Generated by Doxygen 1.10.0

1 ModbusLib	1
1.0.1 Overview	1
1.0.2 Using Library	1
1.0.2.1 Common usage (C++)	1
1.0.2.2 Using with C	4
1.0.2.3 Using with Qt	4
1.0.3 Examples	4
1.0.3.1 democlient	5
1.0.3.2 mbclient	5
1.0.3.3 demoserver	5
1.0.3.4 mbserver	5
1.0.4 Tests	5
1.0.5 Documenations	5
1.0.6 Building	6
1.0.6.1 Build using CMake	6
1.0.6.2 Build using qmake	6
2 Namespace Index	9
2.1 Namespace List	9
3 Hierarchical Index	11
3.1 Class Hierarchy	11
4 Class Index	13
4.1 Class List	13
5 File lades	45
5 File Index	15
5.1 File List	15
6 Namespace Documentation	17
6.1 Modbus Namespace Reference	17
6.1.1 Detailed Description	20
6.1.2 Enumeration Type Documentation	20
6.1.2.1 _MemoryType	20
6.1.2.2 Constants	20
6.1.2.3 FlowControl	20
6.1.2.4 Parity	21
6.1.2.5 ProtocolType	21
6.1.2.6 StatusCode	21
6.1.2.7 StopBits	22
6.1.3 Function Documentation	23
6.1.3.1 addressFromString()	23
6.1.3.2 asciiToBytes()	23
6.1.3.3 asciiToString()	23

6.1.3.4 availableSerialPortList()
6.1.3.5 availableSerialPorts()
6.1.3.6 bytesToAscii()
6.1.3.7 bytesToString()
6.1.3.8 crc16()
6.1.3.9 createClientPort() [1/2] 2
6.1.3.10 createClientPort() [2/2]
6.1.3.11 createPort() [1/2]
6.1.3.12 createPort() [2/2]
6.1.3.13 createServerPort() [1/2]
6.1.3.14 createServerPort() [2/2]
6.1.3.15 enumKey() [1/2]
6.1.3.16 enumKey() [2/2]
6.1.3.17 enumValue() [1/4]
6.1.3.18 enumValue() [2/4]
6.1.3.19 enumValue() [3/4]
6.1.3.20 enumValue() [4/4]
6.1.3.21 getBit()
6.1.3.22 getBits()
6.1.3.23 getBitS()
6.1.3.24 getBitsS()
6.1.3.25 lrc()
6.1.3.26 msleep()
6.1.3.27 readMemBits()
6.1.3.28 readMemRegs()
6.1.3.29 sascii()
6.1.3.30 sbytes()
6.1.3.31 setBit()
6.1.3.32 setBitS()
6.1.3.33 setBits()
6.1.3.34 setBitsS()
6.1.3.35 StatusIsBad()
6.1.3.36 StatusIsGood()
6.1.3.37 StatusIsProcessing()
6.1.3.38 StatusIsStandardError()
6.1.3.39 StatusIsUncertain()
6.1.3.40 timer()
6.1.3.41 toFlowControl() [1/2]
6.1.3.42 toFlowControl() [2/2]
6.1.3.43 toModbusString()
6.1.3.44 toParity() [1/2]
6.1.3.45 toParity() [2/2]

6.1.3.46 toProtocolType() [1/2]	 . 33
<b>6.1.3.47 toProtocolType()</b> [2/2]	 . 33
<b>6.1.3.48 toStopBits()</b> [1/2]	 . 33
<b>6.1.3.49 toStopBits()</b> [2/2]	 . 33
<b>6.1.3.50 toString()</b> [1/5]	 . 34
<b>6.1.3.51 toString()</b> [2/5]	 . 34
<b>6.1.3.52 toString()</b> [3/5]	 . 34
<b>6.1.3.53 toString()</b> [4/5]	 . 34
<b>6.1.3.54 toString()</b> [5/5]	 . 34
6.1.3.55 writeMemBits()	 . 34
6.1.3.56 writeMemRegs()	 . 35
7 Class Documentation	37
7.1 Modbus::Address Class Reference	 . 37
7.1.1 Detailed Description	 . 37
7.1.2 Constructor & Destructor Documentation	 . 37
7.1.2.1 Address() [1/3]	 . 37
7.1.2.2 Address() [2/3]	 . 38
<b>7.1.2.3 Address()</b> [3/3]	 . 38
7.1.3 Member Function Documentation	 . 38
7.1.3.1 isValid()	 . 38
7.1.3.2 number()	 . 38
7.1.3.3 offset()	 . 38
7.1.3.4 operator quint32()	 . 38
7.1.3.5 operator=()	 . 39
7.1.3.6 toString()	 . 39
7.1.3.7 type()	 . 39
7.2 Modbus::Defaults Class Reference	 . 39
7.2.1 Detailed Description	 . 40
7.2.2 Constructor & Destructor Documentation	 . 40
7.2.2.1 Defaults()	 . 40
7.2.3 Member Function Documentation	 . 40
7.2.3.1 instance()	 . 40
7.3 ModbusSerialPort::Defaults Struct Reference	 . 41
7.3.1 Detailed Description	 . 41
7.3.2 Constructor & Destructor Documentation	 . 41
7.3.2.1 Defaults()	 . 41
7.3.3 Member Function Documentation	 . 42
7.3.3.1 instance()	 . 42
7.4 ModbusTcpPort::Defaults Struct Reference	 . 42
7.4.1 Detailed Description	 . 42
7.4.2 Constructor & Destructor Documentation	42

7.4.2.1 Defaults()	42
7.4.3 Member Function Documentation	43
7.4.3.1 instance()	43
7.5 ModbusTcpServer::Defaults Struct Reference	43
7.5.1 Detailed Description	43
7.5.2 Constructor & Destructor Documentation	43
7.5.2.1 Defaults()	43
7.5.3 Member Function Documentation	44
7.5.3.1 instance()	44
7.6 ModbusAscPort Class Reference	44
7.6.1 Detailed Description	45
7.6.2 Constructor & Destructor Documentation	46
7.6.2.1 ModbusAscPort()	46
7.6.2.2 ~ModbusAscPort()	46
7.6.3 Member Function Documentation	46
7.6.3.1 readBuffer()	46
7.6.3.2 type()	46
7.6.3.3 writeBuffer()	46
7.7 ModbusClient Class Reference	47
7.7.1 Detailed Description	48
7.7.2 Constructor & Destructor Documentation	48
7.7.2.1 ModbusClient()	48
7.7.3 Member Function Documentation	48
7.7.3.1 isOpen()	48
7.7.3.2 lastPortErrorStatus()	49
7.7.3.3 lastPortErrorText()	49
7.7.3.4 lastPortStatus()	49
7.7.3.5 port()	49
7.7.3.6 readCoils()	49
7.7.3.7 readCoilsAsBoolArray()	49
7.7.3.8 readDiscreteInputs()	50
7.7.3.9 readDiscreteInputsAsBoolArray()	50
7.7.3.10 readExceptionStatus()	50
7.7.3.11 readHoldingRegisters()	50
7.7.3.12 readInputRegisters()	50
7.7.3.13 setUnit()	51
7.7.3.14 type()	51
7.7.3.15 unit()	51
7.7.3.16 writeMultipleCoils()	51
7.7.3.17 writeMultipleCoilsAsBoolArray()	51
7.7.3.18 writeMultipleRegisters()	51
7.7.3.19 writeSingleCoil()	52

7.7.3.20 writeSingleRegister()	52
7.8 ModbusClientPort Class Reference	52
7.8.1 Detailed Description	54
7.8.2 Constructor & Destructor Documentation	55
7.8.2.1 ModbusClientPort()	55
7.8.3 Member Function Documentation	55
7.8.3.1 cancelRequest()	55
7.8.3.2 close()	55
7.8.3.3 currentClient()	55
7.8.3.4 getRequestStatus()	56
7.8.3.5 isOpen()	56
7.8.3.6 lastErrorStatus()	56
7.8.3.7 lastErrorText()	56
7.8.3.8 lastStatus()	56
7.8.3.9 port()	56
7.8.3.10 readCoils() [1/2]	56
7.8.3.11 readCoils() [2/2]	57
7.8.3.12 readCoilsAsBoolArray() [1/2]	57
7.8.3.13 readCoilsAsBoolArray() [2/2]	57
7.8.3.14 readDiscreteInputs() [1/2]	58
7.8.3.15 readDiscreteInputs() [2/2]	58
7.8.3.16 readDiscreteInputsAsBoolArray() [1/2]	58
7.8.3.17 readDiscreteInputsAsBoolArray() [2/2]	59
7.8.3.18 readExceptionStatus() [1/2]	59
7.8.3.19 readExceptionStatus() [2/2]	59
7.8.3.20 readHoldingRegisters() [1/2]	59
7.8.3.21 readHoldingRegisters() [2/2]	60
7.8.3.22 readInputRegisters() [1/2]	60
7.8.3.23 readInputRegisters() [2/2]	60
7.8.3.24 repeatCount()	61
7.8.3.25 setRepeatCount()	61
7.8.3.26 signalClosed()	61
7.8.3.27 signalError()	61
7.8.3.28 signalOpened()	61
7.8.3.29 signalRx()	61
7.8.3.30 signalTx()	62
7.8.3.31 type()	62
7.8.3.32 writeMultipleCoils() [1/2]	62
7.8.3.33 writeMultipleCoils() [2/2]	62
7.8.3.34 writeMultipleCoilsAsBoolArray() [1/2]	63
7.8.3.35 writeMultipleCoilsAsBoolArray() [2/2]	63
7.8.3.36 writeMultipleRegisters() [1/2]	63

