\_t &func, uint8\_t \*buff, uint16\_t maxSzBuff, uint16\_t \*szOutBuff) override

## Protected Member Functions inherited from ModbusSerialPort

- Modbus::StatusCode write () override
- Modbus::StatusCode read () override

# Protected Member Functions inherited from ModbusPort

• Modbus::StatusCode setError (Modbus::StatusCode status, const Modbus::Char \*text)

#### 7.12.1 Detailed Description

Implements RTU version of the Modbus communication protocol.

 ${\tt ModbusRtuPort\ derived\ from\ ModbusSerialPort\ and\ implements\ write} Buffer\ and\ read Buffer\ for\ RTU\ version\ of\ Modbus\ communication\ protocol.$ 

#### 7.12.2 Constructor & Destructor Documentation

# 7.12.2.1 ModbusRtuPort()

```
ModbusRtuPort::ModbusRtuPort (
          bool blocking = false)
```

Constructor of the class. if blocking = true then defines blocking mode, non blocking otherwise.

## 7.12.2.2 ~ModbusRtuPort()

```
ModbusRtuPort::~ModbusRtuPort ()
```

Destructor of the class.

# 7.12.3 Member Function Documentation

# 7.12.3.1 readBuffer()

The function parses the packet that the read() function puts into the buffer, checks it for correctness, extracts its parameters, and returns the status of the operation.

Implements ModbusPort.

# 7.12.3.2 type()

```
Modbus::ProtocolType ModbusRtuPort::type () const [inline], [override], [virtual]
```

 $\textbf{Returns the Modbus protocol type. For \verb|ModbusAscPort| returns \verb|Modbus::RTU|.}$ 

Implements ModbusPort.

# 7.12.3.3 writeBuffer()

The function directly generates a packet and places it in the buffer for further sending. Returns the status of the operation.

Implements ModbusPort.

The documentation for this class was generated from the following file:

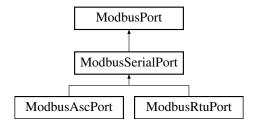
• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusRtuPort.h

# 7.13 ModbusSerialPort Class Reference

The abstract class ModbusSerialPort is the base class serial port Modbus communications.

#include <ModbusSerialPort.h>

Inheritance diagram for ModbusSerialPort:



#### Classes

struct Defaults

Holds the default values of the settings.

#### **Public Member Functions**

- ∼ModbusSerialPort ()
- Modbus::Handle handle () const override
- Modbus::StatusCode open () override
- Modbus::StatusCode close () override
- bool isOpen () const override
- const Modbus::Char \* portName () const
- void setPortName (const Modbus::Char \*portName)
- int32\_t baudRate () const
- void setBaudRate (int32\_t baudRate)
- int8\_t dataBits () const
- void setDataBits (int8 t dataBits)
- Modbus::Parity parity () const
- void setParity (Modbus::Parity parity)
- Modbus::StopBits stopBits () const
- void setStopBits (Modbus::StopBits stopBits)
- · Modbus::FlowControl flowControl () const
- void setFlowControl (Modbus::FlowControl flowControl)
- uint32\_t timeoutFirstByte () const
- void setTimeoutFirstByte (uint32\_t timeout)
- uint32\_t timeoutInterByte () const
- void setTimeoutInterByte (uint32\_t timeout)
- const uint8\_t \* readBufferData () const override
- uint16\_t readBufferSize () const override
- const uint8\_t \* writeBufferData () const override
- uint16\_t writeBufferSize () const override

#### Public Member Functions inherited from ModbusPort

- virtual ∼ModbusPort ()
- virtual Modbus::ProtocolType type () const =0
- virtual void setNextRequestRepeated (bool v)
- bool isChanged () const
- bool isServerMode () const
- virtual void setServerMode (bool mode)
- bool isBlocking () const
- · bool isNonBlocking () const
- uint32\_t timeout () const
- void setTimeout (uint32\_t timeout)
- Modbus::StatusCode lastErrorStatus () const
- const Modbus::Char \* lastErrorText () const
- virtual Modbus::StatusCode writeBuffer (uint8 t unit, uint8 t func, uint8 t \*buff, uint16 t szInBuff)=0
- virtual Modbus::StatusCode readBuffer (uint8\_t &unit, uint8\_t &func, uint8\_t \*buff, uint16\_t maxSzBuff, uint16\_t \*szOutBuff)=0

#### **Protected Member Functions**

- Modbus::StatusCode write () override
- Modbus::StatusCode read () override

# Protected Member Functions inherited from ModbusPort

• Modbus::StatusCode setError (Modbus::StatusCode status, const Modbus::Char \*text)

# 7.13.1 Detailed Description

The abstract class ModbusSerialPort is the base class serial port Modbus communications.

The abstract class ModbusSerialPort is the base class for a specific implementation of the Modbus communication protocol that using Serial Port. It implements functions which are common for the serial port: open, close, read and write.

# 7.13.2 Constructor & Destructor Documentation

# 7.13.2.1 ~ ModbusSerialPort()

ModbusSerialPort::~ModbusSerialPort ()

Virtual destructor. Closes serial port before destruction.

## 7.13.3 Member Function Documentation

# 7.13.3.1 baudRate()

int32\_t ModbusSerialPort::baudRate () const

Returns current serial port baud rate, e.g. 1200, 2400, 9600, 115200 etc.

#### 7.13.3.2 close()

```
Modbus::StatusCode ModbusSerialPort::close () [override], [virtual]
```

Close serial port and returns Modbus::Status\_Good.

Implements ModbusPort.

## 7.13.3.3 dataBits()

```
int8_t ModbusSerialPort::dataBits () const
```

Returns current serial port data bits, e.g. 5, 6, 7 or 8.

# 7.13.3.4 flowControl()

```
Modbus::FlowControl ModbusSerialPort::flowControl () const
```

Returns current serial port Modbus::FlowControl enum value.

# 7.13.3.5 handle()

```
Modbus::Handle ModbusSerialPort::handle () const [override], [virtual]
```

Returns native OS serial port handle, e.g.  ${\tt HANDLE}$  value for Windows.

Implements ModbusPort.

# 7.13.3.6 isOpen()

```
bool ModbusSerialPort::isOpen () const [override], [virtual]
```

Returns true if the serial port is open, false otherwise.

Implements ModbusPort.

# 7.13.3.7 open()

```
Modbus::StatusCode ModbusSerialPort::open () [override], [virtual]
```

Try to open serial port and returns Modbus::Status\_Good if success or Modbus::Status\_BadSerialOpen otherwise.

Implements ModbusPort.

# 7.13.3.8 parity()

```
Modbus::Parity ModbusSerialPort::parity () const
```

Returns current serial port Modbus::Parity enum value.

# 7.13.3.9 portName()

```
const Modbus::Char * ModbusSerialPort::portName () const
```

Returns current serial port name, e.g. COM1 for Windows or /dev/ttyS0 for Unix.

# 7.13.3.10 read()

```
Modbus::StatusCode ModbusSerialPort::read () [override], [protected], [virtual]
```

Implements the algorithm for reading from the port and returns the status of the operation.

Implements ModbusPort.

#### 7.13.3.11 readBufferData()

```
const uint8_t * ModbusSerialPort::readBufferData () const [override], [virtual]
```

Returns pointer to data of read buffer.

Implements ModbusPort.

# 7.13.3.12 readBufferSize()

```
uint16_t ModbusSerialPort::readBufferSize () const [override], [virtual]
```

Returns size of data of read buffer.

Implements ModbusPort.

# 7.13.3.13 setBaudRate()

Set current serial port baud rate.

# 7.13.3.14 setDataBits()

Set current serial port baud data bits.

## 7.13.3.15 setFlowControl()

Set current serial port Modbus::FlowControl enum value.

#### 7.13.3.16 setParity()

Set current serial port Modbus::Parity enum value.

# 7.13.3.17 setPortName()

Set current serial port name.

#### 7.13.3.18 setStopBits()

Set current serial port Modbus::StopBits enum value.

#### 7.13.3.19 setTimeoutFirstByte()

Set current serial port timeout of waiting first byte of incomming packet (in milliseconds).

# 7.13.3.20 setTimeoutInterByte()

Set current serial port timeout of waiting next byte (inter byte waiting tgimeout) of incomming packet (in milliseconds).

# 7.13.3.21 stopBits()

```
Modbus::StopBits ModbusSerialPort::stopBits () const
```

Returns current serial port  ${\tt Modbus::StopBits}$  enum value.

# 7.13.3.22 timeoutFirstByte()

```
uint32_t ModbusSerialPort::timeoutFirstByte () const [inline]
```

Returns current serial port timeout of waiting first byte of incomming packet (in milliseconds).

# 7.13.3.23 timeoutInterByte()

```
uint32_t ModbusSerialPort::timeoutInterByte () const
```

Returns current serial port timeout of waiting next byte (inter byte waiting tgimeout) of incomming packet (in milliseconds).

#### 7.13.3.24 write()

```
Modbus::StatusCode ModbusSerialPort::write () [override], [protected], [virtual]
```

Implements the algorithm for writing to the port and returns the status of the operation.

Implements ModbusPort.

# 7.13.3.25 writeBufferData()

```
const uint8_t * ModbusSerialPort::writeBufferData () const [override], [virtual]
```

Returns pointer to data of write buffer.

Implements ModbusPort.

# 7.13.3.26 writeBufferSize()

```
uint16_t ModbusSerialPort::writeBufferSize () const [override], [virtual]
```

Returns size of data of write buffer.

Implements ModbusPort.

The documentation for this class was generated from the following file:

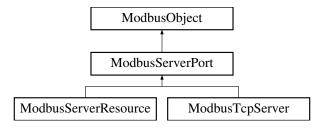
• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusSerialPort.h

# 7.14 ModbusServerPort Class Reference

Abstract base class for direct control of ModbusPort derived classes (TCP or serial) for server side.

#include <ModbusServerPort.h>

Inheritance diagram for ModbusServerPort:



#### **Public Member Functions**

- ModbusInterface \* device () const
- void setDevice (ModbusInterface \*device)
- virtual Modbus::ProtocolType type () const =0
- virtual bool isTcpServer () const
- virtual Modbus::StatusCode open ()=0
- virtual Modbus::StatusCode close ()=0
- virtual bool isOpen () const =0
- bool isBroadcastEnabled () const
- virtual void setBroadcastEnabled (bool enable)
- const void \* unitMap () const
- virtual void setUnitMap (const void \*unitmap)
- void \* context () const
- void setContext (void \*context)
- virtual Modbus::StatusCode process ()=0
- bool isStateClosed () const
- void signalOpened (const Modbus::Char \*source)
- void signalClosed (const Modbus::Char \*source)
- void signalTx (const Modbus::Char \*source, const uint8 t \*buff, uint16 t size)
- void signalRx (const Modbus::Char \*source, const uint8 t \*buff, uint16 t size)
- void signalError (const Modbus::Char \*source, Modbus::StatusCode status, const Modbus::Char \*text)

# Public Member Functions inherited from ModbusObject

- ModbusObject ()
- virtual ∼ModbusObject ()
- const Modbus::Char \* objectName () const
- void setObjectName (const Modbus::Char \*name)
- template < class SignalClass , class T , class ReturnType , class ... Args > void connect (ModbusMethodPointer < SignalClass, ReturnType, Args ... > signalMethodPtr, T \*object, ModbusMethodPointer < T, ReturnType, Args ... > objectMethodPtr)
- template < class SignalClass , class ReturnType , class ... Args > void connect (ModbusMethodPointer < SignalClass, ReturnType, Args ... > signalMethodPtr, ModbusFunctionPointer < ReturnType, Args ... > funcPtr)
- template<class ReturnType , class ... Args> void disconnect (ModbusFunctionPointer< ReturnType, Args ... > funcPtr)
- void disconnectFunc (void \*funcPtr)
- template < class T , class ReturnType , class ... Args > void disconnect (T \*object, ModbusMethodPointer < T, ReturnType, Args ... > objectMethodPtr)
- template < class T >
   void disconnect (T \*object)

#### **Protected Member Functions**

ModbusObject ()

# Protected Member Functions inherited from ModbusObject

template < class T , class ... Args >
 void emitSignal (const char \*thisMethodId, ModbusMethodPointer < T, void, Args ... > thisMethod, Args ...
 args)

#### **Additional Inherited Members**

# Static Public Member Functions inherited from ModbusObject

static ModbusObject \* sender ()

# 7.14.1 Detailed Description

Abstract base class for direct control of ModbusPort derived classes (TCP or serial) for server side.

Pointer to ModbusPort object must be passed to ModbusServerPort derived class constructor.

Also assumed that ModbusServerPort derived classes must accept ModbusInterface object in its constructor to process every Modbus function request.

#### 7.14.2 Member Function Documentation

## 7.14.2.1 close()

```
virtual Modbus::StatusCode ModbusServerPort::close () [pure virtual]
```

Closes port/connection and returns status of the operation.

Implemented in ModbusServerResource, and ModbusTcpServer.

# 7.14.2.2 context()

```
void * ModbusServerPort::context () const
```

Return context of the port previously set by setContext function or nullptr by default.

#### 7.14.2.3 device()

```
ModbusInterface * ModbusServerPort::device () const
```

Returns pointer to  ${\tt ModbusInterface}$  object/device that was previously passed in constructor. This device must process every input  ${\tt Modbus}$  function request for this server port.

## 7.14.2.4 isBroadcastEnabled()

```
bool ModbusServerPort::isBroadcastEnabled () const
```

Returns true if broadcast mode for 0 unit address is enabled, false otherwise. Broadcast mode for 0 unit address is required by  $\frac{1}{2}$  Modbus protocol so it is enabled by default

# 7.14.2.5 isOpen()

```
virtual bool ModbusServerPort::isOpen () const [pure virtual]
```

Returns true if inner port is open, false otherwise.

Implemented in ModbusServerResource, and ModbusTcpServer.

# 7.14.2.6 isStateClosed()

```
bool ModbusServerPort::isStateClosed () const
```

Returns true if current port has closed inner state, false otherwise.

# 7.14.2.7 isTcpServer()

```
virtual bool ModbusServerPort::isTcpServer () const [virtual]
```

Returns true if current server port is TCP server, false otherwise.

Reimplemented in ModbusTcpServer.

#### 7.14.2.8 ModbusObject()

```
ModbusObject::ModbusObject () [protected]
```

Constructor of the class.

# 7.14.2.9 open()

```
virtual Modbus::StatusCode ModbusServerPort::open () [pure virtual]
```

Open inner port/connection to begin working and returns status of the operation. User do not need to call this method directly.

Implemented in ModbusServerResource, and ModbusTcpServer.

#### 7.14.2.10 process()

```
virtual Modbus::StatusCode ModbusServerPort::process () [pure virtual]
```

Main function of the class. Must be called in the cycle. Return statuc code is not very useful but can indicate that inner server operations are good, bad or in process.

Implemented in ModbusServerResource, and ModbusTcpServer.

#### 7.14.2.11 setBroadcastEnabled()

Enables broadcast mode for 0 unit address. It is enabled by default.

See also

```
isBroadcastEnabled()
```

Reimplemented in ModbusTcpServer.

#### 7.14.2.12 setContext()

```
void ModbusServerPort::setContext (
     void * context)
```

Set context of the port.

#### 7.14.2.13 setDevice()

Set pointer to ModbusInterface object/device to transfer all request ot it. This device must process every input Modbus function request for this server port.

#### 7.14.2.14 setUnitMap()

Set units map of current server. Server make a copy of units map data.

See also

```
unitMap()
```

Reimplemented in ModbusTcpServer.

## 7.14.2.15 signalClosed()

Signal occured when inner port was closed. source - current port name.

# 7.14.2.16 signalError()

Signal occured when error is occured with error's status and text. source - current port name.

# 7.14.2.17 signalOpened()

Signal occured when inner port was opened. source - current port name.

## 7.14.2.18 signalRx()

Signal occured when the incoming packet 'Rx' from the internal list of callbacks, passing them the input array 'buff' and its size 'size'. source - current port name.

# 7.14.2.19 signalTx()

Signal occured when the original packet 'Tx' from the internal list of callbacks, passing them the original array 'buff' and its size 'size'. source - current port name.

## 7.14.2.20 type()

```
virtual Modbus::ProtocolType ModbusServerPort::type () const [pure virtual]
```

Returns type of Modbus protocol.

 $Implemented\ in\ ModbusServerResource,\ and\ ModbusTcpServer.$ 

## 7.14.2.21 unitMap()

const void \* ModbusServerPort::unitMap () const

Return pointer to the units map byte array of the current server. By default unit map is not set so return value is nullptr. Unit map is data type with size of 32 bytes in which every bit represents unit address from 0 to 255. So bit 0 of byte 0 represents unit address 0, bit 1 of byte 0 represents unit address 1 and so on. Bit 0 of byte 1 represent unit address 8, bit 7 of byte 31 represents unit address 255. If set unit map can enable or disable (depends on respecting 1/0 bit value) unit address for further processing. It is not set by default and function returns nullptr.

The documentation for this class was generated from the following file:

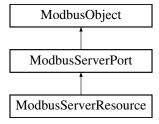
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusServerPort.h

## 7.15 ModbusServerResource Class Reference

Implements direct control for ModbusPort derived classes (TCP or serial) for server side.

#include <ModbusServerResource.h>

Inheritance diagram for ModbusServerResource:



#### **Public Member Functions**

- ModbusServerResource (ModbusPort \*port, ModbusInterface \*device)
- ModbusPort \* port () const
- Modbus::ProtocolType type () const override
- Modbus::StatusCode open () override
- · Modbus::StatusCode close () override
- · bool isOpen () const override
- · Modbus::StatusCode process () override

#### Public Member Functions inherited from ModbusServerPort

- ModbusInterface \* device () const
- void setDevice (ModbusInterface \*device)
- virtual bool isTcpServer () const
- bool isBroadcastEnabled () const
- virtual void setBroadcastEnabled (bool enable)
- const void \* unitMap () const
- virtual void setUnitMap (const void \*unitmap)
- void \* context () const
- void setContext (void \*context)
- bool isStateClosed () const
- void signalOpened (const Modbus::Char \*source)
- void signalClosed (const Modbus::Char \*source)
- void signalTx (const Modbus::Char \*source, const uint8\_t \*buff, uint16\_t size)
- void signalRx (const Modbus::Char \*source, const uint8\_t \*buff, uint16\_t size)
- void signalError (const Modbus::Char \*source, Modbus::StatusCode status, const Modbus::Char \*text)

# Public Member Functions inherited from ModbusObject

- ModbusObject ()
- virtual ∼ModbusObject ()
- const Modbus::Char \* objectName () const
- void setObjectName (const Modbus::Char \*name)
- template < class SignalClass , class T , class ReturnType , class ... Args > void connect (ModbusMethodPointer < SignalClass, ReturnType, Args ... > signalMethodPtr, T \*object, ModbusMethodPointer < T, ReturnType, Args ... > objectMethodPtr)
- template < class SignalClass , class ReturnType , class ... Args > void connect (ModbusMethodPointer < SignalClass, ReturnType, Args ... > signalMethodPtr, ModbusFunctionPointer < ReturnType, Args ... > funcPtr)
- template<class ReturnType , class ... Args> void disconnect (ModbusFunctionPointer< ReturnType, Args ... > funcPtr)
- void disconnectFunc (void \*funcPtr)
- template < class T , class ReturnType , class ... Args > void disconnect (T \*object, ModbusMethodPointer < T, ReturnType, Args ... > objectMethodPtr)
- template < class T > void disconnect (T \*object)

#### **Protected Member Functions**

- virtual Modbus::StatusCode processInputData (const uint8\_t \*buff, uint16\_t sz)
- virtual Modbus::StatusCode processDevice ()
- virtual Modbus::StatusCode processOutputData (uint8\_t \*buff, uint16\_t &sz)

#### Protected Member Functions inherited from ModbusServerPort

• ModbusObject ()

#### Protected Member Functions inherited from ModbusObject

```
    template < class T , class ... Args > void emitSignal (const char *thisMethodId, ModbusMethodPointer < T, void, Args ... > thisMethod, Args ... args)
```

#### **Additional Inherited Members**

# Static Public Member Functions inherited from ModbusObject

static ModbusObject \* sender ()

#### 7.15.1 Detailed Description

Implements direct control for ModbusPort derived classes (TCP or serial) for server side.

ModbusServerResource derived from ModbusServerPort and makes ModbusPort object behaves like server port. Pointer to ModbusPort object is passed to ModbusServerResource constructor.

Also  ${\tt ModbusServerResource}$  have  ${\tt ModbusInterface}$  object as second parameter of constructor which process every  ${\tt Modbus}$  function request.

# 7.15.2 Constructor & Destructor Documentation

# 7.15.2.1 ModbusServerResource()

Constructor of the class.

#### **Parameters**

in	port	Pointer to the ModbusPort which is managed by the current class object.	
in	device	Pointer to the ModbusInterface implementation to which all requests for Modbus	
		functions are forwarded.	

#### 7.15.3 Member Function Documentation

#### 7.15.3.1 close()

```
Modbus::StatusCode ModbusServerResource::close () [override], [virtual]
```

Closes port/connection and returns status of the operation.

Implements ModbusServerPort.

# 7.15.3.2 isOpen()

```
bool ModbusServerResource::isOpen () const [override], [virtual]
```

Returns true if inner port is open, false otherwise.

Implements ModbusServerPort.

## 7.15.3.3 open()

```
Modbus::StatusCode ModbusServerResource::open () [override], [virtual]
```

Open inner port/connection to begin working and returns status of the operation. User do not need to call this method directly.

Implements ModbusServerPort.

# 7.15.3.4 port()

```
ModbusPort * ModbusServerResource::port () const
```

Returns pointer to inner port which was previously passed in constructor.

#### 7.15.3.5 process()

```
Modbus::StatusCode ModbusServerResource::process () [override], [virtual]
```

Main function of the class. Must be called in the cycle. Return statuc code is not very useful but can indicate that inner server operations are good, bad or in process.

Implements ModbusServerPort.

# 7.15.3.6 processDevice()

```
virtual Modbus::StatusCode ModbusServerResource::processDevice () [protected], [virtual]
```

Transfer input request Modbus function to inner device and returns status of the operation.

# 7.15.3.7 processInputData()

Process input data buff with size and returns status of the operation.

# 7.15.3.8 processOutputData()

Process output data buff with size and returns status of the operation.

# 7.15.3.9 type()

```
Modbus::ProtocolType ModbusServerResource::type () const [override], [virtual]
```

Returns type of Modbus protocol. Same as port () ->type ().

Implements ModbusServerPort.

The documentation for this class was generated from the following file:

• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusServerResource.h

# 7.16 ModbusSlotBase< ReturnType, Args > Class Template Reference

ModbusSlotBase base template for slot (method or function)

```
#include <ModbusObject.h>
```

#### **Public Member Functions**

- virtual ∼ModbusSlotBase ()
- virtual void \* object () const
- virtual void \* methodOrFunction () const =0
- virtual ReturnType exec (Args ... args)=0

# 7.16.1 Detailed Description

```
template<class ReturnType, class ... Args> class ModbusSlotBase< ReturnType, Args >
```

ModbusSlotBase base template for slot (method or function)

#### 7.16.2 Constructor & Destructor Documentation

# 7.16.2.1 ∼ModbusSlotBase()

```
template<class ReturnType , class ... Args>
virtual ModbusSlotBase< ReturnType, Args >::~ModbusSlotBase () [inline], [virtual]
```

Virtual destructor of the class

# 7.16.3 Member Function Documentation

# 7.16.3.1 exec()

Execute method or function slot

Implemented in ModbusSlotFunction < ReturnType, Args >, and ModbusSlotMethod < T, ReturnType, Args >.

# 7.16.3.2 methodOrFunction()

```
template<class ReturnType , class ... Args>
virtual void * ModbusSlotBase< ReturnType, Args >::methodOrFunction () const [pure virtual]
```

Return pointer to method (in case of method slot) or function (in case of function slot)

Implemented in ModbusSlotFunction < ReturnType, Args >, and ModbusSlotMethod < T, ReturnType, Args >.

## 7.16.3.3 object()

```
template<class ReturnType , class ... Args>
virtual void * ModbusSlotBase< ReturnType, Args >::object () const [inline], [virtual]
```

Return pointer to object which method belongs to (in case of method slot) or nullptr in case of function slot Reimplemented in ModbusSlotMethod < T, ReturnType, Args >.

The documentation for this class was generated from the following file:

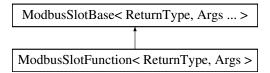
• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusObject.h

# 7.17 ModbusSlotFunction< ReturnType, Args > Class Template Reference

 ${\tt ModbusSlotFunction}\ \textbf{template class hold pointer to slot function}$ 

```
#include <ModbusObject.h>
```

Inheritance diagram for ModbusSlotFunction < ReturnType, Args >:



#### **Public Member Functions**

- ModbusSlotFunction (ModbusFunctionPointer< ReturnType, Args... > funcPtr)
- void \* methodOrFunction () const override
- ReturnType exec (Args ... args) override

# Public Member Functions inherited from ModbusSlotBase< ReturnType, Args ... >

- virtual ∼ModbusSlotBase ()
- virtual void \* object () const

# 7.17.1 Detailed Description

```
template<class ReturnType, class ... Args> class ModbusSlotFunction< ReturnType, Args >
```

ModbusSlotFunction template class hold pointer to slot function

# 7.17.2 Constructor & Destructor Documentation

## 7.17.2.1 ModbusSlotFunction()

Constructor of the slot.

#### **Parameters**

in <i>funcPtr</i>	Pointer to slot function.
-------------------	---------------------------

# 7.17.3 Member Function Documentation

#### 7.17.3.1 exec()

Execute method or function slot

Implements ModbusSlotBase< ReturnType, Args ... >.

#### 7.17.3.2 methodOrFunction()

```
template<class ReturnType , class ... Args>
void * ModbusSlotFunction< ReturnType, Args >::methodOrFunction () const [inline], [override],
[virtual]
```

Return pointer to method (in case of method slot) or function (in case of function slot)

 $Implements\ ModbusSlotBase < ReturnType,\ Args\ ...\ >.$ 

The documentation for this class was generated from the following file:

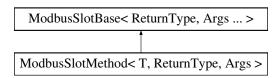
• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusObject.h

# 7.18 ModbusSlotMethod< T, ReturnType, Args > Class Template Reference

 ${\tt ModbusSlotMethod} \ template \ class \ hold \ pointer \ to \ object \ and \ its \ method$ 

```
#include <ModbusObject.h>
```

 $Inheritance\ diagram\ for\ ModbusSlotMethod{<}\ T,\ ReturnType,\ Args>:$ 



#### **Public Member Functions**

- ModbusSlotMethod (T \*object, ModbusMethodPointer< T, ReturnType, Args... > methodPtr)
- void \* object () const override
- void \* methodOrFunction () const override
- ReturnType exec (Args ... args) override

# Public Member Functions inherited from ModbusSlotBase< ReturnType, Args ... >

virtual ∼ModbusSlotBase ()

# 7.18.1 Detailed Description

```
template<class T, class ReturnType, class ... Args> class ModbusSlotMethod< T, ReturnType, Args >
```

ModbusSlotMethod template class hold pointer to object and its method

#### 7.18.2 Constructor & Destructor Documentation

#### 7.18.2.1 ModbusSlotMethod()

Constructor of the slot.

#### **Parameters**

in	object	Pointer to object.
in	methodPtr	Pointer to object's method.

# 7.18.3 Member Function Documentation

## 7.18.3.1 exec()

Execute method or function slot

Implements ModbusSlotBase< ReturnType, Args ... >.

#### 7.18.3.2 methodOrFunction()

```
template<class T , class ReturnType , class ... Args>
void * ModbusSlotMethod< T, ReturnType, Args >::methodOrFunction () const [inline], [override],
[virtual]
```

Return pointer to method (in case of method slot) or function (in case of function slot)

 $Implements\ ModbusSlotBase < ReturnType,\ Args\ ...\ >.$ 

# 7.18.3.3 object()

```
template<class T , class ReturnType , class ... Args>
void * ModbusSlotMethod< T, ReturnType, Args >::object () const [inline], [override], [virtual]
```

Return pointer to object which method belongs to (in case of method slot) or nullptr in case of function slot

Reimplemented from ModbusSlotBase< ReturnType, Args ... >.

The documentation for this class was generated from the following file:

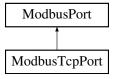
• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusObject.h

# 7.19 ModbusTcpPort Class Reference

 ${\bf Class\ ModbusTcpPort\ implements\ TCP\ version\ of\ {\color{blue}Modbus\ protocol.}}$ 

```
#include <ModbusTcpPort.h>
```

Inheritance diagram for ModbusTcpPort:



#### **Classes**

struct Defaults

Defaults class constain default settings values for ModbusTcpPort.

#### **Public Member Functions**

- ModbusTcpPort (ModbusTcpSocket \*socket, bool blocking=false)
- ModbusTcpPort (bool blocking=false)
- ∼ModbusTcpPort ()
- Modbus::ProtocolType type () const override
- · Modbus::Handle handle () const override
- · Modbus::StatusCode open () override
- · Modbus::StatusCode close () override
- · bool isOpen () const override
- const Modbus::Char \* host () const
- void setHost (const Modbus::Char \*host)
- uint16\_t port () const
- void setPort (uint16\_t port)
- void setNextRequestRepeated (bool v) override
- bool autoIncrement () const
- const uint8 t \* readBufferData () const override
- uint16\_t readBufferSize () const override
- const uint8 t \* writeBufferData () const override
- uint16\_t writeBufferSize () const override

#### Public Member Functions inherited from ModbusPort

- virtual ∼ModbusPort ()
- · bool isChanged () const
- bool isServerMode () const
- · virtual void setServerMode (bool mode)
- bool isBlocking () const
- bool isNonBlocking () const
- uint32\_t timeout () const
- void setTimeout (uint32 t timeout)
- Modbus::StatusCode lastErrorStatus () const
- const Modbus::Char \* lastErrorText () const

# **Protected Member Functions**

- Modbus::StatusCode write () override
- Modbus::StatusCode read () override
- Modbus::StatusCode writeBuffer (uint8\_t unit, uint8\_t func, uint8\_t \*buff, uint16\_t szInBuff) override
- Modbus::StatusCode readBuffer (uint8\_t &unit, uint8\_t &func, uint8\_t \*buff, uint16\_t maxSzBuff, uint16\_t \*szOutBuff) override

#### Protected Member Functions inherited from ModbusPort

• Modbus::StatusCode setError (Modbus::StatusCode status, const Modbus::Char \*text)

#### 7.19.1 Detailed Description

Class ModbusTcpPort implements TCP version of Modbus protocol.

ModbusPort contains function to work with TCP-port (connection).

#### 7.19.2 Constructor & Destructor Documentation

#### 7.19.2.1 ModbusTcpPort() [1/2]

Constructor of the class.

#### 7.19.2.2 ModbusTcpPort() [2/2]

```
ModbusTcpPort::ModbusTcpPort (
          bool blocking = false)
```

Constructor of the class.

#### 7.19.2.3 ∼ModbusTcpPort()

```
ModbusTcpPort::~ModbusTcpPort ()
```

Destructor of the class. Close socket if it was not closed previously

# 7.19.3 Member Function Documentation

#### 7.19.3.1 autoIncrement()

```
bool ModbusTcpPort::autoIncrement () const
```

Returns 'true' if the identifier of each subsequent parcel is automatically incremented by 1, 'false' otherwise.

# 7.19.3.2 close()

```
Modbus::StatusCode ModbusTcpPort::close () [override], [virtual]
```

Closes the port (breaks the connection) and returns the status the result status.

Implements ModbusPort.

#### 7.19.3.3 handle()

```
Modbus::Handle ModbusTcpPort::handle () const [override], [virtual]
```

Native OS handle for the socket.

Implements ModbusPort.

#### 7.19.3.4 host()

```
const Modbus::Char * ModbusTcpPort::host () const
```

Returns the settings for the IP address or DNS name of the remote device.

#### 7.19.3.5 isOpen()

```
bool ModbusTcpPort::isOpen () const [override], [virtual]
```

Returns true if the port is open/communication with the remote device is established, false otherwise.

Implements ModbusPort.

#### 7.19.3.6 open()

```
Modbus::StatusCode ModbusTcpPort::open () [override], [virtual]
```

Opens port (create connection) for further operations and returns the result status.

Implements ModbusPort.

#### 7.19.3.7 port()

```
uint16_t ModbusTcpPort::port () const
```

Returns the setting for the TCP port number of the remote device.

#### 7.19.3.8 read()

```
Modbus::StatusCode ModbusTcpPort::read () [override], [protected], [virtual]
```

Implements the algorithm for reading from the port and returns the status of the operation.

Implements ModbusPort.

# 7.19.3.9 readBuffer()

The function parses the packet that the read() function puts into the buffer, checks it for correctness, extracts its parameters, and returns the status of the operation.

Implements ModbusPort.

#### 7.19.3.10 readBufferData()

```
const uint8_t * ModbusTcpPort::readBufferData () const [override], [virtual]
```

Returns pointer to data of read buffer.

Implements ModbusPort.

#### 7.19.3.11 readBufferSize()

```
uint16_t ModbusTcpPort::readBufferSize () const [override], [virtual]
```

Returns size of data of read buffer.

Implements ModbusPort.

#### 7.19.3.12 setHost()

Sets the settings for the IP address or DNS name of the remote device.

# 7.19.3.13 setNextRequestRepeated()

Repeat next request parameters (for Modbus TCP transaction Id).

Reimplemented from ModbusPort.

# 7.19.3.14 setPort()

Sets the settings for the TCP port number of the remote device.

# 7.19.3.15 type()

```
Modbus::ProtocolType ModbusTcpPort::type () const [inline], [override], [virtual]
```

Returns the Modbus protocol type. In this case it is Modbus::TCP.

Implements ModbusPort.

#### 7.19.3.16 write()

```
Modbus::StatusCode ModbusTcpPort::write () [override], [protected], [virtual]
```

Implements the algorithm for writing to the port and returns the status of the operation.

Implements ModbusPort.

#### 7.19.3.17 writeBuffer()

The function directly generates a packet and places it in the buffer for further sending. Returns the status of the operation.

Implements ModbusPort.

#### 7.19.3.18 writeBufferData()

```
const uint8_t * ModbusTcpPort::writeBufferData () const [override], [virtual]
```

Returns pointer to data of write buffer.

Implements ModbusPort.

#### 7.19.3.19 writeBufferSize()

```
uint16_t ModbusTcpPort::writeBufferSize () const [override], [virtual]
```

Returns size of data of write buffer.

Implements ModbusPort.

The documentation for this class was generated from the following file:

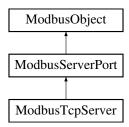
• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusTcpPort.h

# 7.20 ModbusTcpServer Class Reference

The ModbusTcpServer class implements TCP server part of the Modbus protocol.

```
#include <ModbusTcpServer.h>
```

Inheritance diagram for ModbusTcpServer:



#### Classes

struct Defaults

Defaults class constain default settings values for ModbusTcpServer.

#### **Public Member Functions**

- ModbusTcpServer (ModbusInterface \*device)
- ∼ModbusTcpServer ()
- uint16\_t port () const
- void setPort (uint16\_t port)
- uint32\_t timeout () const
- void setTimeout (uint32 t timeout)
- Modbus::ProtocolType type () const override
- bool isTcpServer () const override
- Modbus::StatusCode open () override
- Modbus::StatusCode close () override
- · bool isOpen () const override
- void setBroadcastEnabled (bool enable) override
- void setUnitMap (const void \*unitmap) override
- Modbus::StatusCode process () override
- virtual ModbusServerPort \* createTcpPort (ModbusTcpSocket \*socket)
- virtual void deleteTcpPort (ModbusServerPort \*port)
- void signalNewConnection (const Modbus::Char \*source)
- void signalCloseConnection (const Modbus::Char \*source)

#### Public Member Functions inherited from ModbusServerPort

- ModbusInterface \* device () const
- void setDevice (ModbusInterface \*device)
- bool isBroadcastEnabled () const
- const void \* unitMap () const
- void \* context () const
- void setContext (void \*context)
- bool isStateClosed () const
- void signalOpened (const Modbus::Char \*source)
- void signalClosed (const Modbus::Char \*source)
- void signalTx (const Modbus::Char \*source, const uint8 t \*buff, uint16 t size)
- void signalRx (const Modbus::Char \*source, const uint8\_t \*buff, uint16\_t size)
- void signalError (const Modbus::Char \*source, Modbus::StatusCode status, const Modbus::Char \*text)

# Public Member Functions inherited from ModbusObject

- ModbusObject ()
- virtual ∼ModbusObject ()
- const Modbus::Char \* objectName () const
- void setObjectName (const Modbus::Char \*name)
- template < class SignalClass , class T , class ReturnType , class ... Args > void connect (ModbusMethodPointer < SignalClass, ReturnType, Args ... > signalMethodPtr, T \*object, ModbusMethodPointer < T, ReturnType, Args ... > objectMethodPtr)

- template < class SignalClass , class ReturnType , class ... Args > void connect (ModbusMethodPointer < SignalClass, ReturnType, Args ... > signalMethodPtr, ModbusFunctionPointer < ReturnType, Args ... > funcPtr)
   template < class ReturnType , class ... Args > void disconnect (ModbusFunctionPointer < ReturnType, Args ... > funcPtr)
   void disconnectFunc (void \*funcPtr)
- template < class T , class ReturnType , class ... Args > void disconnect (T \*object, ModbusMethodPointer < T, ReturnType, Args ... > objectMethodPtr)
- template < class T >
   void disconnect (T \*object)

#### **Protected Member Functions**

- ModbusTcpSocket \* nextPendingConnection ()
- void clearConnections ()

#### Protected Member Functions inherited from ModbusServerPort

ModbusObject ()

# Protected Member Functions inherited from ModbusObject

```
    template < class T , class ... Args > void emitSignal (const char *thisMethodId, ModbusMethodPointer < T, void, Args ... > thisMethod, Args ... args)
```

#### **Additional Inherited Members**

# Static Public Member Functions inherited from ModbusObject

• static ModbusObject \* sender ()

# 7.20.1 Detailed Description

The ModbusTcpServer class implements TCP server part of the Modbus protocol.

ModbusTcpServer...

# 7.20.2 Constructor & Destructor Documentation

# 7.20.2.1 ModbusTcpServer()

Constructor of the class. device param is object which might process incoming requests for read/write memory.

#### 7.20.2.2 ~ModbusTcpServer()

```
ModbusTcpServer::~ModbusTcpServer ()
```

Destructor of the class. Clear all unclosed connections.

#### 7.20.3 Member Function Documentation

#### 7.20.3.1 clearConnections()

```
void ModbusTcpServer::clearConnections () [protected]
```

Clear all allocated memory for previously established connections.

#### 7.20.3.2 close()

```
Modbus::StatusCode ModbusTcpServer::close () [override], [virtual]
```

Stop listening for incoming connections and close all previously opened connections.