7.8.3.37 writeMultipleRegisters() [2/2]	63
7.8.3.38 writeSingleCoil() [1/2]	64
7.8.3.39 writeSingleCoil() [2/2]	64
7.8.3.40 writeSingleRegister() [1/2]	64
7.8.3.41 writeSingleRegister() [2/2]	65
7.9 ModbusInterface Class Reference	65
7.9.1 Detailed Description	66
7.9.2 Member Function Documentation	66
7.9.2.1 readCoils()	66
7.9.2.2 readDiscreteInputs()	66
7.9.2.3 readExceptionStatus()	67
7.9.2.4 readHoldingRegisters()	67
7.9.2.5 readInputRegisters()	68
7.9.2.6 writeMultipleCoils()	68
7.9.2.7 writeMultipleRegisters()	69
7.9.2.8 writeSingleCoil()	69
7.9.2.9 writeSingleRegister()	69
7.10 ModbusObject Class Reference	70
7.10.1 Detailed Description	71
7.10.2 Constructor & Destructor Documentation	71
7.10.2.1 ModbusObject()	71
7.10.2.2 ~ModbusObject()	71
7.10.3 Member Function Documentation	71
7.10.3.1 connect() [1/2]	71
7.10.3.2 connect() [2/2]	72
7.10.3.3 disconnect() [1/3]	72
7.10.3.4 disconnect() [2/3]	72
7.10.3.5 disconnect() [3/3]	72
7.10.3.6 disconnectFunc()	72
7.10.3.7 emitSignal()	73
7.10.3.8 objectName()	73
7.10.3.9 sender()	73
7.10.3.10 setObjectName()	73
7.11 ModbusPort Class Reference	73
7.11.1 Detailed Description	74
7.11.2 Constructor & Destructor Documentation	74
7.11.2.1 ~ModbusPort()	74
7.11.3 Member Function Documentation	75
7.11.3.1 close()	75
7.11.3.2 handle()	75
7.11.3.3 isBlocking()	75
7.11.3.4 isChanged()	75

7.11.3.5 isNonBlocking()	 75
7.11.3.6 isOpen()	 75
7.11.3.7 isServerMode()	 76
7.11.3.8 lastErrorStatus()	 76
7.11.3.9 lastErrorText()	 76
7.11.3.10 open()	 76
7.11.3.11 read()	 76
7.11.3.12 readBuffer()	 76
7.11.3.13 readBufferData()	 77
7.11.3.14 readBufferSize()	 77
7.11.3.15 setError()	 77
7.11.3.16 setNextRequestRepeated()	 77
7.11.3.17 setServerMode()	 77
7.11.3.18 type()	 77
7.11.3.19 write()	 78
7.11.3.20 writeBuffer()	 78
7.11.3.21 writeBufferData()	 78
7.11.3.22 writeBufferSize()	 78
7.12 ModbusRtuPort Class Reference	 78
7.12.1 Detailed Description	 80
7.12.2 Constructor & Destructor Documentation	 80
7.12.2.1 ModbusRtuPort()	 80
7.12.2.2 ~ ModbusRtuPort()	 80
7.12.3 Member Function Documentation	 80
7.12.3.1 readBuffer()	 80
7.12.3.2 type()	 81
7.12.3.3 writeBuffer()	 81
7.13 ModbusSerialPort Class Reference	 81
7.13.1 Detailed Description	 83
7.13.2 Member Function Documentation	 83
7.13.2.1 baudRate()	 83
7.13.2.2 close()	 83
7.13.2.3 dataBits()	 83
7.13.2.4 flowControl()	 83
7.13.2.5 handle()	 83
7.13.2.6 isOpen()	 84
7.13.2.7 open()	 84
7.13.2.8 parity()	 84
7.13.2.9 portName()	 84
7.13.2.10 read()	 84
7.13.2.11 readBufferData()	 84
7.13.2.12 readBufferSize()	 85

7.13.2.13 setBaudRate()	 	85
7.13.2.14 setDataBits()		
7.13.2.15 setFlowControl()		
7.13.2.16 setParity()		
7.13.2.17 setPortName()		
7.13.2.18 setStopBits()	 	85
7.13.2.19 setTimeoutFirstByte()	 	86
7.13.2.20 setTimeoutInterByte()	 	86
7.13.2.21 stopBits()	 	86
7.13.2.22 timeoutFirstByte()	 	86
7.13.2.23 timeoutInterByte()	 	86
7.13.2.24 write()	 	86
7.13.2.25 writeBufferData()	 	86
7.13.2.26 writeBufferSize()	 	87
7.14 ModbusServerPort Class Reference	 	87
7.14.1 Detailed Description	 	88
7.14.2 Member Function Documentation	 	88
7.14.2.1 close()	 	88
7.14.2.2 device()	 	89
7.14.2.3 isOpen()	 	89
7.14.2.4 isStateClosed()	 	89
7.14.2.5 isTcpServer()	 	89
7.14.2.6 ModbusObject()	 	89
7.14.2.7 open()	 	89
7.14.2.8 process()	 	90
7.14.2.9 signalClosed()	 	90
7.14.2.10 signalError()	 	90
7.14.2.11 signalOpened()	 	90
7.14.2.12 signalRx()	 	90
7.14.2.13 signalTx()	 	90
7.14.2.14 type()	 	91
7.15 ModbusServerResource Class Reference	 	91
7.15.1 Detailed Description	 	92
7.15.2 Constructor & Destructor Documentation	 	93
7.15.2.1 ModbusServerResource()	 	93
7.15.3 Member Function Documentation	 	93
7.15.3.1 close()	 	93
7.15.3.2 isOpen()	 	93
7.15.3.3 open()	 	93
7.15.3.4 port()	 	93
7.15.3.5 process()	 	94
7.15.3.6 processDevice()	 	94

7.15.3.7 processInputData()	94
7.15.3.8 processOutputData()	94
7.15.3.9 type()	94
7.16 ModbusSlotBase< ReturnType, Args > Class Template Reference	94
7.16.1 Detailed Description	95
7.16.2 Constructor & Destructor Documentation	95
7.16.2.1 ~ModbusSlotBase()	95
7.16.3 Member Function Documentation	95
7.16.3.1 exec()	95
7.16.3.2 methodOrFunction()	95
7.16.3.3 object()	96
7.17 ModbusSlotFunction< ReturnType, Args > Class Template Reference	96
7.17.1 Detailed Description	96
7.17.2 Constructor & Destructor Documentation	96
7.17.2.1 ModbusSlotFunction()	96
7.17.3 Member Function Documentation	97
7.17.3.1 exec()	97
7.17.3.2 methodOrFunction()	97
7.18 ModbusSlotMethod< T, ReturnType, Args > Class Template Reference	97
7.18.1 Detailed Description	98
7.18.2 Constructor & Destructor Documentation	98
7.18.2.1 ModbusSlotMethod()	98
7.18.3 Member Function Documentation	98
7.18.3.1 exec()	98
7.18.3.2 methodOrFunction()	99
7.18.3.3 object()	99
7.19 ModbusTcpPort Class Reference	99
7.19.1 Detailed Description	100
7.19.2 Constructor & Destructor Documentation	101
7.19.2.1 ModbusTcpPort() [1/2]	101
7.19.2.2 ModbusTcpPort() [2/2]	101
7.19.3 Member Function Documentation	101
7.19.3.1 autoIncrement()	101
7.19.3.2 close()	101
7.19.3.3 handle()	101
7.19.3.4 host()	101
7.19.3.5 isOpen()	102
7.19.3.6 open()	102
7.19.3.7 port()	102
7.19.3.8 read()	102
7.19.3.9 readBuffer()	102
7.19.3.10 readBufferData()	102

7.19.3.11 readBufferSize()	103
7.19.3.12 setHost()	103
7.19.3.13 setNextRequestRepeated()	103
7.19.3.14 setPort()	103
7.19.3.15 setTimeout()	103
7.19.3.16 timeout()	103
7.19.3.17 type()	104
7.19.3.18 write()	104
7.19.3.19 writeBuffer()	104
7.19.3.20 writeBufferData()	104
7.19.3.21 writeBufferSize()	104
7.20 ModbusTcpServer Class Reference	105
7.20.1 Detailed Description	106
7.20.2 Constructor & Destructor Documentation	107
7.20.2.1 ModbusTcpServer()	107
7.20.3 Member Function Documentation	107
7.20.3.1 clearConnections()	107
7.20.3.2 close()	107
7.20.3.3 createTcpPort()	107
7.20.3.4 isOpen()	107
7.20.3.5 isTcpServer()	108
7.20.3.6 nextPendingConnection()	108
7.20.3.7 open()	108
7.20.3.8 port()	108
7.20.3.9 process()	108
7.20.3.10 setPort()	109
7.20.3.11 setTimeout()	109
7.20.3.12 signalCloseConnection()	109
7.20.3.13 signalNewConnection()	109
7.20.3.14 timeout()	109
7.20.3.15 type()	109
7.21 Modbus::SerialSettings Struct Reference	110
7.21.1 Detailed Description	110
7.22 Modbus::Strings Class Reference	110
7.22.1 Detailed Description	111
7.22.2 Constructor & Destructor Documentation	111
7.22.2.1 Strings()	111
7.22.3 Member Function Documentation	111
7.22.3.1 instance()	111
7.23 Modbus::TcpSettings Struct Reference	112
7.23.1 Detailed Description	112