Returns

- Modbus::Status\_Good on success
- Modbus::Status\_Processing when operation is not complete

Implements ModbusServerPort.

# 7.20.3.3 createTcpPort()

 $\label{thm:continuous} \textbf{Creates} \ \ \underline{\textbf{ModbusServerPort}} \ \ \textbf{for new incoming connection defined by} \ \ \underline{\textbf{ModbusTcpSocket pointer May be}} \ \ \textbf{reimplemented in subclasses}.$ 

#### 7.20.3.4 deleteTcpPort()

Deletes ModbusServerPort by default. May be reimplemented in subclasses.

# 7.20.3.5 isOpen()

```
bool ModbusTcpServer::isOpen () const [override], [virtual]
```

Returns true if the server is currently listening for incoming connections, false otherwise.

Implements ModbusServerPort.

#### 7.20.3.6 isTcpServer()

```
bool ModbusTcpServer::isTcpServer () const [inline], [override], [virtual]
```

Returns true.

Reimplemented from ModbusServerPort.

#### 7.20.3.7 nextPendingConnection()

```
{\tt ModbusTcpSocket} \ * \ {\tt ModbusTcpServer::} nextPendingConnection \ () \quad [protected]
```

Checks for incoming connections and returns pointer ModbusTcpSocket if new connection established, nullptr otherwise.

#### 7.20.3.8 open()

```
Modbus::StatusCode ModbusTcpServer::open () [override], [virtual]
```

Try to listen for incoming connections on TCP port that was previously set (port ()).

#### Returns

- Modbus::Status\_Good on success
- Modbus::Status\_Processing when operation is not complete
- Modbus::Status\_BadTcpCreate when can't create TCP socket
- Modbus::Status\_BadTcpBind when can't bind TCP socket
- ${\tt Modbus::Status\_BadTcpListen}$  when can't listen TCP socket

Implements ModbusServerPort.

#### 7.20.3.9 port()

```
uint16_t ModbusTcpServer::port () const
```

Returns the setting for the TCP port number of the server.

# 7.20.3.10 process()

```
Modbus::StatusCode ModbusTcpServer::process () [override], [virtual]
```

Main function of TCP server. Must be called in cycle to perform all incoming TCP connections.

Implements ModbusServerPort.

#### 7.20.3.11 setBroadcastEnabled()

Enables broadcast mode for 0 unit address. It is enabled by default.

See also

```
isBroadcastEnabled()
```

Reimplemented from ModbusServerPort.

#### 7.20.3.12 setPort()

Sets the settings for the TCP port number of the server.

# 7.20.3.13 setTimeout()

Sets the setting for the read timeout of every single conncetion.

# 7.20.3.14 setUnitMap()

Set units map of current server. Server make a copy of units map data.

See also

```
unitMap()
```

Reimplemented from ModbusServerPort.

# 7.20.3.15 signalCloseConnection()

Signal occured when TCP connection was closed. source - name of the current connection.

#### 7.20.3.16 signalNewConnection()

Signal occured when new TCP connection was accepted. source - name of the current connection.

#### 7.20.3.17 timeout()

```
uint32_t ModbusTcpServer::timeout () const
```

Returns the setting for the read timeout of every single conncetion.

## 7.20.3.18 type()

```
Modbus::ProtocolType ModbusTcpServer::type () const [inline], [override], [virtual]
```

Returns the Modbus protocol type. In this case it is Modbus::TCP.

Implements ModbusServerPort.

The documentation for this class was generated from the following file:

• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusTcpServer.h

# 7.21 Modbus::SerialSettings Struct Reference

Struct to define settings for Serial Port.

```
#include <ModbusGlobal.h>
```

#### **Public Attributes**

const Char \* portName

Value for the serial port name.

· int32 t baudRate

Value for the serial port's baud rate.

int8\_t dataBits

Value for the serial port's data bits.

Parity parity

Value for the serial port's patiry.

StopBits stopBits

Value for the serial port's stop bits.

FlowControl flowControl

Value for the serial port's flow control.

• uint32\_t timeoutFirstByte

Value for the serial port's timeout waiting first byte of packet.

• uint32\_t timeoutInterByte

Value for the serial port's timeout waiting next byte of packet.

# 7.21.1 Detailed Description

Struct to define settings for Serial Port.

The documentation for this struct was generated from the following file:

• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusGlobal.h

# 7.22 Modbus::Strings Class Reference

Sets constant key values for the map of settings.

```
#include <ModbusQt.h>
```

#### **Public Member Functions**

• Strings ()

#### **Static Public Member Functions**

• static const Strings & instance ()

#### **Public Attributes**

const QString unit

Setting key for the unit number of remote device.

· const QString type

Setting key for the type of Modbus protocol.

const QString tries

Setting key for the number of tries a Modbus request is repeated if it fails.

· const QString host

Setting key for the IP address or DNS name of the remote device.

const QString port

Setting key for the TCP port number of the remote device.

• const QString timeout

Setting key for connection timeout (milliseconds)

· const QString serialPortName

Setting key for the serial port name.

• const QString baudRate

Setting key for the serial port's baud rate.

· const QString dataBits

Setting key for the serial port's data bits.

· const QString parity

Setting key for the serial port's parity.

· const QString stopBits

Setting key for the serial port's stop bits.

· const QString flowControl

Setting key for the serial port's flow control.

const QString timeoutFirstByte

Setting key for the serial port's timeout waiting first byte of packet.

· const QString timeoutInterByte

Setting key for the serial port's timeout waiting next byte of packet.

const QString isBroadcastEnabled

Setting key for the serial port enables broadcast mode for 0 unit address.

const QString NoParity

String constant for repr of NoParity enum value.

· const QString EvenParity

String constant for repr of EvenParity enum value.

const QString OddParity

String constant for repr of OddParity enum value.

· const QString SpaceParity

String constant for repr of SpaceParity enum value.

· const QString MarkParity

String constant for repr of MarkParity enum value.

const QString OneStop

String constant for repr of OneStop enum value.

const QString OneAndHalfStop

String constant for repr of OneAndHalfStop enum value.

const QString TwoStop

String constant for repr of TwoStop enum value.

const QString NoFlowControl

String constant for repr of NoFlowControl enum value.

· const QString HardwareControl

String constant for repr of HardwareControl enum value.

const QString SoftwareControl

String constant for repr of SoftwareControl enum value.

#### 7.22.1 Detailed Description

Sets constant key values for the map of settings.

#### 7.22.2 Constructor & Destructor Documentation

#### 7.22.2.1 Strings()

```
Modbus::Strings::Strings ()
```

Constructor ot the class.

#### 7.22.3 Member Function Documentation

#### 7.22.3.1 instance()

```
static const Strings & Modbus::Strings::instance () [static]
```

Returns a reference to the global Modbus::Strings object.

The documentation for this class was generated from the following file:

c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusQt.h

# 7.23 Modbus::TcpSettings Struct Reference

Struct to define settings for TCP connection.

```
#include <ModbusGlobal.h>
```

# **Public Attributes**

const Char \* host

Value for the IP address or DNS name of the remote device.

• uint16\_t port

Value for the TCP port number of the remote device.

• uint16\_t timeout

Value for connection timeout (milliseconds)

# 7.23.1 Detailed Description

Struct to define settings for TCP connection.

The documentation for this struct was generated from the following file:

• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusGlobal.h

# **Chapter 8**

# **File Documentation**

# 8.1 c:/Users/march/Dropbox/PRJ/ModbusLib/src/cModbus.h File Reference

Contains library interface for C language.

```
#include <stdbool.h>
#include "ModbusGlobal.h"
```

# **Typedefs**

• typedef ModbusPort \* cModbusPort

Handle (pointer) of ModbusPort for C interface.

typedef ModbusClientPort \* cModbusClientPort

Handle (pointer) of ModbusClientPort for C interface.

typedef ModbusClient \* cModbusClient

 ${\it Handle (pointer) of ModbusClient for C interface.}$ 

 $\bullet \ \ typedef \ \underline{\mathsf{ModbusServerPort}} * \mathbf{cModbusServerPort}$ 

Handle (pointer) of ModbusServerPort for C interface.

typedef ModbusInterface \* cModbusInterface

Handle (pointer) of ModbusInterface for C interface.

typedef void \* cModbusDevice

Handle (pointer) of ModbusDevice for C interface.

- typedef StatusCode(\* pfReadCoils) (cModbusDevice dev, uint8\_t unit, uint16\_t offset, uint16\_t count, void \*values)
- typedef StatusCode(\* pfReadDiscreteInputs) (cModbusDevice dev, uint8\_t unit, uint16\_t offset, uint16\_
   t count, void \*values)
- typedef StatusCode(\* pfReadHoldingRegisters) (cModbusDevice dev, uint8\_t unit, uint16\_t offset, uint16\_t count, uint16\_t \*values)
- typedef StatusCode(\* pfReadInputRegisters) (cModbusDevice dev, uint8\_t unit, uint16\_t offset, uint16\_
   t count, uint16\_t \*values)
- typedef StatusCode(\* pfWriteSingleCoil) (cModbusDevice dev, uint8\_t unit, uint16\_t offset, bool value)
- typedef StatusCode(\* pfWriteSingleRegister) (cModbusDevice dev, uint8\_t unit, uint16\_t offset, uint16\_← t value)
- typedef StatusCode(\* pfReadExceptionStatus) (cModbusDevice dev, uint8\_t unit, uint8\_t \*status)

• typedef StatusCode(\* pfDiagnostics) (cModbusDevice dev, uint8\_t unit, uint16\_t subfunc, uint8\_t insize, const uint8\_t \*indata, uint8\_t \*outsize, uint8\_t \*outdata)

- typedef StatusCode(\* pfGetCommEventCounter) (cModbusDevice dev, uint8\_t unit, uint16\_t \*status, uint16\_t \*eventCount)
- typedef StatusCode(\* pfGetCommEventLog) (cModbusDevice dev, uint8\_t unit, uint16\_t \*status, uint16\_t \*eventCount, uint16\_t \*messageCount, uint8\_t \*eventBuffSize, uint8\_t \*eventBuff)
- typedef StatusCode(\* pfWriteMultipleCoils) (cModbusDevice dev, uint8\_t unit, uint16\_t offset, uint16\_t count, const void \*values)
- typedef StatusCode(\* pfWriteMultipleRegisters) (cModbusDevice dev, uint8\_t unit, uint16\_t offset, uint16\_t count, const uint16\_t \*values)
- typedef StatusCode(\* pfReportServerID) (cModbusDevice dev, uint8 t unit, uint8 t \*count, uint8 t \*data)
- typedef StatusCode(\* pfMaskWriteRegister) (cModbusDevice dev, uint8\_t unit, uint16\_t offset, uint16\_t and
   — Mask, uint16\_t orMask)
- typedef StatusCode(\* pfReadWriteMultipleRegisters) (cModbusDevice dev, uint8\_t unit, uint16\_t readOffset, uint16\_t readCount, uint16\_t \*readValues, uint16\_t writeOffset, uint16\_t writeCount, const uint16\_t \*write← Values)
- typedef StatusCode(\* pfReadFIFOQueue) (cModbusDevice dev, uint8\_t unit, uint16\_t fifoadr, uint16\_
   t \*count, uint16\_t \*values)
- typedef void(\* pfSlotOpened) (const Char \*source)
- typedef void(\* pfSlotClosed) (const Char \*source)
- typedef void(\* pfSlotTx) (const Char \*source, const uint8\_t \*buff, uint16\_t size)
- typedef void(\* pfSlotRx) (const Char \*source, const uint8 t \*buff, uint16 t size)
- typedef void(\* pfSlotError) (const Char \*source, StatusCode status, const Char \*text)
- typedef void(\* pfSlotNewConnection) (const Char \*source)
- typedef void(\* pfSlotCloseConnection) (const Char \*source)

#### **Functions**

- MODBUS\_EXPORT void cDeleteModbusDevice (cModbusInterface dev)
- MODBUS\_EXPORT cModbusPort cPortCreate (ProtocolType type, const void \*settings, bool blocking)
- MODBUS EXPORT void cPortDelete (cModbusPort port)
- MODBUS EXPORT cModbusClientPort cCpoCreate (ProtocolType type, const void \*settings, bool blocking)
- MODBUS EXPORT cModbusClientPort cCpoCreateForPort (cModbusPort port)
- MODBUS EXPORT void cCpoDelete (cModbusClientPort clientPort)
- MODBUS\_EXPORT const Char \* cCpoGetObjectName (cModbusClientPort clientPort)
- MODBUS\_EXPORT void cCpoSetObjectName (cModbusClientPort clientPort, const Char \*name)
- MODBUS EXPORT ProtocolType cCpoGetType (cModbusClientPort clientPort)
- MODBUS EXPORT bool cCpolsOpen (cModbusClientPort clientPort)
- MODBUS\_EXPORT bool cCpoClose (cModbusClientPort clientPort)
- MODBUS\_EXPORT uint32\_t cCpoGetRepeatCount (cModbusClientPort clientPort)
- MODBUS EXPORT void cCpoSetRepeatCount (cModbusClientPort clientPort, uint32 t count)
- MODBUS\_EXPORT StatusCode cCpoReadCoils (cModbusClientPort clientPort, uint8\_t unit, uint16\_t offset, uint16\_t count, void \*values)
- MODBUS\_EXPORT StatusCode cCpoReadDiscreteInputs (cModbusClientPort clientPort, uint8\_t unit, uint16 t offset, uint16 t count, void \*values)
- MODBUS\_EXPORT StatusCode cCpoReadHoldingRegisters (cModbusClientPort clientPort, uint8\_t unit, uint16\_t offset, uint16\_t count, uint16\_t \*values)

- MODBUS\_EXPORT StatusCode cCpoReadInputRegisters (cModbusClientPort clientPort, uint8\_t unit, uint16 t offset, uint16 t count, uint16 t \*values)
- MODBUS\_EXPORT StatusCode cCpoWriteSingleCoil (cModbusClientPort clientPort, uint8\_t unit, uint16\_t offset, bool value)
- MODBUS\_EXPORT StatusCode cCpoWriteSingleRegister (cModbusClientPort clientPort, uint8\_t unit, uint16 t offset, uint16 t value)
- MODBUS\_EXPORT StatusCode cCpoReadExceptionStatus (cModbusClientPort clientPort, uint8\_t unit, uint8 t \*value)
- MODBUS\_EXPORT StatusCode cCpoDiagnostics (cModbusClientPort clientPort, uint8\_t unit, uint16\_t subfunc, uint8\_t insize, const uint8\_t \*indata, uint8\_t \*outsize, uint8\_t \*outdata)
- MODBUS\_EXPORT StatusCode cCpoGetCommEventCounter (cModbusClientPort clientPort, uint8\_t unit, uint16 t \*status, uint16 t \*eventCount)
- MODBUS\_EXPORT StatusCode cCpoGetCommEventLog (cModbusClientPort clientPort, uint8\_t unit, uint16 t \*status, uint16 t \*eventCount, uint16 t \*eventBuff) t \*eventBuff)
- MODBUS\_EXPORT StatusCode cCpoWriteMultipleCoils (cModbusClientPort clientPort, uint8\_t unit, uint16\_t offset, uint16\_t count, const void \*values)
- MODBUS\_EXPORT StatusCode cCpoWriteMultipleRegisters (cModbusClientPort clientPort, uint8\_t unit, uint16\_t offset, uint16\_t count, const uint16\_t \*values)
- MODBUS\_EXPORT StatusCode cCpoReportServerID (cModbusClientPort clientPort, uint8\_t unit, uint8\_←
   t \*count, uint8 t \*data)
- MODBUS\_EXPORT StatusCode cCpoMaskWriteRegister (cModbusClientPort clientPort, uint8\_t unit, uint16 t offset, uint16 t andMask, uint16 t orMask)
- MODBUS\_EXPORT StatusCode cCpoReadWriteMultipleRegisters (cModbusClientPort clientPort, uint8\_
   t unit, uint16\_t readOffset, uint16\_t readCount, uint16\_t \*readValues, uint16\_t writeOffset, uint16\_t write
   Count, const uint16\_t \*writeValues)
- MODBUS\_EXPORT StatusCode cCpoReadFIFOQueue (cModbusClientPort clientPort, uint8\_t unit, uint16

  \_t fifoadr, uint16\_t \*count, uint16\_t \*values)
- MODBUS\_EXPORT StatusCode cCpoReadCoilsAsBoolArray (cModbusClientPort clientPort, uint8\_t unit, uint16\_t offset, uint16\_t count, bool \*values)
- MODBUS\_EXPORT StatusCode cCpoReadDiscreteInputsAsBoolArray (cModbusClientPort clientPort, uint8\_t unit, uint16\_t offset, uint16\_t count, bool \*values)
- MODBUS\_EXPORT StatusCode cCpoWriteMultipleCoilsAsBoolArray (cModbusClientPort clientPort, uint8← \_t unit, uint16\_t offset, uint16\_t count, const bool \*values)
- MODBUS EXPORT StatusCode cCpoGetLastStatus (cModbusClientPort clientPort)
- MODBUS EXPORT StatusCode cCpoGetLastErrorStatus (cModbusClientPort clientPort)
- MODBUS EXPORT const Char \* cCpoGetLastErrorText (cModbusClientPort clientPort)
- MODBUS\_EXPORT void cCpoConnectOpened (cModbusClientPort clientPort, pfSlotOpened funcPtr)
- MODBUS EXPORT void cCpoConnectClosed (cModbusClientPort clientPort, pfSlotClosed funcPtr)
- MODBUS EXPORT void cCpoConnectTx (cModbusClientPort clientPort, pfSlotTx funcPtr)
- MODBUS EXPORT void cCpoConnectRx (cModbusClientPort clientPort, pfSlotRx funcPtr)
- MODBUS EXPORT void cCpoConnectError (cModbusClientPort clientPort, pfSlotError funcPtr)
- MODBUS\_EXPORT void cCpoDisconnectFunc (cModbusClientPort clientPort, void \*funcPtr)
- MODBUS\_EXPORT cModbusClient cCliCreate (uint8\_t unit, ProtocolType type, const void \*settings, bool blocking)
- MODBUS EXPORT cModbusClient cCliCreateForClientPort (uint8 t unit, cModbusClientPort clientPort)
- MODBUS EXPORT void cCliDelete (cModbusClient client)
- MODBUS EXPORT const Char \* cCliGetObjectName (cModbusClient client)
- MODBUS EXPORT void cCliSetObjectName (cModbusClient client, const Char \*name)
- MODBUS\_EXPORT ProtocolType cCliGetType (cModbusClient client)
- MODBUS\_EXPORT uint8\_t cCliGetUnit (cModbusClient client)
- MODBUS EXPORT void cCliSetUnit (cModbusClient client, uint8 t unit)
- MODBUS EXPORT bool cClilsOpen (cModbusClient client)
- MODBUS\_EXPORT cModbusClientPort cCliGetPort (cModbusClient client)
- MODBUS\_EXPORT StatusCode cReadCoils (cModbusClient client, uint16\_t offset, uint16\_t count, void \*values)

MODBUS\_EXPORT StatusCode cReadDiscreteInputs (cModbusClient client, uint16\_t offset, uint16\_t count, void \*values)

- MODBUS\_EXPORT StatusCode cReadHoldingRegisters (cModbusClient client, uint16\_t offset, uint16\_
   t count, uint16\_t \*values)
- MODBUS\_EXPORT StatusCode cReadInputRegisters (cModbusClient client, uint16\_t offset, uint16\_t count, uint16\_t \*values)
- MODBUS EXPORT StatusCode cWriteSingleCoil (cModbusClient client, uint16 t offset, bool value)
- MODBUS\_EXPORT StatusCode cWriteSingleRegister (cModbusClient client, uint16\_t offset, uint16\_t value)
- MODBUS\_EXPORT StatusCode cReadExceptionStatus (cModbusClient client, uint8\_t \*value)
- MODBUS\_EXPORT StatusCode cWriteMultipleCoils (cModbusClient client, uint16\_t offset, uint16\_t count, const void \*values)
- MODBUS\_EXPORT StatusCode cWriteMultipleRegisters (cModbusClient client, uint16\_t offset, uint16\_
   t count, const uint16 t \*values)
- MODBUS\_EXPORT StatusCode cMaskWriteRegister (cModbusClient client, uint16\_t offset, uint16\_t and
   — Mask, uint16\_t orMask)
- MODBUS\_EXPORT StatusCode cReadWriteMultipleRegisters (cModbusClient client, uint16\_t readOffset, uint16\_t readCount, uint16\_t \*readValues, uint16\_t writeOffset, uint16\_t writeCount, const uint16\_t \*write← Values)
- MODBUS\_EXPORT StatusCode cReadCoilsAsBoolArray (cModbusClient client, uint16\_t offset, uint16\_←
  t count, bool \*values)
- MODBUS\_EXPORT StatusCode cReadDiscreteInputsAsBoolArray (cModbusClient client, uint16\_t offset, uint16\_t count, bool \*values)
- MODBUS\_EXPORT StatusCode cWriteMultipleCoilsAsBoolArray (cModbusClient client, uint16\_t offset, uint16 t count, const bool \*values)
- MODBUS\_EXPORT StatusCode cCliGetLastPortStatus (cModbusClient client)
- MODBUS EXPORT StatusCode cCliGetLastPortErrorStatus (cModbusClient client)
- MODBUS EXPORT const Char \* cCliGetLastPortErrorText (cModbusClient client)
- MODBUS\_EXPORT cModbusServerPort cSpoCreate (cModbusInterface device, ProtocolType type, const void \*settings, bool blocking)
- MODBUS\_EXPORT void cSpoDelete (cModbusServerPort serverPort)
- MODBUS\_EXPORT const Char \* cSpoGetObjectName (cModbusServerPort serverPort)
- MODBUS EXPORT void cSpoSetObjectName (cModbusServerPort serverPort, const Char \*name)
- MODBUS EXPORT ProtocolType cSpoGetType (cModbusServerPort serverPort)
- MODBUS\_EXPORT bool cSpolsTcpServer (cModbusServerPort serverPort)
- MODBUS\_EXPORT cModbusInterface cSpoGetDevice (cModbusServerPort serverPort)
- MODBUS\_EXPORT bool cSpolsOpen (cModbusServerPort serverPort)
- MODBUS\_EXPORT StatusCode cSpoOpen (cModbusServerPort serverPort)
- MODBUS\_EXPORT StatusCode cSpoClose (cModbusServerPort serverPort)
- MODBUS\_EXPORT StatusCode cSpoProcess (cModbusServerPort serverPort)
- MODBUS EXPORT void cSpoConnectOpened (cModbusServerPort serverPort, pfSlotOpened funcPtr)
- MODBUS EXPORT void cSpoConnectClosed (cModbusServerPort serverPort, pfSlotClosed funcPtr)
- MODBUS\_EXPORT void cSpoConnectTx (cModbusServerPort serverPort, pfSlotTx funcPtr)
- MODBUS\_EXPORT void cSpoConnectRx (cModbusServerPort serverPort, pfSlotRx funcPtr)
- MODBUS EXPORT void cSpoConnectError (cModbusServerPort serverPort, pfSlotError funcPtr)
- MODBUS\_EXPORT void cSpoConnectNewConnection (cModbusServerPort serverPort, pfSlotNewConnection funcPtr)
- MODBUS\_EXPORT void cSpoConnectCloseConnection (cModbusServerPort serverPort, pfSlotCloseConnection funcPtr)
- MODBUS EXPORT void cSpoDisconnectFunc (cModbusServerPort serverPort, void \*funcPtr)

# 8.1.1 Detailed Description

Contains library interface for C language.

Author

serhmarch

Date

May 2024

# 8.1.2 Typedef Documentation

# 8.1.2.1 pfDiagnostics

```
typedef StatusCode(* pfDiagnostics) (cModbusDevice dev, uint8_t unit, uint16_t subfunc, uint8← _t insize, const uint8_t *indata, uint8_t *outsize, uint8_t *outdata)
```

Pointer to C function for diagnostics. dev - pointer to any struct that can hold memory data.

See also

ModbusInterface::diagnostics

#### 8.1.2.2 pfGetCommEventCounter

```
typedef StatusCode(* pfGetCommEventCounter) (cModbusDevice dev, uint8_t unit, uint16_t *status,
uint16_t *eventCount)
```

Pointer to C function for get communication event counter. dev - pointer to any struct that can hold memory data.

See also

ModbusInterface::getCommEventCounter

#### 8.1.2.3 pfGetCommEventLog

```
typedef StatusCode(* pfGetCommEventLog) (cModbusDevice dev, uint8_t unit, uint16_t *status,
uint16_t *eventCount, uint16_t *messageCount, uint8_t *eventBuffSize, uint8_t *eventBuff)
```

Pointer to C function for get communication event logs. dev - pointer to any struct that can hold memory data.

See also

ModbusInterface::getCommEventLog

#### 8.1.2.4 pfMaskWriteRegister

```
typedef StatusCode(* pfMaskWriteRegister) (cModbusDevice dev, uint8_t unit, uint16_t offset,
uint16_t andMask, uint16_t orMask)
```

Pointer to C function for mask write registers (4x). dev - pointer to any struct that can hold memory data.

See also

ModbusInterface::maskWriteRegister

#### 8.1.2.5 pfReadCoils

```
typedef StatusCode(* pfReadCoils) (cModbusDevice dev, uint8_t unit, uint16_t offset, uint16_t
count, void *values)
```

Pointer to C function for read coils (0x). dev - pointer to any struct that can hold memory data.

See also

ModbusInterface::readCoils

#### 8.1.2.6 pfReadDiscreteInputs

```
typedef StatusCode(* pfReadDiscreteInputs) (cModbusDevice dev, uint8_t unit, uint16_t offset,
uint16_t count, void *values)
```

Pointer to C function for read discrete inputs (1x). dev - pointer to any struct that can hold memory data.

See also

ModbusInterface::readDiscreteInputs

# 8.1.2.7 pfReadExceptionStatus

```
typedef StatusCode(* pfReadExceptionStatus) (cModbusDevice dev, uint8_t unit, uint8_t *status)
```

Pointer to C function for read exception status bits. dev - pointer to any struct that can hold memory data.

See also

ModbusInterface::readExceptionStatus

#### 8.1.2.8 pfReadFIFOQueue

```
typedef StatusCode(* pfReadFIFOQueue) (cModbusDevice dev, uint8_t unit, uint16_t fifoadr,
uint16_t *count, uint16_t *values)
```

Pointer to C function for read FIFO queue. dev - pointer to any struct that can hold memory data.

See also

ModbusInterface::readFIF0Oueue

#### 8.1.2.9 pfReadHoldingRegisters

```
typedef StatusCode(* pfReadHoldingRegisters) (cModbusDevice dev, uint8_t unit, uint16_t offset,
uint16_t count, uint16_t *values)
```

Pointer to C function for read holding registers (4x). dev - pointer to any struct that can hold memory data.

See also

ModbusInterface::readHoldingRegisters

#### 8.1.2.10 pfReadInputRegisters

```
typedef StatusCode(* pfReadInputRegisters) (cModbusDevice dev, uint8_t unit, uint16_t offset,
uint16_t count, uint16_t *values)
```

Pointer to C function for read input registers (3x). dev - pointer to any struct that can hold memory data.

See also

ModbusInterface::readInputRegisters

# 8.1.2.11 pfReadWriteMultipleRegisters

```
typedef StatusCode(* pfReadWriteMultipleRegisters) (cModbusDevice dev, uint8_t unit, uint16_t readOffset, uint16_t readCount, uint16_t *readValues, uint16_t writeOffset, uint16_t write← Count, const uint16_t *writeValues)
```

Pointer to C function for write registers (4x). dev - pointer to any struct that can hold memory data.

See also

ModbusInterface::writeMultipleRegisters

#### 8.1.2.12 pfReportServerID

```
typedef StatusCode(* pfReportServerID) (cModbusDevice dev, uint8_t unit, uint8_t *count, uint8\leftarrow_t *data)
```

Pointer to C function for report server id. dev - pointer to any struct that can hold memory data.

See also

ModbusInterface::reportServerID

#### 8.1.2.13 pfSlotCloseConnection

```
typedef void(* pfSlotCloseConnection) (const Char *source)
```

Pointer to C callback function. dev - pointer to any struct that can hold memory data.

See also

ModbusTcpServer::signalCloseConnection

#### 8.1.2.14 pfSlotClosed

```
typedef void(* pfSlotClosed) (const Char *source)
```

Pointer to C callback function. dev - pointer to any struct that can hold memory data.

See also

ModbusClientPort::signalClosed and ModbusServerPort::signalClosed

# 8.1.2.15 pfSlotError

```
typedef void(* pfSlotError) (const Char *source, StatusCode status, const Char *text)
```

Pointer to C callback function. dev - pointer to any struct that can hold memory data.

See also

 ${\tt ModbusClientPort::} signal {\tt Error} \ \textbf{and} \ {\tt ModbusServerPort::} signal {\tt Error}$ 

#### 8.1.2.16 pfSlotNewConnection

```
typedef void(* pfSlotNewConnection) (const Char *source)
```

Pointer to C callback function. dev - pointer to any struct that can hold memory data.

See also

ModbusTcpServer::signalNewConnection

#### 8.1.2.17 pfSlotOpened

```
typedef void(* pfSlotOpened) (const Char *source)
```

Pointer to C callback function. dev - pointer to any struct that can hold memory data.

See also

ModbusClientPort::signalOpened and ModbusServerPort::signalOpened

#### 8.1.2.18 pfSlotRx

```
typedef void(* pfSlotRx) (const Char *source, const uint8_t *buff, uint16_t size)
```

Pointer to C callback function. dev - pointer to any struct that can hold memory data.

See also

```
ModbusClientPort::signalRx and ModbusServerPort::signalRx
```

#### 8.1.2.19 pfSlotTx

```
typedef void(* pfSlotTx) (const Char *source, const uint8_t *buff, uint16_t size)
```

Pointer to C callback function. dev - pointer to any struct that can hold memory data.

See also

```
ModbusClientPort::signalTx and ModbusServerPort::signalTx
```

# 8.1.2.20 pfWriteMultipleCoils

```
typedef StatusCode(* pfWriteMultipleCoils) (cModbusDevice dev, uint8_t unit, uint16_t offset,
uint16_t count, const void *values)
```

Pointer to C function for write coils (0x). dev - pointer to any struct that can hold memory data.

See also

```
ModbusInterface::writeMultipleCoils
```

#### 8.1.2.21 pfWriteMultipleRegisters

```
typedef StatusCode(* pfWriteMultipleRegisters) (cModbusDevice dev, uint8_t unit, uint16_\leftrightarrow t offset, uint16_t count, const uint16_t *values)
```

Pointer to C function for write registers (4x). dev - pointer to any struct that can hold memory data.

See also

```
ModbusInterface::writeMultipleRegisters
```

#### 8.1.2.22 pfWriteSingleCoil

```
typedef StatusCode(* pfWriteSingleCoil) (cModbusDevice dev, uint8_t unit, uint16_t offset,
bool value)
```

Pointer to C function for write single coil (0x). dev - pointer to any struct that can hold memory data.

See also

```
ModbusInterface::writeSingleCoil
```

#### 8.1.2.23 pfWriteSingleRegister

```
typedef StatusCode(* pfWriteSingleRegister) (cModbusDevice dev, uint8_t unit, uint16_t offset,
uint16_t value)
```

Pointer to C function for write single register (4x). dev - pointer to any struct that can hold memory data.

See also

```
ModbusInterface::writeSingleRegister
```

#### 8.1.3 Function Documentation

# 8.1.3.1 cCliCreate()

 $\label{lem:condition} \textbf{Creates} \ \texttt{ModbusClient} \ \textbf{object} \ \textbf{and} \ \textbf{returns} \ \textbf{handle} \ \textbf{to} \ \textbf{it}.$ 

See also

```
Modbus::createClient
```

#### 8.1.3.2 cCliCreateForClientPort()

Creates ModbusClient object with unit for port clientPort and returns handle to it.

#### 8.1.3.3 cCliDelete()

Deletes previously created ModbusClient object represented by client handle

#### 8.1.3.4 cCliGetLastPortErrorStatus()

Wrapper for ModbusClient::lastPortErrorStatus

#### 8.1.3.5 cCliGetLastPortErrorText()

Wrapper for ModbusClient::lastPortErrorText

#### 8.1.3.6 cCliGetLastPortStatus()

Wrapper for ModbusClient::lastPortStatus

#### 8.1.3.7 cCliGetObjectName()

Wrapper for ModbusClient::objectName

# 8.1.3.8 cCliGetPort()

Wrapper for ModbusClient::port

# 8.1.3.9 cCliGetType()

Wrapper for ModbusClient::type

#### 8.1.3.10 cCliGetUnit()

Wrapper for ModbusClient::isOpen

cModbusClient client)

# 8.1.3.12 cCliSetObjectName()

Wrapper for ModbusClient::setObjectName

#### 8.1.3.13 cCliSetUnit()

Wrapper for ModbusClient::setUnit

#### 8.1.3.14 cCpoClose()

Wrapper for ModbusClientPort::close

#### 8.1.3.15 cCpoConnectClosed()

Connects funcPtr-function to ModbusClientPort::signalClosed signal

#### 8.1.3.16 cCpoConnectError()

Connects funcPtr-function to ModbusClientPort::signalError signal

#### 8.1.3.17 cCpoConnectOpened()

Connects funcPtr-function to ModbusClientPort::signalOpened signal

# 8.1.3.18 cCpoConnectRx()

Connects funcPtr-function to ModbusClientPort::signalRx signal

#### 8.1.3.19 cCpoConnectTx()

Connects funcPtr-function to ModbusClientPort::signalTx signal

#### 8.1.3.20 cCpoCreate()

 ${\bf Creates} \ {\tt ModbusClientPort} \ {\bf object} \ {\bf and} \ {\bf returns} \ {\bf handle} \ {\bf to} \ {\bf it}.$ 

See also

```
Modbus::createClientPort
```

#### 8.1.3.21 cCpoCreateForPort()

 ${\bf Creates} \ {\tt ModbusClientPort} \ {\bf object} \ {\bf and} \ {\bf returns} \ {\bf handle} \ {\bf to} \ {\bf it}.$ 

#### 8.1.3.22 cCpoDelete()

Deletes previously created ModbusClientPort object represented by port handle

#### 8.1.3.23 cCpoDiagnostics()

Wrapper for ModbusClientPort::diagnostics

#### 8.1.3.24 cCpoDisconnectFunc()

Disconnects funcPtr-function from ModbusClientPort

#### 8.1.3.25 cCpoGetCommEventCounter()

Wrapper for ModbusClientPort::getCommEventCounter

# 8.1.3.26 cCpoGetCommEventLog()

Wrapper for ModbusClientPort::getCommEventLog

#### 8.1.3.27 cCpoGetLastErrorStatus()

#### 8.1.3.28 cCpoGetLastErrorText()

Wrapper for ModbusClientPort::getLastErrorText

# 8.1.3.29 cCpoGetLastStatus()

Wrapper for ModbusClientPort::getLastStatus

#### 8.1.3.30 cCpoGetObjectName()

Wrapper for ModbusClientPort::objectName

#### 8.1.3.31 cCpoGetRepeatCount()

Wrapper for ModbusClientPort::repeatCount

# 8.1.3.32 cCpoGetType()

Wrapper for ModbusClientPort::type

# 8.1.3.33 cCpolsOpen()

Wrapper for ModbusClientPort::isOpen

#### 8.1.3.34 cCpoMaskWriteRegister()

Wrapper for ModbusClientPort::maskWriteRegister

#### 8.1.3.35 cCpoReadCoils()

Wrapper for ModbusClientPort::readCoils

#### 8.1.3.36 cCpoReadCoilsAsBoolArray()

Wrapper for ModbusClientPort::readCoilsAsBoolArray

#### 8.1.3.37 cCpoReadDiscreteInputs()

Wrapper for ModbusClientPort::readDiscreteInputs

#### 8.1.3.38 cCpoReadDiscreteInputsAsBoolArray()

Wrapper for ModbusClientPort::readDiscreteInputsAsBoolArray

#### 8.1.3.39 cCpoReadExceptionStatus()

Wrapper for ModbusClientPort::readExceptionStatus

#### 8.1.3.40 cCpoReadFIFOQueue()

Wrapper for ModbusClientPort::readFIFOQueue

#### 8.1.3.41 cCpoReadHoldingRegisters()

Wrapper for ModbusClientPort::readHoldingRegisters

# 8.1.3.42 cCpoReadInputRegisters()

Wrapper for ModbusClientPort::readInputRegisters

#### 8.1.3.43 cCpoReadWriteMultipleRegisters()

Wrapper for ModbusClientPort::readWriteMultipleRegisters

#### 8.1.3.44 cCpoReportServerID()

Wrapper for ModbusClientPort::reportServerID

#### 8.1.3.45 cCpoSetObjectName()

Wrapper for ModbusClientPort::setObjectName

# 8.1.3.46 cCpoSetRepeatCount()

Wrapper for ModbusClientPort::setRepeatCount

#### 8.1.3.47 cCpoWriteMultipleCoils()

Wrapper for ModbusClientPort::writeMultipleCoils

## 8.1.3.48 cCpoWriteMultipleCoilsAsBoolArray()

Wrapper for ModbusClientPort::writeMultipleCoilsAsBoolArray

#### 8.1.3.49 cCpoWriteMultipleRegisters()

Wrapper for ModbusClientPort::writeMultipleRegisters

#### 8.1.3.50 cCpoWriteSingleCoil()

Wrapper for ModbusClientPort::writeSingleCoil

#### 8.1.3.51 cCpoWriteSingleRegister()

Wrapper for ModbusClientPort::writeSingleRegister

#### 8.1.3.52 cCreateModbusDevice()

```
MODBUS_EXPORT cModbusInterface cCreateModbusDevice (
             cModbusDevice device,
             pfReadCoils readCoils,
             pfReadDiscreteInputs readDiscreteInputs,
             pfReadHoldingRegisters readHoldingRegisters,
             pfReadInputRegisters readInputRegisters,
             pfWriteSingleCoil writeSingleCoil,
             pfWriteSingleRegister writeSingleRegister,
             pfReadExceptionStatus readExceptionStatus,
             pfDiagnostics diagnostics,
             pfGetCommEventCounter getCommEventCounter,
             pfGetCommEventLog getCommEventLog,
             pfWriteMultipleCoils writeMultipleCoils,
             pfWriteMultipleRegisters writeMultipleRegisters,
             pfReportServerID reportServerID,
             pfMaskWriteRegister maskWriteRegister,
             {\tt pfReadWriteMultipleRegisters}\ read {\tt WriteMultipleRegisters},
             pfReadFIFOQueue readFIFOQueue)
```

Function create ModbusInterface object and returns pointer to it for server. dev - pointer to any struct that can hold memory data. readCoils, readDiscreteInputs, readHoldingRegisters, readInputRegisters, write SingleCoil, writeSingleRegister, readExceptionStatus, diagnostics, getCommEventCounter, getCommEventLog, writeMultipleCoils writeMultipleRegisters reportServerID, maskWriteRegister, readWriteMultipleRegisters, read FIFOQueue - pointers to corresponding Modbus functions to process data. Any pointer can have NULL value. In this case server will return Status\_BadIllegalFunction.

#### 8.1.3.53 cDeleteModbusDevice()

Deletes previously created ModbusInterface object represented by dev handle

#### 8.1.3.54 cMaskWriteRegister()

Wrapper for ModbusClient::maskWriteRegister

#### 8.1.3.55 cPortCreate()

Creates ModbusPort object and returns handle to it.

See also

```
Modbus::createPort
```

#### 8.1.3.56 cPortDelete()

Deletes previously created ModbusPort object represented by port handle

#### 8.1.3.57 cReadCoils()

Wrapper for ModbusClient::readCoils

#### 8.1.3.58 cReadCoilsAsBoolArray()

Wrapper for ModbusClient::readCoilsAsBoolArray

#### 8.1.3.59 cReadDiscreteInputs()

Wrapper for ModbusClient::readDiscreteInputs

#### 8.1.3.60 cReadDiscreteInputsAsBoolArray()

Wrapper for ModbusClient::readDiscreteInputsAsBoolArray

#### 8.1.3.61 cReadExceptionStatus()

Wrapper for ModbusClient::readExceptionStatus

#### 8.1.3.62 cReadHoldingRegisters()

Wrapper for ModbusClient::readHoldingRegisters

#### 8.1.3.63 cReadInputRegisters()

Wrapper for ModbusClient::readInputRegisters

#### 8.1.3.64 cReadWriteMultipleRegisters()

Wrapper for ModbusClient::readWriteMultipleRegisters

#### 8.1.3.65 cSpoClose()

Wrapper for ModbusServerPort::close

# 8.1.3.66 cSpoConnectCloseConnection()

Connects funcPtr-function to ModbusServerPort::signalCloseConnection signal

#### 8.1.3.67 cSpoConnectClosed()

Connects funcPtr-function to ModbusServerPort::signalClosed signal

#### 8.1.3.68 cSpoConnectError()

Connects funcPtr-function to ModbusServerPort::signalError signal

## 8.1.3.69 cSpoConnectNewConnection()

Connects funcPtr-function to ModbusServerPort::signalNewConnection signal

#### 8.1.3.70 cSpoConnectOpened()

Connects funcPtr-function to ModbusServerPort::signalOpened signal

## 8.1.3.71 cSpoConnectRx()

Connects funcPtr-function to ModbusServerPort::signalRx signal

## 8.1.3.72 cSpoConnectTx()

Connects funcPtr-function to ModbusServerPort::signalTx signal

#### 8.1.3.73 cSpoCreate()

Creates ModbusServerPort object and returns handle to it.

See also

```
Modbus::createServerPort
```

#### 8.1.3.74 cSpoDelete()

Deletes previously created ModbusServerPort object represented by serverPort handle

## 8.1.3.75 cSpoDisconnectFunc()

Disconnects funcPtr-function from ModbusServerPort

## 8.1.3.76 cSpoGetDevice()

Wrapper for ModbusServerPort::device

## 8.1.3.77 cSpoGetObjectName()

Wrapper for ModbusServerPort::objectName

## 8.1.3.78 cSpoGetType()

Wrapper for ModbusServerPort::type

## 8.1.3.79 cSpolsOpen()

Wrapper for ModbusServerPort::isOpen

#### 8.1.3.80 cSpolsTcpServer()

Wrapper for ModbusServerPort::isTcpServer

#### 8.1.3.81 cSpoOpen()

#### 8.1.3.82 cSpoProcess()

Wrapper for ModbusServerPort::process

## 8.1.3.83 cSpoSetObjectName()

Wrapper for ModbusServerPort::setObjectName

## 8.1.3.84 cWriteMultipleCoils()

Wrapper for ModbusClient::writeMultipleCoils

## 8.1.3.85 cWriteMultipleCoilsAsBoolArray()

Wrapper for ModbusClient::lastPortStatus

## 8.1.3.86 cWriteMultipleRegisters()

Wrapper for ModbusClient::writeMultipleRegisters

#### 8.1.3.87 cWriteSingleCoil()

Wrapper for ModbusClient::writeSingleCoil

#### 8.1.3.88 cWriteSingleRegister()

Wrapper for ModbusClient::writeSingleRegister

## 8.2 cModbus.h

#### Go to the documentation of this file.

```
00001
00008 #ifndef CMODBUS H
00009 #define CMODBUS H
00010
00011 #include <stdbool.h>
00012 #include "ModbusGlobal.h"
00013
00014 #ifdef __cplusplus
00015 using namespace Modbus;
00016 extern "C" {
00017 #endif
00018
00019 #ifdef __cplusplus
00020 class ModbusPort
00021 class ModbusInterface;
00022 class ModbusClientPort;
00023 class ModbusClient
00024 class ModbusServerPort;
00025
00026 #else
00027 typedef struct ModbusPort
                                      ModbusPort
00028 typedef struct ModbusInterface ModbusInterface;
00029 typedef struct ModbusClientPort ModbusClientPort;
00030 typedef struct ModbusClient
                                       ModbusClient
00031 typedef struct ModbusServerPort ModbusServerPort;
00032 #endif
00033
00035 typedef ModbusPort* cModbusPort;
00036
00038 typedef ModbusClientPort* cModbusClientPort;
00039
00041 typedef ModbusClient* cModbusClient;
00042
00044 typedef ModbusServerPort* cModbusServerPort;
00045
00047 typedef ModbusInterface* cModbusInterface;
00048
00050 typedef void* cModbusDevice;
00051
00052 #ifndef MBF READ COILS DISABLE
00054 typedef StatusCode (*pfReadCoils)(cModbusDevice dev, uint8_t unit, uint16_t offset, uint16_t count,
      void *values);
00055 #endif // MBF_READ_COILS_DISABLE
00056
00057 #ifndef MBF_READ_DISCRETE_INPUTS_DISABLE
00059 typedef StatusCode (*pfReadDiscreteInputs)(cModbusDevice dev, uint8_t unit, uint16_t offset, uint16_t
count, void *values);
00060 #endif // MBF_READ_DISCRETE_INPUTS_DISABLE
00061
```

8.2 cModbus.h

```
00062 #ifndef MBF_READ_HOLDING_REGISTERS_DISABLE
00064 typedef StatusCode (*pfReadHoldingRegisters)(cModbusDevice dev, uint8_t unit, uint16_t offset,
         uint16_t count, uint16_t *values);
00065 #endif // MBF_READ_HOLDING_REGISTERS_DISABLE
00066
00067 #ifndef MBF_READ_INPUT_REGISTERS_DISABLE
00069 typedef StatusCode (*pfReadInputRegisters)(cModbusDevice dev, uint8_t unit, uint16_t offset, uint16_t
         count, uint16_t *values);
00070 #endif // MBF_READ_INPUT_REGISTERS_DISABLE
00071
00072 #ifndef MBF WRITE SINGLE COIL DISABLE
00074 typedef StatusCode (*pfWriteSingleCoil) (cModbusDevice dev, uint8_t unit, uint16_t offset, bool value);
00075 #endif // MBF_WRITE_SINGLE_COIL_DISABLE
00076
00077 #ifndef MBF_WRITE_SINGLE_REGISTER_DISABLE
00079 typedef StatusCode (*pfWriteSingleRegister)(cModbusDevice dev, uint8_t unit, uint16_t offset, uint16_t
        value);
00080 #endif // MBF WRITE SINGLE REGISTER DISABLE
00081
00082 #ifndef MBF_READ_EXCEPTION_STATUS_DISABLE
00084 typedef StatusCode (*pfReadExceptionStatus)(cModbusDevice dev, uint8_t unit, uint8_t *status);
00085 #endif // MBF_READ_EXCEPTION_STATUS_DISABLE
00086
00087 #ifndef MBF_DIAGNOSTICS_DISABLE
00089 typedef StatusCode (*pfDiagnostics) (cModbusDevice dev, uint8_t unit, uint16_t subfunc, uint8_t insize,
         const uint8_t *indata, uint8_t *outsize, uint8_t *outdata);
00090 #endif // MBF_DIAGNOSTICS_DISABLE
00091
00092 #ifndef MBF_GET_COMM_EVENT_COUNTER_DISABLE
00094 typedef StatusCode (*pfGetCommEventCounter) (cModbusDevice dev, uint8_t unit, uint16_t *status,
         uint16_t *eventCount);
00095 #endif // MBF_GET_COMM_EVENT_COUNTER_DISABLE
00096
00097 #ifndef MBF_GET_COMM_EVENT_LOG_DISABLE
00099 typedef StatusCode (*pfGetCommEventLog) (cModbusDevice dev, uint8_t unit, uint16_t *status, uint16_t *eventCount, uint16_t *messageCount, uint8_t *eventBuffSize, uint8_t *eventBuff);
00100 #endif // MBF_GET_COMM_EVENT_LOG_DISABLE
00102 #ifndef MBF_WRITE_MULTIPLE_COILS_DISABLE
00104 typedef StatusCode (*pfWriteMultipleCoils)(cModbusDevice dev, uint8_t unit, uint16_t offset, uint16_t count, const void *values);
00105 #endif // MBF_WRITE_MULTIPLE_COILS_DISABLE
00106
00107 #ifndef MBF_WRITE_MULTIPLE_REGISTERS_DISABLE
00109 typedef StatusCode (*pfWriteMultipleRegisters)(cModbusDevice dev, uint8_t unit, uint16_t offset,
         uint16_t count, const uint16_t *values);
00110 #endif // MBF_WRITE_MULTIPLE_REGISTERS_DISABLE
00111
00112 #ifndef MBF REPORT SERVER ID DISABLE
00114 typedef StatusCode (*pfReportServerID) (cModbusDevice dev, uint8_t unit, uint8_t *count, uint8_t
          *data);
00115 #endif // MBF_REPORT_SERVER_ID_DISABLE
00116
00117 #ifndef MBF MASK WRITE REGISTER DISABLE
00119 typedef StatusCode (*pfMaskWriteRegister)(cModbusDevice dev, uint8_t unit, uint16_t offset, uint16_t
andMask, uint16_t orMask);
00120 #endif // MBF_MASK_WRITE_REGISTER_DISABLE
00121
00122 #ifndef MBF_READ_WRITE_MULTIPLE_REGISTERS_DISABLE
00124 typedef StatusCode (*pfReadWriteMultipleRegisters)(cModbusDevice dev, uint8_t unit, uint16_t
         readOffset, uint16_t readCount, uint16_t *readValues, uint16_t writeOffset, uint16_t writeCount, const
         uint16_t *writeValues);
00125 #endif // MBF_READ_WRITE_MULTIPLE_REGISTERS_DISABLE
00126
00127 #ifndef MBF_READ_FIFO_QUEUE_DISABLE
00129 typedef StatusCode (*pfReadFIFOQueue) (cModbusDevice dev, uint8_t unit, uint16_t fifoadr, uint16_t
         *count, uint16_t *values);
00130 #endif // MBF_READ_FIFO_QUEUE_DISABLE
00131
00133 typedef void (*pfSlotOpened) (const Char *source);
00134
00136 typedef void (*pfSlotClosed)(const Char *source);
00137
00139 typedef void (*pfSlotTx) (const Char *source, const uint8_t* buff, uint16_t size);
00140
00142 typedef void (*pfSlotRx)(const Char *source, const uint8_t* buff, uint16_t size);
00143
00145 typedef void (*pfSlotError) (const Char *source, StatusCode status, const Char *text);
00146
00148 typedef void (*pfSlotNewConnection) (const Char *source):
00149
00151 typedef void (*pfSlotCloseConnection) (const Char *source);
00152
{\tt 00172\ MODBUS\_EXPORT\ cModbusInterface\ cCreateModbusDevice} (cModbusDevice) and {\tt 00172\ MODBUS\_EXPORT\ cModbusDevice} (cModbusDevice\ cModbusDevice\ cModbusDevice\
                                                                                                                                        device
00173 #ifndef MBF_READ_COILS_DISABLE
                                                                                      , pfReadCoils
00174
                                                                                                                                        readCoils
00175 #endif // MBF_READ_COILS_DISABLE
```

```
00176 #ifndef MBF_READ_DISCRETE_INPUTS_DISABLE
00177
                                                     , pfReadDiscreteInputs
                                                                                   readDiscreteInputs
00178 #endif // MBF_READ_DISCRETE_INPUTS_DISABLE
00179 #ifndef MBF_READ_HOLDING_REGISTERS_DISABLE
                                                     , pfReadHoldingRegisters
00180
                                                                                   readHoldingRegisters
00181 #endif // MBF_READ_HOLDING_REGISTERS_DISABLE
00182 #ifndef MBF_READ_INPUT_REGISTERS_DISABLE
00183
                                                     , pfReadInputRegisters
                                                                                   readInputRegisters
00184 #endif // MBF_READ_INPUT_REGISTERS_DISABLE
00185 #ifndef MBF_WRITE_SINGLE_COIL_DISABLE
                                                     , pfWriteSingleCoil
00186
                                                                                   writeSingleCoil
00187 #endif // MBF_WRITE_SINGLE_COIL_DISABLE
00188 #ifndef MBF_WRITE_SINGLE_REGISTER_DISABLE
00189
                                                     , pfWriteSingleRegister
                                                                                   writeSingleRegister
00190 #endif // MBF_WRITE_SINGLE_REGISTER_DISABLE
00191 #ifndef MBF_READ_EXCEPTION_STATUS_DISABLE
                                                     , pfReadExceptionStatus
00192
                                                                                   readExceptionStatus
00193 #endif // MBF READ EXCEPTION STATUS DISABLE
00194 #ifndef MBF_DIAGNOSTICS_DISABLE
00195
                                                     , pfDiagnostics
                                                                                   diagnostics
00196 #endif // MBF_DIAGNOSTICS_DISABLE
00197 #ifndef MBF_GET_COMM_EVENT_COUNTER_DISABLE
                                                     , pfGetCommEventCounter
00198
                                                                                   getCommEventCounter
00199 #endif // MBF_GET_COMM_EVENT_COUNTER_DISABLE
00200 #ifndef MBF_GET_COMM_EVENT_LOG_DISABLE
00201
                                                     , pfGetCommEventLog
                                                                                   getCommEventLog
00202 #endif // MBF_GET_COMM_EVENT_LOG_DISABLE
00203 #ifndef MBF_WRITE_MULTIPLE_COILS_DISABLE
00204
                                                     , pfWriteMultipleCoils
                                                                                   writeMultipleCoils
00205 #endif // MBF_WRITE_MULTIPLE_COILS_DISABLE
00206 #ifndef MBF WRITE MULTIPLE REGISTERS DISABLE
00207
                                                     , pfWriteMultipleRegisters
                                                                                   writeMultipleRegisters
00208 #endif // MBF_WRITE_MULTIPLE_REGISTERS_DISABLE
00209 #ifndef MBF_REPORT_SERVER_ID_DISABLE
                                                                                   reportServerID
                                                     , pfReportServerID
00210
00211 #endif // MBF REPORT SERVER ID DISABLE
00212 #ifndef MBF_MASK_WRITE_REGISTER_DISABLE
                                                     , pfMaskWriteRegister
                                                                                 maskWriteRegister
00214 #endif // MBF_MASK_WRITE_REGISTER_DISABLE
00215 #ifndef MBF_READ_WRITE_MULTIPLE_REGISTERS_DISABLE
                                                     , pfReadWriteMultipleRegisters
00216
     readWriteMultipleRegisters
00217 #endif // MBF READ WRITE MULTIPLE REGISTERS DISABLE
00218 #ifndef MBF_READ_FIFO_QUEUE_DISABLE
00219
                                                     , pfReadFIFOQueue
                                                                                 readFIFOOueue
00220 #endif // MBF_READ_FIFO_QUEUE_DISABLE
00221
00222
00223
00225 MODBUS_EXPORT void cDeleteModbusDevice(cModbusInterface dev);
00226
00228 // ----- ModbusPort
00229 //
00230
00232 MODBUS_EXPORT cModbusPort cPortCreate(ProtocolType type, const void *settings, bool blocking);
00233
00235 MODBUS EXPORT void cPortDelete (cModbusPort port):
00236
00237
00238 //
00239 // ----- ModbusClientPort
00240 //
00241 #ifndef MB_CLIENT_DISABLE
00242
00244 MODBUS_EXPORT cModbusClientPort cCpoCreate(ProtocolType type, const void *settings, bool blocking);
00245
00247 MODBUS_EXPORT cModbusClientPort cCpoCreateForPort(cModbusPort port);
00248
00250 MODBUS_EXPORT void cCpoDelete(cModbusClientPort clientPort);
00251
00253 MODBUS_EXPORT const Char *cCpoGetObjectName(cModbusClientPort clientPort);
00254
00256 MODBUS EXPORT void cCpoSetObjectName(cModbusClientPort clientPort, const Char *name):
00257
00259 MODBUS_EXPORT ProtocolType cCpoGetType(cModbusClientPort clientPort);
00260
00262 MODBUS_EXPORT bool cCpoIsOpen(cModbusClientPort clientPort);
00263
00265 MODBUS_EXPORT bool cCpoClose(cModbusClientPort clientPort);
00266
```

8.2 cModbus.h

```
00268 MODBUS_EXPORT uint32_t cCpoGetRepeatCount(cModbusClientPort clientPort);
00271 MODBUS_EXPORT void cCpoSetRepeatCount(cModbusClientPort clientPort, uint32_t count);
00272
00273 #ifndef MBF READ COILS DISABLE
00275 MODBUS_EXPORT StatusCode cCpoReadCoils(cModbusClientPort clientPort, uint8_t unit, uint16_t offset,
        uint16_t count, void *values);
00276 #endif // MBF_READ_COILS_DISABLE
00277
00278 #ifndef MBF_READ_DISCRETE_INPUTS_DISABLE
00280 MODBUS_EXPORT StatusCode cCpoReadDiscreteInputs(cModbusClientPort clientPort, uint8_t unit, uint16_t
        offset, uint16_t count, void *values);
00281 #endif // MBF_READ_DISCRETE_INPUTS_DISABLE
00283 #ifndef MBF_READ_HOLDING_REGISTERS_DISABLE
\tt 00285\ MODBUS\_EXPORT\ StatusCode\ cCpoReadHoldingRegisters (cModbusClientPort\ clientPort,\ uint8\_t\ unit,\ uint16\_t
offset, uint16_t count, uint16_t *values);
00286 #endif // MBF_READ_HOLDING_REGISTERS_DISABLE
00288 #ifndef MBF_READ_INPUT_REGISTERS_DISABLE
00290 MODBUS_EXPORT StatusCode cCpoReadInputRegisters(cModbusClientPort clientPort, uint8_t unit, uint16_t
         offset, uint16_t count, uint16_t *values);
00291 #endif // MBF_READ_INPUT_REGISTERS_DISABLE
00292
00293 #ifndef MBF_WRITE_SINGLE_COIL_DISABLE
00295 MODBUS_EXPORT StatusCode cCpoWriteSingleCoil(cModbusClientPort clientPort, uint8_t unit, uint16_t
         offset, bool value);
00296 #endif // MBF_WRITE_SINGLE_COIL_DISABLE
00297
00298 #ifndef MBF WRITE SINGLE REGISTER DISABLE
{\tt 00300~MODBUS\_EXPORT~StatusCode~cCpoWriteSingleRegister(cModbusClientPort~clientPort,~uint16\_t~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~clientPort~
         offset, uint16_t value);
00301 #endif // MBF_READ_EXCEPTION_STATUS_DISABLE
00302
00303 #ifndef MBF_DIAGNOSTICS_DISABLE
00305 MODBUS_EXPORT StatusCode cCpoReadExceptionStatus(cModbusClientPort clientPort, uint8_t unit, uint8_t
         *value);
00306 #endif // MBF_DIAGNOSTICS_DISABLE
00307
00308 #ifndef MBF_DIAGNOSTICS_DISABLE
00310 MODBUS_EXPORT StatusCode cCpoDiagnostics(cModbusClientPort clientPort, uint8_t unit, uint16_t subfunc,
         uint8_t insize, const uint8_t *indata, uint8_t *outsize, uint8_t *outdata);
00311 #endif // MBF_DIAGNOSTICS_DISABLE
00313 #ifndef MBF_GET_COMM_EVENT_COUNTER_DISABLE
00315 MODBUS_EXPORT StatusCode cCpoGetCommEventCounter(cModbusClientPort clientPort, uint8_t unit, uint16_t
         *status, uint16_t *eventCount);
00316 #endif // MBF_GET_COMM_EVENT_COUNTER_DISABLE
00317
00318 #ifndef MBF GET COMM EVENT LOG DISABLE
00320 MODBUS_EXPORT StatusCode cCpoGetCommEventLog(cModbusClientPort clientPort, uint8_t unit, uint16_t
         *status, uint16_t *eventCount, uint16_t *messageCount, uint8_t *eventBuffSize, uint8_t *eventBuff);
00321 #endif // MBF_GET_COMM_EVENT_LOG_DISABLE
00322
00323 #ifndef MBF_WRITE_MULTIPLE_COILS_DISABLE
00325 MODBUS_EXPORT StatusCode cCpoWriteMultipleCoils(cModbusClientPort clientPort, uint8_t unit, uint16_t
         offset, uint16_t count, const void *values);
00326 #endif // MBF_WRITE_MULTIPLE_COILS_DISABLE
00327
00328 #ifndef MBF_WRITE_MULTIPLE_REGISTERS_DISABLE
00330 MODBUS_EXPORT StatusCode cCpoWriteMultipleRegisters(cModbusClientPort clientPort, uint8_t unit, uint16_t offset, uint16_t count, const uint16_t *values);
00331 #endif // MBF_WRITE_MULTIPLE_REGISTERS_DISABLE
00333 #ifndef MBF_REPORT_SERVER_ID_DISABLE
00335 MODBUS_EXPORT StatusCode cCpoReportServerID(cModbusClientPort clientPort, uint8_t unit, uint8_t
*count, uint8_t *data);
00336 #endif // MBF_REPORT_SERVER_ID_DISABLE
00337
00338 #ifndef MBF_MASK_WRITE_REGISTER_DISABLE
00340 MODBUS_EXPORT StatusCode cCpoMaskWriteRegister(cModbusClientPort clientPort, uint8_t unit, uint16_t
         offset, uint16_t andMask, uint16_t orMask);
00341 #endif // MBF_MASK_WRITE_REGISTER_DISABLE
00342
00343 #ifndef MBF READ WRITE MULTIPLE REGISTERS DISABLE
00345 MODBUS_EXPORT StatusCode cCpoReadWriteMultipleRegisters(cModbusClientPort clientPort, uint8_t unit,
         uint16_t readOffset, uint16_t readCount, uint16_t *readValues, uint16_t writeOffset, uint16_t
         writeCount, const uint16_t *writeValues);
00346 #endif // MBF_READ_WRITE_MULTIPLE_REGISTERS_DISABLE
00347
00348 #ifndef MBF READ FIFO QUEUE DISABLE
00350 MODBUS_EXPORT StatusCode cCpoReadFIFOQueue(cModbusClientPort clientPort, uint8_t unit, uint16_t
fifoadr, uint16_t *count, uint16_t *values);
00351 #endif // MBF_READ_FIFO_QUEUE_DISABLE
00352
00353 #ifndef MBF READ COILS DISABLE
00355 MODBUS_EXPORT StatusCode cCpoReadCoilsAsBoolArray(cModbusClientPort clientPort, uint8_t unit, uint16_t
```

```
offset, uint16_t count, bool *values);
00356 #endif // MBF_READ_COILS_DISABLE
00357
00358 #ifndef MBF READ DISCRETE INPUTS DISABLE
00360 MODBUS_EXPORT StatusCode cCpoReadDiscreteInputsAsBoolArray(cModbusClientPort clientPort, uint8_t unit, uint16_t offset, uint16_t count, bool *values);
00361 #endif // MBF_READ_DISCRETE_INPUTS_DISABLE
00362
00363 #ifndef MBF_WRITE_MULTIPLE_COILS_DISABLE
00365 MODBUS_EXPORT StatusCode cCpoWriteMultipleCoilsAsBoolArray(cModbusClientPort clientPort, uint8_t unit,
      uint16_t offset, uint16_t count, const bool *values);
00366 #endif // MBF WRITE MULTIPLE COILS DISABLE
00367
00369 MODBUS_EXPORT StatusCode cCpoGetLastStatus(cModbusClientPort clientPort);
00370
00372 MODBUS_EXPORT StatusCode cCpoGetLastErrorStatus(cModbusClientPort clientPort);
00373
00375 MODBUS EXPORT const Char *cCpoGetLastErrorText(cModbusClientPort clientPort);
00376
00378 MODBUS_EXPORT void cCpoConnectOpened(cModbusClientPort clientPort, pfSlotOpened funcPtr);
00379
00381 MODBUS_EXPORT void cCpoConnectClosed(cModbusClientPort clientPort, pfSlotClosed funcPtr);
00382
00384 MODBUS EXPORT void cCpoConnectTx(cModbusClientPort clientPort, pfSlotTx funcPtr);
00385
00387 MODBUS_EXPORT void cCpoConnectRx(cModbusClientPort clientPort, pfSlotRx funcPtr);
00388
00390 MODBUS_EXPORT void cCpoConnectError(cModbusClientPort clientPort, pfSlotError funcPtr);
00391
00393 MODBUS EXPORT void cCooDisconnectFunc(cModbusClientPort clientPort, void *funcPtr);
00394
00395
00396 //
                          ----- ModbusClient
00397 // ---
                        _____
00398 //
00399
00401 MODBUS_EXPORT cModbusClient cCliCreate(uint8_t unit, ProtocolType type, const void *settings, bool
      blocking);
00402
00404 MODBUS EXPORT cModbusClient cCliCreateForClientPort(uint8 t unit, cModbusClientPort clientPort);
00405
00407 MODBUS EXPORT void cCliDelete(cModbusClient client);
00408
00410 MODBUS_EXPORT const Char *cCliGetObjectName(cModbusClient client);
00411
00413 MODBUS EXPORT void cCliSetObjectName(cModbusClient client, const Char *name);
00414
00416 MODBUS_EXPORT ProtocolType cCliGetType(cModbusClient client);
00417
00419 MODBUS_EXPORT uint8_t cCliGetUnit(cModbusClient client);
00420
00422 MODBUS EXPORT void cCliSetUnit(cModbusClient client, uint8_t unit);
00423
00425 MODBUS_EXPORT bool cCliIsOpen(cModbusClient client);
00426
00428 MODBUS_EXPORT cModbusClientPort cCliGetPort(cModbusClient client);
00429
00431 MODBUS EXPORT StatusCode cReadCoils(cModbusClient client, uint16 t offset, uint16 t count, void
      *values);
00432
00434 MODBUS_EXPORT StatusCode cReadDiscreteInputs(cModbusClient client, uint16_t offset, uint16_t count,
      void *values);
00435
00437 MODBUS_EXPORT StatusCode cReadHoldingRegisters(cModbusClient client, uint16_t offset, uint16_t count,
      uint16 t *values):
00438
00440 MODBUS_EXPORT StatusCode cReadInputRegisters(cModbusClient client, uint16_t offset, uint16_t count,
      uint16_t *values);
00441
00443 MODBUS_EXPORT StatusCode cWriteSingleCoil(cModbusClient client, uint16_t offset, bool value);
00444
00446 MODBUS EXPORT StatusCode cWriteSingleRegister(cModbusClient client, uint16 t offset, uint16 t value);
00447
00449 MODBUS_EXPORT StatusCode cReadExceptionStatus(cModbusClient client, uint8_t *value);
00450
00452 MODBUS_EXPORT StatusCode cWriteMultipleCoils(cModbusClient client, uint16_t offset, uint16_t count,
      const void *values):
00453
00455 MODBUS_EXPORT StatusCode cWriteMultipleRegisters(cModbusClient client, uint16_t offset, uint16_t
      count, const uint16 t *values);
00456
00458 MODBUS_EXPORT StatusCode cMaskWriteRegister(cModbusClient client, uint16_t offset, uint16_t andMask,
      uint16_t orMask);
00459
```

8.2 cModbus.h

```
00461 MODBUS_EXPORT StatusCode cReadWriteMultipleRegisters(cModbusClient client, uint16_t readOffset,
      uint16_t readCount, uint16_t *readValues, uint16_t writeOffset, uint16_t writeCount, const uint16_t
      *writeValues);
00462
00464 MODBUS EXPORT StatusCode cReadCoilsAsBoolArray(cModbusClient client, uint16 t offset, uint16 t count,
     bool *values);
00467 MODBUS_EXPORT StatusCode cReadDiscreteInputsAsBoolArray(cModbusClient client, uint16_t offset,
     uint16_t count, bool *values);
00468
00470 MODBUS_EXPORT StatusCode cWriteMultipleCoilsAsBoolArray(cModbusClient client, uint16_t offset,
     uint16_t count, const bool *values);
00471
00473 MODBUS_EXPORT StatusCode cCliGetLastPortStatus(cModbusClient client);
00474
00476 MODBUS_EXPORT StatusCode cCliGetLastPortErrorStatus(cModbusClient client);
00477
00479 MODBUS EXPORT const Char *cCliGetLastPortErrorText(cModbusClient client);
00480
00481 #endif // MB_CLIENT_DISABLE
00482
00483 //
00484 // -----
                     ----- ModbusServerPort
00485 //
00486
00487 #ifndef MB_SERVER_DISABLE
00488
00490 MODBUS_EXPORT cModbusServerPort cSpoCreate(cModbusInterface device, ProtocolType type, const void
     *settings, bool blocking);
00491
00493 MODBUS_EXPORT void cSpoDelete(cModbusServerPort serverPort);
00494
00496 MODBUS EXPORT const Char *cSpoGetObjectName(cModbusServerPort serverPort);
00497
00499 MODBUS_EXPORT void cSpoSetObjectName(cModbusServerPort serverPort, const Char *name);
00500
00502 MODBUS_EXPORT ProtocolType cSpoGetType(cModbusServerPort serverPort);
00503
00505 MODBUS_EXPORT bool cSpoIsTcpServer(cModbusServerPort serverPort);
00506
00508 MODBUS_EXPORT cModbusInterface cSpoGetDevice(cModbusServerPort serverPort);
00509
00511 MODBUS_EXPORT bool cSpoIsOpen(cModbusServerPort serverPort);
00512
{\tt 00514\ MODBUS\_EXPORT\ StatusCode\ cSpoOpen(cModbusServerPort\ serverPort);}
00515
00517 MODBUS_EXPORT StatusCode cSpoClose(cModbusServerPort serverPort);
00518
00520 MODBUS_EXPORT StatusCode cSpoProcess(cModbusServerPort serverPort);
00521
00523 MODBUS_EXPORT void cSpoConnectOpened(cModbusServerPort serverPort, pfSlotOpened funcPtr);
00524
00526 MODBUS EXPORT void cSpoConnectClosed(cModbusServerPort serverPort, pfSlotClosed funcPtr);
00529 MODBUS_EXPORT void cSpoConnectTx(cModbusServerPort serverPort, pfSlotTx funcPtr);
00530
00532 MODBUS_EXPORT void cSpoConnectRx(cModbusServerPort serverPort, pfSlotRx funcPtr);
00533
00535 MODBUS_EXPORT void cSpoConnectError(cModbusServerPort serverPort, pfSlotError funcPtr);
00536
00538 MODBUS_EXPORT void cSpoConnectNewConnection(cModbusServerPort serverPort, pfSlotNewConnection
      funcPtr);
00539
00541 MODBUS_EXPORT void cSpoConnectCloseConnection(cModbusServerPort serverPort, pfSlotCloseConnection
     funcPtr):
00542
00544 MODBUS_EXPORT void cSpoDisconnectFunc(cModbusServerPort serverPort, void *funcPtr);
00545
00546 #endif // MB_SERVER_DISABLE
00547
00548 #ifdef
              cplusplus
00549 } // extern "C
00550 #endif
00551
00552 #endif // CMODBUS_H
```

## 8.3 c:/Users/march/Dropbox/PRJ/ModbusLib/src/Modbus.h File Reference

Contains general definitions of the Modbus protocol.

```
#include <string>
#include <list>
#include "ModbusGlobal.h"
```

#### Classes

· class ModbusInterface

Main interface of Modbus communication protocol.

#### **Namespaces**

namespace Modbus

Main Modbus namespace. Contains classes, functions and constants to work with Modbus-protocol.

## **Typedefs**

· typedef std::string Modbus::String

Modbus::String class for strings.

• template<class T >

using Modbus::List = std::list<T>

Modbus::List template class.

#### **Functions**

- MODBUS\_EXPORT String Modbus::getLastErrorText ()
- String Modbus::toModbusString (int val)
- MODBUS EXPORT String Modbus::bytesToString (const uint8 t \*buff, uint32 t count)
- MODBUS\_EXPORT String Modbus::asciiToString (const uint8\_t \*buff, uint32\_t count)
- MODBUS\_EXPORT List< String > Modbus::availableSerialPorts ()
- MODBUS\_EXPORT List< int32\_t > Modbus::availableBaudRate ()
- MODBUS\_EXPORT List< int8\_t > Modbus::availableDataBits ()
- MODBUS\_EXPORT List< Parity > Modbus::availableParity ()
- MODBUS\_EXPORT List< StopBits > Modbus::availableStopBits ()
- MODBUS\_EXPORT List< FlowControl > Modbus::availableFlowControl ()
- MODBUS\_EXPORT ModbusPort \* Modbus::createPort (ProtocolType type, const void \*settings, bool blocking)
- MODBUS\_EXPORT ModbusClientPort \* Modbus::createClientPort (ProtocolType type, const void \*settings, bool blocking)
- MODBUS\_EXPORT ModbusServerPort \* Modbus::createServerPort (ModbusInterface \*device, ProtocolType type, const void \*settings, bool blocking)
- StatusCode Modbus::readMemRegs (uint32\_t offset, uint32\_t count, void \*values, const void \*memBuff, uint32\_t memRegCount)
- StatusCode Modbus::writeMemRegs (uint32\_t offset, uint32\_t count, const void \*values, void \*memBuff, uint32\_t memRegCount)
- StatusCode Modbus::readMemBits (uint32\_t offset, uint32\_t count, void \*values, const void \*memBuff, uint32\_t memBitCount)
- StatusCode Modbus::writeMemBits (uint32\_t offset, uint32\_t count, const void \*values, void \*memBuff, uint32\_t memBitCount)

8.4 Modbus.h 171

#### **Detailed Description** 8.3.1

Contains general definitions of the Modbus protocol.