8 File Documentation	113
8.1 c:/Users/march/Dropbox/PRJ/ModbusLib/src/cModbus.h File Reference	113
8.1.1 Detailed Description	116
8.1.2 Typedef Documentation	116
8.1.2.1 pfReadCoils	116
8.1.2.2 pfReadDiscreteInputs	116
8.1.2.3 pfReadExceptionStatus	117
8.1.2.4 pfReadHoldingRegisters	117
8.1.2.5 pfReadInputRegisters	117
8.1.2.6 pfSlotCloseConnection	117
8.1.2.7 pfSlotClosed	117
8.1.2.8 pfSlotError	118
8.1.2.9 pfSlotNewConnection	118
8.1.2.10 pfSlotOpened	118
8.1.2.11 pfSlotRx	118
8.1.2.12 pfSlotTx	118
8.1.2.13 pfWriteMultipleCoils	119
8.1.2.14 pfWriteMultipleRegisters	119
8.1.2.15 pfWriteSingleCoil	119
8.1.2.16 pfWriteSingleRegister	119
8.1.3 Function Documentation	120
8.1.3.1 cCliCreate()	120
8.1.3.2 cCliCreateForClientPort()	120
8.1.3.3 cCliDelete()	120
8.1.3.4 cCliGetLastPortErrorStatus()	120
8.1.3.5 cCliGetLastPortErrorText()	120
8.1.3.6 cCliGetLastPortStatus()	121
8.1.3.7 cCliGetObjectName()	121
8.1.3.8 cCliGetPort()	121
8.1.3.9 cCliGetType()	121
8.1.3.10 cCliGetUnit()	121
8.1.3.11 cClilsOpen()	121
8.1.3.12 cCliSetObjectName()	121
8.1.3.13 cCliSetUnit()	122
8.1.3.14 cCpoClose()	122
8.1.3.15 cCpoConnectClosed()	122
8.1.3.16 cCpoConnectError()	122
8.1.3.17 cCpoConnectOpened()	122
8.1.3.18 cCpoConnectRx()	122
8.1.3.19 cCpoConnectTx()	123
8.1.3.20 cCpoCreate()	123
8.1.3.21 cCpoCreateForPort()	123

8.1.3.22 cCpoDelete()
8.1.3.23 cCpoDisconnectFunc()
8.1.3.24 cCpoGetLastErrorStatus()
8.1.3.25 cCpoGetLastErrorText()
8.1.3.26 cCpoGetLastStatus()
8.1.3.27 cCpoGetObjectName()
8.1.3.28 cCpoGetRepeatCount()
8.1.3.29 cCpoGetType()
8.1.3.30 cCpolsOpen()
8.1.3.31 cCpoReadCoils()
8.1.3.32 cCpoReadCoilsAsBoolArray()
8.1.3.33 cCpoReadDiscreteInputs()
8.1.3.34 cCpoReadDiscreteInputsAsBoolArray()
8.1.3.35 cCpoReadExceptionStatus()
8.1.3.36 cCpoReadHoldingRegisters()
8.1.3.37 cCpoReadInputRegisters()
8.1.3.38 cCpoSetObjectName()
8.1.3.39 cCpoSetRepeatCount()
8.1.3.40 cCpoWriteMultipleCoils()
8.1.3.41 cCpoWriteMultipleCoilsAsBoolArray()
8.1.3.42 cCpoWriteMultipleRegisters()
8.1.3.43 cCpoWriteSingleCoil()
8.1.3.44 cCpoWriteSingleRegister()
8.1.3.45 cCreateModbusDevice()
8.1.3.46 cDeleteModbusDevice()
8.1.3.47 cPortCreate()
8.1.3.48 cPortDelete()
8.1.3.49 cReadCoils()
8.1.3.50 cReadCoilsAsBoolArray()
8.1.3.51 cReadDiscreteInputs()
8.1.3.52 cReadDiscreteInputsAsBoolArray()
8.1.3.53 cReadExceptionStatus()
8.1.3.54 cReadHoldingRegisters()
8.1.3.55 cReadInputRegisters()
8.1.3.56 cSpoClose()
8.1.3.57 cSpoConnectCloseConnection()
8.1.3.58 cSpoConnectClosed()
8.1.3.59 cSpoConnectError()
8.1.3.60 cSpoConnectNewConnection()
8.1.3.61 cSpoConnectOpened()
8.1.3.62 cSpoConnectRx()
8.1.3.63 cSpoConnectTx()

8.1.3.64 cSpoCreate()
8.1.3.65 cSpoDelete()
8.1.3.66 cSpoDisconnectFunc()
8.1.3.67 cSpoGetDevice()
8.1.3.68 cSpoGetObjectName()
8.1.3.69 cSpoGetType()
8.1.3.70 cSpolsOpen()
8.1.3.71 cSpolsTcpServer()
8.1.3.72 cSpoOpen()
8.1.3.73 cSpoProcess()
8.1.3.74 cSpoSetObjectName()
8.1.3.75 cWriteMultipleCoils()
8.1.3.76 cWriteMultipleCoilsAsBoolArray()
8.1.3.77 cWriteMultipleRegisters()
8.1.3.78 cWriteSingleCoil()
8.1.3.79 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
8.12.2 Macro Definition Documentation
8.12.2.1 GET_BITS
8.12.2.2 MB_RTU_IO_BUFF_SZ
8.12.2.3 SET_BIT
8.12.2.4 SET_BITS
8.13 ModbusGlobal.h
8.14 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusObject.h File Reference
8.14.1 Detailed Description
8.15 ModbusObject.h
8.16 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusPlatform.h File Reference

	8.16.1 Detailed Description	157
	8.17 ModbusPlatform.h	158
	8.18 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusPort.h File Reference	158
	8.18.1 Detailed Description	158
	8.19 ModbusPort.h	159
	8.20 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusQt.h File Reference	160
	8.20.1 Detailed Description	161
	8.21 ModbusQt.h	161
	8.22 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusRtuPort.h File Reference	163
	8.22.1 Detailed Description	164
	8.23 ModbusRtuPort.h	164
	8.24 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusSerialPort.h File Reference	164
	8.24.1 Detailed Description	165
	8.25 ModbusSerialPort.h	165
	8.26 ModbusServerPort.h	166
	8.27 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusServerResource.h File Reference	167
	8.27.1 Detailed Description	167
	8.28 ModbusServerResource.h	167
	8.29 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusTcpPort.h File Reference	168
	8.29.1 Detailed Description	168
	8.30 ModbusTcpPort.h	168
	8.31 c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusTcpServer.h File Reference	169
	8.31.1 Detailed Description	169
	8.32 ModbusTcpServer.h	170
L	da	474
ın	dex	171

## **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
- 15 (0x0F) WRITE\_MULTIPLE\_COILS
- 16 (0x10) WRITE\_MULTIPLE\_REGISTERS

### 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.

#### Client

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

```
#include <ModbusClientPort.h>
void main()
    Modbus::TcpSettings settings;
    settings.host = "someadr.plc";
settings.port = 502;
    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::StatusIsGood(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.

#### 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:

```
{
            memcpy(values, mem4x, count*sizeof(uint16_t));
            return Modbus::Status_Good;
        return Modbus::Status BadIllegalDataAddress;
};
void main()
    MvModbusDevice device:
    Modbus::TcpSettings settings;
    settings.port = 502;
    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 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.

#### 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)) {
    // ...
}
//...
```

### 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 order which it 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);
      //...
}
```

### 1.0.2.2 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 = 502;
    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);
```

### 1.0.2.3 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

democlient 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 -help

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:

5./mbserver -?

```
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
$ 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):

```
>cmake -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
```