**Author** 

serhmarch

Date

May 2024

#### 8.4 Modbus.h

#### Go to the documentation of this file.

```
00001
00008 #ifndef MODBUS_H
00009 #define MODBUS_H
00010
00011 #include <string>
00012 #include <list>
00013
00014 #include "ModbusGlobal.h"
00016 class ModbusPort;
00017 class ModbusClientPort;
00018 class ModbusServerPort;
00019
00020 //
00021 // ----- Modbus interface
00022 //
00023
00047 class MODBUS_EXPORT ModbusInterface
00048 {
00049 public:
00050
00051 #ifndef MBF_READ_COILS_DISABLE
00058 virtual Modbus::StatusCode readCoils(uint8_t unit, uint16_t offset, uint16_t count, void *values);
00059 #endif // MBF_READ_COILS_DISABLE
00061 #ifndef MBF_READ_DISCRETE_INPUTS_DISABLE
00068
         virtual Modbus::StatusCode readDiscreteInputs(uint8_t unit, uint16_t offset, uint16_t count, void
      *values);
00069 #endif // MBF_READ_DISCRETE_INPUTS_DISABLE
00071 #ifndef MBF_READ_HOLDING_REGISTERS_DISABLE
          virtual Modbus::StatusCode readHoldingRegisters(uint8_t unit, uint16_t offset, uint16_t count,
00078
uint16_t *values);
00079 #endif // MBF_READ_HOLDING_REGISTERS_DISABLE
08000
00081 #ifndef MBF_READ_INPUT_REGISTERS_DISABLE
          virtual Modbus::StatusCode readInputRegisters(uint8_t unit, uint16_t offset, uint16_t count,
88000
      uint16_t *values);
00089 #endif // MBF_READ_INPUT_REGISTERS_DISABLE
00090
00091 #ifndef MBF_WRITE_SINGLE_COIL_DISABLE
         virtual Modbus::StatusCode writeSingleCoil(uint8_t unit, uint16_t offset, bool value);
00097
00098 #endif // MBF_WRITE_SINGLE_COIL_DISABLE
00099
00100 #ifndef MBF_WRITE_SINGLE_REGISTER_DISABLE
00106
          virtual Modbus::StatusCode writeSingleRegister(uint8_t unit, uint16_t offset, uint16_t value);
00107 #endif // MBF_WRITE_SINGLE_REGISTER_DISABLE
00108
00109 #ifndef MBF_READ_EXCEPTION_STATUS_DISABLE
         virtual Modbus::StatusCode readExceptionStatus(uint8_t unit, uint8_t *status);
00115 #endif // MBF_READ_EXCEPTION_STATUS_DISABLE
00116
00117 #ifndef MBF_DIAGNOSTICS_DISABLE
00127 virtual Modbus::StatusCode diagnostics(uint8_t unit, uint16_t subfunc, uint8_t insize, const
      uint8_t *indata, uint8_t *outsize, uint8_t *outdata);
00128 #endif // MBF_DIAGNOSTICS_DISABLE
```

```
00129
00130 #ifndef MBF_GET_COMM_EVENT_COUNTER_DISABLE
         virtual Modbus::StatusCode getCommEventCounter(uint8_t unit, uint16_t *status, uint16_t
00136
     *eventCount):
00137 #endif // MBF GET COMM EVENT COUNTER DISABLE
00138
00139 #ifndef MBF_GET_COMM_EVENT_LOG_DISABLE
          virtual Modbus::StatusCode getCommEventLog(uint8_t unit, uint16_t *status, uint16_t *eventCount,
00148
     uint16_t *messageCount, uint8_t *eventBuffSize, uint8_t *eventBuff);
00149 #endif // MBF_GET_COMM_EVENT_LOG_DISABLE
00150
00151 #ifndef MBF WRITE MULTIPLE COILS DISABLE
         virtual Modbus::StatusCode writeMultipleCoils(uint8_t unit, uint16_t offset, uint16_t count, const
00159
     void *values);
00160 #endif // MBF_WRITE_MULTIPLE_COILS_DISABLE
00161
00162 #ifndef MBF_WRITE_MULTIPLE_REGISTERS_DISABLE
00169 virtual Modbus::StatusCode writeMultipleRegisters(uint8_t unit, uint16_t offset, uint16_t count,
     const uint16_t *values);
00170 #endif // MBF_WRITE_MULTIPLE_REGISTERS_DISABLE
00171
00172 #ifndef MBF_REPORT_SERVER_ID_DISABLE
         virtual Modbus::StatusCode reportServerID(uint8_t unit, uint8_t *count, uint8_t *data);
00178
00179 #endif // MBF_REPORT_SERVER_ID_DISABLE
00180
00181 #ifndef MBF_MASK_WRITE_REGISTER_DISABLE
          virtual Modbus::StatusCode maskWriteRegister(uint8_t unit, uint16_t offset, uint16_t andMask,
00191
     uint16_t orMask);
00192 #endif // MBF_MASK_WRITE_REGISTER_DISABLE
00193
00194 #ifndef MBF_READ_WRITE_MULTIPLE_REGISTERS_DISABLE
00204
          virtual Modbus::StatusCode readWriteMultipleRegisters(uint8_t unit, uint16_t readOffset, uint16_t
     readCount, uint16_t *readValues, uint16_t writeOffset, uint16_t writeCount, const uint16_t
      *writeValues);
00205 #endif // MBF_READ_WRITE_MULTIPLE_REGISTERS_DISABLE
00206
00207 #ifndef MBF READ FIFO QUEUE DISABLE
         virtual Modbus::StatusCode readFIFOQueue(uint8_t unit, uint16_t fifoadr, uint16_t *count, uint16_t
     *values);
00215 #endif // MBF_READ_FIFO_QUEUE_DISABLE
00216
00217 };
00218
00219 //
                     ----- Modbus namespace
00220 // -----
                   ._____
00221 //
00222
00224 namespace Modbus {
00225
00227 typedef std::string String;
00228
00230 template <class T>
00231 using List = std::list<T>;
00232
00234 MODBUS_EXPORT String getLastErrorText();
00235
00238 inline String toModbusString(int val) { return std::to string(val); }
00239
00241 MODBUS_EXPORT String bytesToString(const uint8_t* buff, uint32_t count);
00242
00244 MODBUS_EXPORT String asciiToString(const uint8_t* buff, uint32_t count);
00245
00247 MODBUS EXPORT List<String> availableSerialPorts();
00248
00250 MODBUS_EXPORT List<int32_t> availableBaudRate();
00253 MODBUS_EXPORT List<int8_t> availableDataBits();
00254
00256 MODBUS_EXPORT List<Parity> availableParity();
00257
00259 MODBUS_EXPORT List<StopBits> availableStopBits();
00260
00262 MODBUS_EXPORT List<FlowControl> availableFlowControl();
00263
00268 MODBUS_EXPORT ModbusPort *createPort(ProtocolType type, const void *settings, bool blocking);
00269
00270 #ifndef MB CLIENT DISABLE
00275 MODBUS_EXPORT ModbusClientPort *createClientPort (ProtocolType type, const void *settings, bool
     blocking);
00276 #endif // MB_CLIENT_DISABLE
00277
00278 #ifndef MB SERVER DISABLE
00284 MODBUS_EXPORT ModbusServerPort *createServerPort (ModbusInterface *device, ProtocolType type, const
```

8.5 Modbus\_config.h

```
void *settings, bool blocking);
00285 #endif // MB_SERVER_DISABLE
00286
00288 inline StatusCode readMemRegs(uint32_t offset, uint32_t count, void *values, const void *memBuff,
      uint32_t memRegCount)
00289 {
00290
          return readMemRegs(offset, count , values, memBuff, memRegCount, nullptr);
00291 }
00292
00294 inline StatusCode writeMemRegs(uint32_t offset, uint32_t count, const void *values, void *memBuff,
     uint32_t memRegCount)
00295 {
00296
          return writeMemRegs(offset, count , values, memBuff, memRegCount, nullptr);
00297 }
00298
00300 inline StatusCode readMemBits(uint32_t offset, uint32_t count, void *values, const void *memBuff,
      uint32_t memBitCount)
00301 {
00302
          return readMemBits(offset, count , values, memBuff, memBitCount, nullptr);
00303 }
00304
00306 inline StatusCode writeMemBits(uint32_t offset, uint32_t count, const void *values, void *memBuff,
     uint32_t memBitCount)
00307 {
00308
          return writeMemBits(offset, count , values, memBuff, memBitCount, nullptr);
00309 }
00310
00311 } //namespace Modbus
00312
00313 #endif // MODBUS H
```

## 8.5 Modbus\_config.h

```
00001 #ifndef MODBUS_CONFIG_H
00002 #define MODBUS_CONFIG_H
00003
00004 #define MODBUSLIB_VERSION_MAJOR 0
00005 #define MODBUSLIB_VERSION_MINOR 4
00006 #define MODBUSLIB VERSION PATCH 2
00007
00008 #define MB_DYNAMIC_LINKING
00009
00010 /* #undef MB_CLIENT_DISABLE */
00011 /* #undef MB_SERVER_DISABLE */
00012 /* #undef MB_C_SUPPORT_DISABLE */
00013
00014 /* #undef MBF_READ_COILS_DISABLE */
00015 /* #undef MBF_READ_DISCRETE_INPUTS_DISABLE */
00016 /* #undef MBF_READ_HOLDING_REGISTERS_DISABLE */
00017 /* #undef MBF_READ_INPUT_REGISTERS_DISABLE */
00018 /* #undef MBF_WRITE_SINGLE_COIL_DISABLE */
00019 /* #undef MBF_WRITE_SINGLE_REGISTER_DISABLE */
00020 /* #undef MBF_READ_EXCEPTION_STATUS_DISABLE */
00021 /* #undef MBF_DIAGNOSTICS_DISABLE */
00022 /* #undef MBF_GET_COMM_EVENT_COUNTER_DISABLE */
00023 /* #undef MBF_GET_COMM_EVENT_LOG_DISABLE */
00024 /* #undef MBF_WRITE_MULTIPLE_COILS_DISABLE */
00025 /* #undef MBF_WRITE_MULTIPLE_REGISTERS_DISABLE */
00026 /* #undef MBF_REPORT_SERVER_ID_DISABLE */
00027 /* #undef MBF_READ_FILE_RECORD_DISABLE */
00028 /* #undef MBF_WRITE_FILE_RECORD */
00029 /* #undef MBF_MASK_WRITE_REGISTER_DISABLE */
00030 /* #undef MBF_READ_WRITE_MULTIPLE_REGISTERS_DISABLE */
00031 /* #undef MBF_READ_FIFO_QUEUE_DISABLE */
00034 #endif // MODBUS_CONFIG_H
```

# 8.6 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusAscPort.h File Reference

Contains definition of ASCII serial port class.

```
#include "ModbusSerialPort.h"
```

#### Classes

· class ModbusAscPort

Implements ASCII version of the Modbus communication protocol.

## 8.6.1 Detailed Description

Contains definition of ASCII serial port class.

Contains definition of base server side port class.

**Author** 

serhmarch

Date

May 2024

## 8.7 ModbusAscPort.h

### Go to the documentation of this file.

```
00001
00008 #ifndef MODBUSASCPORT_H
00009 #define MODBUSASCPORT_H
00010
00011 #include "ModbusSerialPort.h"
00012
00019 class MODBUS_EXPORT ModbusAscPort : public ModbusSerialPort
00020 {
00021 public:
00023
         ModbusAscPort(bool blocking = false);
00024
00026
         ~ModbusAscPort();
00027
00028 public:
         Modbus::ProtocolType type() const override { return Modbus::ASC; }
00031
00032 protected:
00033
         Modbus::StatusCode writeBuffer(uint8_t unit, uint8_t func, uint8_t *buff, uint16_t szInBuff)
     override;
00034
         Modbus::StatusCode readBuffer(uint8_t &unit, uint8_t &func, uint8_t *buff, uint16_t maxSzBuff,
     uint16_t *szOutBuff) override;
00035
00036 protected:
00037
          using ModbusSerialPort::ModbusSerialPort;
00038 };
00039
00040 #endif // MODBUSASCPORT_H
```

# 8.8 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusClient.h File Reference

Header file of Modbus client.

```
#include "ModbusObject.h"
```

8.9 ModbusClient.h

#### **Classes**

· class ModbusClient

The ModbusClient class implements the interface of the client part of the Modbus protocol.

## 8.8.1 Detailed Description

Header file of Modbus client.

**Author** 

serhmarch

Date

May 2024

## 8.9 ModbusClient.h

#### Go to the documentation of this file.

```
00008 #ifndef MODBUSCLIENT H
00009 #define MODBUSCLIENT_H
00010
00011 #include "ModbusObject.h"
00013 class ModbusClientPort;
00014
00024 class MODBUS_EXPORT ModbusClient : public ModbusObject
00025 {
00026 public:
          ModbusClient(uint8_t unit, ModbusClientPort *port);
00031
00032 public:
00034
         Modbus::ProtocolType type() const;
00035
00037
          uint8 t unit() const;
00038
00040
          void setUnit(uint8_t unit);
00041
00043
          bool isOpen() const;
00044
00046
          ModbusClientPort *port() const;
00047
00048 public:
00049
00050 #ifndef MBF_READ_COILS_DISABLE
00052
         Modbus::StatusCode readCoils(uint16_t offset, uint16_t count, void *values);
00053 #endif // MBF_READ_COILS_DISABLE
00054
00055 #ifndef MBF_READ_DISCRETE_INPUTS_DISABLE
00057
          Modbus::StatusCode readDiscreteInputs(uint16_t offset, uint16_t count, void *values);
00058 #endif // MBF_READ_DISCRETE_INPUTS_DISABLE
00059
00060 #ifndef MBF_READ_HOLDING_REGISTERS_DISABLE
00062
          Modbus::StatusCode readHoldingRegisters(uint16_t offset, uint16_t count, uint16_t *values);
00063 #endif // MBF_READ_HOLDING_REGISTERS_DISABLE
00064
00065 #ifndef MBF_READ_INPUT_REGISTERS_DISABLE
00067
          Modbus::StatusCode readInputRegisters(uint16_t offset, uint16_t count, uint16_t *values);
00068 #endif // MBF_READ_INPUT_REGISTERS_DISABLE
00069
00070 #ifndef MBF_WRITE_SINGLE_COIL_DISABLE
00072
         Modbus::StatusCode writeSingleCoil(uint16_t offset, bool value);
00073 #endif // MBF_WRITE_SINGLE_COIL_DISABLE
00074
00075 #ifndef MBF_WRITE_SINGLE_REGISTER_DISABLE
00077 Modbus::StatusCode writeSingleRegister(uint16_t offset, uint16_t value);
00078 #endif // MBF_WRITE_SINGLE_REGISTER_DISABLE
```

```
00080 #ifndef MBF_READ_EXCEPTION_STATUS_DISABLE
         Modbus::StatusCode readExceptionStatus(uint8_t *value);
00083 #endif // MBF_READ_EXCEPTION_STATUS_DISABLE
00084
00085 #ifndef MBF DIAGNOSTICS DISABLE
00087
         Modbus::StatusCode diagnostics(uint16 t subfunc, uint8 t insize, const uint8 t *indata, uint8 t
      *outsize, uint8_t *outdata);
00088 #endif // MBF_DIAGNOSTICS_DISABLE
00089
00090 #ifndef MBF_GET_COMM_EVENT_COUNTER_DISABLE
         Modbus::StatusCode getCommEventCounter(uint16_t *status, uint16_t *eventCount);
00092
00093 #endif // MBF_GET_COMM_EVENT_COUNTER_DISABLE
00094
00095 #ifndef MBF_GET_COMM_EVENT_LOG_DISABLE
00097
         Modbus::StatusCode getCommEventLog(uint16_t *status, uint16_t *eventCount, uint16_t *messageCount,
     uint8_t *eventBuffSize, uint8_t *eventBuff);
00098 #endif // MBF_GET_COMM_EVENT_LOG_DISABLE
00099
00100 #ifndef MBF_WRITE_MULTIPLE_COILS_DISABLE
         Modbus::StatusCode writeMultipleCoils(uint16_t offset, uint16_t count, const void *values);
00103 #endif // MBF_WRITE_MULTIPLE_COILS_DISABLE
00104
00105 #ifndef MBF WRITE MULTIPLE REGISTERS DISABLE
         Modbus::StatusCode writeMultipleRegisters(uint16_t offset, uint16_t count, const uint16_t
00107
      *values);
00108 #endif // MBF_WRITE_MULTIPLE_REGISTERS_DISABLE
00109
00110 #ifndef MBF_REPORT_SERVER_ID_DISABLE
00112
         Modbus::StatusCode reportServerID(uint8_t *count, uint8_t *data);
00113 #endif // MBF_REPORT_SERVER_ID_DISABLE
00114
00115 #ifndef MBF_MASK_WRITE_REGISTER_DISABLE
00117
         Modbus::StatusCode maskWriteRegister(uint16_t offset, uint16_t andMask, uint16_t orMask);
00118 #endif // MBF_MASK_WRITE_REGISTER_DISABLE
00119
00120 #ifndef MBF_READ_WRITE_MULTIPLE_REGISTERS DISABLE
         Modbus::StatusCode readWriteMultipleRegisters(uint16_t readOffset, uint16_t readCount, uint16_t
00122
      *readValues, uint16_t writeOffset, uint16_t writeCount, const uint16_t *writeValues);
00123 #endif // MBF_READ_WRITE_MULTIPLE_REGISTERS_DISABLE
00124
00125 #ifndef MBF_READ_FIFO_QUEUE_DISABLE
         Modbus::StatusCode readFIFOQueue(uint16_t fifoadr, uint16_t *count, uint16_t *values);
00127
00128 #endif // MBF READ FIFO QUEUE DISABLE
00130 #ifndef MBF_READ_COILS_DISABLE
00132
         Modbus::StatusCode readCoilsAsBoolArray(uint16_t offset, uint16_t count, bool *values);
00133 #endif // MBF_READ_COILS_DISABLE
00134
00135 #ifndef MBF READ DISCRETE INPUTS DISABLE
         Modbus::StatusCode readDiscreteInputsAsBoolArray(uint16_t offset, uint16_t count, bool *values);
00137
00138 #endif // MBF_READ_DISCRETE_INPUTS_DISABLE
00139
00140 #ifndef MBF_WRITE_MULTIPLE_COILS_DISABLE
00142
         Modbus::StatusCode writeMultipleCoilsAsBoolArray(uint16_t offset, uint16_t count, const bool
     *values);
00143 #endif // MBF_WRITE_MULTIPLE_COILS_DISABLE
00145 public:
00147
         Modbus::StatusCode lastPortStatus() const;
00148
00150
         Modbus::StatusCode lastPortErrorStatus() const;
00151
00153
         const Modbus::Char *lastPortErrorText() const;
00154
00155 protected:
00157
         using ModbusObject::ModbusObject;
00159 };
00160
00161 #endif // MODBUSCLIENT_H
```

## 8.10 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusClientPort.h File Reference

General file of the algorithm of the client part of the Modbus protocol port.

```
#include "ModbusObject.h"
```

8.11 ModbusClientPort.h 177

#### **Classes**

· class ModbusClientPort

The ModbusClientPort class implements the algorithm of the client part of the Modbus communication protocol port.

## 8.10.1 Detailed Description

General file of the algorithm of the client part of the Modbus protocol port.

**Author** 

serhmarch

Date

May 2024

## 8.11 ModbusClientPort.h

#### Go to the documentation of this file.

```
00001
00008 #ifndef MODBUSCLIENTPORT_H
00009 #define MODBUSCLIENTPORT_H
00010
00011 #include "ModbusObject.h"
00012
00013 class ModbusPort:
00014
00054 class MODBUS_EXPORT ModbusClientPort : public ModbusObject, public ModbusInterface
00055 {
00056 public:
00059
          enum RequestStatus
00060
00061
              Enable,
00062
              Disable,
00063
              Process
00064
00065
00066 public:
00070
          ModbusClientPort(ModbusPort *port);
00071
00072 public:
00074
         Modbus::ProtocolType type() const;
00075
00077
          ModbusPort *port() const;
00078
08000
          void setPort(ModbusPort *port);
00081
00083
          Modbus::StatusCode close();
00084
00086
          bool isOpen() const;
00087
00089
          uint32_t tries() const;
00090
00092
          void setTries(uint32_t v);
00093
00095
          inline uint32_t repeatCount() const { return tries(); }
00096
00098
          inline void setRepeatCount(uint32_t v) { setTries(v); }
00099
00102
          bool isBroadcastEnabled() const;
00103
00106
          void setBroadcastEnabled(bool enable);
00107
00108 public: // Main interface
00110 #ifndef MBF_READ_COILS_DISABLE
```

```
Modbus::StatusCode readCoils(ModbusObject *client, uint8_t unit, uint16_t offset, uint16_t count,
      void *values);
00113 #endif // MBF_READ_COILS_DISABLE
00114
00115 #ifndef MBF_READ_DISCRETE INPUTS DISABLE
         Modbus::StatusCode readDiscreteInputs(ModbusObject *client, uint8_t unit, uint16_t offset,
00117
      uint16_t count, void *values);
00118 #endif // MBF_READ_DISCRETE_INPUTS_DISABLE
00119
00120 #ifndef MBF_READ_HOLDING_REGISTERS_DISABLE
         Modbus::StatusCode readHoldingRegisters(ModbusObject *client, uint8_t unit, uint16_t offset,
00122
     uint16_t count, uint16_t *values);
00123 #endif // MBF_READ_HOLDING_REGISTERS_DISABLE
00125 #ifndef MBF_READ_INPUT_REGISTERS_DISABLE
     Modbus::StatusCode readInputRegisters(ModbusObject *client, uint8_t unit, uint16_t offset,
uint16_t count, uint16_t *values);
00127
00128 #endif // MBF_READ_INPUT_REGISTERS_DISABLE
00130 #ifndef MBF_WRITE_SINGLE_COIL_DISABLE
          Modbus::StatusCode writeSingleCoil(ModbusObject *client, uint8_t unit, uint16_t offset, bool
     value);
00133 #endif // MBF_WRITE_SINGLE_COIL_DISABLE
00134
00135 #ifndef MBF_WRITE_SINGLE_REGISTER_DISABLE
         Modbus::StatusCode writeSingleRegister(ModbusObject *client, uint8_t unit, uint16_t offset,
00137
     uint16_t value);
00138 #endif // MBF_WRITE_SINGLE_REGISTER_DISABLE
00139
00140 #ifndef MBF_READ_EXCEPTION_STATUS_DISABLE
        Modbus::StatusCode readExceptionStatus(ModbusObject *client, uint8_t unit, uint8_t *value);
00142
00143 #endif // MBF_READ_EXCEPTION_STATUS_DISABLE
00144
00145 #ifndef MBF_DIAGNOSTICS_DISABLE
         Modbus::StatusCode diagnostics(ModbusObject *client, uint8_t unit, uint16_t subfunc, uint8_t
00147
     insize, const uint8_t *indata, uint8_t *outsize, uint8_t *outdata);
00148 #endif // MBF_DIAGNOSTICS_DISABLE
00150 #ifndef MBF_GET_COMM_EVENT_COUNTER_DISABLE
         Modbus::StatusCode getCommEventCounter(ModbusObject *client, uint8_t unit, uint16_t *status,
     uint16_t *eventCount);
00153 #endif // MBF_GET_COMM_EVENT_COUNTER_DISABLE
00154
00155 #ifndef MBF_GET_COMM_EVENT_LOG_DISABLE
        Modbus::StatusCode getCommEventLog(ModbusObject *client, uint8_t unit, uint16_t *status, uint16_t
     *eventCount, uint16_t *messageCount, uint8_t *eventBuffSize, uint8_t *eventBuff);
00158 #endif // MBF_GET_COMM_EVENT_LOG_DISABLE
00159
00160 #ifndef MBF WRITE MULTIPLE COILS DISABLE
         Modbus::StatusCode writeMultipleCoils(ModbusObject *client, uint8_t unit, uint16_t offset,
00162
      uint16_t count, const void *values);
00163 #endif // MBF_WRITE_MULTIPLE_COILS_DISABLE
00164
00165 #ifndef MBF_WRITE_MULTIPLE_REGISTERS_DISABLE
         Modbus::StatusCode writeMultipleRegisters(ModbusObject *client, uint8_t unit, uint16_t offset,
00167
      uint16_t count, const uint16_t *values);
00168 #endif // MBF_WRITE_MULTIPLE_REGISTERS_DISABLE
00169
00170 #ifndef MBF_REPORT_SERVER_ID_DISABLE
00172
         Modbus::StatusCode reportServerID (ModbusObject *client, uint8_t unit, uint8_t *count, uint8_t
      *data):
00173 #endif // MBF_REPORT_SERVER_ID_DISABLE
00174
00175 #ifndef MBF_MASK_WRITE_REGISTER_DISABLE
         Modbus::StatusCode maskWriteRegister(ModbusObject *client, uint8_t unit, uint16_t offset, uint16_t
00177
andMask, uint16_t orMask);
00178 #endif // MBF_MASK_WRITE_REGISTER_DISABLE
00179
00180 #ifndef MBF_READ_WRITE_MULTIPLE_REGISTERS_DISABLE
         Modbus::StatusCode readWriteMultipleRegisters(ModbusObject *client, uint8_t unit, uint16_t
      readOffset, uint16_t readCount, uint16_t *readValues, uint16_t writeOffset, uint16_t writeCount, const
      uint16_t *writeValues);
00183 #endif // MBF_READ_WRITE_MULTIPLE_REGISTERS_DISABLE
00184
00185 #ifndef MBF_READ_FIFO_QUEUE_DISABLE
        Modbus::StatusCode readFIFOQueue(ModbusObject *client, uint8_t unit, uint16_t fifoadr, uint16_t
      *count, uint16_t *values);
00188 #endif // MBF_READ_FIFO_QUEUE_DISABLE
00189
00190 #ifndef MBF READ COILS DISABLE
         Modbus::StatusCode readCoilsAsBoolArray(ModbusObject *client, uint8_t unit, uint16_t offset,
00192
      uint16_t count, bool *values);
00193 #endif // MBF_READ_COILS_DISABLE
00194
00195 #ifndef MBF_READ_DISCRETE_INPUTS_DISABLE
         Modbus::StatusCode readDiscreteInputsAsBoolArray(ModbusObject *client, uint8_t unit, uint16_t
00197
      offset, uint16_t count, bool *values);
```

8.11 ModbusClientPort.h 179

```
00198 #endif // MBF_READ_DISCRETE_INPUTS_DISABLE
00199
00200 #ifndef MBF_WRITE_MULTIPLE_COILS_DISABLE
         Modbus::StatusCode writeMultipleCoilsAsBoolArray(ModbusObject *client, uint8_t unit, uint16_t
00202
     offset, uint16_t count, const bool *values);
00203 #endif // MBF_WRITE_MULTIPLE_COILS_DISABLE
00205 public: // Modbus Interface
00206
00207 #ifndef MBF READ COILS DISABLE
         Modbus::StatusCode readCoils(uint8_t unit, uint16_t offset, uint16_t count, void *values)
00208
     override;
00209 #endif // MBF_READ_COILS_DISABLE
00210
00211 #ifndef MBF_READ_DISCRETE_INPUTS_DISABLE
         Modbus::StatusCode readDiscreteInputs(uint8_t unit, uint16_t offset, uint16_t count, void *values)
00212
     override:
00213 #endif // MBF_READ_DISCRETE_INPUTS_DISABLE
00215 #ifndef MBF_READ_HOLDING_REGISTERS_DISABLE
          Modbus::StatusCode readHoldingRegisters(uint8_t unit, uint16_t offset, uint16_t count, uint16_t
      *values) override;
00217 #endif // MBF_READ_HOLDING_REGISTERS_DISABLE
00218
00219 #ifndef MBF_READ_INPUT_REGISTERS_DISABLE
         Modbus::StatusCode readInputRegisters(uint8_t unit, uint16_t offset, uint16_t count, uint16_t
00220
      *values) override;
00221 #endif // MBF_READ_INPUT_REGISTERS_DISABLE
00222
00223 #ifndef MBF_WRITE_SINGLE_COIL_DISABLE
        Modbus::StatusCode writeSingleCoil(uint8 t unit, uint16 t offset, bool value) override;
00224
00225 #endif // MBF_WRITE_SINGLE_COIL_DISABLE
00226
00227 #ifndef MBF_WRITE_SINGLE_REGISTER_DISABLE
00228
         Modbus::StatusCode writeSingleRegister(uint8_t unit, uint16_t offset, uint16_t value) override;
00229 #endif // MBF_WRITE_SINGLE_REGISTER_DISABLE
00230
00231 #ifndef MBF_READ_EXCEPTION_STATUS_DISABLE
00232
         Modbus::StatusCode readExceptionStatus(uint8_t unit, uint8_t *value) override;
00233 #endif // MBF_READ_EXCEPTION_STATUS_DISABLE
00234
00235 #ifndef MBF DIAGNOSTICS DISABLE
         Modbus::StatusCode diagnostics(uint8_t unit, uint16_t subfunc, uint8_t insize, const uint8_t
00236
      *indata, uint8_t *outsize, uint8_t *outdata) override;
00237 #endif // MBF_DIAGNOSTICS_DISABLE
00238
00239 #ifndef MBF GET COMM EVENT COUNTER DISABLE
00240
       Modbus::StatusCode getCommEventCounter(uint8_t unit, uint16_t *status, uint16_t *eventCount)
     override:
00241 #endif // MBF_GET_COMM_EVENT_COUNTER_DISABLE
00243 #ifndef MBF_GET_COMM_EVENT_LOG_DISABLE
00244
         Modbus::StatusCode getCommEventLog(uint8_t unit, uint16_t *status, uint16_t *eventCount, uint16_t
      *messageCount, \ uint8\_t \ *eventBuffSize, \ uint8\_t \ *eventBuff) \ override;
00245 #endif // MBF_GET_COMM_EVENT_LOG_DISABLE
00246
00247 #ifndef MBF WRITE MULTIPLE COILS DISABLE
         Modbus::StatusCode writeMultipleCoils(uint8_t unit, uint16_t offset, uint16_t count, const void
     *values) override;
00249 #endif // MBF_WRITE_MULTIPLE_COILS_DISABLE
00250
00251 #ifndef MBF WRITE MULTIPLE REGISTERS DISABLE
00252
         Modbus::StatusCode writeMultipleRegisters(uint8_t unit, uint16_t offset, uint16_t count, const
      uint16_t *values) override;
00253 #endif // MBF_WRITE_MULTIPLE_REGISTERS_DISABLE
00254
00255 #ifndef MBF_REPORT_SERVER_ID_DISABLE
00256 Modbus::StatusCode reportServerID(uint8_t unit, uint8_t *count, uint8_t *data) override;
00257 #endif // MBF_REPORT_SERVER_ID_DISABLE
00259 #ifndef MBF_MASK_WRITE_REGISTER_DISABLE
         Modbus::StatusCode maskWriteRegister(uint8_t unit, uint16_t offset, uint16_t andMask, uint16_t
00260
     orMask) override;
00261 #endif // MBF_MASK_WRITE_REGISTER_DISABLE
00262
00263 #ifndef MBF_READ_WRITE_MULTIPLE_REGISTERS_DISABLE
          Modbus::StatusCode readWriteMultipleRegisters(uint8_t unit, uint16_t readOffset, uint16_t
00264
      readCount, uint16_t *readValues, uint16_t writeOffset, uint16_t writeCount, const uint16_t
      *writeValues) override;
00265 #endif // MBF READ WRITE MULTIPLE REGISTERS DISABLE
00266
00267 #ifndef MBF_READ_FIFO_QUEUE_DISABLE
          Modbus::StatusCode readFIFOQueue(uint8_t unit, uint16_t fifoadr, uint16_t *count, uint16_t
      *values) override;
00269 #endif // MBF_READ_FIFO_QUEUE_DISABLE
00270
00271 #ifndef MBF_READ_COILS_DISABLE
```

```
inline Modbus::StatusCode readCoilsAsBoolArray(uint8_t unit, uint16_t offset, uint16_t count, bool
      *values) { return readCoilsAsBoolArray(this, unit, offset, count, values); }
00274 #endif // MBF_READ_COILS_DISABLE
00275
00276 #ifndef MBF_READ_DISCRETE_INPUTS_DISABLE
         inline Modbus::StatusCode readDiscreteInputsAsBoolArray(uint8_t unit, uint16_t offset, uint16_t
00278
      count, bool *values) { return readDiscreteInputsAsBoolArray(this, unit, offset, count, values); }
00279 #endif // MBF_READ_DISCRETE_INPUTS_DISABLE
00280
00281 #ifndef MBF_WRITE_MULTIPLE_COILS_DISABLE
         inline Modbus::StatusCode writeMultipleCoilsAsBoolArray(uint8_t unit, uint16_t offset, uint16_t
00283
      count, const bool *values) { return writeMultipleCoilsAsBoolArray(this, unit, offset, count, values);
00284 #endif // MBF_WRITE_MULTIPLE_COILS_DISABLE
00285
00286 public:
00288
          Modbus::StatusCode lastStatus() const;
00289
          Modbus::Timestamp lastStatusTimestamp() const;
00292
00294
          Modbus::StatusCode lastErrorStatus() const;
00295
00297
         const Modbus::Char *lastErrorText() const;
00298
00299 public:
         const ModbusObject *currentClient() const;
00302
00308
          RequestStatus getRequestStatus(ModbusObject *client);
00309
00311
          void cancelRequest(ModbusObject *client);
00312
00313 public: // SIGNALS
00315
         void signalOpened(const Modbus::Char *source);
00316
00318
          void signalClosed(const Modbus::Char *source);
00319
00321
          void signalTx(const Modbus::Char *source, const uint8 t* buff, uint16 t size);
00322
00324
          void signalRx(const Modbus::Char *source, const uint8_t* buff, uint16_t size);
00325
00327
          void signalError(const Modbus::Char *source, Modbus::StatusCode status, const Modbus::Char *text);
00328
00329 private:
00330
         Modbus::StatusCode request(uint8_t unit, uint8_t func, uint8_t *buff, uint16_t szInBuff, uint16_t
     maxSzBuff, uint16_t *szOutBuff);
          Modbus::StatusCode process();
00331
00332
          friend class ModbusClient;
00333 };
00334
00335 #endif // MODBUSCLIENTPORT_H
```

## 8.12 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusGlobal.h File Reference

Contains general definitions of the Modbus libarary (for C++ and "pure" C).

```
#include <stdint.h>
#include <string.h>
#include "ModbusPlatform.h"
#include "Modbus_config.h"
```

#### Classes

• struct Modbus::SerialSettings

Struct to define settings for Serial Port.

struct Modbus::TcpSettings

Struct to define settings for TCP connection.

#### **Namespaces**

namespace Modbus

Main Modbus namespace. Contains classes, functions and constants to work with Modbus-protocol.

#### **Macros**

ModbusLib version value that defines as MODBUSLIB\_VERSION = (major << 16) + (minor << 8) + patch.

#define MODBUSLIB\_VERSION\_STR MODBUSLIB\_VERSION\_STR\_MAKE(MODBUSLIB\_VERSION\_

 MAJOR,MODBUSLIB VERSION MINOR,MODBUSLIB VERSION PATCH)

ModbusLib version value that defines as MODBUSLIB\_VERSION\_STR "major.minor.patch".

#define MODBUS\_EXPORT MB\_DECL\_IMPORT

MODBUS\_EXPORT defines macro for import/export functions and classes.

#define StringLiteral(cstr)

Macro for creating string literal, must be used like: StringLiteral ("Some string")

• #define CharLiteral(cchar)

Macro for creating char literal, must be used like: 'CharLiteral('A')'.

#define GET BIT(bitBuff, bitNum)

Macro for get bit with number bitNum from array bitBuff.

#define SET\_BIT(bitBuff, bitNum, value)

Macro for set bit value with number bit Num to array bit Buff.

#define GET\_BITS(bitBuff, bitNum, bitCount, boolBuff)

Macro for get bits begins with number bitNum with count from input bit array bitBuff to output bool array boolBuff.

#define SET\_BITS(bitBuff, bitNum, bitCount, boolBuff)

Macro for set bits begins with number bitNum with count from input bool array boolBuffto output bit array bitBuff.

- #define MB\_UNITMAP\_SIZE 32
- #define MB\_UNITMAP\_GET\_BIT(unitmap, unit)
- #define MB\_UNITMAP\_SET\_BIT(unitmap, unit, value)
- #define MB BYTE SZ BITES 8

8 = count bits in byte (byte size in bits)

#define MB\_REGE\_SZ\_BITES 16

16 = count bits in 16 bit register (register size in bits)

• #define MB\_REGE\_SZ\_BYTES 2

2 = count bytes in 16 bit register (register size in bytes)

• #define MB\_MAX\_BYTES 255

255 - count\_of\_bytes in function readHoldingRegisters, readCoils etc

• #define MB\_MAX\_REGISTERS 127

127 = 255(count\_of\_bytes in function readHoldingRegisters etc) / 2 (register size in bytes)

#define MB MAX DISCRETS 2040

2040 = 255(count\_of\_bytes in function readCoils etc) \* 8 (bits in byte)

#define MB\_VALUE\_BUFF\_SZ 255

Same as MB\_MAX\_BYTES

#define MB\_RTU\_IO\_BUFF\_SZ 264

Maximum func data size: WriteMultipleCoils 261 = 1 byte(function) + 2 bytes (starting offset) + 2 bytes (count) + 1 bytes (byte count) + 255 bytes(maximum data length)

• #define MB\_ASC\_IO\_BUFF\_SZ 529

1 byte(start symbol ':')+(( 1 byte(unit) + 261 (max func data size: WriteMultipleCoils)) + 1 byte(LRC) ))\*2+2 bytes(CR+LF)

#define MB\_TCP\_IO\_BUFF\_SZ 268

6 bytes(tcp-prefix)+1 byte(unit)+261 (max func data size: WriteMultipleCoils)

• #define GET COMM EVENT LOG MAX 64

Maximum events for GetCommEventLog function.

#define READ\_FIFO\_QUEUE\_MAX 31

Maximum events for GetCommEventLog function.

#### **Modbus Functions**

Modbus Function's codes.

- #define MBF READ COILS 1
- #define MBF\_READ\_DISCRETE\_INPUTS 2
- #define MBF\_READ\_HOLDING\_REGISTERS 3
- #define MBF READ INPUT REGISTERS 4
- #define MBF WRITE SINGLE COIL 5
- #define MBF\_WRITE\_SINGLE\_REGISTER 6
- #define MBF\_READ\_EXCEPTION\_STATUS 7
- #define MBF\_DIAGNOSTICS 8
- #define MBF\_GET\_COMM\_EVENT\_COUNTER 11
- #define MBF\_GET\_COMM\_EVENT\_LOG 12#define MBF\_WRITE\_MULTIPLE\_COILS 15
- #define MBF\_WRITE\_MULTIPLE\_REGISTERS 16
- #define MBF\_REPORT\_SERVER\_ID 17
- #define MBF READ\_FILE\_RECORD 20
- #define MBF WRITE FILE RECORD 21
- #define MBF\_MASK\_WRITE\_REGISTER 22
- #define MBF READ WRITE MULTIPLE REGISTERS 23
- #define MBF\_READ\_FIFO\_QUEUE 24
- #define MBF\_ENCAPSULATED\_INTERFACE\_TRANSPORT 43
- #define MBF\_ILLEGAL\_FUNCTION 73
- #define MBF\_EXCEPTION 128

#### **Typedefs**

typedef void \* Modbus::Handle

Handle type for native OS values.

• typedef char Modbus::Char

Type for Modbus character.

• typedef uint32\_t Modbus::Timer

Type for Modbus timer.

typedef int64 t Modbus::Timestamp

Type for Modbus timestamp (in UNIX millisec format)

typedef enum Modbus:: MemoryType Modbus::MemoryType

Defines type of memory used in Modbus protocol.

## **Enumerations**

 enum Modbus::Constants { Modbus::VALID\_MODBUS\_ADDRESS\_BEGIN = 1, Modbus::VALID\_MODBUS\_ADDRESS\_END = 247, Modbus::STANDARD TCP PORT = 502}

Define list of contants of Modbus protocol.

enum Modbus:: MemoryType {

Modbus::Memory Unknown = 0xFFFF, Modbus::Memory 0x = 0, Modbus::Memory Coils = Memory 0x, Modbus::Memory 1x = 1,

Modbus::Memory DiscreteInputs = Memory 1x, Modbus::Memory 3x = 3, Modbus::Memory InputRegisters = Memory\_3x , Modbus::Memory\_4x = 4 ,

Modbus::Memory\_HoldingRegisters = Memory\_4x }

```
Defines type of memory used in Modbus protocol.
enum Modbus::StatusCode {
     Modbus::Status_Processing = 0x80000000 , Modbus::Status_Good = 0x00000000 , Modbus::Status_Bad =
     0x01000000, Modbus::Status_Uncertain = 0x02000000,
     Modbus::Status BadlllegalFunction = Status Bad | 0x01 , Modbus::Status BadlllegalDataAddress =
     Status Bad | 0x02, Modbus::Status BadlllegalDataValue = Status Bad | 0x03, Modbus::Status BadServerDeviceFailure
     = Status Bad | 0x04,
     Modbus::Status BadAcknowledge = Status Bad | 0x05, Modbus::Status BadServerDeviceBusy = Status↔
      _Bad | 0x06 , Modbus::Status_BadNegativeAcknowledge = Status_Bad | 0x07 , Modbus::Status BadMemoryParityError
     = Status Bad | 0x08,
     Modbus::Status BadGatewayPathUnavailable = Status Bad | 0x0A, Modbus::Status BadGatewayTargetDeviceFailedToRespo
     = Status Bad | 0x0B , Modbus::Status BadEmptyResponse = Status Bad | 0x101 , Modbus::Status BadNotCorrectRequest
     Modbus::Status_BadNotCorrectResponse, Modbus::Status_BadWriteBufferOverflow, Modbus::Status_BadReadBufferOverflow, Modbus::Status_BadReadBufferOve
      , Modbus::Status_BadSerialOpen = Status_Bad | 0x201 ,
     Modbus::Status_BadSerialWrite, Modbus::Status_BadSerialRead, Modbus::Status_BadSerialReadTimeout
      , Modbus::Status BadSerialWriteTimeout,
     Modbus::Status\_BadAscMissColon = Status\_Bad \mid 0x301 \text{ , } Modbus::Status\_BadAscMissCrLf \text{ , } Modbus::Status\_BadAscChar \text{ } AscChar \text{ } AscCha
      , Modbus::Status BadLrc ,
     Modbus::Status BadCrc = Status Bad | 0x401, Modbus::Status BadTcpCreate = Status Bad | 0x501,
     Modbus::Status BadTcpConnect, Modbus::Status BadTcpWrite,
     Modbus::Status_BadTcpRead, Modbus::Status_BadTcpBind, Modbus::Status_BadTcpListen, Modbus::Status_BadTcpAccep
     Modbus::Status_BadTcpDisconnect }
               Defines status of executed Modbus functions.

    enum Modbus::ProtocolType { Modbus::ASC , Modbus::RTU , Modbus::TCP }

               Defines type of Modbus protocol.
enum Modbus::Parity {
     Modbus::NoParity, Modbus::EvenParity, Modbus::OddParity, Modbus::SpaceParity,
     Modbus::MarkParity }
               Defines Parity for serial port.

    enum Modbus::StopBits { Modbus::OneStop , Modbus::OneAndHalfStop , Modbus::TwoStop }

               Defines Stop Bits for serial port.

    enum Modbus::FlowControl { Modbus::NoFlowControl , Modbus::HardwareControl , Modbus::SoftwareControl
```

#### **Functions**

}

- bool Modbus::StatusIsProcessing (StatusCode status)
- bool Modbus::StatusIsGood (StatusCode status)
- bool Modbus::StatusIsBad (StatusCode status)

FlowControl Parity for serial port.