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.5.15.3: /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: /usr/lib/x86_64-linux-gnu/libQt5Help.so.5.15
```

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.  $\sim$ /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:

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

10. Finaly to make current set of programs print:

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 with Modbus-	
protocol				17

10 Namespace Index

## **Hierarchical Index**

## 3.1 Class Hierarchy

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

Modbus::Address
Modbus::Defaults
ModbusSerialPort::Defaults
ModbusTcpPort::Defaults
ModbusTcpServer::Defaults
ModbusInterface
ModbusClientPort
ModbusObject
ModbusClient
ModbusClientPort
ModbusServerPort
ModbusServerResource
ModbusTcpServer
ModbusPort
ModbusSerialPort
ModbusAscPort
ModbusRtuPort
ModbusTcpPort
ModbusSlotBase < ReturnType, Args >
ModbusSlotBase < ReturnType, Args >
ModbusSlotFunction< ReturnType, Args >
ModbusSlotMethod< T, ReturnType, Args >
Modbus::SerialSettings
Modbus::Strings
Modbus::TcpSettings

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	37
Modbus::Defaults	
Holds the default values of the settings	39
ModbusSerialPort::Defaults	
Holds the default values of the settings	41
ModbusTcpPort::Defaults	
Defaults class constain default settings values for ModbusTcpPort	42
ModbusTcpServer::Defaults	
Defaults class constain default settings values for ModbusTcpServer	43
ModbusAscPort	
Implements ASCII version of the Modbus communication protocol	44
ModbusClient	
The ModbusClient class implements the interface of the client part of the Modbus protocol	47
ModbusClientPort	
The ModbusClientPort class implements the algorithm of the client part of the Modbus	
communication protocol port	52
ModbusInterface	
Main interface of Modbus communication protocol	65
ModbusObject	
The ModbusObject class is the base class for objects that use signal/slot mechanism	70
ModbusPort	
The abstract class ModbusPort is the base class for a specific implementation of the Modbus	
communication protocol	73
ModbusRtuPort	
Implements RTU version of the Modbus communication protocol	78
ModbusSerialPort	
The abstract class ModbusSerialPort is the base class serial port Modbus communications	81
ModbusServerPort	
Abstract base class for direct control of ModbusPort derived classes (TCP or serial) for server	
side	87
ModbusServerResource	
Implements direct control for ModbusPort derived classes (TCP or serial) for server side	91
ModbusSlotBase < ReturnType, Args >	
ModbusSlotBase base template for slot (method or function)	94

14 Class Index

ModbusSlotFunction < ReturnType, Args >	
ModbusSlotFunction template class hold pointer to slot function	96
ModbusSlotMethod< T, ReturnType, Args >	
ModbusSlotMethod template class hold pointer to object and its method	97
ModbusTcpPort	
Class ModbusTcpPort implements TCP version of Modbus protocol	99
ModbusTcpServer	
The ModbusTcpServer class implements TCP server part of the Modbus protocol	105
Modbus::SerialSettings	
Struct to define settings for Serial Port	110
Modbus::Strings	
Sets constant key values for the map of settings	110
Modbus::TcpSettings	
Struct to define settings for TCP connection	112

# 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	113
c:/Users/march/Dropbox/PRJ/ModbusLib/src/Modbus.h	
Contains general definitions of the Modbus protocol	137
c:/Users/march/Dropbox/PRJ/ModbusLib/src/Modbus_config.h	140
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusAscPort.h	
Contains definition of ASCII serial port class	140
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusClient.h	
Header file of Modbus client	141
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusClientPort.h	
General file of the algorithm of the client part of the Modbus protocol port	142
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusGlobal.h	
Contains general definitions of the Modbus libarary (for C++ and "pure" C)	144
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusObject.h	
The header file defines the class templates used to create signal/slot-like mechanism	154
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusPlatform.h	
Definition of platform specific macros	157
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusPort.h	
Header file of abstract class ModbusPort	158
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusQt.h	
Qt support file for ModbusLib	160
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusRtuPort.h	
Contains definition of RTU serial port class	163
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusSerialPort.h	
Contains definition of base serial port class	164
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusServerPort.h	166
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusServerResource.h	
The header file defines the class that controls specific port	167
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusTcpPort.h	
Header file of class Modbus TcpPort	168
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusTcpServer.h	
Header file of Modhus TCP server	169

16 File Index

## **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 enum Modbus::\_MemoryType MemoryType

Defines type of memory used in Modbus protocol.

typedef QHash< QString, QVariant > Settings

 $\textit{Map for settings of $Modbus$ protocol where key has type $\it QString$ and value is $\it QVariant.$$ 

#### **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 BadSerialOpen = Status Bad | 0x201,
 Status BadSerialWrite , Status BadSerialRead , 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**

- 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 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)
- 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 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)
- MODBUS\_EXPORT StatusCode writeMemRegs (uint32\_t offset, uint32\_t count, const void \*values, void \*memBuff, uint32\_t memRegCount)
- MODBUS\_EXPORT StatusCode readMemBits (uint32\_t offset, uint32\_t count, void \*values, const void \*memBuff, uint32\_t memBitCount)
- MODBUS\_EXPORT StatusCode writeMemBits (uint32\_t offset, uint32\_t count, const void \*values, void \*memBuff, uint32\_t memBitCount)
- 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 Timer timer ()
- MODBUS\_EXPORT void msleep (uint32\_t msec)
- Address addressFromString (const QString &s)
- template < class EnumType >

QString enumKey (int value)

template < class EnumType >

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

 $\bullet \ \ \mathsf{template} \mathord{<} \mathsf{class} \ \mathsf{EnumType} \mathord{>}$ 

EnumType enumValue (const QString &key, bool \*ok=nullptr)

template < class EnumType >

EnumType enumValue (const QVariant &value, bool \*ok)

 $\bullet \ \ \mathsf{template} {<} \mathsf{class} \ \mathsf{EnumType} >$ 

EnumType enumValue (const QVariant &value, EnumType defaultValue)

 $\bullet \ \ \text{template}{<} \text{class EnumType} >$ 

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 Parity to Parity (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)
- 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.
Status_BadIllegalFunction	Standard error. The feature is not supported.
Status_BadIllegalDataAddress	Standard error. Invalid data address.
Status_BadIllegalDataValue	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.

# Enumerator

	tandard error. The server is busy processing a long ommand. The request must be repeated later.
	tandard error. The programming function cannot be erformed.
file	tandard error. The server attempted to read a record e but detected a parity error in memory.
to inp Us	tandard error. Indicates that the gateway was unable allocate an internal communication path from the put port o the output port for processing the request. sually means that the gateway is misconfigured or verloaded.
ob	tandard error. Indicates that no response was otained from the target device. Usually means that e device is not present on the network.
Status_BadEmptyResponse Er	rror. Empty request/response body.
Status_BadNotCorrectRequest Er	rror. Invalid request.
Status_BadNotCorrectResponse Er	rror. Invalid response.
Status_BadWriteBufferOverflow Er	rror. Write buffer overflow.
Status_BadReadBufferOverflow Er	rror. Request receive buffer overflow.
Status_BadSerialOpen Er	rror. Serial port cannot be opened.
Status_BadSerialWrite Er	rror. Cannot send a parcel to the serial port.
Status_BadSerialRead Er	rror. Reading the serial port (timeout)
Status_BadAscMissColon Er	rror (ASC). Missing packet start character ':'.
Status_BadAscMissCrLf Er	rror (ASC). '\r\n' end of packet character missing.
Status_BadAscChar Er	rror (ASC). Invalid ASCII character.
Status_BadLrc Er	rror (ASC). Invalid checksum.
Status_BadCrc Er	rror (RTU). Wrong checksum.
Status_BadTcpCreate Er	rror. Unable to create a TCP socket.
Status_BadTcpConnect Er	rror. Unable to create a TCP connection.
Status_BadTcpWrite Er	rror. Unable to send a TCP packet.
Status_BadTcpRead Er	rror. Unable to receive a TCP packet.
Status_BadTcpBind Er	rror. Unable to bind a TCP socket (server side)
Status_BadTcpListen Er	rror. Unable to listen a TCP socket (server side)
Status_BadTcpAccept Er	rror. Unable accept bind a TCP socket (server side)
Status_BadTcpDisconnect Er	rror. 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()

Make string representation of ASCII array and separate bytes by space

## 6.1.3.4 availableSerialPortList()

```
{\tt MODBUS\_EXPORT\ QStringList\ Modbus::} available {\tt SerialPortList\ (\ )}
```

Returns list of string that represent names of serial ports

### 6.1.3.5 availableSerialPorts()

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

Return list of names of available serial ports

#### 6.1.3.6 bytesToAscii()

Function converts byte array bytesBuff to ASCII repr of byte array. Every byte of 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 count bytes of bytesBuff.

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}

#### 6.1.3.7 bytesToString()

Make string representation of bytes array and separate bytes by space

#### 6.1.3.8 crc16()

CRC16 checksum hash function (for Modbus RTU).

Returns

Returns a 16-bit unsigned integer value of the checksum

#### 6.1.3.9 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.

## 6.1.3.10 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.
Ī	in	blocking	If true blocking will be set, non blocking otherwise.

#### 6.1.3.11 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.

## 6.1.3.12 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.
	in	blocking	If true blocking will be set, non blocking otherwise.

## 6.1.3.13 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.

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

```
const void * settings,
bool blocking )
```

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.

#### 6.1.3.15 enumKey() [1/2]

Convert value to QString key for type

## 6.1.3.16 enumKey() [2/2]

Convert value to QString key for type

## 6.1.3.17 enumValue() [1/4]

Convert key to value for enumeration by QString key

## 6.1.3.18 enumValue() [2/4]

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

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

Convert QVariant value to enumeration value (int - value, string - key). Stores result of convertion in output parameter ok

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

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

## 6.1.3.21 getBit()

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

#### 6.1.3.22 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.

#### 6.1.3.23 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.

## 6.1.3.24 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.

#### 6.1.3.25 Irc()

LRC checksum hash function (for Modbus ASCII).

#### Returns

Returns an 8-bit unsigned integer value of the checksum

#### 6.1.3.26 msleep()

Make current thread sleep with 'msec' milliseconds.

#### 6.1.3.27 readMemBits()

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

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.

#### 6.1.3.28 readMemRegs()

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.
Ī	in	memRegCount	Size of memory buffer memBuff in 16-bit registers.

## 6.1.3.29 sascii()

Make string representation of ASCII array and separate bytes by space

#### 6.1.3.30 sbytes()

Make string representation of bytes array and separate bytes by space

## 6.1.3.31 setBit()

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

## 6.1.3.32 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.

#### 6.1.3.33 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.

#### 6.1.3.34 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.

#### 6.1.3.35 StatusIsBad()

Returns a general indication that the operation result is unsuccessful.

## 6.1.3.36 StatusIsGood()

Returns a general indication that the operation result is successful.

#### 6.1.3.37 StatusIsProcessing()

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

#### 6.1.3.38 StatusIsStandardError()

Returns a general sign that the result is standard error.

#### 6.1.3.39 StatusIsUncertain()

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

#### 6.1.3.40 timer()

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

Get timer value in milliseconds.

## 6.1.3.41 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.

## 6.1.3.42 toFlowControl() [2/2]

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

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.

#### 6.1.3.43 toModbusString()

```
String Modbus::toModbusString (
    int val ) [inline]
```

Convert interger value to Modbus::String

Returns

Returns new Modbus::String value

#### 6.1.3.44 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.

#### 6.1.3.45 toParity() [2/2]

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.

#### 6.1.3.46 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.

#### 6.1.3.47 toProtocolType() [2/2]

```
MODBUS_EXPORT ProtocolType Modbus::toProtocolType ( const QVariant & v, 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.

## 6.1.3.48 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.

# 6.1.3.49 toStopBits() [2/2]

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.

## 6.1.3.50 toString() [1/5]

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

Returns string representation of FlowControl enum value

# 6.1.3.51 toString() [2/5]

Returns string representation of Parity enum value

## 6.1.3.52 toString() [3/5]

Returns string representation of Protocol Type enum value

#### 6.1.3.53 toString() [4/5]

```
\begin{tabular}{ll} MODBUS\_EXPORT QString Modbus::toString ( \\ StatusCode $v$ ) \end{tabular}
```

Returns string representation of StatusCode enum value

#### 6.1.3.54 toString() [5/5]

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

Returns string representation of StopBits enum value

## 6.1.3.55 writeMemBits()

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

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.

## 6.1.3.56 writeMemRegs()

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

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.

# **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

## 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 ()

#### **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 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 patiry.

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.

## 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 of 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  ${\tt Modbus::Defaults}$  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.

# 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

 ${\tt Defaults} \ \textbf{class} \ \textbf{constain} \ \textbf{default} \ \textbf{settings} \ \textbf{values} \ \textbf{for} \ \texttt{ModbusTcpServer}.$ 

## 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:

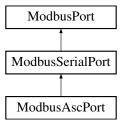
• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusTcpServer.h

## 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)
- ∼ModbusAscPort ()
- Modbus::ProtocolType type () const override

## Public Member Functions inherited from 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
- 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.

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

## 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:

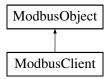
• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusAscPort.h

#### 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 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 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 ∼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)
- $\begin{tabular}{ll} \bullet & template < class Return Type \ , class ... Args > \\ void & disconnect & (Modbus Function Pointer < Return Type, Args ... > func Ptr) \\ \end{tabular}$
- 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()

```
ModbusClient::ModbusClient (
          uint8_t unit,
          ModbusClientPort * port )
```

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 isOpen()

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

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

## 7.7.3.2 lastPortErrorStatus()

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

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

## 7.7.3.3 lastPortErrorText()

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

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

## 7.7.3.4 lastPortStatus()

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

Returns the status of the last operation performed.

#### 7.7.3.5 port()

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

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

## 7.7.3.6 readCoils()

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

#### 7.7.3.7 readCoilsAsBoolArray()

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

#### 7.7.3.8 readDiscreteInputs()

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

#### 7.7.3.9 readDiscreteInputsAsBoolArray()

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

#### 7.7.3.10 readExceptionStatus()

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

#### 7.7.3.11 readHoldingRegisters()

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

#### 7.7.3.12 readInputRegisters()

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

#### 7.7.3.13 setUnit()

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

#### 7.7.3.14 type()

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

Returns the type of the Modbus protocol.

## 7.7.3.15 unit()

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

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

## 7.7.3.16 writeMultipleCoils()

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

#### 7.7.3.17 writeMultipleCoilsAsBoolArray()

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

#### 7.7.3.18 writeMultipleRegisters()

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

## 7.7.3.19 writeSingleCoil()

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

#### 7.7.3.20 writeSingleRegister()

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

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

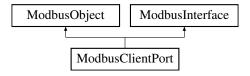
• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusClient.h

## 7.8 ModbusClientPort Class Reference

The  ${\tt ModbusClientPort}$  class implements the algorithm of the client part of the  ${\tt Modbus}$  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
- Modbus::StatusCode close ()
- bool isOpen () const
- uint32 t repeatCount () const
- void setRepeatCount (uint32\_t v)
- 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 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 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 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 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::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)

#### **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  ${\tt ModbusClientPort}$  class implements the algorithm of the client part of the  ${\tt Modbus}$  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 queue will capture it, and so on in a circle.

```
//...
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**

## 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 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.

## 7.8.3.5 isOpen()

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

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

#### 7.8.3.6 lastErrorStatus()

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

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

#### 7.8.3.7 lastErrorText()

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

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

## 7.8.3.8 lastStatus()

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

Returns the status of the last operation performed.

#### 7.8.3.9 port()

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

Returns a pointer to the port object that uses this algorithm.

## 7.8.3.10 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.

### 7.8.3.11 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  $Status\_Bad \leftarrow IllegalFunction$ .

Reimplemented from ModbusInterface.

## 7.8.3.12 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.

## 7.8.3.13 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.

### 7.8.3.14 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.

## 7.8.3.15 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  $Status\_Bad \leftarrow IllegalFunction$ .

Reimplemented from ModbusInterface.

## 7.8.3.16 readDiscreteInputsAsBoolArray() [1/2]

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

### 7.8.3.17 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.

## 7.8.3.18 readExceptionStatus() [1/2]

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

### 7.8.3.19 readExceptionStatus() [2/2]

Function to read ExceptionStatus.

#### **Parameters**

	in	unit	Address of the remote Modbus device.
ĺ	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.

# 7.8.3.20 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.

## 7.8.3.21 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).
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.

## 7.8.3.22 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.

# 7.8.3.23 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  $Status\_Bad \leftarrow IllegalFunction$ .

Reimplemented from ModbusInterface.

## 7.8.3.24 repeatCount()

```
uint32_t ModbusClientPort::repeatCount ( ) const
```

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

### 7.8.3.25 setRepeatCount()

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

## 7.8.3.26 signalClosed()

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

### 7.8.3.27 signalError()

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

### 7.8.3.28 signalOpened()

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

### 7.8.3.29 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'.

### 7.8.3.30 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'.

# 7.8.3.31 type()

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

Returns type of Modbus protocol.

### 7.8.3.32 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.

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

Function for write 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.
out	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.

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

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

## 7.8.3.35 writeMultipleCoilsAsBoolArray() [2/2]

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

#### 7.8.3.36 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.

## 7.8.3.37 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.
out	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.

### 7.8.3.38 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.

## 7.8.3.39 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).
out	value	Boolean value to be set.

# Returns

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

Reimplemented from ModbusInterface.

## 7.8.3.40 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.

## 7.8.3.41 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).
out	value	16-bit unsigned integer value to be set.

#### Returns

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

Reimplemented from ModbusInterface.

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

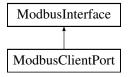
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusClientPort.h

# 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 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)

# 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 15 (0x0F) - WRITE\_MULTIPLE\_COILS 16 (0x10) - WRITE\_MULTIPLE\_REGISTERS

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

### 7.9.2 Member Function Documentation

### 7.9.2.1 readCoils()

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

#### **Parameters**

in	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  $Status\_Bad \leftarrow IllegalFunction$ .

Reimplemented in ModbusClientPort.

## 7.9.2.2 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).	G
out	values	Pointer to the output buffer (bit array) where the read values are stored.	

Generated by Doxygen

#### Returns

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

Reimplemented in ModbusClientPort.

## 7.9.2.3 readExceptionStatus()

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.	Ī

### Returns

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

Reimplemented in ModbusClientPort.

# 7.9.2.4 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.	
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.

## 7.9.2.5 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  ${\tt Modbus::StatusCode}$  of the operation. Default implementation returns  ${\tt Status\_Bad} \leftarrow {\tt IllegalFunction}.$ 

Reimplemented in ModbusClientPort.

## 7.9.2.6 writeMultipleCoils()

Function for write 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.	
out	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.

## 7.9.2.7 writeMultipleRegisters()

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.
out	values	Pointer to the input buffer which values must be written.

#### Returns

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

Reimplemented in ModbusClientPort.

## 7.9.2.8 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).
out	value	Boolean value to be set.

# Returns

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

Reimplemented in ModbusClientPort.

## 7.9.2.9 writeSingleRegister()

```
uint16_t offset,
uint16_t value ) [virtual]
```

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).
out	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 in ModbusClientPort.

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

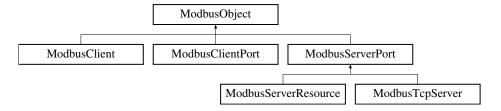
• c:/Users/march/Dropbox/PRJ/ModbusLib/src/Modbus.h

# 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]

Connect this object's signal signal MethodPtr to the objects method objectMethodPtr.

```
class MyClass : public ModbusObject { public: void signalSomething(int a, int b) {
    emitSignal(&MyClass::signalSomething, a, b); } ;
class MyReceiver { public: void slotSomething(int a, int b) { doSomething(); } };
MyClass c;
MyReceiver r;
c.connect(&MyClass::signalSomething, r, &MyReceiver::slotSomething);
```

#### Note

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.

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

```
\label{template} $$\operatorname{T} > $$ void ModbusObject::disconnect ( $$ T * object ) [inline]
```

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  ${\tt funcPtr}$  from all signals of current object, but  ${\tt funcPtr}$  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:

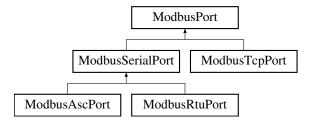
• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusObject.h

# 7.11 ModbusPort Class Reference

The abstract class  ${\tt ModbusPort}$  is the base class for a specific implementation of the  ${\tt Modbus}$  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
- 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 $\sim$ 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} \mbox{virtual void ModbusPort::setNextRequestRepeated (} \\ \mbox{bool } \mbox{$v$} \mbox{) [virtual]}
```

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 type()

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

Returns the Modbus protocol type.

Implemented in ModbusAscPort, ModbusRtuPort, and ModbusTcpPort.

## 7.11.3.19 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.

## 7.11.3.20 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.

### 7.11.3.21 writeBufferData()

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

Returns pointer to data of write buffer.

Implemented in ModbusSerialPort, and ModbusTcpPort.