- bool Modbus::StatusIsUncertain (StatusCode status)
- bool Modbus::StatusIsStandardError (StatusCode status)
- bool Modbus::getBit (const void \*bitBuff, uint16 t bitNum)
- bool Modbus::getBitS (const void \*bitBuff, uint16 t bitNum, uint16 t maxBitCount)
- void Modbus::setBit (void \*bitBuff, uint16\_t bitNum, bool value)
- void Modbus::setBitS (void \*bitBuff, uint16\_t bitNum, bool value, uint16\_t maxBitCount)
- bool \* Modbus::getBits (const void \*bitBuff, uint16\_t bitNum, uint16\_t bitCount, bool \*boolBuff)
- bool \* Modbus::getBitsS (const void \*bitBuff, uint16\_t bitNum, uint16\_t bitCount, bool \*boolBuff, uint16\_t maxBitCount)
- void \* Modbus::setBits (void \*bitBuff, uint16 t bitNum, uint16 t bitCount, const bool \*boolBuff)
- void \* Modbus::setBitsS (void \*bitBuff, uint16\_t bitNum, uint16\_t bitCount, const bool \*boolBuff, uint16\_t maxBitCount)
- MODBUS\_EXPORT uint32\_t Modbus::modbusLibVersion ()

- MODBUS\_EXPORT const Char \* Modbus::modbusLibVersionStr ()
- uint16\_t Modbus::toModbusOffset (uint32\_t adr)
- MODBUS EXPORT uint16 t Modbus::crc16 (const uint8 t \*byteArr, uint32 t count)
- MODBUS EXPORT uint8 t Modbus::Irc (const uint8 t \*byteArr, uint32 t count)
- MODBUS\_EXPORT StatusCode Modbus::readMemRegs (uint32\_t offset, uint32\_t count, void \*values, const void \*memBuff, uint32\_t memRegCount, uint32\_t \*outCount)
- MODBUS\_EXPORT StatusCode Modbus::writeMemRegs (uint32\_t offset, uint32\_t count, const void \*values, void \*memBuff, uint32\_t memRegCount, uint32\_t \*outCount)
- MODBUS\_EXPORT StatusCode Modbus::readMemBits (uint32\_t offset, uint32\_t count, void \*values, const void \*memBuff, uint32\_t memBitCount, uint32\_t \*outCount)
- MODBUS\_EXPORT StatusCode Modbus::writeMemBits (uint32\_t offset, uint32\_t count, const void \*values, void \*memBuff, uint32\_t memBitCount, uint32\_t \*outCount)
- MODBUS\_EXPORT uint32\_t Modbus::bytesToAscii (const uint8\_t \*bytesBuff, uint8\_t \*asciiBuff, uint32\_←
  t count)
- MODBUS\_EXPORT uint32\_t Modbus::asciiToBytes (const uint8\_t \*asciiBuff, uint8\_t \*bytesBuff, uint32\_←
  t count)
- MODBUS\_EXPORT Char \* Modbus::sbytes (const uint8\_t \*buff, uint32\_t count, Char \*str, uint32\_t str-maxlen)
- MODBUS EXPORT Char \* Modbus::sascii (const uint8 t \*buff, uint32 t count, Char \*str, uint32 t strmaxlen)
- MODBUS EXPORT const Char \* Modbus::sprotocolType (ProtocolType type)
- MODBUS\_EXPORT const Char \* Modbus::sparity (Parity parity)
- MODBUS EXPORT const Char \* Modbus::sstopBits (StopBits stopBits)
- MODBUS\_EXPORT const Char \* Modbus::sflowControl (FlowControl flowControl)
- MODBUS EXPORT Timer Modbus::timer ()
- MODBUS EXPORT Timestamp Modbus::currentTimestamp ()
- MODBUS EXPORT void Modbus::msleep (uint32 t msec)

## 8.12.1 Detailed Description

Contains general definitions of the Modbus libarary (for C++ and "pure" C).

**Author** 

serhmarch

Date

May 2024

### 8.12.2 Macro Definition Documentation

### 8.12.2.1 CharLiteral

```
#define CharLiteral(
cchar)
```

## Value:

cchai

Macro for creating char literal, must be used like: 'CharLiteral('A')'.

#### 8.12.2.2 GET\_BIT

Macro for get bit with number bitNum from array bitBuff.

## 8.12.2.3 **GET\_BITS**

Macro for get bits begins with number bitNum with count from input bit array bitBuff to output bool array boolBuff.

## 8.12.2.4 MB\_RTU\_IO\_BUFF\_SZ

```
#define MB_RTU_IO_BUFF_SZ 264
```

Maximum func data size: WriteMultipleCoils 261 = 1 byte(function) + 2 bytes (starting offset) + 2 bytes (count) + 1 bytes (byte count) + 255 bytes(maximum data length)

1 byte(unit) + 261 (max func data size: WriteMultipleCoils) + 2 bytes(CRC)

## 8.12.2.5 MB\_UNITMAP\_GET\_BIT

#### 8.12.2.6 MB UNITMAP SET BIT

## 8.12.2.7 SET\_BIT

Macro for set bit value with number bitNum to array bitBuff.

## 8.12.2.8 SET\_BITS

Macro for set bits begins with number bitNum with count from input bool array boolBuffto output bit array bitBuff.

#### 8.12.2.9 StringLiteral

#### Value:

cstr

Macro for creating string literal, must be used like: StringLiteral("Some string")

8.13 ModbusGlobal.h

## 8.13 ModbusGlobal.h

#### Go to the documentation of this file.

```
00001
00008 #ifndef MODBUSGLOBAL H
00009 #define MODBUSGLOBAL_H
00010
00011 #include <stdint.h>
00012 #include <string.h>
00013
00014 #ifdef QT_CORE_LIB
00015 #include <qobjectdefs.h>
00016 #endif
00017
00018 #include "ModbusPlatform.h"
00019 #include "Modbus_config.h"
00020
00022 #define MODBUSLIB VERSION
      ((MODBUSLIB_VERSION_MAJOR«16)|(MODBUSLIB_VERSION_MINOR«8)|(MODBUSLIB_VERSION_PATCH))
00023
00025 #define MODBUSLIB_VERSION_STR_HELPER(major,minor,patch) #major"."#minor"."#patch
00026
00027 #define MODBUSLIB_VERSION_STR_MAKE(major,minor,patch) MODBUSLIB_VERSION_STR_HELPER(major,minor,patch)
00029
00031 #define MODBUSLIB_VERSION_STR
     MODBUSLIB_VERSION_STR_MAKE (MODBUSLIB_VERSION_MAJOR, MODBUSLIB_VERSION_MINOR, MODBUSLIB_VERSION_PATCH)
00032
00033
00034
00036 #ifdef MB_DYNAMIC_LINKING
00037
00038 #if defined(MODBUS_EXPORTS) && defined(MB_DECL_EXPORT)
00039 #define MODBUS_EXPORT MB_DECL_EXPORT
00040 #elif defined(MB_DECL_IMPORT)
00041 #define MODBUS_EXPORT MB_DECL_IMPORT
00042 #else
00043 #define MODBUS EXPORT
00044 #endif
00045
00046 #else // MB_DYNAMIC_LINKING
00047
00048 #define MODBUS EXPORT
00049
00050 #endif // MB_DYNAMIC_LINKING
00052
00053
00055 #define StringLiteral(cstr) cstr
00056
00058 #define CharLiteral(cchar) cchar
00060 //
00061 // -----
                 ----- Helper macros
00062 //
00063
00065
     00066
00068 #define SET BIT (bitBuff, bitNum, value)
00069
         if (value)
00070
             ((uint8_t*)(bitBuff))[(bitNum)/8] = (1 < ((bitNum) %8));
00071
00072
             ((uint8_t*)(bitBuff))[(bitNum)/8] &= (~(1«((bitNum)%8)));
00073
00075
     #define GET_BITS(bitBuff, bitNum, bitCount, boolBuff)
00076
         for (uint16_t __i_ = 0; __i_ < bitCount; __i_++)
             boolBuff[_i_] = (((const uint8_t*)(bitBuff))[((bitNum)+__i__)/8] & (1«(((bitNum)+__i__)%8)))
00077
       = 0;
00078
08000
     #define SET_BITS(bitBuff, bitNum, bitCount, boolBuff)
00081
         for (uint16_t __i_ = 0; __i_ < bitCount; __i_
00082
             if (boolBuff[__i__])
00083
                 ((uint8_t*)(bitBuff))[((bitNum)+__i__)/8] = (1 (((bitNum)+__i__) %8));
```

```
00084
             else
00085
                ((uint8_t*)(bitBuff))[((bitNum)+__i__)/8] &= (~(1«(((bitNum)+__i__)%8)));
00086
00088 #define MB UNITMAP SIZE 32
00089
00091 #define MB_UNITMAP_GET_BIT(unitmap, unit) ((((const uint8_t*)(unitmap))[(unit)/8] & (1«((unit)%8))) !=
00092
00094 #define MB_UNITMAP_SET_BIT(unitmap, unit, value)
00095
          if (value)
00096
            ((uint8_t*)(unitmap))[(unit)/8] |= (1«((unit)%8));
00097
00098
             ((uint8_t*)(unitmap))[(unit)/8] &= (~(1«((unit)%8)));
00099
00100 //
00101 // ----- Modbus function codes
00102 //
00103
00107 #define MBF_READ_COILS
00108 #define MBF_READ_DISCRETE_INPUTS
00109 #define MBF_READ_HOLDING_REGISTERS
00110 #define MBF_READ_INPUT_REGISTERS
00111 #define MBF_WRITE_SINGLE_COIL
00112 #define MBF_WRITE_SINGLE_REGISTER
00113 #define MBF_READ_EXCEPTION_STATUS
00114 #define MBF_DIAGNOSTICS
00115 #define MBF_GET_COMM_EVENT_COUNTER 00116 #define MBF_GET_COMM_EVENT_LOG
00117 #define MBF_WRITE_MULTIPLE_COILS
00118 #define MBF_WRITE_MULTIPLE_REGISTERS
00119 #define MBF_REPORT_SERVER_ID
00120 #define MBF_READ_FILE_RECORD
                                                      20
00121 #define MBF_WRITE_FILE_RECORD
00122 #define MBF_MASK_WRITE_REGISTER
00123 #define MBF_READ_WRITE_MULTIPLE_REGISTERS
00124 #define MBF_READ_FIFO_QUEUE
00125 #define MBF_ENCAPSULATED_INTERFACE_TRANSPORT
00126 #define MBF_ILLEGAL_FUNCTION
00127 #define MBF_EXCEPTION
00129
00130
00131 //
00132 // ----- Modbus count constants
00133 //
00134
00136 #define MB_BYTE_SZ_BITES 8
00137
00139 #define MB_REGE_SZ_BITES 16
00140
00142 #define MB REGE SZ BYTES 2
00143
00145 #define MB_MAX_BYTES 255
00146
00148 #define MB_MAX_REGISTERS 127
00149
00151 #define MB_MAX_DISCRETS 2040
00152
00154 #define MB_VALUE_BUFF_SZ 255
00155
00158
00160 #define MB_RTU_IO_BUFF_SZ 264
00161
00163 #define MB ASC IO BUFF SZ 529
00164
00166 #define MB_TCP_IO_BUFF_SZ 268
00167
00169 #define GET_COMM_EVENT_LOG_MAX 64
00170
00172 #define READ FIFO OUEUE MAX 31
00173
00174 #ifdef __cplusplus
00175
00176 namespace Modbus {
00177
00178 #ifdef QT_CORE_LIB
00179 Q_NAMESPACE
```

8.13 ModbusGlobal.h

```
00180 #endif
00181
00182 #endif // __cplusplus
00183
00185 typedef void* Handle:
00186
00188 typedef char Char;
00189
00191 typedef uint32_t Timer;
00192
00194 typedef int64_t Timestamp;
00195
00197 enum Constants
00198 {
00199
         VALID_MODBUS_ADDRESS_BEGIN = 1
UU201 STANDARD_TCP_PORT
00202 };
         VALID_MODBUS_ADDRESS_END = 247,
                                    = 502
00204 //===== Modbus protocol types ========
00205
00207 typedef enum _MemoryType
00208 {
          Memory_Unknown = 0xFFFF,
00209
00210
         Memory_0x = 0,
         Memory_Coils = Memory_0x,
00211
00212
         Memory_1x = 1,
00213
        Memory_DiscreteInputs = Memory_1x,
       Memory_InputRegisters = Memory_3x,
Memory_4x = 4,
00214
00215
00216
00217
         Memory_HoldingRegisters = Memory_4x,
00218 } MemoryType;
00219
00221 #ifdef __cplusplus // Note: for Qt/moc support
00222 enum StatusCode
00223 #else
00224 typedef enum _StatusCode
00225 #endif
00226 {
00227
          Status_Processing
                                         = 0x800000000
         Status_Good
                                         = 0 \times 000000000.
00228
                                         = 0 \times 01000000.
00229
         Status Bad
00230
         Status_Uncertain
                                         = 0x02000000,
00231
00232
         //---- Modbus standart errors begin -----
00233
         // from 0 to 255
         Status_BadIllegalFunction
00234
                                                      = Status_Bad | 0x01,
         Status_BadIllegalDataAddress
00235
                                                      = Status Bad |
                                                                      0x02.
00236
          Status_BadIllegalDataValue
                                                      = Status Bad |
                                                                      0x03,
          Status_BadServerDeviceFailure
                                                      = Status_Bad |
00238
          Status_BadAcknowledge
                                                      = Status_Bad |
00239
          Status_BadServerDeviceBusy
                                                      = Status_Bad |
                                                                      0x06,
                                                                      0x07,
00240
          Status_BadNegativeAcknowledge
                                                      = Status Bad |
                                                     = Status_Bad |
00241
          Status_BadMemoryParityError
                                                                      0x08.
00242
          Status_BadGatewayPathUnavailable
                                                       = Status Bad |
                                                                      0x0A,
00243
          Status_BadGatewayTargetDeviceFailedToRespond = Status_Bad | 0x0B,
00244
          //---- Modbus standart errors end ---
00245
00246
          //---- Modbus common errors begin ----
          Status_BadEmptyResponse
                                     = Status_Bad | 0x101,
00247
00248
          Status BadNotCorrectRequest
00249
          Status_BadNotCorrectResponse
00250
          Status_BadWriteBufferOverflow
00251
          Status_BadReadBufferOverflow
00252
00253
          //---- Modbus common errors end -----
00254
00255
          //--_ Modbus serial specified errors begin --
          Status_BadSerialOpen = Status_Bad | 0x201,
00257
          Status_BadSerialWrite
00258
          Status_BadSerialRead
00259
          Status_BadSerialReadTimeout
00260
          Status BadSerialWriteTimeout
          //--- Modbus serial specified errors end ---
00261
00262
00263
          //---- Modbus ASC specified errors begin ----
00264
          Status_BadAscMissColon = Status_Bad | 0x301,
00265
          Status_BadAscMissCrLf
          Status_BadAscChar
00266
00267
          Status BadLrc
00268
          //--- Modbus ASC specified errors end ----
00269
00270
          //--- Modbus RTU specified errors begin ----
00271
         Status_BadCrc
                                         = Status_Bad | 0x401,
          //---- Modbus RTU specified errors end ---
00272
00273
```

```
//-- Modbus TCP specified errors begin --
00275
          Status_BadTcpCreate
                                           = Status_Bad | 0x501,
00276
          Status_BadTcpConnect,
00277
          Status_BadTcpWrite,
          Status_BadTcpRead,
00278
00279
          Status_BadTcpBind,
          Status_BadTcpListen,
00281
          Status_BadTcpAccept,
00282
          Status_BadTcpDisconnect,
          //--- Modbus TCP specified errors end ---
00283
00284 }
00285 #ifdef __cplusplus
00286 ;
00287 #else
00288 StatusCode;
00289 #endif
00290
00292 #ifdef __cplusplus // Note: for Qt/moc support 00293 enum ProtocolType
00294 #else
00295 typedef enum _ProtocolType
00296 #endif
00297 {
00298
          ASC,
00299
          RTU,
00300
          TCP
00301 }
00302 #ifdef __cplusplus
00303 ;
00304 #else
00305 ProtocolType;
00306 #endif
00307
00308
00310 #ifdef __cplusplus // Note: for Qt/moc support
00311 enum Parity
00312 #else
00313 typedef enum _Parity
00314 #endif
00315 {
00316
          NoParity
         EvenParity,
00317
00318
          OddParity
          SpaceParity,
00319
00320
         MarkParity
00321 }
00322 #ifdef __cplusplus
00323 ;
00324 #else
00325 Parity;
00326 #endif
00327
00328
00330 #ifdef __cplusplus // Note: for Qt/moc support
00331 enum StopBits
00332 #else
00333 typedef enum _StopBits
00334 #endif
00335 {
00336
          OneStop
00337
          OneAndHalfStop,
00338
          TwoStop
00339 }
00340 #ifdef __cplusplus
00341 ;
00342 #else
00343 StopBits;
00344 #endif
00345
00347 #ifdef __cplusplus // Note: for Qt/moc support 00348 enum FlowControl
00349 #else
00350 typedef enum _FlowControl
00351 #endif
00352 {
00353 NoFlowControl
00354
          HardwareControl,
00355
          SoftwareControl
00356 }
00357 #ifdef __cplusplus
00358 ;
00359 #else
00360 FlowControl;
00361 #endif
00362
00363 #ifdef QT_CORE_LIB
00364 O_ENUM_NS(StatusCode)
```

8.13 ModbusGlobal.h

```
00365 Q_ENUM_NS(ProtocolType)
00366 Q_ENUM_NS(Parity)
00367 Q_ENUM_NS(StopBits)
00368 Q_ENUM_NS(FlowControl)
00369 #endif
00370
00372 typedef struct
00373 {
00374
          const Char *portName
          int32_t
00375
                     baudRate
00376
          int8 t
                      dataBits
00377
         Parity
                      parity
00378
          StopBits
                      stopBits
00379
          FlowControl flowControl
00380
         uint32_t timeoutFirstByte;
00381
         uint32_t
                     timeoutInterByte;
00382 } SerialSettings;
00383
00385 typedef struct
00386 {
00387
          const Char *host ;
00388
         uint16_t
                     port
00389
         uint16 t
                      timeout;
00390 } TcpSettings;
00391
00392 #ifdef __cplusplus
00393 extern "C" {
00394 #endif
00395
00397 inline bool StatusIsProcessing(StatusCode status) { return status == Status_Processing; }
00398
00400 inline bool StatusIsGood(StatusCode status) { return (status & 0xFF000000) == Status_Good; }
00401
00403 inline bool StatusIsBad(StatusCode status) { return (status & Status_Bad) != 0; }
00404
00406 inline bool StatusIsUncertain(StatusCode status) { return (status & Status_Uncertain) != 0; }
00407
00409 inline bool StatusIsStandardError(StatusCode status) { return (status & Status_Bad) && ((status &
      0xFF00) == 0); }
00410
00412 inline bool getBit(const void *bitBuff, uint16_t bitNum) { return GET_BIT (bitBuff, bitNum); }
00413
00415 inline bool getBitS(const void *bitBuff, uint16_t bitNum, uint16_t maxBitCount) { return (bitNum <
      maxBitCount) ? getBit(bitBuff, bitNum) : false; }
00416
00418 inline void setBit(void *bitBuff, uint16_t bitNum, bool value) { SET_BIT (bitBuff, bitNum, value) }
00419
00421 inline void setBitS(void *bitBuff, uint16_t bitNum, bool value, uint16_t maxBitCount) { if (bitNum <
      maxBitCount) setBit(bitBuff, bitNum, value); }
00422
00426 inline bool *getBits(const void *bitBuff, uint16_t bitNum, uint16_t bitCount, bool *boolBuff) {
      GET_BITS(bitBuff, bitNum, bitCount, boolBuff) return boolBuff; }
00427
00430 inline bool *getBitsS(const void *bitBuff, uint16_t bitNum, uint16_t bitCount, bool *boolBuff,
      uint16_t maxBitCount) { if ((bitNum+bitCount) <= maxBitCount) getBits(bitBuff, bitNum, bitCount,
      boolBuff); return boolBuff; }
00431
00435 inline void *setBits(void *bitBuff, uint16_t bitNum, uint16_t bitCount, const bool *boolBuff) {
      SET_BITS(bitBuff, bitNum, bitCount, boolBuff) return bitBuff; }
00436
00439 inline void *setBitsS(void *bitBuff, uint16_t bitNum, uint16_t bitCount, const bool *boolBuff,
      uint16 t maxBitCount) { if ((bitNum + bitCount) <= maxBitCount) setBits(bitBuff, bitNum, bitCount,
      boolBuff); return bitBuff; }
00440
00442 MODBUS_EXPORT uint32_t modbusLibVersion();
00443
00445 MODBUS EXPORT const Char* modbusLibVersionStr();
00446
00448 inline uint16 t toModbusOffset(uint32 t adr) { return (uint16 t) (adr - 1); }
00452 MODBUS_EXPORT uint16_t crc16(const uint8_t *byteArr, uint32_t count);
00453
00456 MODBUS_EXPORT uint8_t lrc(const uint8_t *byteArr, uint32_t count);
00457
00466 MODBUS EXPORT StatusCode readMemRegs(uint32 t offset, uint32 t count, void *values, const void
      *memBuff, uint32_t memRegCount, uint32_t *outCount);
00467
00476 MODBUS_EXPORT StatusCode writeMemRegs (uint32_t offset, uint32_t count, const void *values, void
      *memBuff, uint32_t memRegCount, uint32_t *outCount);
00477
00486 MODBUS_EXPORT StatusCode readMemBits(uint32_t offset, uint32_t count, void *values, const void
      *memBuff, uint32_t memBitCount, uint32_t *outCount);
00496 MODBUS_EXPORT StatusCode writeMemBits(uint32_t offset, uint32_t count, const void *values, void
      *memBuff, uint32_t memBitCount, uint32_t *outCount);
00497
00505 MODBUS EXPORT uint32 t bytesToAscii(const uint8 t* bytesBuff, uint8 t* asciiBuff, uint32 t count);
```

```
00514 MODBUS_EXPORT uint32_t asciiToBytes(const uint8_t* asciiBuff, uint8_t* bytesBuff, uint32_t count);
00515
00517 MODBUS_EXPORT Char *sbytes(const uint8_t* buff, uint32_t count, Char *str, uint32_t strmaxlen);
00518
00520 MODBUS_EXPORT Char *sascii(const uint8_t* buff, uint32_t count, Char *str, uint32_t strmaxlen);
00521
00524 MODBUS_EXPORT const Char *sprotocolType(ProtocolType type);
00525
00528 MODBUS_EXPORT const Char *sparity(Parity parity);
00529
00532 MODBUS_EXPORT const Char *sstopBits(StopBits stopBits);
00533
00536 MODBUS_EXPORT const Char *sflowControl (FlowControl flowControl);
00537
00539 MODBUS_EXPORT Timer timer();
00540
00542 MODBUS EXPORT Timestamp currentTimestamp();
00545 MODBUS_EXPORT void msleep(uint32_t msec);
00546
00547 #ifdef
00548 } //extern "C"
00549 #endif
00550
00551 #ifdef __cplusplus
00552 } //namespace Modbus
00553 #endif
00554
00555 #endif // MODBUSGLOBAL_H
```

## 8.14 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusObject.h File Reference

The header file defines the class templates used to create signal/slot-like mechanism.

```
#include "Modbus.h"
```

## Classes

class ModbusSlotBase< ReturnType, Args >

ModbusSlotBase base template for slot (method or function)

class ModbusSlotMethod
 T, ReturnType, Args

ModbusSlotMethod template class hold pointer to object and its method

- class ModbusSlotFunction< ReturnType, Args >

ModbusSlotFunction template class hold pointer to slot function

· class ModbusObject

 $\textit{The } \underline{\textit{ModbusObject class}} \textit{ is the base class for objects that use signal/slot mechanism.}$ 

## **Typedefs**

template < class T , class ReturnType , class ... Args > using ModbusMethodPointer = ReturnType(T::\*)(Args...)

 ${\it Modbus Method Pointer-pointer\ to\ class\ method\ template\ type}$ 

template < class ReturnType , class ... Args > using ModbusFunctionPointer = ReturnType (\*)(Args...)

ModbusFunctionPointer pointer to function template type

8.15 ModbusObject.h

## 8.14.1 Detailed Description

The header file defines the class templates used to create signal/slot-like mechanism.

**Author** 

serhmarch

Date

May 2024

## 8.15 ModbusObject.h

#### Go to the documentation of this file.

```
00001
00008 #ifndef MODBUSOBJECT_H
00009 #define MODBUSOBJECT_H
00010
00011 #include "Modbus.h"
00012
00014 template <class T, class ReturnType, class ... Args>
00015 using ModbusMethodPointer = ReturnType(T::*)(Args...);
00018 template <class ReturnType, class ... Args>
00019 using ModbusFunctionPointer = ReturnType (*)(Args...);
00020
00022 template <class ReturnType, class ... Args>
00023 class ModbusSlotBase
00024 {
00025 public:
00027
         virtual ~ModbusSlotBase() {}
00028
00031
         virtual void *object() const { return nullptr; }
00032
00034
         virtual void *methodOrFunction() const = 0;
00035
00037
          virtual ReturnType exec(Args ... args) = 0;
00038 };
00039
00040
00041
00043 template <class T, class ReturnType, class ... Args>
00044 class ModbusSlotMethod : public ModbusSlotBase<ReturnType, Args ...>
00045 (
00046 public:
         ModbusSlotMethod(T* object, ModbusMethodPointer<T, ReturnType, Args...> methodPtr) :
00050
     m_object(object), m_methodPtr(methodPtr) {}
00051
00052 public:
00053
          void *object() const override { return m_object; }
00054
          void *methodOrFunction() const override { return reinterpret_cast<void*>(m_voidPtr); }
00055
00056
          ReturnType exec(Args ... args) override
         {
00058
              return (m_object->*m_methodPtr) (args...);
00059
00060
00061 private:
00062
          T* m object;
00063
         union
00064
         {
00065
              ModbusMethodPointer<T, ReturnType, Args...> m_methodPtr;
00066
              void *m_voidPtr;
00067
         };
00068 };
00069
00070
00072 template <class ReturnType, class ... Args>
00073 class ModbusSlotFunction : public ModbusSlotBase<ReturnType, Args ...>
00074 {
00075 public:
00078
         ModbusSlotFunction(ModbusFunctionPointer<ReturnType, Args...> funcPtr) : m_funcPtr(funcPtr) {}
```

```
00080 public:
          void *methodOrFunction() const override { return m_voidPtr; }
00081
00082
          ReturnType exec(Args ... args) override
00083
00084
              return m funcPtr(args...);
00085
          }
00086
00087 private:
00088
         union
00089
          {
00090
              ModbusFunctionPointer<ReturnType, Args...> m_funcPtr;
00091
              void *m_voidPtr;
00092
          };
00093 };
00094
00095 class ModbusObjectPrivate;
00096
00114 class MODBUS EXPORT ModbusObject
00115 {
00116 public:
00120
          static ModbusObject *sender();
00121
00122 public:
          ModbusObject();
00124
00125
00127
          virtual ~ModbusObject();
00128
00129 public:
00131
         const Modbus::Char *objectName() const;
00132
00134
          void setObjectName(const Modbus::Char *name);
00135
00136 public:
          template <class SignalClass, class T, class ReturnType, class ... Args>
00147
00148
          void connect(ModbusMethodPointer<SignalClass, ReturnType, Args ...> signalMethodPtr, T *object,
     ModbusMethodPointer<T, ReturnType, Args ... > objectMethodPtr)
00149
         {
00150
              ModbusSlotMethod<T, ReturnType, Args ...> *slotMethod = new ModbusSlotMethod<T, ReturnType,
     Args ...>(object, objectMethodPtr);
00151
             union {
00152
                  ModbusMethodPointer<SignalClass, ReturnType, Args ...> signalMethodPtr;
                  void* voidPtr;
00153
00154
              } converter:
00155
              converter.signalMethodPtr = signalMethodPtr;
00156
              setSlot(converter.voidPtr, slotMethod);
00157
          }
00158
          template <class SignalClass, class ReturnType, class ... Args>
void connect(ModbusMethodPointer<SignalClass, ReturnType, Args ...> signalMethodPtr,
00161
00162
     ModbusFunctionPointer<ReturnType, Args ...> funcPtr)
00163
          {
              ModbusSlotFunction<ReturnType, Args ...> *slotFunc = new ModbusSlotFunction<ReturnType, Args
00164
      ...>(funcPtr);
00165
              union {
                  ModbusMethodPointer<SignalClass, ReturnType, Args ...> signalMethodPtr;
00166
                  void* voidPtr;
00167
00168
              } converter;
00169
              converter.signalMethodPtr = signalMethodPtr;
00170
              setSlot(converter.voidPtr, slotFunc);
00171
          }
00172
00174
          template <class ReturnType, class ... Args>
00175
          inline void disconnect(ModbusFunctionPointer<ReturnType, Args ...> funcPtr)
00176
          {
00177
              disconnect(nullptr, funcPtr);
00178
          }
00179
00181
          inline void disconnectFunc(void *funcPtr)
00182
          {
00183
              disconnect(nullptr, funcPtr);
00184
00185
00187
          template <class T, class ReturnType, class ... Args>
          inline void disconnect(T *object, ModbusMethodPointer<T, ReturnType, Args ...> objectMethodPtr)
00188
00189
          {
00190
00191
                  ModbusMethodPointer<T, ReturnType, Args ...> objectMethodPtr;
00192
                  void* voidPtr;
00193
              } converter;
              converter.objectMethodPtr = objectMethodPtr;
00194
00195
              disconnect(object, converter.voidPtr);
00196
          }
00197
          template <class T>
00199
00200
          inline void disconnect(T *object)
00201
00202
              disconnect(object, nullptr);
```

```
00203
          }
00204
00205
00206 protected:
       template <class T, class ... Args>
00208
         void emitSignal(const char *thisMethodId, ModbusMethodPointer<T, void, Args ...> thisMethod, Args
00209
... args)
00211
              dummy = thisMethodId; // Note: present because of weird MSVC compiler optimization,
00212
                                    // when diff signals can have same address
             //printf("Emit signal: %s\n", thisMethodId);
00213
00214
             union {
             ModbusMethodPointer<T, void, Args ...> thisMethod;
00215
                  void* voidPtr;
00216
00217
             } converter;
00218
             converter.thisMethod = thisMethod;
00219
00220
             pushSender(this);
00221
             int i = 0;
00222
              while (void* itemSlot = slot(converter.voidPtr, i++))
00223
             {
00224
                  ModbusSlotBase<void, Args...> *slotBase = reinterpret_cast<ModbusSlotBase<void, Args...>
*>(itemSlot);
00225
                 slotBase->exec(args...);
00226
00227
             popSender();
00228
00229
00230 private:
00231
       void *slot(void *signalMethodPtr, int i) const;
00232
         void setSlot(void *signalMethodPtr, void *slotPtr);
00233
         void disconnect(void *object, void *methodOrFunc);
00234
00235 private:
00236
       static void pushSender(ModbusObject *sender);
00237
         static void popSender();
00238
00239 protected:
00241 static const char* dummy; // Note: prevent weird MSVC compiler optimization 00242 ModbusObjectPrivate *d_ptr;
00243
         ModbusObject(ModbusObjectPrivate *d);
00245 };
00246
00247 #endif // MODBUSOBJECT_H
```

## 8.16 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusPlatform.h File Reference

Definition of platform specific macros.

## 8.16.1 Detailed Description

Definition of platform specific macros.

Author

serhmarch

Date

May 2024

## 8.17 ModbusPlatform.h

#### Go to the documentation of this file.

```
00008 #ifndef MODBUSPLATFORM_H
00009 #define MODBUSPLATFORM_H
00010
00011 #if defined (_WIN32)|| defined(_WIN64)|| defined(__WIN32__) || defined(__WINDOWS__)
00012 #define MB_OS_WINDOWS
00013 #endif
00014
00015 // Linux, BSD and Solaris define "unix", OSX doesn't, even though it derives from BSD
00016 #if defined(unix) || defined(__unix__) || defined(__unix)
00017 #define MB_PLATFORM_UNIX
00018 #endif
00019
00020 #if BSD \ge 0
00021 #define MB_OS_BSD 00022 #endif
00023
00024 #if __APPLE_
00025 #define MB_OS_OSX
00026 #endif
00027
00028
00029 #ifdef _MSC_VER
00030
00031 #define MB_DECL_IMPORT __declspec (dllimport) 00032 #define MB_DECL_EXPORT __declspec (dllexport)
00033
00034 #else
00035
00036 #define MB_DECL_IMPORT
00037 #define MB_DECL_EXPORT
00038
00039 #endif
00040
00041 #endif // MODBUSPLATFORM_H
```

# 8.18 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusPort.h File Reference

Header file of abstract class ModbusPort.

```
#include <string>
#include <list>
#include "Modbus.h"
```

#### **Classes**

class ModbusPort

The abstract class ModbusPort is the base class for a specific implementation of the Modbus communication protocol.

## 8.18.1 Detailed Description

Header file of abstract class ModbusPort.

**Author** 

serhmarch

Date

May 2024

8.19 ModbusPort.h

#### 8.19 ModbusPort.h

```
00001
00008 #ifndef MODBUSPORT H
00009 #define MODBUSPORT_H
00010
00011 #include <string>
00012 #include <list>
00013
00014 #include "Modbus.h"
00015
00016 class ModbusPortPrivate;
00024 class MODBUS_EXPORT ModbusPort
00025 {
00026 public:
00028
          virtual ~ModbusPort();
00029
00030 public:
00032
          virtual Modbus::ProtocolType type() const = 0;
00033
00035
          virtual Modbus::Handle handle() const = 0;
00036
00038
          virtual Modbus::StatusCode open() = 0;
00039
00041
          virtual Modbus::StatusCode close() = 0;
00042
00044
          virtual bool isOpen() const = 0;
00045
00048
          virtual void setNextRequestRepeated(bool v);
00049
00050 public:
00052
          bool isChanged() const;
00053
00055
          bool isServerMode() const;
00056
00058
          virtual void setServerMode(bool mode);
00059
00061
          bool isBlocking() const;
00062
00064
          bool isNonBlocking() const;
00065
00067
          uint32_t timeout() const;
00068
00070
          void setTimeout(uint32_t timeout);
00071
00072 public: // errors
00074
          Modbus::StatusCode lastErrorStatus() const;
00075
00077
          const Modbus::Char *lastErrorText() const;
00078
00079 public:
00081
         virtual Modbus::StatusCode writeBuffer(uint8_t unit, uint8_t func, uint8_t *buff, uint16_t
      szInBuff) = 0;
00082
00084
          virtual Modbus::StatusCode readBuffer(uint8_t &unit, uint8_t &func, uint8_t *buff, uint16_t
      maxSzBuff, uint16_t *szOutBuff) = 0;
00085
00087
          virtual Modbus::StatusCode write() = 0;
00088
00090
          virtual Modbus::StatusCode read() = 0;
00091
00092 public: // buffer
00094
          virtual const uint8_t *readBufferData() const = 0;
00095
00097
          virtual uint16_t readBufferSize() const = 0;
00098
00100
          virtual const uint8_t *writeBufferData() const = 0;
00103
          virtual uint16_t writeBufferSize() const = 0;
00104
00105 protected:
00107
          Modbus::StatusCode setError(Modbus::StatusCode status, const Modbus::Char *text);
00108
00109 protected:
00111
          ModbusPortPrivate *d_ptr;
00112
          ModbusPort(ModbusPortPrivate *d);
00114 };
00115
00116 #endif // MODBUSPORT_H
```

### 8.20 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusQt.h File Reference

#### Qt support file for ModbusLib.

```
#include "Modbus.h"
#include <QMetaEnum>
#include <QHash>
#include <QVariant>
```

#### Classes

class Modbus::Strings

Sets constant key values for the map of settings.

class Modbus::Defaults

Holds the default values of the settings.

class Modbus::Address

Class for convinient manipulation with Modbus Data Address.

#### **Namespaces**

namespace Modbus

Main Modbus namespace. Contains classes, functions and constants to work with Modbus-protocol.