#### 7.11.3.22 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:

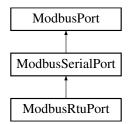
• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusPort.h

# 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

- · 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
- 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]
```

Returns the Modbus protocol type. For ModbusAscPort returns 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:

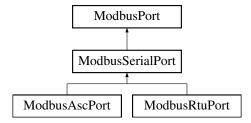
• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusRtuPort.h

# 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**

- · 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
- 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 Member Function Documentation

### 7.13.2.1 baudRate()

```
int32_t ModbusSerialPort::baudRate ( ) const
```

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

### 7.13.2.2 close()

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

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

Implements ModbusPort.

## 7.13.2.3 dataBits()

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

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

## 7.13.2.4 flowControl()

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

Returns current serial port Modbus::FlowControl enum value.

# 7.13.2.5 handle()

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

Returns native OS serial port handle, e.g. HANDLE value for Windows.

Implements ModbusPort.

## 7.13.2.6 isOpen()

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

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

Implements ModbusPort.

### 7.13.2.7 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.

# 7.13.2.8 parity()

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

Returns current serial port Modbus::Parity enum value.

# 7.13.2.9 portName()

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

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

## 7.13.2.10 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.

### 7.13.2.11 readBufferData()

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

Returns pointer to data of read buffer.

Implements ModbusPort.

## 7.13.2.12 readBufferSize()

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

Returns size of data of read buffer.

Implements ModbusPort.

# 7.13.2.13 setBaudRate()

Set current serial port baud rate.

## 7.13.2.14 setDataBits()

Set current serial port baud data bits.

## 7.13.2.15 setFlowControl()

Set current serial port Modbus::FlowControl enum value.

# 7.13.2.16 setParity()

Set current serial port Modbus::Parity enum value.

# 7.13.2.17 setPortName()

Set current serial port name.

### 7.13.2.18 setStopBits()

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

## 7.13.2.19 setTimeoutFirstByte()

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

# 7.13.2.20 setTimeoutInterByte()

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

### 7.13.2.21 stopBits()

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

Returns current serial port Modbus::StopBits enum value.

## 7.13.2.22 timeoutFirstByte()

```
uint32_t ModbusSerialPort::timeoutFirstByte ( ) const
```

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

## 7.13.2.23 timeoutInterByte()

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

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

### 7.13.2.24 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.

# 7.13.2.25 writeBufferData()

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

Returns pointer to data of write buffer.

Implements ModbusPort.

## 7.13.2.26 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:

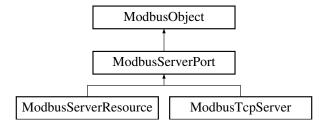
• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusSerialPort.h

# 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
- 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
- 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.

 $Implemented \ in \ Modbus Server Resource, \ and \ Modbus Tcp Server.$ 

## 7.14.2.2 device()

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

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

## 7.14.2.3 isOpen()

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

Returns true if inner port is open, false otherwise.

Implemented in ModbusServerResource, and ModbusTcpServer.

## 7.14.2.4 isStateClosed()

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

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

## 7.14.2.5 isTcpServer()

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

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

Reimplemented in ModbusTcpServer.

## 7.14.2.6 ModbusObject()

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

Constructor of the class.

# 7.14.2.7 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.

## 7.14.2.8 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.

### 7.14.2.9 signalClosed()

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

## 7.14.2.10 signalError()

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

## 7.14.2.11 signalOpened()

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

## 7.14.2.12 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.

### 7.14.2.13 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.

#### 7.14.2.14 type()

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

Returns type of Modbus protocol.

Implemented in ModbusServerResource, and ModbusTcpServer.

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

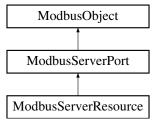
• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusServerPort.h

# 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
- virtual bool isTcpServer () const
- 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 ModbusSlotMethod< T, ReturnType, Args >, and ModbusSlotFunction< 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 ModbusSlotMethod< T, ReturnType, Args >, and ModbusSlotFunction< 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:

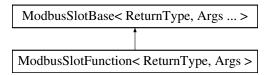
• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusObject.h

# 7.17 ModbusSlotFunction< ReturnType, Args > Class Template Reference

ModbusSlotFunction 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   funcPtr   Pointer to slot t
----------------------------------

# 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:

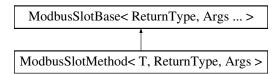
• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusObject.h

# 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:

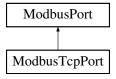
• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusObject.h

# 7.19 ModbusTcpPort Class Reference

Class ModbusTcpPort implements TCP version of 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)
- 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)
- uint32\_t timeout () const
- void setTimeout (uint32\_t timeout)
- 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
- 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.

#### 7.19.3 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()

```
\verb|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()

```
\label{local_post_post} \mbox{void ModbusTcpPort::setNextRequestRepeated (} \\ \mbox{bool } \mbox{$v$ ) [override], [virtual]}
```

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 from ModbusPort.

# 7.19.3.14 setPort()

Sets the settings for the TCP port number of the remote device.

# 7.19.3.15 setTimeout()

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

# 7.19.3.16 timeout()

```
uint32_t ModbusTcpPort::timeout ( ) const
```

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

# 7.19.3.17 type()

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

Returns the Modbus protocol type. In this case it is Modbus::TCP.

Implements ModbusPort.

#### 7.19.3.18 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.

# 7.19.3.19 writeBuffer()

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

Implements ModbusPort.

# 7.19.3.20 writeBufferData()

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

Returns pointer to data of write buffer.

Implements ModbusPort.

# 7.19.3.21 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:

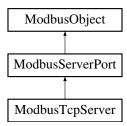
• c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusTcpPort.h

# 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)
- uint16\_t port () const
- void setPort (uint16\_t port)
- int timeout () const
- void setTimeout (int timeout)
- Modbus::ProtocolType type () const override
- bool isTcpServer () const override
- Modbus::StatusCode open () override
- Modbus::StatusCode close () override
- bool isOpen () const override
- Modbus::StatusCode process () override
- virtual ModbusServerPort \* createTcpPort (ModbusTcpSocket \*socket)
- void signalNewConnection (const Modbus::Char \*source)
- void signalCloseConnection (const Modbus::Char \*source)

# Public Member Functions inherited from ModbusServerPort

- ModbusInterface \* device () const
- 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.

#### 7.20.3 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()

Creates ModbusServerPort for new incoming connection defined by ModbusTcpSocket pointer/

# 7.20.3.4 isOpen()

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

Returns  ${\tt true}$  if the server is currently listening for incoming connections,  ${\tt false}$  otherwise.

Implements ModbusServerPort.

# 7.20.3.5 isTcpServer()

```
bool ModbusTcpServer::isTcpServer ( ) const [inline], [override], [virtual]
```

Returns true.

Reimplemented from ModbusServerPort.

#### 7.20.3.6 nextPendingConnection()

```
{\tt ModbusTcpSocket} \ * \ {\tt ModbusTcpServer::nextPendingConnection} \ \ (\ ) \quad [protected]
```

Checks for incoming connections and returns pointer ModbusTcpSocket if new connection established, nullptr otherwise.

#### 7.20.3.7 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
- $\bullet \ \texttt{Modbus::Status\_BadTcpListen} \ \textbf{when can't listen TCP socket}$

Implements ModbusServerPort.

#### 7.20.3.8 port()

```
uint16_t ModbusTcpServer::port ( ) const
```

Returns the setting for the TCP port number of the server.

# 7.20.3.9 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.

#### 7.20.3.10 setPort()

Sets the settings for the TCP port number of the server.

# 7.20.3.11 setTimeout()

Sets the setting for the read timeout of every single conncetion.

# 7.20.3.12 signalCloseConnection()

Signal occured when TCP connection was closed. source - name of the current connection.

#### 7.20.3.13 signalNewConnection()

Signal occured when new TCP connection was accepted. source - name of the current connection.

#### 7.20.3.14 timeout()

```
int ModbusTcpServer::timeout ( ) const
```

Returns the setting for the read timeout of every single connection.

# 7.20.3.15 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 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 patiry.

· 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.

# 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 of 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

 ${\it Handle (pointer) of ModbusClient for C interface.}$ 

• typedef ModbusServerPort \* 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(\* 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 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 cModbusInterface cCreateModbusDevice (cModbusDevice device, pfReadCoils readCoils, pfReadDiscreteInputs readDiscreteInputs, pfReadHoldingRegisters readHoldingRegisters, pfReadInputRegisters readInputRegisters, pfWriteSingleCoil writeSingleCoil, pfWriteSingleRegister write SingleRegister, pfReadExceptionStatus readExceptionStatus, pfWriteMultipleCoils writeMultipleCoils, pfWriteMultipleRegisters writeMultipleRegisters)
- 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 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 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 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 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

#### 8.1.2.2 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

#### 8.1.2.3 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

#### 8.1.2.4 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

#### 8.1.2.5 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

# 8.1.2.6 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

# 8.1.2.7 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

# 8.1.2.8 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

ModbusClientPort::signalError and ModbusServerPort::signalError

# 8.1.2.9 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

#### 8.1.2.10 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

#### 8.1.2.11 pfSlotRx

```
typedef void(* pfSlotRx) (const Char *source, const uint8_t *buff, uint16_t size)
```

Pointer to C callback function.  ${\tt dev}$  - pointer to any struct that can hold memory data.

See also

ModbusClientPort::signalRx and ModbusServerPort::signalRx

# 8.1.2.12 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

 ${\tt ModbusClientPort::signalTx} \ \textbf{and} \ {\tt ModbusServerPort::signalTx}$ 

# 8.1.2.13 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

# 8.1.2.14 pfWriteMultipleRegisters

 $typedef\ StatusCode\ (*\ pfWriteMultipleRegisters)\ (cModbusDevice\ dev,\ uint8\_t\ unit,\ uint16\_{\leftarrow}\ t\ offset,\ uint16\_t\ count,\ const\ uint16\_t\ *values)$ 

Pointer to C function for write registers (4x). dev - pointer to any struct that can hold memory data.

See also

ModbusInterface::writeMultipleRegisters

#### 8.1.2.15 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

# 8.1.2.16 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()

```
MODBUS_EXPORT cModbusClient cCliCreate (
          uint8_t unit,
          ProtocolType type,
          const void * settings,
          bool blocking )
```

 ${\bf Creates} \ {\bf ModbusClient} \ {\bf object} \ {\bf and} \ {\bf returns} \ {\bf handle} \ {\bf to} \ {\bf it}.$ 

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()

 $\textbf{Deletes previously created } \underline{\texttt{ModbusClient}} \ \textbf{object represented by } \underline{\texttt{client}} \ \textbf{handle}$ 

#### 8.1.3.4 cCliGetLastPortErrorStatus()

```
\begin{tabular}{ll} MODBUS\_EXPORT & StatusCode & cCliGetLastPortErrorStatus & ( & cModbusClient & client & ) \\ \end{tabular}
```

Wrapper for ModbusClient::lastPortErrorStatus

# 8.1.3.5 cCliGetLastPortErrorText()

Wrapper for ModbusClient::lastPortErrorText

#### 8.1.3.6 cCliGetLastPortStatus()

```
MODBUS_EXPORT StatusCode cCliGetLastPortStatus (
            cModbusClient client )
Wrapper for ModbusClient::lastPortStatus
8.1.3.7 cCliGetObjectName()
MODBUS_EXPORT const Char * cCliGetObjectName (
            cModbusClient client )
Wrapper for ModbusClient::objectName
8.1.3.8 cCliGetPort()
MODBUS_EXPORT cModbusClientPort cCliGetPort (
            cModbusClient client )
Wrapper for ModbusClient::port
8.1.3.9 cCliGetType()
MODBUS_EXPORT ProtocolType cCliGetType (
            cModbusClient client )
Wrapper for ModbusClient::type
8.1.3.10 cCliGetUnit()
MODBUS_EXPORT uint8_t cCliGetUnit (
            cModbusClient client )
Wrapper for ModbusClient::unit
8.1.3.11 cClilsOpen()
MODBUS_EXPORT bool cCliIsOpen (
            cModbusClient client )
Wrapper for ModbusClient::isOpen
8.1.3.12 cCliSetObjectName()
MODBUS_EXPORT void cCliSetObjectName (
            cModbusClient client,
            const Char * name )
```

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()

```
MODBUS_EXPORT cModbusClientPort cCpoCreate (
          ProtocolType type,
          const void * settings,
          bool blocking )
```

Creates ModbusClientPort object and returns handle to it.

See also

```
Modbus::createClientPort
```

#### 8.1.3.21 cCpoCreateForPort()

Creates ModbusClientPort object and returns handle to it.

# 8.1.3.22 cCpoDelete()

Deletes previously created ModbusClientPort object represented by port handle

#### 8.1.3.23 cCpoDisconnectFunc()

Disconnects funcPtr-function from ModbusClientPort

# 8.1.3.24 cCpoGetLastErrorStatus()

Wrapper for ModbusClientPort::getLastErrorStatus

#### 8.1.3.25 cCpoGetLastErrorText()

```
MODBUS_EXPORT const Char * cCpoGetLastErrorText (
            cModbusClientPort clientPort )
Wrapper for ModbusClientPort::getLastErrorText
8.1.3.26 cCpoGetLastStatus()
MODBUS_EXPORT StatusCode cCpoGetLastStatus (
            cModbusClientPort clientPort )
Wrapper for ModbusClientPort::getLastStatus
8.1.3.27 cCpoGetObjectName()
MODBUS_EXPORT const Char * cCpoGetObjectName (
            cModbusClientPort clientPort )
Wrapper for ModbusClientPort::objectName
8.1.3.28 cCpoGetRepeatCount()
MODBUS_EXPORT uint32_t cCpoGetRepeatCount (
            cModbusClientPort clientPort )
Wrapper for ModbusClientPort::repeatCount
8.1.3.29 cCpoGetType()
MODBUS_EXPORT ProtocolType cCpoGetType (
            cModbusClientPort clientPort )
Wrapper for ModbusClientPort::type
8.1.3.30 cCpolsOpen()
MODBUS_EXPORT bool cCpoIsOpen (
            cModbusClientPort clientPort )
Wrapper for ModbusClientPort::isOpen
8.1.3.31 cCpoReadCoils()
MODBUS_EXPORT StatusCode cCpoReadCoils (
            cModbusClientPort clientPort,
            uint8_t unit,
            uint16_t offset,
```

uint16\_t count,
void \* values )

Wrapper for ModbusClientPort::readCoils

# 8.1.3.32 cCpoReadCoilsAsBoolArray()

Wrapper for ModbusClientPort::readCoilsAsBoolArray

# 8.1.3.33 cCpoReadDiscreteInputs()

Wrapper for ModbusClientPort::readDiscreteInputs

# 8.1.3.34 cCpoReadDiscreteInputsAsBoolArray()

Wrapper for ModbusClientPort::readDiscreteInputsAsBoolArray

#### 8.1.3.35 cCpoReadExceptionStatus()

Wrapper for ModbusClientPort::readExceptionStatus

# 8.1.3.36 cCpoReadHoldingRegisters()

Wrapper for ModbusClientPort::readHoldingRegisters

# 8.1.3.37 cCpoReadInputRegisters()

Wrapper for ModbusClientPort::readInputRegisters

# 8.1.3.38 cCpoSetObjectName()

Wrapper for ModbusClientPort::setObjectName

# 8.1.3.39 cCpoSetRepeatCount()

Wrapper for ModbusClientPort::repeatCount

# 8.1.3.40 cCpoWriteMultipleCoils()

Wrapper for ModbusClientPort::writeMultipleCoils

# 8.1.3.41 cCpoWriteMultipleCoilsAsBoolArray()

Wrapper for ModbusClientPort::writeMultipleCoilsAsBoolArray

# 8.1.3.42 cCpoWriteMultipleRegisters()

Wrapper for ModbusClientPort::writeMultipleRegisters

# 8.1.3.43 cCpoWriteSingleCoil()

Wrapper for ModbusClientPort::writeSingleCoil

#### 8.1.3.44 cCpoWriteSingleRegister()

Wrapper for ModbusClientPort::writeSingleRegister

#### 8.1.3.45 cCreateModbusDevice()

Function create  ${\tt ModbusInterface}$  object and returns pointer to it for server.  ${\tt dev}$  - pointer to any struct that can hold memory data. readCoils, readDiscreteInputs, readHoldingRegisters, readInputRegisters, writeSingle Coil, writeSingleRegister, readExceptionStatus, writeMultipleCoils - pointers to corresponding Modbus functions to process data. Any pointer can have NULL value. In this case server will return  ${\tt Status\_BadIllegal} \leftarrow {\tt Function}$ .

# 8.1.3.46 cDeleteModbusDevice()

```
\begin{tabular}{ll} MODBUS\_EXPORT & void & cDeleteModbusDevice & ( \\ & & cModbusInterface & dev & ) \end{tabular}
```

Deletes previously created  ${\tt ModbusInterface}$  object represented by  ${\tt dev}$  handle

# 8.1.3.47 cPortCreate()

```
MODBUS_EXPORT cModbusPort cPortCreate (
          ProtocolType type,
          const void * settings,
          bool blocking )
```

Creates ModbusPort object and returns handle to it.

See also

```
Modbus::createPort
```

# 8.1.3.48 cPortDelete()

Deletes previously created ModbusPort object represented by port handle

#### 8.1.3.49 cReadCoils()

Wrapper for ModbusClient::readCoils

# 8.1.3.50 cReadCoilsAsBoolArray()

Wrapper for ModbusClient::readCoilsAsBoolArray

## 8.1.3.51 cReadDiscreteInputs()

Wrapper for ModbusClient::readDiscreteInputs

## 8.1.3.52 cReadDiscreteInputsAsBoolArray()

Wrapper for ModbusClient::readDiscreteInputsAsBoolArray

## 8.1.3.53 cReadExceptionStatus()

Wrapper for ModbusClient::readExceptionStatus

## 8.1.3.54 cReadHoldingRegisters()

Wrapper for ModbusClient::readHoldingRegisters

## 8.1.3.55 cReadInputRegisters()

Wrapper for ModbusClient::readInputRegisters

## 8.1.3.56 cSpoClose()

Wrapper for ModbusServerPort::close

# 8.1.3.57 cSpoConnectCloseConnection()

Connects funcPtr-function to ModbusServerPort::signalCloseConnection signal

# 8.1.3.58 cSpoConnectClosed()

Connects funcPtr-function to ModbusServerPort::signalClosed signal

## 8.1.3.59 cSpoConnectError()

Connects funcPtr-function to ModbusServerPort::signalError signal

# 8.1.3.60 cSpoConnectNewConnection()

Connects funcPtr-function to ModbusServerPort::signalNewConnection signal

# 8.1.3.61 cSpoConnectOpened()

Connects funcPtr-function to ModbusServerPort::signalOpened signal

## 8.1.3.62 cSpoConnectRx()

Connects funcPtr-function to ModbusServerPort::signalRx signal

## 8.1.3.63 cSpoConnectTx()

Connects funcPtr-function to ModbusServerPort::signalTx signal

## 8.1.3.64 cSpoCreate()

Creates ModbusServerPort object and returns handle to it.