#### **Typedefs**

typedef QHash< QString, QVariant > Modbus::Settings

 $\textit{Map for settings of $Modbus$ protocol where key has type $\it QString$ and value is $\it QVariant.$}$ 

#### **Functions**

- MODBUS\_EXPORT uint8\_t Modbus::getSettingUnit (const Settings &s, bool \*ok=nullptr)
- MODBUS\_EXPORT ProtocolType Modbus::getSettingType (const Settings &s, bool \*ok=nullptr)
- MODBUS\_EXPORT uint32\_t Modbus::getSettingTries (const Settings &s, bool \*ok=nullptr)
- MODBUS EXPORT QString Modbus::getSettingHost (const Settings &s, bool \*ok=nullptr)
- MODBUS\_EXPORT uint16\_t Modbus::getSettingPort (const Settings &s, bool \*ok=nullptr)
- MODBUS EXPORT uint32 t Modbus::getSettingTimeout (const Settings &s, bool \*ok=nullptr)
- MODBUS\_EXPORT QString Modbus::getSettingSerialPortName (const Settings &s, bool \*ok=nullptr)
- MODBUS\_EXPORT int32\_t Modbus::getSettingBaudRate (const Settings &s, bool \*ok=nullptr)
- MODBUS\_EXPORT int8\_t Modbus::getSettingDataBits (const Settings &s, bool \*ok=nullptr)
- MODBUS EXPORT Parity Modbus::getSettingParity (const Settings &s, bool \*ok=nullptr)
- MODBUS\_EXPORT StopBits Modbus::getSettingStopBits (const Settings &s, bool \*ok=nullptr)
- MODBUS\_EXPORT FlowControl Modbus::getSettingFlowControl (const Settings &s, bool \*ok=nullptr)
- MODBUS\_EXPORT uint32\_t Modbus::getSettingTimeoutFirstByte (const Settings &s, bool \*ok=nullptr)
- MODBUS\_EXPORT uint32\_t Modbus::getSettingTimeoutInterByte (const Settings &s, bool \*ok=nullptr)
- MODBUS EXPORT bool Modbus::getSettingBroadcastEnabled (const Settings &s, bool \*ok=nullptr)
- MODBUS\_EXPORT void Modbus::setSettingUnit (Settings &s, uint8\_t v)
- MODBUS EXPORT void Modbus::setSettingType (Settings &s, ProtocolType v)
- MODBUS\_EXPORT void Modbus::setSettingTries (Settings &s, uint32\_t)

- MODBUS\_EXPORT void Modbus::setSettingHost (Settings &s, const QString &v)
- MODBUS\_EXPORT void Modbus::setSettingPort (Settings &s, uint16\_t v)
- MODBUS EXPORT void Modbus::setSettingTimeout (Settings &s, uint32 t v)
- MODBUS EXPORT void Modbus::setSettingSerialPortName (Settings &s, const QString &v)
- MODBUS\_EXPORT void Modbus::setSettingBaudRate (Settings &s, int32\_t v)
- MODBUS\_EXPORT void Modbus::setSettingDataBits (Settings &s, int8\_t v)
- MODBUS\_EXPORT void Modbus::setSettingParity (Settings &s, Parity v)
- MODBUS EXPORT void Modbus::setSettingStopBits (Settings &s, StopBits v)
- MODBUS\_EXPORT void Modbus::setSettingFlowControl (Settings &s, FlowControl v)
- MODBUS EXPORT void Modbus::setSettingTimeoutFirstByte (Settings &s, uint32 t v)
- MODBUS EXPORT void Modbus::setSettingTimeoutInterByte (Settings &s, uint32 t v)
- MODBUS\_EXPORT void Modbus::setSettingBroadcastEnabled (Settings &s, bool v)
- Address Modbus::addressFromString (const QString &s)
- template < class EnumType >

QString Modbus::enumKey (int value)

template < class EnumType >

QString Modbus::enumKey (EnumType value, const QString &byDef=QString())

template < class EnumType >

EnumType Modbus::enumValue (const QString &key, bool \*ok=nullptr, EnumType defaultValue=static\_cast< EnumType >(-1))

template < class EnumType >

EnumType Modbus::enumValue (const QVariant &value, bool \*ok=nullptr, EnumType defaultValue=static\_cast< EnumType >(-1))

template < class EnumType >

EnumType Modbus::enumValue (const QVariant &value, EnumType defaultValue)

template < class EnumType >

EnumType Modbus::enumValue (const QVariant &value)

- MODBUS\_EXPORT ProtocolType Modbus::toProtocolType (const QString &s, bool \*ok=nullptr)
- MODBUS\_EXPORT ProtocolType Modbus::toProtocolType (const QVariant &v, bool \*ok=nullptr)
- MODBUS EXPORT int32 t Modbus::toBaudRate (const QString &s, bool \*ok=nullptr)
- MODBUS EXPORT int32 t Modbus::toBaudRate (const QVariant &v, bool \*ok=nullptr)
- MODBUS EXPORT int8 t Modbus::toDataBits (const QString &s, bool \*ok=nullptr)
- MODBUS\_EXPORT int8\_t Modbus::toDataBits (const QVariant &v, bool \*ok=nullptr)
- MODBUS EXPORT Parity Modbus::toParity (const QString &s, bool \*ok=nullptr)
- MODBUS EXPORT Parity Modbus::toParity (const QVariant &v, bool \*ok=nullptr)
- MODBUS\_EXPORT StopBits Modbus::toStopBits (const QString &s, bool \*ok=nullptr)
- MODBUS\_EXPORT StopBits Modbus::toStopBits (const QVariant &v, bool \*ok=nullptr)
- MODBUS\_EXPORT FlowControl Modbus::toFlowControl (const QString &s, bool \*ok=nullptr)
- MODBUS\_EXPORT FlowControl Modbus::toFlowControl (const QVariant &v, bool \*ok=nullptr)
- MODBUS\_EXPORT QString Modbus::toString (StatusCode v)
- MODBUS\_EXPORT QString Modbus::toString (ProtocolType v)
- MODBUS\_EXPORT QString Modbus::toString (Parity v)
- MODBUS\_EXPORT QString Modbus::toString (StopBits v)
- MODBUS\_EXPORT QString Modbus::toString (FlowControl v)
- QString Modbus::bytesToString (const QByteArray &v)
- QString Modbus::asciiToString (const QByteArray &v)
- MODBUS EXPORT QStringList Modbus::availableSerialPortList ()
- MODBUS\_EXPORT ModbusPort \* Modbus::createPort (const Settings &settings, bool blocking=false)
- MODBUS\_EXPORT ModbusClientPort \* Modbus::createClientPort (const Settings &settings, bool blocking=false)
- MODBUS\_EXPORT ModbusServerPort \* Modbus::createServerPort (ModbusInterface \*device, const Settings &settings, bool blocking=false)

### 8.20.1 Detailed Description

Qt support file for ModbusLib.

**Author** 

serhmarch

Date

May 2024

#### 8.21 ModbusQt.h

```
00001
00008 #ifndef MODBUSQT_H
00009 #define MODBUSQT_H
00010
00011 #include "Modbus.h"
00012
00013 #include <QMetaEnum>
00014 #include <QHash>
00015 #include <QVariant>
00016
00017 namespace Modbus {
00018
00020 typedef QHash<QString, QVariant> Settings;
00024 class MODBUS_EXPORT Strings
00025 {
00026 public:
00027 const OString unit
00028 const OSt
         const QString type
00029
         const OString tries
       const QString host
const QString port
00031
00032
         const QString timeout
00033
         const QString serialPortName
00034
         const QString baudRate
00035
         const QString dataBits
         const QString parity
00036
00037
          const QString stopBits
00038
          const QString flowControl
00039
          const QString timeoutFirstByte
00040
          const QString timeoutInterByte
00041
         const QString isBroadcastEnabled;
00042
00043
          const QString NoParity
00044
          const QString EvenParity
00045
          const QString OddParity
00046
          const QString SpaceParity
          const QString MarkParity
00047
00048
00049
          const QString OneStop
00050
          const QString OneAndHalfStop
00051
          const QString TwoStop
00052
00053
          const QString NoFlowControl
00054
          const QString HardwareControl
          const QString SoftwareControl
00055
00056
00058
          Strings();
00059
00061
          static const Strings &instance():
00062 };
00066 class MODBUS_EXPORT Defaults
00067 {
00068 public:
       const uint8_t
00069
                             unit
         const ProtocolType type
00070
         const uint32_t tries
00071
        const QString
                             host
```

8.21 ModbusQt.h 201

```
const uint16_t
00073
                             port
00074
          const uint32 t
                             timeout
          const QString
00075
                             serialPortName
00076
          const int32_t
                             baudRate
00077
         const int8 t
                             dataBits
00078
          const Parity
                             parity
00079
          const StopBits
                             stopBits
00080
          const FlowControl
                             flowControl
00081
          const uint32_t
                             timeoutFirstByte
00082
          const uint32 t
                             timeoutInterByte
00083
         const bool
                             isBroadcastEnabled:
00084
00086
         Defaults();
00087
00089
          static const Defaults &instance();
00090 };
00091
00094 MODBUS EXPORT uint8 t getSettingUnit (const Settings &s, bool *ok = nullptr);
00095
00098 MODBUS_EXPORT ProtocolType getSettingType(const Settings &s, bool *ok = nullptr);
00099
00102 MODBUS_EXPORT uint32_t getSettingTries(const Settings &s, bool \starok = nullptr);
00103
00106 MODBUS EXPORT OString getSettingHost(const Settings &s, bool *ok = nullptr);
00107
00110 MODBUS_EXPORT uint16_t getSettingPort(const Settings &s, bool *ok = nullptr);
00111
00114 MODBUS_EXPORT uint32_t getSettingTimeout(const Settings &s, bool *ok = nullptr);
00115
00118 MODBUS_EXPORT QString getSettingSerialPortName(const Settings &s, bool *ok = nullptr);
00119
00122 MODBUS_EXPORT int32_t getSettingBaudRate(const Settings &s, bool *ok = nullptr);
00123
00126 MODBUS_EXPORT int8_t getSettingDataBits(const Settings &s, bool *ok = nullptr);
00127
00130 MODBUS_EXPORT Parity getSettingParity(const Settings &s, bool *ok = nullptr);
00131
00134 MODBUS_EXPORT StopBits getSettingStopBits(const Settings &s, bool *ok = nullptr);
00135
00138 MODBUS_EXPORT FlowControl getSettingFlowControl(const Settings &s, bool *ok = nullptr);
00139
00142 MODBUS_EXPORT uint32_t getSettingTimeoutFirstByte(const Settings &s, bool *ok = nullptr);
00143
00146 MODBUS_EXPORT uint32_t getSettingTimeoutInterByte(const Settings &s, bool *ok = nullptr);
00147
00150 MODBUS_EXPORT bool getSettingBroadcastEnabled(const Settings &s, bool *ok = nullptr);
00151
00153 MODBUS_EXPORT void setSettingUnit(Settings &s, uint8_t v);
00154
00156 MODBUS EXPORT void setSettingType (Settings &s. ProtocolType v):
00157
00159 MODBUS_EXPORT void setSettingTries(Settings &s, uint32_t);
00160
00162 MODBUS_EXPORT void setSettingHost(Settings &s, const QString &v);
00163
00165 MODBUS EXPORT void setSettingPort(Settings &s, uint16 t v);
00166
00168 MODBUS_EXPORT void setSettingTimeout(Settings &s, uint32_t v);
00169
00171 MODBUS_EXPORT void setSettingSerialPortName (Settings &s, const QString&v);
00172
00174 MODBUS EXPORT void setSettingBaudRate (Settings &s, int32 t v);
00175
00177 MODBUS_EXPORT void setSettingDataBits(Settings &s, int8_t v);
00178
00180 MODBUS_EXPORT void setSettingParity(Settings &s, Parity v);
00181
00183 MODBUS EXPORT void setSettingStopBits(Settings &s. StopBits v):
00184
00186 MODBUS_EXPORT void setSettingFlowControl(Settings &s, FlowControl v);
00187
00189 MODBUS_EXPORT void setSettingTimeoutFirstByte(Settings &s, uint32_t v);
00190
00192 MODBUS_EXPORT void setSettingTimeoutInterByte(Settings &s, uint32_t v);
00193
00195 MODBUS EXPORT void setSettingBroadcastEnabled(Settings &s. bool v):
00196
00197
00200 class MODBUS EXPORT Address
00201 {
00202 public:
00204
          Address();
00205
00207
          Address (Modbus::MemoryType, quint16 offset);
00208
00211
          Address (quint 32 adr);
00212
```

```
00213 public:
         inline bool isValid() const { return m_type != Memory_Unknown; }
00215
00216
00218
         inline MemoryType type() const { return static_cast<MemoryType>(m_type); }
00219
00221
         inline quint16 offset() const { return m offset; }
00222
00224
         inline quint32 number() const { return m_offset+1; }
00225
00228
         OString toString() const;
00229
         inline operator quint32 () const { return number() + (m_type*100000); }
00232
00233
00235
         Address & operator = (quint32 v);
00236
00237 private:
         quint16 m_type;
00238
00239
         quint16 m_offset;
00240 };
00241
00243 inline Address addressFromString(const QString &s) { return Address(s.toUInt()); }
00244
00246 template <class EnumType>
00247 inline QString enumKey(int value)
00248 {
00249
         const QMetaEnum me = QMetaEnum::fromType<EnumType>();
00250
         return QString(me.valueToKey(value));
00251 }
00252
00254 template <class EnumType>
00255 inline QString enumKey(EnumType value, const QString &byDef = QString())
00256 {
00257
          const QMetaEnum me = QMetaEnum::fromType<EnumType>();
00258
         const char *key = me.valueToKey(value);
00259
         if (key)
00260
             return QString(me.valueToKey(value));
00261
         else
00262
             return byDef;
00263 }
00264
00266 template <class EnumType>
00267 inline EnumType enumValue(const QString& key, bool* ok = nullptr, EnumType defaultValue =
     static cast < EnumType > (-1))
00268 {
00269
         bool okInner;
00270
         const QMetaEnum me = QMetaEnum::fromType<EnumType>();
00271
         00272
         if (ok)
00273
             *ok = okInner:
         if (okInner)
00274
00275
             return v;
00276
         return defaultValue;
00277 }
00278
00282 template <class EnumType>
00283 inline EnumType enumValue(const QVariant& value, bool *ok = nullptr, EnumType defaultValue =
     static_cast<EnumType>(-1))
00284 {
00285
         bool okInner;
00286
         int v = value.toInt(&okInner);
         if (okInner)
00287
00288
         {
00289
             const QMetaEnum me = QMetaEnum::fromType<EnumType>();
00290
              if (me.valueToKey(v)) // check value exists
00291
00292
                 if (ok)
                     *ok = true;
00293
00294
                 return static_cast<EnumType>(v);
00295
00296
             if (ok)
00297
                  *ok = false;
00298
             return defaultValue;
00299
00300
         return enumValue<EnumType>(value.toString(), ok, defaultValue);
00301 }
00302
00305 template <class EnumType>
00306 inline EnumType enumValue(const QVariant& value, EnumType defaultValue)
00307 {
00308
         return enumValue<EnumType>(value, nullptr, defaultValue);
00309 }
00310
00312 template <class EnumType>
00313 inline EnumType enumValue(const QVariant& value)
00314 {
00315
          return enumValue<EnumType>(value, nullptr);
00316 }
```

```
00317
00320 MODBUS_EXPORT ProtocolType toProtocolType (const QString &s, bool *ok = nullptr);
00321
00324 MODBUS_EXPORT ProtocolType toProtocolType (const QVariant &v, bool *ok = nullptr);
00325
00328 MODBUS_EXPORT int32_t toBaudRate(const QString &s, bool *ok = nullptr);
00332 MODBUS_EXPORT int32_t toBaudRate(const QVariant &v, bool *ok = nullptr);
00333
00336 MODBUS EXPORT int8 t toDataBits (const OString &s, bool *ok = nullptr);
00337
00340 MODBUS EXPORT int8 t toDataBits(const OVariant &v. bool *ok = nullptr);
00341
00344 MODBUS_EXPORT Parity toParity(const QString &s, bool *ok = nullptr);
00345
00348 MODBUS_EXPORT Parity toParity (const QVariant &v, bool *ok = nullptr);
00349
00352 MODBUS EXPORT StopBits toStopBits (const OString &s, bool *ok = nullptr);
00356 MODBUS_EXPORT StopBits toStopBits (const QVariant &v, bool *ok = nullptr);
00357
00360 MODBUS_EXPORT FlowControl toFlowControl(const QString &s, bool *ok = nullptr);
00361
00364 MODBUS EXPORT FlowControl toFlowControl(const OVariant &v, bool *ok = nullptr);
00365
00367 MODBUS_EXPORT QString toString(StatusCode v);
00368
00370 MODBUS_EXPORT QString toString(ProtocolType v);
00371
00373 MODBUS_EXPORT QString toString(Parity v);
00374
00376 MODBUS_EXPORT QString toString(StopBits v);
00377
00379 MODBUS_EXPORT QString toString(FlowControl v);
00380
00382 inline QString bytesToString(const QByteArray &v) { return bytesToString(reinterpret_cast<const
     uint8_t*>(v.constData()), v.size()).data(); }
00383
00385 inline QString asciiToString(const QByteArray &v) { return asciiToString(reinterpret_cast<const
     uint8_t*>(v.constData()), v.size()).data();
00386
00388 MODBUS_EXPORT QStringList availableSerialPortList();
00389
00392 MODBUS_EXPORT ModbusPort *createPort(const Settings &settings, bool blocking = false);
00393
00396 MODBUS_EXPORT ModbusClientPort *createClientPort (const Settings &settings, bool blocking = false);
00397
00400 MODBUS_EXPORT ModbusServerPort *createServerPort (ModbusInterface *device, const Settings,
     bool blocking = false);
00401
00402 } // namespace Modbus
00403
00404 #endif // MODBUSQT_H
```

### 8.22 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusRtuPort.h File Reference

Contains definition of RTU serial port class.

```
#include "ModbusSerialPort.h"
```

#### Classes

· class ModbusRtuPort

Implements RTU version of the Modbus communication protocol.

#### 8.22.1 Detailed Description

Contains definition of RTU serial port class.

**Author** 

serhmarch

Date

May 2024

#### 8.23 ModbusRtuPort.h

#### Go to the documentation of this file.

```
00008 #ifndef MODBUSRTUPORT_H
00009 #define MODBUSRTUPORT_H
00010
00011 #include "ModbusSerialPort.h"
00012
00019 class MODBUS_EXPORT ModbusRtuPort : public ModbusSerialPort
00020 {
00021 public:
00023
         ModbusRtuPort(bool blocking = false);
00024
00026
          ~ModbusRtuPort();
00027
00028 public:
00030
         Modbus::ProtocolType type() const override { return Modbus::RTU; }
00032 protected:
00033
         Modbus::StatusCode writeBuffer(uint8_t unit, uint8_t func, uint8_t *buff, uint16_t szInBuff)
00034
          Modbus::StatusCode readBuffer(uint8_t &unit, uint8_t &func, uint8_t *buff, uint16_t maxSzBuff,
     uint16_t *szOutBuff) override;
00035
00036 protected:
00037
          using ModbusSerialPort::ModbusSerialPort;
00038 };
00039
00040 #endif // MODBUSRTUPORT_H
```

## 8.24 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusSerialPort.h File Reference

Contains definition of base serial port class.

```
#include "ModbusPort.h"
```

#### Classes

· class ModbusSerialPort

The abstract class ModbusSerialPort is the base class serial port Modbus communications.

struct ModbusSerialPort::Defaults

Holds the default values of the settings.

8.25 ModbusSerialPort.h 205

#### 8.24.1 Detailed Description

Contains definition of base serial port class.

**Author** 

serhmarch

Date

May 2024

#### 8.25 ModbusSerialPort.h

```
00001
00008 #ifndef MODBUSSERIALPORT_H
00009 #define MODBUSSERIALPORT_H
00010
00011 #include "ModbusPort.h"
00012
00020 class MODBUS EXPORT ModbusSerialPort : public ModbusPort
00021 {
00022 public:
00025
       struct MODBUS_EXPORT Defaults
00026
00027
              const Modbus::Char
                                       *portName
             const int32_t
00028
                                       baudRate
00029
             const int8_t
                                        dataBits
00030
             const Modbus::Parity
                                      parity
00031
             const Modbus::StopBits
                                        stopBits
00032
              const Modbus::FlowControl flowControl
00033
              const uint32_t
                                        timeoutFirstByte;
00034
             const uint32_t
                                        timeoutInterByte;
00035
00037
             Defaults();
00038
00040
              static const Defaults &instance();
00041
         };
00042
00043 public:
00045
         ~ModbusSerialPort();
00046
00047 public:
00049
          Modbus::Handle handle() const override;
00050
00052
          Modbus::StatusCode open() override;
00053
00055
          Modbus::StatusCode close() override;
00056
00058
         bool isOpen() const override;
00059
00060 public: // settings
00062
         const Modbus::Char *portName() const;
00063
00065
          void setPortName(const Modbus::Char *portName);
00066
00068
          int32_t baudRate() const;
00069
00071
          void setBaudRate(int32 t baudRate);
00072
00074
          int8_t dataBits() const;
00075
00077
          void setDataBits(int8_t dataBits);
00078
00080
          Modbus::Parity parity() const;
00081
00083
          void setParity(Modbus::Parity parity);
00084
00086
          Modbus::StopBits stopBits() const;
00087
00089
          void setStopBits(Modbus::StopBits stopBits);
00090
00092
          Modbus::FlowControl flowControl() const;
```

```
00093
00095
          void setFlowControl (Modbus::FlowControl flowControl);
00096
00098
          inline uint32_t timeoutFirstByte() const { return timeout(); }
00099
00101
          inline void setTimeoutFirstByte(uint32_t timeout) { setTimeout(timeout); }
00102
00104
          uint32_t timeoutInterByte() const;
00105
00107
          void setTimeoutInterByte(uint32_t timeout);
00108
00109 public:
00110
          const uint8_t *readBufferData() const override;
00111
          uint16_t readBufferSize() const override;
00112
          const uint8_t *writeBufferData() const override;
00113
          uint16_t writeBufferSize() const override;
00114
00115 protected:
00116
         Modbus::StatusCode write() override;
00117
         Modbus::StatusCode read() override;
00118
00119 protected:
00121
         using ModbusPort::ModbusPort;
00123 };
00124
00125 #endif // MODBUSSERIALPORT_H
```

#### 8.26 ModbusServerPort.h

```
00001
00008 #ifndef MODBUSSERVERPORT_H
00009 #define MODBUSSERVERPORT_H
00011 #include "ModbusObject.h"
00012
00021 class MODBUS_EXPORT ModbusServerPort : public ModbusObject
00022 {
00023 public:
00026
         ModbusInterface *device() const;
00027
00030
          void setDevice(ModbusInterface *device);
00031
00032 public: // server port interface
00034
         virtual Modbus::ProtocolType type() const = 0;
00035
00037
          virtual bool isTcpServer() const;
00038
00041
          virtual Modbus::StatusCode open() = 0;
00042
00044
          virtual Modbus::StatusCode close() = 0;
00045
00047
          virtual bool isOpen() const = 0;
00048
00051
          bool isBroadcastEnabled() const;
00052
00055
          virtual void setBroadcastEnabled(bool enable);
00056
00064
          const void *unitMap() const;
00065
00068
          virtual void setUnitMap(const void *unitmap);
00069
00071
          void *context() const;
00072
00074
          void setContext(void *context);
00075
00078
          virtual Modbus::StatusCode process() = 0;
00079
00080 public:
00082
          bool isStateClosed() const;
00083
00084 public: // SIGNALS
00086
          void signalOpened(const Modbus::Char *source);
00087
00089
          void signalClosed(const Modbus::Char *source);
00090
00093
          void signalTx(const Modbus::Char *source, const uint8_t* buff, uint16_t size);
00094
00097
          void signalRx(const Modbus::Char *source, const uint8_t* buff, uint16_t size);
00098
00100
          void signalError(const Modbus::Char *source, Modbus::StatusCode status, const Modbus::Char *text);
00101
00102 protected:
00103
          using ModbusObject::ModbusObject;
```

```
00104 };
00105
00106 #endif // MODBUSSERVERPORT_H
00107
```

# 8.27 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusServer⊸ Resource.h File Reference

The header file defines the class that controls specific port.

```
#include "ModbusServerPort.h"
```

#### **Classes**

· class ModbusServerResource

Implements direct control for ModbusPort derived classes (TCP or serial) for server side.

#### 8.27.1 Detailed Description

The header file defines the class that controls specific port.

**Author** 

serhmarch

Date

May 2024

### 8.28 ModbusServerResource.h

```
00008 #ifndef MODBUSSERVERRESOURCE_H
00009 #define MODBUSSERVERRESOURCE_H
00010
00011 #include "ModbusServerPort.h"
00013 class ModbusPort;
00014
00024 class MODBUS_EXPORT ModbusServerResource : public ModbusServerPort
00025 {
00026 public:
00030
         ModbusServerResource(ModbusPort *port, ModbusInterface *device);
00031
00032 public:
00034
         ModbusPort *port() const;
00035
00036 public: // server port interface
00038
         Modbus::ProtocolType type() const override;
00039
00040
         Modbus::StatusCode open() override;
00041
00042
         Modbus::StatusCode close() override;
00043
00044
         bool isOpen() const override;
00045
```

```
Modbus::StatusCode process() override;
00048 protected:
        virtual Modbus::StatusCode processInputData(const uint8_t *buff, uint16_t sz);
00050
00051
00053
         virtual Modbus::StatusCode processDevice();
00056
         virtual Modbus::StatusCode processOutputData(uint8_t *buff, uint16_t &sz);
00057
00058 protected:
         using ModbusServerPort::ModbusServerPort;
00059
00060 };
00061
00062 #endif // MODBUSSERVERRESOURCE_H
```

## 8.29 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusTcpPort.h File Reference

```
Header file of class ModbusTcpPort.
```

```
#include "ModbusPort.h"
```

#### **Classes**

class ModbusTcpPort

Class Modbus TcpPort implements TCP version of Modbus protocol.

struct ModbusTcpPort::Defaults

 ${\it Defaults} \ {\it class} \ {\it constain} \ {\it default} \ {\it settings} \ {\it values} \ {\it for} \ {\it ModbusTcpPort}.$ 

#### 8.29.1 Detailed Description

Header file of class ModbusTcpPort.

**Author** 

serhmarch

Date

April 2024

### 8.30 ModbusTcpPort.h

```
00027
         {
00028
              const Modbus::Char *host
              const uint16_t
00029
00030
             const uint32_t
                                 timeout;
00031
00033
             Defaults();
00036
              static const Defaults &instance();
00037
00038
00039 public:
00041
         ModbusTcpPort(ModbusTcpSocket *socket, bool blocking = false);
00042
00044
          ModbusTcpPort(bool blocking = false);
00045
00047
          ~ModbusTcpPort();
00048
00049 public:
00051
         Modbus::ProtocolType type() const override { return Modbus::TCP; }
00052
00054
          Modbus::Handle handle() const override;
00055
00056
         Modbus::StatusCode open() override;
00057
          Modbus::StatusCode close() override;
00058
         bool isOpen() const override;
00059
00060 public:
00062
         const Modbus::Char *host() const;
00063
00065
         void setHost(const Modbus::Char *host);
00066
00068
         uint16_t port() const;
00069
00071
         void setPort(uint16_t port);
00072
00074
         void setNextRequestRepeated(bool v) override;
00075
         bool autoIncrement() const;
00078
00079 public:
00080
         const uint8_t *readBufferData() const override;
         uint16_t readBufferSize() const override;
00081
00082
         const_uint8 t *writeBufferData() const_override:
00083
         uint16_t writeBufferSize() const override;
00084
00085 protected:
00086 Modbus::StatusCode write() override;
00087
         Modbus::StatusCode read() override;
         Modbus::StatusCode writeBuffer(uint8_t unit, uint8_t func, uint8_t *buff, uint16_t szInBuff)
00088
     override:
00089
         Modbus::StatusCode readBuffer(uint8_t &unit, uint8_t &func, uint8_t *buff, uint16_t maxSzBuff,
     uint16_t *szOutBuff) override;
00090
00091 protected:
00092
         using ModbusPort::ModbusPort;
00093 };
00094
00095 #endif // MODBUSTCPPORT_H
```

## 8.31 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusTcpServer.h File Reference

```
Header file of Modbus TCP server.
```

```
#include "ModbusServerPort.h"
```

#### Classes

• class ModbusTcpServer

The ModbusTcpServer class implements TCP server part of the Modbus protocol.

struct ModbusTcpServer::Defaults

Defaults class constain default settings values for ModbusTcpServer.

#### 8.31.1 Detailed Description

Header file of Modbus TCP server.

**Author** 

serhmarch

Date

May 2024

### 8.32 ModbusTcpServer.h

```
00008 #ifndef MODBUSSERVERTCP_H
00009 #define MODBUSSERVERTCP_H
00010
00011 #include "ModbusServerPort.h"
00012
00013 class ModbusTcpSocket;
00014
00021 class MODBUS_EXPORT ModbusTcpServer : public ModbusServerPort
00023 public:
00026
          struct MODBUS_EXPORT Defaults
00027
              const uint16_t port ;
const uint32_t timeout;
00028
00029
00030
00032
00033
00035
              static const Defaults &instance();
00036
          };
00037
00038 public:
00040
          ModbusTcpServer(ModbusInterface *device);
00041
00043
          ~ModbusTcpServer();
00044
00045 public:
00047
          uint16_t port() const;
00048
00050
          void setPort(uint16_t port);
00051
00053
          uint32_t timeout() const;
00054
00056
          void setTimeout(uint32_t timeout);
00057
00058 public:
00060
          Modbus::ProtocolType type() const override { return Modbus::TCP; }
00061
00063
          bool isTcpServer() const override { return true; }
00064
00071
          Modbus::StatusCode open() override;
00072
00076
          Modbus::StatusCode close() override;
00077
00079
          bool isOpen() const override;
08000
00083
          void setBroadcastEnabled(bool enable) override;
00084
00087
          void setUnitMap(const void *unitmap) override;
00088
00090
          Modbus::StatusCode process() override;
00091
00092 public:
00095
          virtual ModbusServerPort *createTcpPort(ModbusTcpSocket *socket);
00096
00099
          virtual void deleteTcpPort(ModbusServerPort *port);
00100
00101 public: // SIGNALS
00103
          void signalNewConnection(const Modbus::Char *source);
00104
```

```
00106     void signalCloseConnection(const Modbus::Char *source);
00107
00108 protected:
00110     ModbusTcpSocket *nextPendingConnection();
00111
00113     void clearConnections();
00114
00115 protected:
00116     using ModbusServerPort::ModbusServerPort;
00117 };
00118
00119 #endif // MODBUSSERVERTCP_H
```

### Index

_MemoryType	bytesToString
Modbus, 21	Modbus, 27
$\sim$ ModbusAscPort	
ModbusAscPort, 56	c:/Users/march/Dropbox/PRJ/ModbusLib/src/cModbus.h,
$\sim$ ModbusObject	139, 164
ModbusObject, 94	c:/Users/march/Dropbox/PRJ/ModbusLib/src/Modbus.h,
$\sim$ ModbusPort	170, 171
ModbusPort, 97	c:/Users/march/Dropbox/PRJ/ModbusLib/src/Modbus_config.h,
$\sim$ ModbusRtuPort	173
ModbusRtuPort, 103	c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusAscPort.h,
$\sim$ ModbusSerialPort	173, 174
ModbusSerialPort, 106	c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusClient.h,
$\sim$ ModbusSlotBase	174, 175
ModbusSlotBase< ReturnType, Args >, 120	c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusClientPort.h,
$\sim$ ModbusTcpPort	176, 177
ModbusTcpPort, 126	c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusGlobal.h,
$\sim$ ModbusTcpServer	180, 187
ModbusTcpServer, 131	c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusObject.h,
,	192, 193
Address	c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusPlatform.h,
Modbus::Address, 47, 48	195, 196
addressFromString	c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusPort.h,
Modbus, 25	196, 197
ASC	c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusQt.h,
Modbus, 22	198, 200
asciiToBytes	c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusRtuPort.h,
Modbus, 25	203, 204
asciiToString	c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusSerialPort.h,
Modbus, 25	204, 205
autoIncrement	c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusServerPort.h,
ModbusTcpPort, 126	206
availableBaudRate	c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusServerResource.h,
Modbus, 25	207
availableDataBits	c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusTcpPort.h,
Modbus, 26	208
availableFlowControl	c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusTcpServer.h,
Modbus, 26	209, 210
availableParity	cancelRequest
Modbus, 26	ModbusClientPort, 68
availableSerialPortList	cCliCreate
Modbus, 26	cModbus.h, 148
availableSerialPorts	cCliCreateForClientPort
Modbus, 26	cModbus.h, 148
availableStopBits	cCliDelete
Modbus, 26	cModbus.h, 148
	cCliGetLastPortErrorStatus
baudRate	cModbus.h, 149
ModbusSerialPort, 106	cCliGetLastPortErrorText
bytesToAscii	cModbus.h, 149
Modbus, 26	cCliGetLastPortStatus

oModbuo b. 140	oModbuo b. 154
cModbus.h, 149 cCliGetObjectName	cModbus.h, 154 cCpoReadCoilsAsBoolArray
cModbus.h, 149	cModbus.h, 154
cCliGetPort	
	cCpoReadDiscreteInputs
cModbus.h, 149	cModbus.h, 154
cCliGetType	cCpoReadDiscreteInputsAsBoolArray
cModbus.h, 149	cModbus.h, 154
cCliGetUnit	cCpoReadExceptionStatus
cModbus.h, 149	cModbus.h, 154
cClilsOpen	cCpoReadFIFOQueue
cModbus.h, 150	cModbus.h, 155
cCliSetObjectName	cCpoReadHoldingRegisters
cModbus.h, 150	cModbus.h, 155
cCliSetUnit	cCpoReadInputRegisters
cModbus.h, 150	cModbus.h, 155
cCpoClose	cCpoReadWriteMultipleRegisters
cModbus.h, 150	cModbus.h, 155
cCpoConnectClosed	cCpoReportServerID
cModbus.h, 150	cModbus.h, 155
cCpoConnectError	cCpoSetObjectName
cModbus.h, 150	cModbus.h, 156
cCpoConnectOpened	cCpoSetRepeatCount
cModbus.h, 151	cModbus.h, 156
cCpoConnectRx	cCpoWriteMultipleCoils
cModbus.h, 151	cModbus.h, 156
cCpoConnectTx	cCpoWriteMultipleCoilsAsBoolArray
cModbus.h, 151	cModbus.h, 156
cCpoCreate	cCpoWriteMultipleRegisters
cModbus.h, 151	cModbus.h, 156
cCpoCreateForPort	cCpoWriteSingleCoil
cModbus.h, 151	cModbus.h, 157
cCpoDelete	cCpoWriteSingleRegister
cModbus.h, 151	cModbus.h, 157
cCpoDiagnostics	cCreateModbusDevice
cModbus.h, 152	cModbus.h, 157
cCpoDisconnectFunc	cDeleteModbusDevice
cModbus.h, 152	cModbus.h, 157
cCpoGetCommEventCounter	CharLiteral
cModbus.h, 152	ModbusGlobal.h, 184
cCpoGetCommEventLog	clearConnections
cModbus.h, 152	ModbusTcpServer, 132
cCpoGetLastErrorStatus	close
cModbus.h, 152	ModbusClientPort, 68
cCpoGetLastErrorText	ModbusPort, 98
cModbus.h, 153	ModbusSerialPort, 106
cCpoGetLastStatus	ModbusServerPort, 112
cModbus.h, 153	ModbusServerResource, 118
cCpoGetObjectName	ModbusTcpPort, 126
cModbus.h, 153	ModbusTcpServer, 132
cCpoGetRepeatCount	cMaskWriteRegister
cModbus.h, 153	cModbus.h, 158
cCpoGetType	cModbus.h
cModbus.h, 153	cCliCreateForClientPort 148
cCpolsOpen	cCliCreateForClientPort, 148
cModbus.h, 153	cCliDelete, 148
cCpoMaskWriteRegister cModbus.h, 153	cCliGetLastPortErrorStatus, 149 cCliGetLastPortErrorText, 149
cCpoReadCoils	cCliGetLastPortStatus, 149
	COllidelLastifultotatus, 149

	0 01 100
cCliGetObjectName, 149	cSpoClose, 160
cCliGetPort, 149	cSpoConnectCloseConnection, 160
cCliGetType, 149	cSpoConnectClosed, 160
cCliGetUnit, 149	cSpoConnectError, 160
cClilsOpen, 150	cSpoConnectNewConnection, 161
cCliSetObjectName, 150	cSpoConnectOpened, 161
cCliSetUnit, 150	cSpoConnectRx, 161
cCpoClose, 150	cSpoConnectTx, 161
cCpoConnectClosed, 150	cSpoCreate, 161
cCpoConnectError, 150	cSpoDelete, 161
cCpoConnectOpened, 151	cSpoDisconnectFunc, 162
cCpoConnectRx, 151	cSpoGetDevice, 162
cCpoConnectTx, 151	cSpoGetObjectName, 162
cCpoCreate, 151	cSpoGetType, 162
cCpoCreateForPort, 151	cSpolsOpen, 162
cCpoDelete, 151	cSpolsTcpServer, 162
cCpoDiagnostics, 152	cSpoOpen, 162
cCpoDisconnectFunc, 152	cSpoProcess, 163
cCpoGetCommEventCounter, 152	cSpoSetObjectName, 163
cCpoGetCommEventLog, 152	cWriteMultipleCoils, 163
cCpoGetLastErrorStatus, 152	cWriteMultipleCoilsAsBoolArray, 163
cCpoGetLastErrorText, 153	cWriteMultipleRegisters, 163
cCpoGetLastStatus, 153	cWriteSingleCoil, 163
cCpoGetObjectName, 153	cWriteSingleRegister, 164
cCpoGetRepeatCount, 153	pfDiagnostics, 143
cCpoGetType, 153	pfGetCommEventCounter, 143
cCpolsOpen, 153	pfGetCommEventLog, 143
cCpoMaskWriteRegister, 153	pfMaskWriteRegister, 143
cCpoReadCoils, 154	pfReadCoils, 144
•	•
cCpoReadCoilsAsBoolArray, 154	pfReadDiscreteInputs, 144
cCpoReadDiscreteInputs, 154	pfReadExceptionStatus, 144
cCpoReadDiscreteInputsAsBoolArray, 154	pfReadFIFOQueue, 144 pfReadHoldingRegisters, 145
cCpoReadExceptionStatus, 154	
cCpoReadFIFOQueue, 155	pfReadInputRegisters, 145
cCpoReadHoldingRegisters, 155	pfReadWriteMultipleRegisters, 145
cCpoReadInputRegisters, 155	pfReportServerID, 145
cCpoReadWriteMultipleRegisters, 155	pfSlotCloseConnection, 146
cCpoReportServerID, 155	pfSlotClosed, 146
cCpoSetObjectName, 156	pfSlotError, 146
cCpoSetRepeatCount, 156	pfSlotNewConnection, 146
cCpoWriteMultipleCoils, 156	pfSlotOpened, 146
cCpoWriteMultipleCoilsAsBoolArray, 156	pfSlotRx, 147
cCpoWriteMultipleRegisters, 156	pfSlotTx, 147
cCpoWriteSingleCoil, 157	pfWriteMultipleCoils, 147
cCpoWriteSingleRegister, 157	pfWriteMultipleRegisters, 147
cCreateModbusDevice, 157	pfWriteSingleCoil, 147
cDeleteModbusDevice, 157	pfWriteSingleRegister, 148
cMaskWriteRegister, 158	connect
cPortCreate, 158	ModbusObject, 94
cPortDelete, 158	Constants
cReadCoils, 158	Modbus, 21
cReadCoilsAsBoolArray, 158	context
cReadDiscreteInputs, 159	ModbusServerPort, 112
cReadDiscreteInputsAsBoolArray, 159	cPortCreate
cReadExceptionStatus, 159	cModbus.h, 158
cReadHoldingRegisters, 159	cPortDelete
cReadInputRegisters, 159	cModbus.h, 158
cReadWriteMultipleRegisters, 160	crc16

Modbus, 27	cModbus.h, 162
cReadCoils	cSpoProcess
cModbus.h, 158	cModbus.h, 163
cReadCoilsAsBoolArray	cSpoSetObjectName
cModbus.h, 158	cModbus.h, 163
cReadDiscreteInputs	currentClient
cModbus.h, 159	ModbusClientPort, 68
cReadDiscreteInputsAsBoolArray	currentTimestamp
cModbus.h, 159	Modbus, 29
cReadExceptionStatus	cWriteMultipleCoils
cModbus.h, 159	cModbus.h, 163
cReadHoldingRegisters	cWriteMultipleCoilsAsBoolArray
cModbus.h, 159	cModbus.h, 163
cReadInputRegisters	cWriteMultipleRegisters
cModbus.h, 159	cModbus.h, 163
cReadWriteMultipleRegisters	cWriteSingleCoil
cModbus.h, 160 createClientPort	cModbus.h, 163
	cWriteSingleRegister
Modbus, 27, 28	cModbus.h, 164
createPort	dataBits
Modbus, 28 createServerPort	ModbusSerialPort, 107
Modbus, 28, 29	Defaults
createTcpPort	Modbus::Defaults, 50
ModbusTcpServer, 132	ModbusSerialPort::Defaults, 52
cSpoClose	ModbusTcpPort::Defaults, 53
cModbus.h, 160	ModbusTcpServer::Defaults, 54
cSpoConnectCloseConnection	deleteTcpPort
cModbus.h, 160	ModbusTcpServer, 132
cSpoConnectClosed	device
cModbus.h, 160	ModbusServerPort, 112
cSpoConnectError	diagnostics
cModbus.h, 160	ModbusClient, 59
cSpoConnectNewConnection	ModbusClientPort, 68
cModbus.h, 161	ModbusInterface, 86
cSpoConnectOpened	disconnect
cModbus.h, 161	ModbusObject, 95
cSpoConnectRx	disconnectFunc
cModbus.h, 161	ModbusObject, 95
cSpoConnectTx	
cModbus.h, 161	emitSignal
cSpoCreate	ModbusObject, 95
cModbus.h, 161	enumKey
cSpoDelete	Modbus, 29
cModbus.h, 161	enumValue
cSpoDisconnectFunc	Modbus, 29, 30
cModbus.h, 162	EvenParity
cSpoGetDevice	Modbus, 22
cModbus.h, 162	exec
cSpoGetObjectName	ModbusSlotBase < ReturnType, Args >, 120
cModbus.h, 162	ModbusSlotFunction < ReturnType, Args >, 122
cSpoGetType	ModbusSlotMethod< T, ReturnType, Args >, 123
cModbus.h, 162	FlowControl
cSpolsOpen	Modbus, 21
cModbus.h, 162	flowControl
cSpolsTcpServer	ModbusSerialPort, 107
cModbus.h, 162	modbassian ort, 107
cSpoOpen	GET_BIT
	_

Marallacia Olahari Ia. 404	Marillana 00
ModbusGlobal.h, 184	Modbus, 22
GET_BITS  ModbusClobal b. 185	host ModbugTopPort 136
ModbusGlobal.h, 185 qetBit	ModbusTcpPort, 126
Modbus, 30	instance
getBitS	Modbus::Defaults, 51
Modbus, 30	Modbus::Strings, 137
getBits	ModbusSerialPort::Defaults, 52
Modbus, 31	ModbusTcpPort::Defaults, 53
getBitsS	ModbusTcpServer::Defaults, 54
Modbus, 31	isBlocking
getCommEventCounter	ModbusPort, 98
ModbusClient, 59	isBroadcastEnabled
ModbusClientPort, 69	ModbusClientPort, 70
ModbusInterface, 86	ModbusServerPort, 112
getCommEventLog	isChanged
ModbusClient, 59	ModbusPort, 98
ModbusClientPort, 69, 70	isNonBlocking
ModbusInterface, 86	ModbusPort, 98
getLastErrorText	isOpen
Modbus, 31	ModbusClient, 59
getRequestStatus	ModbusClientPort, 71
ModbusClientPort, 70	ModbusPort, 98
getSettingBaudRate	ModbusSerialPort, 107
Modbus, 31	ModbusServerPort, 113
getSettingBroadcastEnabled	ModbusServerResource, 118
Modbus, 32	ModbusTcpPort, 127
getSettingDataBits	ModbusTcpServer, 132
Modbus, 32	isServerMode
getSettingFlowControl	ModbusPort, 98
Modbus, 32	isStateClosed
getSettingHost	ModbusServerPort, 113
Modbus, 32	isTcpServer
getSettingParity	ModbusServerPort, 113
Modbus, 32	ModbusTcpServer, 132
getSettingPort	isValid
Modbus, 32	Modbus::Address, 48
getSettingSerialPortName	lastErrorStatus
Modbus, 33	ModbusClientPort, 71
getSettingStopBits	ModbusPort, 99
Modbus, 33	lastErrorText
getSettingTimeout	ModbusClientPort, 71
Modbus, 33	ModbusPort, 99
getSettingTimeoutFirstByte  Modbus, 33	lastPortErrorStatus
•	ModbusClient, 60
getSettingTimeoutInterByte	lastPortErrorText
Modbus, 33 getSettingTries	ModbusClient, 60
Modbus, 33	lastPortStatus
getSettingType	ModbusClient, 60
Modbus, 34	lastStatus
getSettingUnit	ModbusClientPort, 71
Modbus, 34	lastStatusTimestamp
Wodbab, or	ModbusClientPort, 71
handle	Irc
ModbusPort, 98	Modbus, 34
ModbusSerialPort, 107	
ModbusTcpPort, 126	MarkParity
HardwareControl	Modbus, 22

maskWriteRegister	getBitS, 30
ModbusClient, 60	getBits, 31
ModbusClientPort, 71	getBitsS, 31
ModbusInterface, 87	getLastErrorText, 31
MB_RTU_IO_BUFF_SZ	getSettingBaudRate, 31
ModbusGlobal.h, 185	getSettingBroadcastEnabled, 32
MB_UNITMAP_GET_BIT	getSettingDataBits, 32
ModbusGlobal.h, 185	getSettingFlowControl, 32
MB_UNITMAP_SET_BIT	getSettingHost, 32
	-
ModbusGlobal.h, 185	getSettingParity, 32
Memory_0x	getSettingPort, 32
Modbus, 21	getSettingSerialPortName, 33
Memory_1x	getSettingStopBits, 33
Modbus, 21	getSettingTimeout, 33
Memory_3x	getSettingTimeoutFirstByte, 33
Modbus, 21	getSettingTimeoutInterByte, 33
Memory_4x	getSettingTries, 33
Modbus, 21	getSettingType, 34
Memory_Coils	getSettingUnit, 34
Modbus, 21	HardwareControl, 22
Memory_DiscreteInputs	Irc, 34
Modbus, 21	MarkParity, 22
Memory_HoldingRegisters	Memory_0x, 21
Modbus, 21	Memory_1x, 21
	Memory_3x, 21
Memory_InputRegisters	•
Modbus, 21	Memory_4x, 21
Memory_Unknown	Memory_Coils, 21
Modbus, 21	Memory_DiscreteInputs, 21
methodOrFunction	Memory_HoldingRegisters, 21
ModbusSlotBase< ReturnType, Args >, 120	Memory_InputRegisters, 21
ModbusSlotFunction < ReturnType, Args >, 122	Memory_Unknown, 21
ModbusSlotMethod< T, ReturnType, Args >, 123	modbusLibVersion, 34
Modbus, 17	modbusLibVersionStr, 34
_MemoryType, 21	msleep, 34
addressFromString, 25	NoFlowControl, 22
ASC, 22	NoParity, 22
asciiToBytes, 25	OddParity, 22
asciiToString, 25	OneAndHalfStop, 25
availableBaudRate, 25	OneStop, 25
	•
availableDataBits, 26	Parity, 22
availableFlowControl, 26	ProtocolType, 22
availableParity, 26	readMemBits, 35
availableSerialPortList, 26	readMemRegs, 35
availableSerialPorts, 26	RTU, 22
availableStopBits, 26	sascii, 36
bytesToAscii, 26	sbytes, 36
bytesToString, 27	setBit, 36
Constants, 21	setBitS, 36
crc16, 27	setBits, 37
createClientPort, 27, 28	setBitsS, 37
createPort, 28	setSettingBaudRate, 37
createServerPort, 28, 29	setSettingBroadcastEnabled, 37
currentTimestamp, 29	setSettingDataBits, 38
enumKey, 29	setSettingFlowControl, 38
•	_
enumValue, 29, 30	setSettingHost, 38
EvenParity, 22	setSettingParity, 38
FlowControl, 21	setSettingPort, 38
getBit, 30	setSettingSerialPortName, 38

setSettingStopBits, 39	StopBits, 23
setSettingTimeout, 39	TCP, 22
setSettingTimeoutFirstByte, 39	timer, 41
setSettingTimeoutInterByte, 39	toBaudRate, 41
setSettingTries, 39	toDataBits, 42
setSettingType, 39	toFlowControl, 42
setSettingUnit, 40	toModbusOffset, 42
sflowControl, 40	toModbusString, 42
SoftwareControl, 22	toParity, 43
SpaceParity, 22	toProtocolType, 43
sparity, 40	toStopBits, 43, 44
sprotocolType, 40	toString, 44
sstopBits, 40	TwoStop, 25
STANDARD_TCP_PORT, 21	VALID MODBUS ADDRESS BEGIN, 21
Status Bad, 22	VALID_MODBUS_ADDRESS_END, 21
<del>-</del> · · ·	
Status_BadAcknowledge, 23	writeMemBits, 45
Status_BadAscChar, 23	writeMemRegs, 45
Status_BadAscMissColon, 23	Modbus::Address, 47
Status_BadAscMissCrLf, 23	Address, 47, 48
Status_BadCrc, 23	isValid, 48
Status_BadEmptyResponse, 23	number, 48
Status_BadGatewayPathUnavailable, 23	offset, 48
Status_BadGatewayTargetDeviceFailedToRespond,	operator quint32, 48
23	operator=, 48
Status_BadIllegalDataAddress, 23	toString, 49
Status_BadIllegalDataValue, 23	type, 49
Status_BadIllegalFunction, 23	Modbus::Defaults, 49
Status BadLrc, 23	Defaults, 50
Status_BadMemoryParityError, 23	instance, 51
Status_BadNegativeAcknowledge, 23	Modbus::SerialSettings, 135
Status_BadNotCorrectRequest, 23	Modbus::Strings, 136
Status_BadNotCorrectResponse, 23	instance, 137
Status BadReadBufferOverflow, 23	Strings, 137
Status_BadSerialOpen, 23	Modbus::TcpSettings, 138
Status BadSerialRead, 23	ModbusAscPort, 54
Status BadSerialReadTimeout, 23	~ModbusAscPort, 56
<del>_</del>	
Status_BadSerialWrite, 23	ModbusAscPort, 56
Status_BadSerialWriteTimeout, 23	readBuffer, 56
Status_BadServerDeviceBusy, 23	type, 56
Status_BadServerDeviceFailure, 23	writeBuffer, 56
Status_BadTcpAccept, 23	ModbusClient, 57
Status_BadTcpBind, 23	diagnostics, 59
Status_BadTcpConnect, 23	getCommEventCounter, 59
Status_BadTcpCreate, 23	getCommEventLog, 59
Status_BadTcpDisconnect, 23	isOpen, 59
Status_BadTcpListen, 23	lastPortErrorStatus, 60
Status_BadTcpRead, 23	lastPortErrorText, 60
Status_BadTcpWrite, 23	lastPortStatus, 60
Status_BadWriteBufferOverflow, 23	maskWriteRegister, 60
Status_Good, 22	ModbusClient, 59
Status_Processing, 22	port, 60
Status_Uncertain, 22	readCoils, 60
StatusCode, 22	readCoilsAsBoolArray, 60
StatusIsBad, 40	readDiscreteInputs, 61
StatusIsGood, 41	readDiscreteInputsAsBoolArray, 61
StatusIsProcessing, 41	readExceptionStatus, 61
StatusIsStandardError, 41	readFIFOQueue, 61
StatusIsUncertain, 41	readHoldingRegisters, 61
Cialagio i i i i i i i i i i i i i i i i i i	rodui rolulligi logisters, UT

readInputRegisters, 62	GET_BITS, 185
readWriteMultipleRegisters, 62	MB_RTU_IO_BUFF_SZ, 185
reportServerID, 62	MB_UNITMAP_GET_BIT, 185
setUnit, 62	MB_UNITMAP_SET_BIT, 185
type, 62	SET BIT, 185
unit, 63	SET_BITS, 186
writeMultipleCoils, 63	StringLiteral, 186
•	
writeMultipleCoilsAsBoolArray, 63	ModbusInterface, 85
writeMultipleRegisters, 63	diagnostics, 86
writeSingleCoil, 63	getCommEventCounter, 86
writeSingleRegister, 63	getCommEventLog, 86
ModbusClientPort, 64	maskWriteRegister, 87
cancelRequest, 68	readCoils, 87
close, 68	readDiscreteInputs, 88
currentClient, 68	readExceptionStatus, 88
diagnostics, 68	readFIFOQueue, 89
getCommEventCounter, 69	readHoldingRegisters, 89
getCommEventLog, 69, 70	readInputRegisters, 89
	readWriteMultipleRegisters, 90
getRequestStatus, 70	· · · · · · · · · · · · · · · · · · ·
isBroadcastEnabled, 70	reportServerID, 90
isOpen, 71	writeMultipleCoils, 91
lastErrorStatus, 71	writeMultipleRegisters, 91
lastErrorText, 71	writeSingleCoil, 92
lastStatus, 71	writeSingleRegister, 92
lastStatusTimestamp, 71	ModbusLib, 1
maskWriteRegister, 71	modbusLibVersion
ModbusClientPort, 67	Modbus, 34
port, 72	modbusLibVersionStr
readCoils, 72	Modbus, 34
readCoilsAsBoolArray, 73	Modbus, 34 ModbusObject, 93
readDiscreteInputs, 73	∼ModbusObject, 94
readDiscreteInputsAsBoolArray, 75	connect, 94
readExceptionStatus, 75	disconnect, 95
readFIFOQueue, 76	disconnectFunc, 95
readHoldingRegisters, 76, 77	emitSignal, 95
readInputRegisters, 77	ModbusObject, 94
readWriteMultipleRegisters, 78	ModbusServerPort, 113
repeatCount, 79	objectName, 96
reportServerID, 79	sender, 96
setBroadcastEnabled, 79	setObjectName, 96
setPort, 80	ModbusPort, 96
setRepeatCount, 80	$\sim$ ModbusPort, 97
setTries, 80	close, 98
signalClosed, 80	handle, 98
•	
signalError, 80	isBlocking, 98
signalOpened, 80	isChanged, 98
signalRx, 81	isNonBlocking, 98
signalTx, 81	isOpen, 98
tries, 81	isServerMode, 98
type, 81	lastErrorStatus, 99
writeMultipleCoils, 81	lastErrorText, 99
writeMultipleCoilsAsBoolArray, 82	open, 99
writeMultipleRegisters, 82, 83	read, 99
writeSingleCoil, 83	readBuffer, 99
writeSingleRegister, 84	readBufferData, 99
ModbusGlobal.h	readBufferSize, 100
CharLiteral, 184	setError, 100
GET_BIT, 184	setNextRequestRepeated, 100

setServerMode, 100	setDevice, 114
setTimeout, 100	setUnitMap, 114
timeout, 100	signalClosed, 114
type, 101	signalError, 115
write, 101	signalOpened, 115
writeBuffer, 101	signalRx, 115
writeBufferData, 101	signalTx, 115
writeBufferSize, 101	type, 115
ModbusRtuPort, 102	unitMap, 115
~ModbusRtuPort, 103	ModbusServerResource, 116
ModbusRtuPort, 103	close, 118
readBuffer, 104	isOpen, 118 ModbusServerResource, 118
type, 104 writeBuffer, 104	
ModbusSerialPort, 105	open, 118 port, 118
~ModbusSerialPort, 106	process, 118
baudRate, 106	processDevice, 119
close, 106	processInputData, 119
dataBits, 107	processOutputData, 119
flowControl, 107	type, 119
handle, 107	ModbusSlotBase< ReturnType, Args >, 119
isOpen, 107	~ModbusSlotBase, 120
open, 107	exec, 120
parity, 107	methodOrFunction, 120
portName, 108	object, 120
read, 108	ModbusSlotFunction
readBufferData, 108	ModbusSlotFunction< ReturnType, Args >, 121
readBufferSize, 108	ModbusSlotFunction< ReturnType, Args >, 121
setBaudRate, 108	exec, 122
setDataBits, 108	methodOrFunction, 122
setFlowControl, 108	ModbusSlotFunction, 121
setParity, 109	ModbusSlotMethod
setPortName, 109	ModbusSlotMethod< T, ReturnType, Args >, 123
setStopBits, 109	ModbusSlotMethod< T, ReturnType, Args >, 122
setTimeoutFirstByte, 109	exec, 123
setTimeoutInterByte, 109	methodOrFunction, 123
stopBits, 109	ModbusSlotMethod, 123
timeoutFirstByte, 109	object, 124
timeoutInterByte, 110	ModbusTcpPort, 124
write, 110	∼ModbusTcpPort, 126
writeBufferData, 110	autoIncrement, 126
writeBufferSize, 110	close, 126
ModbusSerialPort::Defaults, 51	handle, 126
Defaults, 52	host, 126
instance, 52	isOpen, 127
ModbusServerPort, 111	ModbusTcpPort, 126
close, 112	open, 127
context, 112	port, 127
device, 112	read, 127
isBroadcastEnabled, 112	readBuffer, 127
isOpen, 113	readBufferData, 127
isStateClosed, 113	readBufferSize, 128
isTcpServer, 113	setHost, 128
ModbusObject, 113	setNextRequestRepeated, 128
open, 113	setPort, 128
process, 113	type, 128
setBroadcastEnabled, 114	write, 128
setContext, 114	writeBuffer, 129

writeBufferData, 129	ModbusTcpPort, 127
writeBufferSize, 129	ModbusTcpServer, 133
ModbusTcpPort::Defaults, 52	operator quint32
Defaults, 53	Modbus::Address, 48
instance, 53	operator=
ModbusTcpServer, 129	Modbus::Address, 48
	ModbusAddress, 46
~ModbusTcpServer, 131	Parity
clearConnections, 132	-
close, 132	Modbus, 22
createTcpPort, 132	parity
deleteTcpPort, 132	ModbusSerialPort, 107
isOpen, 132	pfDiagnostics
isTcpServer, 132	cModbus.h, 143
ModbusTcpServer, 131	pfGetCommEventCounter
nextPendingConnection, 133	cModbus.h, 143
_	pfGetCommEventLog
open, 133	cModbus.h, 143
port, 133	pfMaskWriteRegister
process, 133	-
setBroadcastEnabled, 133	cModbus.h, 143
setPort, 134	pfReadCoils
setTimeout, 134	cModbus.h, 144
setUnitMap, 134	pfReadDiscreteInputs
signalCloseConnection, 134	cModbus.h, 144
signalNewConnection, 134	pfReadExceptionStatus
-	cModbus.h, 144
timeout, 135	pfReadFIFOQueue
type, 135	cModbus.h, 144
ModbusTcpServer::Defaults, 53	
Defaults, 54	pfReadHoldingRegisters
instance, 54	cModbus.h, 145
msleep	pfReadInputRegisters
Modbus, 34	cModbus.h, 145
	pfReadWriteMultipleRegisters
nextPendingConnection	cModbus.h, 145
ModbusTcpServer, 133	pfReportServerID
NoFlowControl	cModbus.h, 145
	pfSlotCloseConnection
Modbus, 22	cModbus.h, 146
NoParity	· ·
Modbus, 22	pfSlotClosed
number	cModbus.h, 146
Modbus::Address, 48	pfSlotError
	cModbus.h, 146
object	pfSlotNewConnection
ModbusSlotBase< ReturnType, Args >, 120	cModbus.h, 146
ModbusSlotMethod< T, ReturnType, Args >, 124	pfSlotOpened
objectName	cModbus.h, 146
ModbusObject, 96	pfSlotRx
	-
OddParity	cModbus.h, 147
Modbus, 22	pfSlotTx
offset	cModbus.h, 147
Modbus::Address, 48	pfWriteMultipleCoils
OneAndHalfStop	cModbus.h, 147
Modbus, 25	pfWriteMultipleRegisters
OneStop	cModbus.h, 147
Modbus, 25	pfWriteSingleCoil
open	cModbus.h, 147
ModbusPort, 99	
	pfWriteSingleRegister
ModbusSerialPort, 107	cModbus.h, 148
ModbusServerPort, 113	port
ModbusServerResource, 118	ModbusClient, 60

ModbusClientPort, 72	ModbusClient, 61
ModbusServerResource, 118	ModbusClientPort, 76, 77
ModbusTcpPort, 127	ModbusInterface, 89
ModbusTcpServer, 133	readInputRegisters
portName	ModbusClient, 62
ModbusSerialPort, 108	ModbusClientPort, 77
process	ModbusInterface, 89
ModbusServerPort, 113	readMemBits
ModbusServerResource, 118	Modbus, 35
ModbusTcpServer, 133	readMemRegs
processDevice	Modbus, 35
ModbusServerResource, 119	readWriteMultipleRegisters
processInputData	ModbusClient, 62
ModbusServerResource, 119	ModbusClientPort, 78
processOutputData	ModbusInterface, 90
ModbusServerResource, 119	repeatCount
ProtocolType	ModbusClientPort, 79
Modbus, 22	reportServerID
	ModbusClient, 62
read	ModbusClientPort, 79
ModbusPort, 99	ModbusInterface, 90
ModbusSerialPort, 108	RTU
ModbusTcpPort, 127	Modbus, 22
readBuffer	,
ModbusAscPort, 56	sascii
ModbusPort, 99	Modbus, 36
ModbusRtuPort, 104	sbytes
ModbusTcpPort, 127	Modbus, 36
readBufferData	sender
ModbusPort, 99	ModbusObject, 96
ModbusSerialPort, 108	SET BIT
ModbusTcpPort, 127	ModbusGlobal.h, 185
readBufferSize	SET BITS
ModbusPort, 100	ModbusGlobal.h, 186
ModbusSerialPort, 108	setBaudRate
ModbusTcpPort, 128	ModbusSerialPort, 108
readCoils	setBit
ModbusClient, 60	Modbus, 36
ModbusClientPort, 72	setBitS
ModbusInterface, 87	Modbus, 36
readCoilsAsBoolArray	setBits
ModbusClient, 60	Modbus, 37
ModbusClientPort, 73	setBitsS
readDiscreteInputs	Modbus, 37
ModbusClient, 61	setBroadcastEnabled
ModbusClientPort, 73	ModbusClientPort, 79
ModbusInterface, 88	ModbusServerPort, 114
	ModbusTcpServer, 133
readDiscreteInputsAsBoolArray	•
ModbusClient, 61	setContext
ModbusClientPort, 75	ModbusServerPort, 114
readExceptionStatus	setDataBits
ModbusClient, 61	ModbusSerialPort, 108
ModbusClientPort, 75	setDevice
ModbusInterface, 88	ModbusServerPort, 114
readFIFOQueue	setError
ModbusClient, 61	ModbusPort, 100
ModbusClientPort, 76	setFlowControl
ModbusInterface, 89	ModbusSerialPort, 108
readHoldingRegisters	setHost

ModbusTcpPort, 128	ModbusClientPort, 80
setNextRequestRepeated	setUnit
ModbusPort, 100	ModbusClient, 62
ModbusTcpPort, 128	setUnitMap
setObjectName	ModbusServerPort, 114
ModbusObject, 96	ModbusTcpServer, 134
setParity	sflowControl
ModbusSerialPort, 109	Modbus, 40
setPort	signalCloseConnection
ModbusClientPort, 80	ModbusTcpServer, 134
ModbusTcpPort, 128	signalClosed
ModbusTcpServer, 134	ModbusClientPort, 80
setPortName	ModbusServerPort, 114
ModbusSerialPort, 109	signalError
setRepeatCount	ModbusClientPort, 80
ModbusClientPort, 80	ModbusServerPort, 115
setServerMode	signalNewConnection
ModbusPort, 100	ModbusTcpServer, 134
setSettingBaudRate	signalOpened
Modbus, 37	ModbusClientPort, 80
setSettingBroadcastEnabled	ModbusServerPort, 115
Modbus, 37	signalRx
setSettingDataBits	ModbusClientPort, 81
Modbus, 38	ModbusServerPort, 115
setSettingFlowControl	
Modbus, 38	signalTx  ModbusClientPort, 81
setSettingHost  Modbus 38	ModbusServerPort, 115 SoftwareControl
Modbus, 38	
setSettingParity	Modbus, 22
Modbus, 38	SpaceParity Madhua 88
setSettingPort	Modbus, 22
Modbus, 38	sparity
setSettingSerialPortName	Modbus, 40
Modbus, 38	sprotocolType
setSettingStopBits	Modbus, 40
Modbus, 39	sstopBits
setSettingTimeout	Modbus, 40
Modbus, 39	STANDARD_TCP_PORT
setSettingTimeoutFirstByte	Modbus, 21
Modbus, 39	Status_Bad
setSettingTimeoutInterByte	Modbus, 22
Modbus, 39	Status_BadAcknowledge
setSettingTries	Modbus, 23
Modbus, 39	Status_BadAscChar
setSettingType	Modbus, 23
Modbus, 39	Status_BadAscMissColon
setSettingUnit	Modbus, 23
Modbus, 40	Status_BadAscMissCrLf
setStopBits	Modbus, 23
ModbusSerialPort, 109	Status_BadCrc
setTimeout	Modbus, 23
ModbusPort, 100	Status_BadEmptyResponse
ModbusTcpServer, 134	Modbus, 23
setTimeoutFirstByte	Status_BadGatewayPathUnavailable
ModbusSerialPort, 109	Modbus, 23
setTimeoutInterByte	$Status\_BadGateway Target Device Failed To Respond$
ModbusSerialPort, 109	Modbus, 23
setTries	Status BadIllegalDataAddress

Modbus, 23 Status_BadllegalDataValue Modbus, 23 Status_BadllegalFunction Modbus, 23 Status_BadllegalFunction Modbus, 23 Status_Badler Modbus, 23 Status_Badler Modbus, 23 Status_BadlkegativeAcknowledge Modbus, 23 Status_BadlkegativeAcknowledge Modbus, 23 Status_BadlyeatoveAcknowledge Modbus, 23 Status_BadlyotCorrectRequest Modbus, 23 Status_BadlyotCorrectRequest Modbus, 23 Status_BadlyotCorrectResponse Modbus, 23 Status_BadleadEufferOverflow Modbus, 23 Status_BadleadEufferOverflow Modbus, 23 Status_BadSerialPead Modbus, 23 Status_BadSerialRead Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWriteTimeout Modbus, 23 Status_BadSerialWriteTimeout Modbus, 23 Status_BadServerDeviceBusy Modbus, 23 Status_BadServerDeviceBusy Modbus, 23 Status_BadTopConnect Modbus, 23 Status_BadTopConnect Modbus, 23 Status_BadTopCreate Modbus, 23 Status_BadTopDisconnect Modbus, 24 Status_BadTopDisconnect Modbus, 23 Status_BadDisconnect Modbus, 24 Status_BadDisconnect Modbus, 25 Status_BadDisconnect Modbus, 26 Modbus, 27 Modbus, 28 Modbus, 29 Modbus, 20 M		
Modbus, 23 Slatus, BadlllegalFunction Modbus, 23 Slatus, BadlllegalFunction Modbus, 23 Slatus, BadllegalFunction Modbus, 23 Slatus, Badlker Modbus, 23 Slatus, Badler Modbus, 24 Slatus, Badler Modbus, 25 Slatus, Badler Modbus, 22 Slatus, Decertain Modbus, 22 Slatus, Cood Modbus, 22 Slat	•	
Status_BadlllegalFunction Modbus, 23 Status_BadLrc Modbus, 23 Status_BadNemoryParityError Modbus, 23 Status_BadNemoryParityError Modbus, 23 Status_BadNetCorrectRequest Modbus, 23 Status_BadNotCorrectRequest Modbus, 23 Status_BadNotCorrectResponse Modbus, 23 Status_BadNotCorrectResponse Modbus, 23 Status_BadSerialOpen Modbus, 23 Status_BadSerialOpen Modbus, 23 Status_BadSerialRead Modbus, 23 Status_BadSerialRead Modbus, 23 Status_BadSerialRead Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWriteTimeout Modbus, 23 Status_BadSerialWriteTimeout Modbus, 23 Status_BadSeriavPerviceBusy Modbus, 23 Status_BadTopConnect Modbus, 23 Status_BadTopConnect Modbus, 23 Status_BadTopConnect Modbus, 23 Status_BadTopCreate Modbus, 23 Status_BadTopCreate Modbus, 23 Status_BadTopDisconnect Modbus, 24 Status_BadTopListen Modbus, 25 Status_BadTopDisconnect Modbus, 26 Modbus, 27 Status_BadTopDisconnect Modbus, 28 Modbus, 29 Modbus, 29 Status_BadTopDisconnect Modbus, 29 Modbus, 20 Modbus, 20 Modbus, 20 Modbus, 20 Modbus, 21 Modbus, 22 Modbus, 22 Status_BadTopDisconnect Modbus, 22 M	<del>_</del>	
Modbus, 23 Status BadLrc Modbus, 25 Status BadMemoryParityError Modbus, 23 Status BadMemoryParityError Modbus, 23 Status BadNegativeAcknowledge Modbus, 23 Status BadNotCorrectRequest Modbus, 23 Status BadNotCorrectRequest Modbus, 23 Status BadNotCorrectResponse Modbus, 23 Status BadNotCorrectResponse Modbus, 23 Status BadNotCorrectResponse Modbus, 23 Status BadSerialPort, 109 Status BadSerialPore Modbus, 23 Status BadSerialPore Modbus, 23 Status BadSerialPore Modbus, 23 Status BadSerialRead Modbus, 23 Status BadSerialRead Modbus, 23 Status BadSerialWrite Modbus, 23 Status BadSerialWrite Imeout Modbus, 23 Status BadSerialWrite Timeout Modbus, 23 Status BadServerDeviceBusy Modbus, 23 Status BadServerDeviceBusy Modbus, 23 Status BadServerDeviceFailure Modbus, 23 Status BadTcpAccept Modbus, 23 Status BadTcpConnect Modbus, 23 Status BadTcpConnect Modbus, 23 Status BadTcpCreate Modbus, 23 Status BadTcpCreate Modbus, 23 Status BadTcpDisconnect Modbus, 24 Modbus, 25 Status BadTcpDisconnect Modbus, 26 Modbus, 27 Modbus, 28 Modbus, 29 Modbus, 29 Modbus, 20 Modb		
Status_BadLrc Modbus, 23 Status_BadMemoryParityError Modbus, 23 Status_BadMegativeAcknowledge Modbus, 23 Status_BadNotCorrectRequest Modbus, 23 Status_BadNotCorrectResponse Modbus, 23 Status_BadNotCorrectResponse Modbus, 23 Status_BadNotCorrectResponse Modbus, 23 Status_BadNotCorrectResponse Modbus, 23 Status_BadRedBufferOverflow Modbus, 23 Status_BadSerialCopen Modbus, 23 Status_BadSerialRead Modbus, 23 Status_BadSerialRead Modbus, 23 Status_BadSerialReadTimeout Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadServerDeviceBusy Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadTopAccept Modbus, 23 Status_BadTopAccept Modbus, 23 Status_BadTopConnect Modbus, 23 Status_BadTopConnect Modbus, 23 Status_BadTopConnect Modbus, 23 Status_BadTopConnect Modbus, 23 Status_BadTopDisconnect Modbus, 23 Status_BadTopListen Modbus, 24 Modbus, 42 Modbus, 43 Modbus, 42 Modbus, 43 Modbus, 43 Modbus, 43 Modbus, 44 Modbus, 45 Modbus, 46 Modbus, 47 Modbus, 49 Modbus, 49 Modbus, 40 Modbus,		
Modbus, 23 Status_BadMemoryParityError Modbus, 23 Status_BadNegativeAcknowledge Modbus, 23 Status_BadNotCorrectRequest Modbus, 23 Status_BadNotCorrectRequest Modbus, 23 Status_BadNotCorrectResponse Modbus, 23 Status_BadNotCorrectResponse Modbus, 23 Status_BadSerialCpen Modbus, 23 Status_BadSerialCpen Modbus, 23 Status_BadSerialCpen Modbus, 23 Status_BadSerialCpen Modbus, 23 Status_BadSerialRead Modbus, 23 Status_BadSerialRead Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWrite imeout Modbus, 23 Status_BadSerialWrite imeout Modbus, 23 Status_BadSerialWrite imeout Modbus, 23 Status_BadSerialWrite imeout Modbus, 23 Status_BadSerialVrite imeout Modbus, 23 Status_BadSerialVrite imeout Modbus, 23 Status_BadSerialVrite imeout Modbus, 23 Status_BadSerialVrite imeout Modbus, 23 Status_BadSerialPrit, 110 Status_BadSerialPrit, 110 Status_BadSerialVrite Modbus, 23 Status_BadSerialPrity imeout Modbus, 23 Status_BadSerialPrity imeout Modbus, 23 Status_BadSerialPrity imeout Modbus, 23 Status_BadTcpAccept Modbus, 23 Status_BadTcpAccept Modbus, 23 Status_BadTcpAccept Modbus, 23 Status_BadTcpConnect Modbus, 23 Status_BadTcpConnect Modbus, 23 Status_BadTcpConnect Modbus, 23 Status_BadTcpCoreate Modbus, 23 Status_BadTcpDisconnect Modbus, 23 Modbus, 24 Modbus, 25 Modbus, 26 Status_BadValus_BadValus_BadValus_BadValus_BadValus_BadValus_BadValus_BadValus_BadValus_BadValu		
Status_BadMemoryParityError Modbus, 23 Status_BadNegativeAcknowledge Modbus, 23 Status_BadNotCorrectRequest Modbus, 23 Status_BadNotCorrectRequest Modbus, 23 Status_BadNotCorrectReponse Modbus, 23 Status_BadNotCorrectResponse Modbus, 23 Status_BadReadBufferOverflow Modbus, 23 Status_BadSerialCpen Modbus, 23 Status_BadSerialCpen Modbus, 23 Status_BadSerialRead Modbus, 23 Status_BadSerialRead Modbus, 23 Status_BadSerialReadTimeout Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWriteTimeout Modbus, 23 Status_BadServerDeviceBusy Modbus, 23 Status_BadServerDeviceBusy Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadTcpAccept Modbus, 23 Status_BadTcpConnect Modbus, 23 Status_BadTcpConnect Modbus, 23 Status_BadTcpCreate Modbus, 23 Status_BadTcpCreate Modbus, 23 Status_BadTcpListen Modbus, 23 Status_BadTcpCreate Modbus, 23 Status_BadTcpListen Modbus, 23 Status_BadTcpListen Modbus, 23 Status_BadTcpListen Modbus, 23 Status_BadTcpListen Modbus, 23 Status_BadTcpCreate Modbus, 23 Status_BadTcpListen Modbus, 24 Modbus, 25 Status_BadTcpListen Modbus, 25 Status_BadTcpListen Modbus, 26 Modbus, 27 Status_BadTcpListen Modbus, 28 Status_BadTcpListen Modbus, 29 Modbus, 20 Status_BadTcpLientlen Modbus, 22 Status_BadTcpLientlen Modbus, 22 Status_BadCeplated Modbus, 22 ModbusClientPort, 81 ModbusClientPort, 81 ModbusClientPort, 81		
Modbus, 23 Status_BadNegativeAcknowledge Modbus, 23 Status_BadNotCorrectRequest Modbus, 23 Status_BadNotCorrectRequest Modbus, 23 Status_BadNotCorrectResponse Modbus, 23 Status_BadNotCorrectResponse Modbus, 23 Status_BadNotCorrectResponse Modbus, 23 Status_BadReadBufferOverflow Modbus, 23 Status_BadSerialOpen Modbus, 23 Status_BadSerialRead Modbus, 23 Status_BadSerialRead Modbus, 23 Status_BadSerialReadTimeout Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWriteTimeout Modbus, 23 Status_BadServerDeviceBusy Modbus, 23 Status_BadServerDeviceBusy Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadTopAccept Modbus, 23 Status_BadTopAccept Modbus, 23 Status_BadTopConnect Modbus, 23 Status_BadTopConnect Modbus, 23 Status_BadTopConnect Modbus, 23 Status_BadTopCreate Modbus, 23 Status_BadTopCreate Modbus, 23 Status_BadTopCreate Modbus, 23 Status_BadTopDisconnect Modbus, 23 Status_BadTopUsten Modbus, 23 Status_BadTopDwrite Modbus, 23 Status_BadTopVrite Modbus, 23 Status_BadTopNrite Modbus, 23 Status_BadVierBufferOverflow Modbus, 23 Status_BadVierBufferOverflow Modbus, 23 Status_BadVierBufferOverflow Modbus, 24 Modbus, 25 Status_DadVierBufferOverflow Modbus, 25 Status_DadVierBufferOverflow Modbus, 22 Status_DadVierBufferOverflow Modbus, 22 Status_DadVierBufferOverflow Modbus, 22 Status_DadVierBufferOverflow Modbus, 22 Status_DadVierBufferOverflow Modbus, 2		,
Status_BadNegativeAcknowledge Modbus, 23 Status_BadNotCorrectRequest Modbus, 23 Status_BadNotCorrectResponse Modbus, 23 Status_BadNotCorrectResponse Modbus, 23 Status_BadNotCorrectResponse Modbus, 23 Status_BadReadBufferOverflow Modbus, 23 Status_BadReadBufferOverflow Modbus, 23 Status_BadSerialQpen Modbus, 23 Status_BadSerialRead Modbus, 23 Status_BadSerialReadTimeout Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWriteTimeout Modbus, 23 Status_BadServerDeviceBusy Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadTopAccept Modbus, 23 Status_BadTopAccept Modbus, 23 Status_BadTopDisconnect Modbus, 23 Status_BadTopConnect Modbus, 23 Status_BadTopConnect Modbus, 23 Status_BadTopConnect Modbus, 23 Status_BadTopDisconnect Modbus, 23 Modbus, 42 Modbus, 42 ToPortocolType Modbus, 43 Modbus, 42 ToPortocolType Modbus, 43 Modbus, 44 Modbus, 45 Modbus, 46 Modbus, 47 Modbus, 49 Modbus, 49 Modbus, 49 Modbus, 49 Modbus, 40 Modbus, 40 Modbus, 41 Modbus, 41 Modbus, 41 Modbus, 42 Modbus, 42 Modbus, 42 Modbus, 43 Modbus, 44 Modbus, 45 Modbus, 46 Modbus, 47 Modbus, 49 Modbus, 40 Modbus, 40 Modbus, 41 Modbus, 41 Modbus, 42 Modbus, 41 Modbus, 42 Modbus, 41 Modbus, 42 Modbus, 42 Modbus, 41 Modbus, 41 Modbus, 42 Modbus,		
Modbus, 23 Status_BadNotCorrectRequest Modbus, 23 Status_BadNotCorrectResponse Modbus, 23 Status_BadNotCorrectResponse Modbus, 23 Status_BadReadBufferOverflow Modbus, 23 Status_BadSerialOpen Modbus, 23 Status_BadSerialPead Modbus, 23 Status_BadSerialRead Modbus, 23 Status_BadSerialRead Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadServerDeviceBusy Modbus, 23 Status_BadServerDeviceBusy Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadTopAccept Modbus, 23 Status_BadTopAccept Modbus, 23 Status_BadTopAccept Modbus, 23 Status_BadTopConnect Modbus, 23 Status_BadTopDisconnect Modbus, 23 Status_BadTopConnect Modbus, 23 Status_BadTopListen Modbus, 23 Status_BadTopWrite Modbus, 23 Status_BadTopWrite Modbus, 23 Status_BadTopWrite Modbus, 23 Status_BadWriteBufferOverflow Modbus, 23 Modbus, 24 Modbus, 25 Status_Dood Modbus, 22 Status_Drocessing Modbus, 22 Status_Uncertain Modbus, 22 Status_Uncertain Modbus, 22 Status_Drocessing Modbus, 22 Status_Drocessing Modbus, 22 Status_Drocessing Modbus, 22 Modbus, 22 Status_Drocessing Modbus, 22		,
Status_BadNotCorrectRequest Modbus, 23 Status_BadNotCorrectResponse Modbus, 23 Status_BadReadBufferOverflow Modbus, 23 Status_BadReadBufferOverflow Modbus, 23 Status_BadSerialOpen Modbus, 23 Status_BadSerialRead Modbus, 23 Status_BadSerialRead Modbus, 23 Status_BadSerialReadTimeout Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWriteTimeout Modbus, 23 Status_BadSerialWriteTimeout Modbus, 23 Status_BadServerDeviceBusy Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadTopAccept Modbus, 23 Status_BadTopCreate Modbus, 23 Status_BadTopDisconnect Modbus, 23 Modbus, 24 Modbus, 43 Modbus, 43 Modbus, 43 Modbus, 43 Modbus, 43 Modbus, 44 Modbus, 45 Modbus, 45 Modbus, 46 Modbus, 47 Modbus, 48 Modbus, 49 Modbus, 49 Modbus, 49 Modbus, 40 Modbus, 40 Modbus, 41 Modbus, 41 Modbus, 42 Modbus, 42 Modbus, 42 Modbus, 42 Modbus, 43 Modbus, 44 Modbus, 45 Modbus, 45 Modbus, 45 Modbus, 46 Modbus, 47 Modbus, 47 Modbus, 49 Modbus, 40 Modbus, 40 Modbus, 40 Modbus, 40 Modbus, 40		•
Modbus, 23 Status BadNotCorrectResponse Modbus, 23 Status BadReadBufferOverflow Modbus, 23 Status BadSerialOpen Modbus, 23 Status BadSerialRead Modbus, 23 Status BadSerialRead Modbus, 23 Status BadSerialRead Modbus, 23 Status BadSerialReadTimeout Modbus, 23 Status BadSerialWrite Modbus, 23 Status BadServerDeviceBusy Modbus, 23 Status BadServerDeviceBusy Modbus, 23 Status BadServerDeviceFailure Modbus, 23 Status BadTopAccept Modbus, 23 Status BadTopAccept Modbus, 23 Status BadTopAccept Modbus, 23 Status BadTopConnect Modbus, 23 Status BadTopConnect Modbus, 23 Status BadTopCreate Modbus, 23 Status BadTopCreate Modbus, 23 Status BadTopCreate Modbus, 23 Status BadTopDisconnect Modbus, 24 Modbus, 24 Modbus, 25 Status BadWriteBufferOverflow Modbus, 24 Mod		
Status_BadNotCorrectResponse Modbus, 23 Status_BadReadButferOverflow Modbus, 23 Modbus:Strings, 137 Status_BadSerialOpen Modbus, 23 Status_BadSerialRead Modbus, 23 Status_BadSerialReadTimeout Modbus, 23 Status_BadSerialReadTimeout Modbus, 23 Status_BadSerialReadTimeout Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWriteTimeout Modbus, 23 Status_BadSerialWriteTimeout Modbus, 23 Status_BadServerDeviceBusy Modbus, 23 Status_BadServerDeviceBusy Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadTopAccept Modbus, 23 Status_BadTopAccept Modbus, 23 Status_BadTopAccept Modbus, 23 Status_BadTopAccept Modbus, 23 Status_BadTopCreate Modbus, 23 Status_BadTopCreate Modbus, 23 Status_BadTopDisconnect Modbus, 24 Modbus, 43 Status_BadTopDisconnect Modbus, 25 Status_BadWriteBufferOverflow Modbus, 26 Modbus, 27 Modbus, 28 Modbus, 28 Modbus, 28 Modbus, 28 Modbus, 28 Modbus, 29 Modbus, 29 Modbus, 29 Modbus, 29 Modbus, 29 Modbus, 20 Modbus, 20 Modbus, 20 Modbus, 20 Modbus, 20 Modbus, 21 Modbus, 22 Modbus,	·	
Modbus, 23 Status BadReadBufferOverflow Modbus, 23 Status BadSerialOpen Modbus, 23 Status BadSerialRead Modbus, 23 Status BadSerialRead Modbus, 23 Status BadSerialRead Modbus, 23 Status BadSerialReadTimeout Modbus, 23 Status BadSerialWrite Modbus, 23 Status BadSerialWrite Timeout Modbus, 23 Status BadServerDeviceBusy Modbus, 23 Status BadServerDeviceBusy Modbus, 23 Status BadServerDeviceFailure Modbus, 23 Status BadTopAccept Modbus, 23 Status BadTopConnect Modbus, 23 Status BadTopConnect Modbus, 23 Status BadTopDisconnect Modbus, 24 Modbus, 25 Status Processing Modbus, 25 Status Processing Modbus, 22 Status Locertain Modbus, 22 Status Modbus, 22 Modbus, 23 Modbus, 23 Modbus		
Status_BadReadBufferOverflow Modbus, 23 Status_BadSerialOpen Modbus, 23 Status_BadSerialRead Modbus, 23 Status_BadSerialRead Modbus, 23 Status_BadSerialReadTimeout Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWriteTimeout Modbus, 23 Status_BadSerialWriteTimeout Modbus, 23 Status_BadServerDeviceBusy Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadTopAccept Modbus, 23 Status_BadTopBrind Modbus, 23 Status_BadTopBrind Modbus, 23 Status_BadTopConnect Modbus, 23 Status_BadTopListen Modbus, 23 Status_BadTopListen Modbus, 23 Status_BadTopListen Modbus, 23 Status_BadTopWrite Modbus, 23 Status_BadVitleBufferOverflow Modbus, 23 Status_BadVitleBufferOverflow Modbus, 23 Status_Processing Modbus, 22 Modbus_ClientPort, 81 Modbus, 22 Status_Uncertain Modbus, 22 ModbusClientPort, 81	-	_
Modbus, 23 Status BadSerialOpen Modbus, 23 Status BadSerialRead Modbus, 23 Status BadSerialReadTimeout Modbus, 23 Status BadSerialReadTimeout Modbus, 23 Status BadSerialWrite Modbus, 23 Status BadSerialWrite Modbus, 23 Status BadSerialWriteTimeout Modbus, 23 Status BadServerDeviceBusy Modbus, 23 Status BadServerDeviceFailure Modbus, 23 Status BadServerDeviceFailure Modbus, 23 Status BadTcpAccept Modbus, 23 Status BadTcpAccept Modbus, 23 Status BadTcpBind Modbus, 23 Status BadTcpConnect Modbus, 23 Status BadTcpConnect Modbus, 23 Status BadTcpConnect Modbus, 23 Status BadTcpDisconnect Modbus, 23 Status BadTcpDisconnect Modbus, 23 Status BadTcpListen Modbus, 23 Status BadTcpDisconnect Modbus, 23 Status BadTcpRead Modbus, 23 Status BadTcpRead Modbus, 23 Status BadTcpRead Modbus, 23 Modbus, 24 Modbus, 43 Modbus, 43 Modbus, 43 Modbus, 44 Modbus, 25 Modbus, 25 Modbus, 25 Modbus, 25 Modbus, 25 Modbus, 25 ModbusClientPort, 81		
Status_BadSerialOpen Modbus, 23 Status_BadSerialRead Modbus, 23 Status_BadSerialReadTimeout Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWriteTimeout Modbus, 23 Status_BadServerDeviceBusy Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadTcpAccept Modbus, 23 Status_BadTcpAccept Modbus, 23 Status_BadTcpConnect Modbus, 23 Status_BadTcpConnect Modbus, 23 Status_BadTcpCreate Modbus, 23 Status_BadTcpCreate Modbus, 23 Status_BadTcpListen Modbus, 23 Status_BadTcpListen Modbus, 23 Status_BadTcpListen Modbus, 23 Status_BadTcpCreate Modbus, 23 Status_BadTcpListen Modbus, 23 Status_BadTcpListen Modbus, 23 Status_BadTcpCreate Modbus, 23 Status_BadTcpListen Modbus, 23 Status_BadTcpListen Modbus, 23 Status_BadTcpWrite Modbus, 23 Status_BadTcpWrite Modbus, 23 Status_BadTcpWrite Modbus, 23 Status_BadTcpWrite Modbus, 23 Status_BadTcpCoefflow Modbus, 24 Modbus, 25 Status_BadTcpCoefflow Modbus, 22 Status_Uncertain Modbus, 22 Status_Uncertain Modbus, 22 Status_Uncertain Modbus, 22 Status_Uncertain Modbus, 22 Status_BadDed Modbus, 22 Status_BadDed Modbus, 22 Status_Badd Modbus, 22 ModbusClientFort, 81	<del>-</del>	
Modbus, 23 Status_BadSerialRead Modbus, 23 Status_BadSerialReadTimeout Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWriteTimeout Modbus, 23 Status_BadServerDeviceBusy Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadTcpAccept Modbus, 23 Status_BadTcpAccept Modbus, 23 Status_BadTcpConnect Modbus, 23 Status_BadTcpConnect Modbus, 23 Status_BadTcpConnect Modbus, 23 Status_BadTcpDisconnect Modbus, 23 Status_BadTcpDisten Modbus, 23 Status_BadTcpDisten Modbus, 23 Status_BadTcpDisten Modbus, 23 Status_BadTcpWrite Modbus, 23 Status_BadTcpDwrite Modbus, 23 Status_BadTcpConnect Modbus, 23 Status_BadTcpDwrite Modbus, 23 Status_BadTcpRead Modbus, 23 Status_BadTcpWrite Modbus, 23 Status_BadTcpRead Modbus, 24 Status_BadTcpRead Modbus, 25 Status_Uncertain Modbus, 22 Status_Uncertain Modbus, 22 Status_Uncertain Modbus, 22 ModbusClientPort, 81 ModbusPort, 101	•	ModbusStrings, 137
Modbus, 23 Status_BadSerialReadTimeout Modbus, 23 Status_BadSerialReadTimeout Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWriteTimeout Modbus, 23 Status_BadServerDeviceBusy Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadSerperDeviceFailure Modbus, 23 Status_BadTcpAccept Modbus, 23 Status_BadTcpBind Modbus, 23 Status_BadTcpConnect Modbus, 23 Status_BadTcpCreate Modbus, 23 Status_BadTcpCreate Modbus, 23 Status_BadTcpDisconnect Modbus, 23 Status_BadTcpDisconnect Modbus, 23 Status_BadTcpListen Modbus, 23 Status_BadTcpListen Modbus, 23 Status_BadTcpListen Modbus, 23 Status_BadTcpCreate Modbus, 23 Status_BadTcpListen Modbus, 23 Status_BadTcpListen Modbus, 23 Status_BadTcpDisconnect Modbus, 23 Status_BadTcpListen Modbus, 23 Status_BadTcpDisconnect Modbus, 23 Status_BadTcpDrite Modbus, 23 Status_BadTcpWrite Modbus, 23 Status_BadTcpWrite Modbus, 23 Status_BadTcpWrite Modbus, 23 Status_BadTcpWrite Modbus, 23 Status_BadTcpRead Modbus, 23 Status_BadTcpWrite Modbus, 24 Modbus, 43 tries Modbus, 44 Modbus::Address, 49 tries Modbus, 25 Status_Ode Modbus, 22 Status_Uncertain Modbus, 22 Status_Uncertain Modbus, 22 ModbusAderes, 49 ModbusAderes, 49 ModbusAderes, 49 ModbusAderes, 49 ModbusClientPort, 81 ModbusPort, 101		TCP
Status_BadSerialReadTimeout Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWriteTimeout Modbus, 23 Status_BadSerialWriteTimeout Modbus, 23 Status_BadServerDeviceBusy Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadTcpAccept Modbus, 23 Status_BadTcpBind Modbus, 23 Status_BadTcpConnect Modbus, 23 Status_BadTcpConnect Modbus, 23 Status_BadTcpCreate Modbus, 23 Status_BadTcpDisconnect Modbus, 23 Status_BadTcpRead Modbus, 24 Status_BadTcpRead Modbus, 24 Status_BadTcpRead Modbus, 24 Status_BadTcpRead Modbus, 25 Status_Decessing Modbus, 22 Status_Decessin		
Status_BadSerialReadTimeout Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWriteTimeout Modbus, 23 Status_BadSerialWriteTimeout Modbus, 23 Status_BadServerDeviceBusy Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadTopAccept Modbus, 23 Status_BadTopDaccept Modbus, 23 Status_BadTopDaccept Modbus, 23 Status_BadTopConnect Modbus, 23 Status_BadTopCreate Modbus, 23 Status_BadTopDisconnect Modbus, 24 Status_BadTopDisconnect Modbus, 24 Status_BadTopDisconnect Modbus, 24 Status_BadTopDisconnect Modbus, 24 Status_BadTopDisconnect Modbus, 25 Status_Code Modbus, 22 ModbusClientPort, 81 ModbusPort, 101	_	
Modbus, 23 Status_BadSerialWrite Modbus, 23 Status_BadSerialWriteTimeout Modbus, 23 Status_BadServerDeviceBusy Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadTcpAccept Modbus, 23 Status_BadTcpAccept Modbus, 23 Status_BadTcpBind Modbus, 23 Status_BadTcpConnect Modbus, 23 Status_BadTcpConnect Modbus, 23 Status_BadTcpConnect Modbus, 23 Status_BadTcpConnect Modbus, 23 Status_BadTcpDisconnect Modbus, 23 Status_BadTcpListen Modbus, 23 Status_BadTcpRead Modbus, 23 Status_BadTcpConnect Modbus, 23 Status_BadTcpRead Modbus, 24 Modbus. 34 Modbu		ModbusPort, 100
Status_BadSerialWrite Modbus, 23 Status_BadSerialWriteTimeout Modbus, 23 Status_BadServerDeviceBusy Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadTopAccept Modbus, 23 Status_BadTopAccept Modbus, 23 Status_BadTopBind Modbus, 23 Status_BadTopCreate Modbus, 23 Status_BadTopCreate Modbus, 23 Status_BadTopDisconnect Modbus, 23 Status_BadTopBead Modbus, 23 Status_BadTopWrite Modbus, 23 Status_BadVriteBufferOverflow Modbus, 23 Status_BadVriteBufferOverflow Modbus, 23 Status_Good Modbus, 22 Status_Uncertain Modbus, 22 Status_Uncertain Modbus, 22 Status_Uncertain Modbus, 22 Status_Code Modbus, 22 StatusSad ModbusClientPort, 81 ModbusClientPort, 81 ModbusClientPort, 81 ModbusClientPort, 81 ModbusClientPort, 81 ModbusClientPort, 81	<del>-</del>	
Modbus, 23 Status_BadSerialWriteTimeout Modbus, 23 Status_BadServerDeviceBusy Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadTcpAccept toDataBits Modbus, 23 Status_BadTcpBind Modbus, 23 Status_BadTcpConnect Modbus, 23 Status_BadTcpCreate toModbus, 23 Status_BadTcpCreate Modbus, 23 Status_BadTcpDisconnect Modbus, 23 Status_BadTcpDisconnect toModbus, 23 Status_BadTcpDisconnect toPlowControl Modbus, 23 Status_BadTcpDisconnect toParity Modbus, 23 Status_BadTcpDead toStopBits Modbus, 43 Modbus, 23 Status_BadTcpWrite toString Modbus, 43 Status_BadTcpWrite toString Modbus, 44 Modbus, 23 Status_BadWriteBufferOverflow Modbus, 23 Status_Good Modbus, 23 Status_Good Modbus, 22 Status_Processing Modbus, 22 Status_Uncertain Modbus, 22 Status_Uncertain Modbus, 22 Status_Code Modbus, 22 StatusCode Modbus, 22 StatusCode Modbus, 22 StatusLocet Modbus, 22 S		•
Status_BadSerialWriteTimeout Modbus, 23  Status_BadServerDeviceBusy Modbus, 23  Status_BadServerDeviceFailure Modbus, 23  Status_BadServerDeviceFailure Modbus, 23  Status_BadTcpAccept Modbus, 23  Status_BadTcpBind Modbus, 23  Status_BadTcpConnect Modbus, 23  Status_BadTcpCreate Modbus, 23  Status_BadTcpDisconnect Modbus, 23  Status_BadTcpWrite Modbus, 23  Status_BadTcpWrite Modbus, 23  Status_BadTcpWrite Modbus, 23  Status_BadVriteBufferOverflow Modbus, 23  Status_BodWriteBufferOverflow Modbus, 23  Status_Good Modbus, 22  Status_Processing Modbus, 22  Status_Uncertain Modbus, 22  Status_Code Modbus, 22  StatusCode Modbus, 22  StatusCode Modbus, 22  StatusIsBad Modbus, 21  ModbusPort, 101	<del>-</del>	-
Modbus, 23 Status_BadServerDeviceBusy Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Modbus, 23 Modbus, 23 Status_BadTcpAccept Modbus, 23 Status_BadTcpBind Modbus, 23 Status_BadTcpConnect Modbus, 23 Status_BadTcpCreate Modbus, 23 Status_BadTcpDisconnect Modbus, 23 Status_BadTcpListen Modbus, 23 Status_BadTcpDisconnect Modbus, 23 Status_BadTcpCoverflow Modbus, 23 Status_BadTcpWrite Modbus, 23 Status_BadTcpWrite Modbus, 23 Status_BadWriteBufferOverflow Modbus, 23 Status_Good Modbus, 22 Status_Processing Modbus, 22 Status_Uncertain Modbus, 22 Status_Code Modbus, 22 Status_Code Modbus, 22 StatusCode Modbus, 22 Modbus, 22 StatusCode Modbus, 22 Modbus, 22 StatusCode Modbus, 22 ModbusClientPort, 81 ModbusPort, 101		
Status_BadServerDeviceBusy Modbus, 23 Status_BadServerDeviceFailure Modbus, 23 Status_BadTcpAccept Modbus, 23 Status_BadTcpAccept Modbus, 23 Status_BadTcpBind Modbus, 23 Status_BadTcpConnect Modbus, 23 Status_BadTcpCreate Modbus, 23 Status_BadTcpCreate Modbus, 23 Status_BadTcpDisconnect Modbus, 23 Status_BadTcpWrite Modbus, 23 Status_BadVwriteBufferOverflow Modbus, 24 Status_BadVwriteBufferOverflow Modbus, 25 Status_BadVwriteBufferOverflow Modbus, 26 Status_BadVwriteBufferOverflow Modbus, 26 S	<del>-</del>	ModbusSerialPort, 110
Modbus, 23  Status_BadServerDeviceFailure Modbus, 23  Status_BadTcpAccept Modbus, 23  Status_BadTcpBind Modbus, 23  Status_BadTcpConnect Modbus, 23  Status_BadTcpConnect Modbus, 23  Status_BadTcpCreate Modbus, 23  Status_BadTcpDisconnect Modbus, 23  Status_BadTcpRead Modbus, 23  Status_BadTcpRead Modbus, 23  Status_BadTcpWrite Modbus, 23  Status_BadTcpWrite Modbus, 23  Status_BadTcpWrite Modbus, 23  Status_BadTcpWrite Modbus, 23  Status_BadVriteBufferOverflow Modbus, 23  Status_BadVriteBufferOverflow Modbus, 25  Status_Good Modbus, 22  Status_Processing Modbus, 22  Status_Uncertain Modbus, 22  Status_Code Modbus, 22  StatusCode Modbus, 22  StatusCode Modbus, 22  StatusCode Modbus, 22  ModbusClientPort, 81 ModbusClientPort, 85 ModbusClientPort, 86 ModbusClientPort, 86 ModbusClientPort, 81 ModbusPort, 101		timer
Status_BadServerDeviceFailure Modbus, 23  Status_BadTcpAccept Modbus, 23  Status_BadTcpBind Modbus, 23  Status_BadTcpConnect Modbus, 23  Status_BadTcpConnect Modbus, 23  Status_BadTcpDisconnect Modbus, 23  Status_BadTcpDisconnect Modbus, 23  Status_BadTcpDisconnect Modbus, 23  Status_BadTcpDisconnect Modbus, 23  Status_BadTcpListen Modbus, 23  Status_BadTcpRead Modbus, 23  Status_BadTcpWrite Modbus, 23  Status_BadVriteBufferOverflow Modbus, 23  Status_BadVwriteBufferOverflow Modbus, 23  Status_BadVwriteBufferOverflow Modbus, 23  Status_Cood Modbus, 22  Status_Uncertain Modbus, 22  Status_Code Modbus, 22  StatusCode Modbus, 22  StatusSdad  ModbusClientPort, 81		Modbus, 41
Modbus, 23  Status_BadTcpAccept Modbus, 23  Status_BadTcpBind Modbus, 23  Status_BadTcpConnect Modbus, 23  Status_BadTcpCreate Modbus, 23  Status_BadTcpCreate Modbus, 23  Status_BadTcpDisconnect Modbus, 23  Status_BadTcpDisconnect Modbus, 23  Status_BadTcpListen Modbus, 23  Status_BadTcpRead Modbus, 23  Status_BadTcpWrite Modbus, 23  Status_BadVriteBufferOverflow Modbus, 23  Status_BadWriteBufferOverflow Modbus, 23  Status_BadWriteBufferOverflow Modbus, 23  Status_BadWriteBufferOverflow Modbus, 25  Status_Uncertain Modbus, 22  Status_Code Modbus, 22  StatusCode Modbus, 22  StatusSbad  ModbusClientPort, 81 ModbusClientPort, 56 ModbusClientPort, 81		toBaudRate
Status_BadTcpAccept Modbus, 23 Status_BadTcpBind Modbus, 23 Status_BadTcpConnect Modbus, 23 Status_BadTcpCeate Modbus, 23 Status_BadTcpCreate Modbus, 23 Status_BadTcpDisconnect Modbus, 23 Status_BadTcpListen Modbus, 23 Status_BadTcpRead Modbus, 23 Status_BadTcpRead Modbus, 23 Status_BadTcpWrite Modbus, 23 Status_BadVriteBufferOverflow Modbus, 23 Status_BadVriteBufferOverflow Modbus, 23 Status_BadWriteBufferOverflow Modbus, 23 Status_Good Modbus, 22 Status_Uncertain Modbus, 22 Status_Uncertain Modbus, 22 StatusCode Modbus, 22 Modbus, 22 Modbus, 22 StatusCode Modbus, 22 ModbusClientPort, 81 ModbusPort, 101	<del>-</del>	Modbus, 41
Modbus, 23 Status_BadTcpBind Modbus, 23 Status_BadTcpConnect Modbus, 23 Status_BadTcpConnect Modbus, 23 Status_BadTcpCreate Modbus, 23 Status_BadTcpDisconnect Modbus, 23 Status_BadTcpDisconnect Modbus, 23 Status_BadTcpListen Modbus, 23 Status_BadTcpRead Modbus, 23 Status_BadTcpWrite Modbus, 23 Status_BadVriteBufferOverflow Modbus, 23 Status_BadWriteBufferOverflow Modbus, 23 Status_BadWriteBufferOverflow Modbus, 23 Status_Good Modbus, 22 Status_Processing Modbus, 22 Status_Uncertain Modbus, 22 Status_Uncertain Modbus, 22 Status_Code Modbus, 22 StatusCode Modbus, 22 ModbusClientPort, 81 ModbusClientPort, 56 ModbusClientPort, 56 ModbusClientPort, 81 ModbusPort, 101		toDataBits
Status_BadTcpBind Modbus, 23 Status_BadTcpConnect Modbus, 23 Status_BadTcpCreate Modbus, 23 Status_BadTcpDreate Modbus, 23 Status_BadTcpDisconnect Modbus, 23 Status_BadTcpDisconnect Modbus, 23 Status_BadTcpListen Modbus, 23 Status_BadTcpRead Modbus, 23 Status_BadTcpWrite Modbus, 23 Status_BadTcpWrite Modbus, 23 Status_BadTcpWrite Modbus, 23 Status_BadTcpWrite Modbus, 23 Status_BadWriteBufferOverflow Modbus, 23 Status_BadWriteBufferOverflow Modbus, 23 Status_BadWriteBufferOverflow Modbus, 23 Status_BadWriteBufferOverflow Modbus, 23 Status_Good Modbus, 22 Status_Processing Modbus, 22 Status_Uncertain Modbus, 22 Status_Uncertain Modbus, 22 StatusCode Modbus, 22 StatusScode Modbus, 22 ModbusClient, 62 ModbusClientPort, 81 ModbusPort, 101	<del>-</del> · ·	Modbus, 42
Modbus, 23  Status_BadTcpConnect Modbus, 23  Status_BadTcpCreate Modbus, 23  Status_BadTcpDisconnect Modbus, 23  Status_BadTcpDisconnect Modbus, 23  Status_BadTcpDisconnect Modbus, 23  Status_BadTcpListen Modbus, 23  Status_BadTcpRead Modbus, 23  Status_BadTcpWrite Modbus, 23  Status_BadTcpWrite Modbus, 23  Status_BadTcpWrite Modbus, 23  Status_BadWriteBufferOverflow Modbus, 23  Status_BadWriteBufferOverflow Modbus, 23  Status_Cood Modbus, 22  Status_Processing Modbus, 22  Status_Uncertain Modbus, 22  StatusCode Modbus, 22  StatusCode Modbus, 22  StatusIsBad  Modbus, 22  StatusSBad  Modbus, 22  StatusCode Modbus, 22  StatusIsBad  Modbus, 22  StatusIsBad  ModbusPort, 101		toFlowControl
Status_BadTcpConnect Modbus, 23 Status_BadTcpCreate Modbus, 23 Status_BadTcpDisconnect Modbus, 23 Status_BadTcpDisconnect Modbus, 23 Status_BadTcpListen Modbus, 23 Status_BadTcpRead Modbus, 23 Status_BadTcpWrite Modbus, 23 Status_BadTcpWrite Modbus, 23 Status_BadVriteBufferOverflow Modbus, 23 Status_BadWriteBufferOverflow Modbus, 23 Status_BadVriteBufferOverflow Modbus, 23 Status_BadVriteBufferOverflow Modbus, 23 Status_Good Modbus, 22 Status_Processing Modbus, 22 Status_Uncertain Modbus, 22 StatusCode Modbus, 22 StatusCode Modbus, 22 StatusIbBad Modbus, 22 StatusIbBad Modbus, 22 StatusIbBad Modbus, 22 ModbusClientPort, 81 ModbusClient, 62 ModbusClientPort, 81 ModbusPort, 101	<del>-</del> ·	Modbus, 42
Modbus, 23  Status_BadTcpCreate     Modbus, 23  Status_BadTcpDisconnect     Modbus, 23  Status_BadTcpListen     Modbus, 23  Status_BadTcpRead     Modbus, 23  Status_BadTcpWrite     Modbus, 23  Status_BadVriteBufferOverflow     Modbus, 23  Status_BadWriteBufferOverflow     Modbus, 23  Status_BadWriteBufferOverflow     Modbus, 23  Status_Cood     Modbus, 25  Status_Processing     Modbus, 22  Status_Uncertain     Modbus, 22  Status_Code     Modbus, 22  StatusCode     Modbus, 22  StatusCode     Modbus, 22  StatusCode     Modbus, 22  StatusSBad  Modbus, 22  StatusCode     Modbus, 22  StatusSBad  ModbusClientPort, 81  ModbusClient, 62  ModbusClientPort, 81  ModbusPort, 101		toModbusOffset
Status_BadTcpCreate Modbus, 23  Status_BadTcpDisconnect Modbus, 23  Status_BadTcpListen Modbus, 23  Status_BadTcpRead Modbus, 23  Status_BadTcpWrite Modbus, 23  Status_BadTcpWrite Modbus, 23  Status_BadTcpWrite Modbus, 23  Status_BadVriteBufferOverflow Modbus, 23  Status_BadVriteBufferOverflow Modbus, 23  Status_Good Modbus, 22  Status_Processing Modbus, 22  Status_Uncertain Modbus, 22  Status_Code Modbus, 22  StatusCode Modbus, 22  StatusCode Modbus, 22  StatusSad ModbusClientPort, 81 ModbusPort, 101		Modbus, 42
Modbus, 23  Status_BadTcpDisconnect     Modbus, 23  Status_BadTcpListen     Modbus, 23  Status_BadTcpRead     Modbus, 23  Status_BadTcpWrite     Modbus, 23  Status_BadVriteBufferOverflow     Modbus, 23  Status_Good     Modbus, 22  Status_Processing     Modbus, 22  Status_Uncertain     Modbus, 22  StatusCode     Modbus, 22  StatusCode     Modbus, 22  StatusSead  Modbus, 22  StatusCode     Modbus, 22  StatusCode     Modbus, 22  StatusSead  Modbus, 22  StatusCode     Modbus, 22  StatusCode     Modbus, 22  StatusSead  ModbusClientPort, 81  ModbusClientPort, 56  ModbusClientPort, 81  ModbusPort, 101		toModbusString
Status_BadTcpDisconnect    Modbus, 23  Status_BadTcpListen    Modbus, 23  Status_BadTcpRead    Modbus, 23  Status_BadTcpWrite    Modbus, 23  Status_BadTcpWrite    Modbus, 23  Status_BadVriteBufferOverflow    Modbus, 23  Status_Good    Modbus, 22  Status_Processing    Modbus, 22  Status_Uncertain    Modbus, 22  StatusCode    Modbus, 22  StatusCode    Modbus, 22  StatusCode    Modbus, 22  StatusSode    Modbus, 22  StatusSode    Modbus, 22  StatusCode    Modbus, 22  StatusSode    ModbusClientPort, 81    ModbusClientPort, 81    ModbusClientPort, 81    ModbusClientPort, 81    ModbusClientPort, 81    ModbusPort, 101		Modbus, 42
Modbus, 23 Status_BadTcpListen Modbus, 23 Status_BadTcpRead Modbus, 23 Status_BadTcpWrite Modbus, 23 Status_BadTcpWrite Modbus, 23 Status_BadWriteBufferOverflow Modbus, 23 Status_Good Modbus, 22 Status_Processing Modbus, 22 Status_Uncertain Modbus, 22 Status_Code Modbus, 22 StatusCode Modbus, 22 StatusScode Modbus, 22 StatusSad Modbus, 22 StatusScode Modbus, 22 StatusSad Modbus, 22 StatusSad Modbus, 22 StatusCode Modbus, 22 StatusSad ModbusPort, 101		•
Modbus, 23 Status_BadTcpRead Modbus, 23 Status_BadTcpWrite Modbus, 23 Status_BadWriteBufferOverflow Modbus, 23 Status_Good Modbus, 22 Status_Processing Modbus, 22 Status_Uncertain Modbus, 22 Status_Uncertain Modbus, 22 StatusCode Modbus, 22 StatusCode Modbus, 22 StatusSead Modbus, 22 StatusSead Modbus, 22 StatusCode Modbus, 22 StatusCode Modbus, 22 StatusSead Modbus, 22 StatusCode Modbus, 22 Modbus, 22 StatusCode Modbus, 22 ModbusClientPort, 81 ModbusClient, 62 ModbusClientPort, 81 ModbusPort, 101		Modbus, 43
Status_BadTcpRead Modbus, 23 Status_BadTcpWrite Modbus, 23 Status_BadWriteBufferOverflow Modbus, 23 Status_Good Modbus, 22 Status_Processing Modbus, 22 Status_Uncertain Modbus, 22 Status_Uncertain Modbus, 22 Status_Code Modbus, 22 StatusCode Modbus, 22 StatusCode Modbus, 22 StatusSead Modbus, 22 StatusSead Modbus, 22 StatusCode Modbus, 22 StatusSead Modbus, 22 StatusSead Modbus, 22 Modbus, 22 Modbus, 22 Modbus, 22 Modbus, 21 Modbus, 22 Modbus, 22 Modbus, 22 Modbus, 22 Modbus, 22 Modbus, 21 Modbus, 22 Modbus, 22 Modbus, 22 Modbus, 21 Modbus, 22 Modbus, 22 Modbus, 22 Modbus, 21 Modbus, 22 Modbus, 22 Modbus, 21 Modbus, 22 Modbus, 21 Modbus, 22 Modbus, 21	Status_BadTcpListen	
Modbus, 23  Status_BadTcpWrite     Modbus, 23  Status_BadWriteBufferOverflow     Modbus, 23  Status_Good     Modbus, 22  Status_Processing     Modbus, 22  Status_Uncertain     Modbus, 22  Status_Code     Modbus, 22  StatusCode     Modbus, 22  StatusSad  Modbus, 22  StatusSad  Modbus, 22  Status_Uncertain     Modbus, 22  Status_Ode     Modbus, 22  StatusCode     Modbus, 22  StatusSad  ModbusPort, 101	Modbus, 23	
Status_BadTcpWrite Modbus, 23 Status_BadWriteBufferOverflow Modbus, 23 Status_Good Modbus, 22 Status_Processing Modbus, 22 Status_Uncertain Modbus, 22 Status_Uncertain Modbus, 22 Status_Code Modbus, 22 StatusCode Modbus, 22 StatusSode Modbus, 22 StatusSad Modbus, 22 StatusSad Modbus, 21 Modbus, 22 Modbus, 21 Modbus Modb	Status_BadTcpRead	-
Modbus, 23 Status_BadWriteBufferOverflow    Modbus, 23 Status_Good    Modbus, 22 Status_Processing    Modbus, 22 Status_Uncertain    Modbus, 22 Status_Code    Modbus, 22 StatusCode    Modbus, 22 StatusSead Modbus, 22 StatusSead Modbus, 22 StatusCode    Modbus, 22 StatusSead Modbus, 22 Modbus, 22 Modbus, 22 StatusCode    Modbus, 22 ModbusClientPort, 81 ModbusClientPort, 81 ModbusPort, 101	Modbus, 23	
Status_BadWriteBufferOverflow Modbus, 23 Status_Good Modbus, 22 Status_Processing Modbus, 22 Status_Uncertain Modbus, 22 Status_Uncertain Modbus, 22 StatusCode Modbus, 22 StatusCode Modbus, 22 StatusSead Modbus, 22 StatusSead Modbus, 22 Modbus, 21 ModbusAscPort, 56 ModbusClient, 62 ModbusClientPort, 81 ModbusPort, 101	Status_BadTcpWrite	S .
Modbus, 23  Status_Good	Modbus, 23	
Status_Good ModbusClientPort, 81  Modbus, 22  Status_Processing Modbus, 25  Modbus, 22  Status_Uncertain Modbus, 22  Status_Code Modbus, 22  StatusCode Modbus, 22  StatusSad ModbusPort, 101	Status_BadWriteBufferOverflow	•
Modbus, 22 Status_Processing Modbus, 22 Status_Uncertain Modbus, 22 StatusCode Modbus, 22 Modbus, 22 StatusCode Modbus, 22 StatusBad  TwoStop Modbus, 25 type  Modbus::Address, 49 ModbusAscPort, 56 ModbusAscPort, 56 ModbusClient, 62 ModbusClientPort, 81 ModbusPort, 101	Modbus, 23	
Status_Processing Modbus, 22 Status_Uncertain Modbus, 22 StatusCode Modbus, 22 Modbus, 22 StatusCode Modbus, 22 StatusBad ModbusAscPort, 56 ModbusClient, 62 ModbusClientPort, 81 ModbusPort, 101	Status_Good	
Modbus, 22 Status_Uncertain Modbus, 22 StatusCode Modbus, 22 StatusCode Modbus, 22 StatusBad  type  Modbus::Address, 49 ModbusAscPort, 56 ModbusClient, 62 ModbusClientPort, 81 ModbusPort, 101	Modbus, 22	-
Status_Uncertain Modbus; 22 StatusCode Modbus, 22 Modbus, 22 Modbus, 22 Modbus, 22 StatuslsBad ModbusClient, 62 ModbusClientPort, 81 ModbusPort, 101	Status_Processing	
Modbus, 22 StatusCode Modbus, 22 Modbus, 22 Modbus, 22 Modbus, 22 Modbus, 21 ModbusClient, 62 ModbusClientPort, 81 ModbusPort, 101	Modbus, 22	
StatusCode ModbusClient, 62 Modbus, 22 ModbusClientPort, 81 StatusIsBad ModbusPort, 101	Status_Uncertain	
Modbus, 22 ModbusClientPort, 81 StatusIsBad ModbusPort, 101	Modbus, 22	
StatusIsBad ModbusPort, 101	StatusCode	
Olalasisbaa	Modbus, 22	
ModbusHtuPort, 104	StatusIsBad	
		เขเบนอนรหเนศอก, 104

```
ModbusServerPort, 115
    ModbusServerResource, 119
    ModbusTcpPort, 128
    ModbusTcpServer, 135
unit
    ModbusClient, 63
unitMap
    ModbusServerPort, 115
VALID_MODBUS_ADDRESS_BEGIN
    Modbus, 21
VALID_MODBUS_ADDRESS_END
    Modbus, 21
write
    ModbusPort, 101
    ModbusSerialPort, 110
    ModbusTcpPort, 128
writeBuffer
    ModbusAscPort, 56
    ModbusPort, 101
    ModbusRtuPort, 104
    ModbusTcpPort, 129
writeBufferData
    ModbusPort, 101
    ModbusSerialPort, 110
    ModbusTcpPort, 129
writeBufferSize
    ModbusPort, 101
    ModbusSerialPort, 110
    ModbusTcpPort, 129
writeMemBits
    Modbus, 45
writeMemRegs
    Modbus, 45
writeMultipleCoils
    ModbusClient, 63
    ModbusClientPort, 81
    ModbusInterface, 91
writeMultipleCoilsAsBoolArray
    ModbusClient, 63
    ModbusClientPort, 82
writeMultipleRegisters
    ModbusClient, 63
    ModbusClientPort, 82, 83
    ModbusInterface, 91
writeSingleCoil
    ModbusClient, 63
    ModbusClientPort, 83
    ModbusInterface, 92
writeSingleRegister
    ModbusClient, 63
    ModbusClientPort, 84
```

ModbusInterface, 92