See also

```
Modbus::createServerPort
```

# 8.1.3.65 cSpoDelete()

Deletes previously created ModbusServerPort object represented by serverPort handle

# 8.1.3.66 cSpoDisconnectFunc()

Disconnects funcPtr-function from ModbusServerPort

## 8.1.3.67 cSpoGetDevice()

Wrapper for ModbusServerPort::device

# 8.1.3.68 cSpoGetObjectName()

```
MODBUS_EXPORT const Char * cSpoGetObjectName (
            cModbusServerPort serverPort )
Wrapper for ModbusServerPort::objectName
8.1.3.69 cSpoGetType()
MODBUS_EXPORT ProtocolType cSpoGetType (
            cModbusServerPort serverPort )
Wrapper for ModbusServerPort::type
8.1.3.70 cSpolsOpen()
MODBUS_EXPORT bool cSpoIsOpen (
            cModbusServerPort serverPort )
Wrapper for ModbusServerPort::isOpen
8.1.3.71 cSpoIsTcpServer()
MODBUS_EXPORT bool cSpoIsTcpServer (
            cModbusServerPort serverPort )
Wrapper for ModbusServerPort::isTcpServer
8.1.3.72 cSpoOpen()
MODBUS_EXPORT StatusCode cSpoOpen (
            cModbusServerPort serverPort )
Wrapper for ModbusServerPort::open
8.1.3.73 cSpoProcess()
MODBUS_EXPORT StatusCode cSpoProcess (
            cModbusServerPort serverPort )
Wrapper for ModbusServerPort::process
8.1.3.74 cSpoSetObjectName()
MODBUS_EXPORT void cSpoSetObjectName (
            cModbusServerPort serverPort,
            const Char * name )
```

Wrapper for ModbusServerPort::setObjectName

## 8.1.3.75 cWriteMultipleCoils()

Wrapper for ModbusClient::writeMultipleCoils

## 8.1.3.76 cWriteMultipleCoilsAsBoolArray()

Wrapper for ModbusClient::lastPortStatus

## 8.1.3.77 cWriteMultipleRegisters()

Wrapper for ModbusClient::writeMultipleRegisters

# 8.1.3.78 cWriteSingleCoil()

Wrapper for ModbusClient::writeSingleCoil

# 8.1.3.79 cWriteSingleRegister()

Wrapper for ModbusClient::writeSingleRegister

# 8.2 cModbus.h

```
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 ModbusClientPort:
00022 class ModbusClient
00023 class ModbusServerPort;
00024 class ModbusInterface;
00025
00026 #else
00027 typedef struct ModbusPort
                                      ModbusPort
00028 typedef struct ModbusClientPort ModbusClientPort;
00029 typedef struct ModbusClient
                                       ModbusClient
00030 typedef struct ModbusServerPort ModbusServerPort;
00031 typedef struct ModbusInterface ModbusInterface;
00032 #endif
00033
00034
00036 typedef ModbusPort* cModbusPort;
00037
00039 typedef ModbusClientPort* cModbusClientPort;
00040
00042 typedef ModbusClient* cModbusClient;
00043
00045 typedef ModbusServerPort* cModbusServerPort;
00046
00048 typedef ModbusInterface* cModbusInterface;
00049
00051 typedef void* cModbusDevice;
00052
00054 typedef StatusCode (*pfReadCoils)(cModbusDevice dev, uint8_t unit, uint16_t offset, uint16_t count,
      void *values);
00055
00057 typedef StatusCode (*pfReadDiscreteInputs) (cModbusDevice dev, uint8_t unit, uint16_t offset, uint16_t
      count, void *values);
00058
00060 typedef StatusCode (*pfReadHoldingRegisters) (cModbusDevice dev, uint8_t unit, uint16_t offset,
      uint16_t count, uint16_t *values);
00061
00063 typedef StatusCode (*pfReadInputRegisters) (cModbusDevice dev, uint8_t unit, uint16_t offset, uint16_t
      count, uint16_t *values);
00064
00066 typedef StatusCode (*pfWriteSingleCoil) (cModbusDevice dev, uint8_t unit, uint16_t offset, bool value);
00067
00069 typedef StatusCode (*pfWriteSingleRegister) (cModbusDevice dev, uint8_t unit, uint16_t offset, uint16_t
00070
00072 typedef StatusCode (*pfReadExceptionStatus)(cModbusDevice dev, uint8_t unit, uint8_t *status);
00073
00075 typedef StatusCode (*pfWriteMultipleCoils) (cModbusDevice dev, uint8_t unit, uint16_t offset, uint16_t
      count, const void *values);
00076
00078 typedef StatusCode (*pfWriteMultipleRegisters)(cModbusDevice dev, uint8_t unit, uint16_t offset,
      uint16_t count, const uint16_t *values);
00079
00081 typedef void (*pfSlotOpened)(const Char *source);
00082
00084 typedef void (*pfSlotClosed)(const Char *source);
00085
00087 typedef void (*pfSlotTx)(const Char *source, const uint8_t* buff, uint16_t size);
00088
00090 typedef void (*pfSlotRx)(const Char *source, const uint8 t* buff, uint16 t size);
00091
00093 typedef void (*pfSlotError) (const Char *source, StatusCode status, const Char *text);
00094
00096 typedef void (*pfSlotNewConnection)(const Char *source);
00097
00099 typedef void (*pfSlotCloseConnection)(const Char *source);
00112 MODBUS EXPORT cModbusInterface cCreateModbusDevice(cModbusDevice
00113
                                                          pfReadCoils
                                                                                    readCoils
00114
                                                           pfReadDiscreteInputs
                                                                                    readDiscreteInputs
```

8.2 cModbus.h 135

```
00115
                                                         pfReadHoldingRegisters
                                                                                 readHoldingRegisters
00116
                                                         pfReadInputRegisters
                                                                                  readInputRegisters
00117
                                                         pfWriteSingleCoil
                                                                                  writeSingleCoil
00118
                                                         pfWriteSingleRegister
                                                                                  writeSingleRegister
00119
                                                         pfReadExceptionStatus
                                                                                  readExceptionStatus
                                                         pfWriteMultipleCoils
00120
                                                                                 writeMultipleCoils
00121
                                                         pfWriteMultipleRegisters writeMultipleRegisters);
00122
00123
00125 MODBUS EXPORT void cDeleteModbusDevice(cModbusInterface dev);
00126
00127 //
00128 // ----- ModbusPort
00129 //
00130
00132 MODBUS_EXPORT cModbusPort cPortCreate(ProtocolType type, const void *settings, bool blocking);
00135 MODBUS_EXPORT void cPortDelete(cModbusPort port);
00136
00137
00138 //
00139 // --
                                ----- ModbusClientPort
00140 //
00141
00143 MODBUS_EXPORT cModbusClientPort cCpoCreate(ProtocolType type, const void *settings, bool blocking);
00144
00146 MODBUS_EXPORT cModbusClientPort cCpoCreateForPort(cModbusPort port);
00147
00149 MODBUS_EXPORT void cCpoDelete(cModbusClientPort clientPort);
00150
00152 MODBUS EXPORT const Char *cCpoGetObjectName(cModbusClientPort clientPort);
00153
00155 MODBUS EXPORT void cCpoSetObjectName(cModbusClientPort clientPort, const Char *name);
00156
00158 MODBUS_EXPORT ProtocolType cCpoGetType(cModbusClientPort clientPort);
00159
00161 MODBUS EXPORT bool cCpoIsOpen(cModbusClientPort clientPort);
00162
00164 MODBUS_EXPORT bool cCpoClose(cModbusClientPort clientPort);
00165
00167 MODBUS_EXPORT uint32_t cCpoGetRepeatCount(cModbusClientPort clientPort);
00168
00170 MODBUS EXPORT void cCpoSetRepeatCount (cModbusClientPort clientPort, uint32 t count);
00171
00173 MODBUS_EXPORT StatusCode cCpoReadCoils(cModbusClientPort clientPort, uint8_t unit, uint16_t offset,
      uint16_t count, void *values);
00174
00176 MODBUS_EXPORT StatusCode cCpoReadDiscreteInputs(cModbusClientPort clientPort, uint8_t unit, uint16_t
      offset, uint16_t count, void *values);
00177
00179 MODBUS_EXPORT StatusCode cCpoReadHoldingRegisters(cModbusClientPort clientPort, uint8_t unit, uint16_t
      offset, uint16_t count, uint16_t *values);
00180
00182 MODBUS_EXPORT StatusCode cCpoReadInputRegisters(cModbusClientPort clientPort, uint8_t unit, uint16_t
      offset, uint16_t count, uint16_t *values);
00183
00185 MODBUS_EXPORT StatusCode cCpoWriteSingleCoil(cModbusClientPort clientPort, uint8_t unit, uint16_t
      offset, bool value);
00186
00188 MODBUS_EXPORT StatusCode cCpoWriteSingleRegister(cModbusClientPort clientPort, uint8_t unit, uint16_t
      offset, uint16_t value);
00189
00191 MODBUS_EXPORT StatusCode cCpoReadExceptionStatus(cModbusClientPort clientPort, uint8_t unit, uint8_t
      *value);
00192
00194 MODBUS_EXPORT StatusCode cCpoWriteMultipleCoils(cModbusClientPort clientPort, uint8_t unit, uint16_t
      offset, uint16_t count, const void *values);
00195
00197 MODBUS_EXPORT StatusCode cCpoWriteMultipleRegisters(cModbusClientPort clientPort, uint8_t unit,
      uint16_t offset, uint16_t count, const uint16_t *values);
00198
00200 MODBUS_EXPORT StatusCode cCpoReadCoilsAsBoolArray(cModbusClientPort clientPort, uint8_t unit, uint16_t
      offset, uint16_t count, bool *values);
00201
00203 MODBUS_EXPORT StatusCode cCpoReadDiscreteInputsAsBoolArray(cModbusClientPort clientPort, uint8_t unit,
      uint16_t offset, uint16_t count, bool *values);
00206 MODBUS_EXPORT StatusCode cCpoWriteMultipleCoilsAsBoolArray(cModbusClientPort clientPort, uint8_t unit,
      uint16_t offset, uint16_t count, const bool *values);
00207
00209 MODBUS EXPORT StatusCode cCpoGetLastStatus(cModbusClientPort clientPort);
```

```
00212 MODBUS_EXPORT StatusCode cCpoGetLastErrorStatus(cModbusClientPort clientPort);
00213
00215 MODBUS EXPORT const Char *cCpoGetLastErrorText(cModbusClientPort clientPort);
00216
00218 MODBUS_EXPORT void cCpoConnectOpened(cModbusClientPort clientPort, pfSlotOpened funcPtr);
00219
00221 MODBUS_EXPORT void cCpoConnectClosed(cModbusClientPort clientPort, pfSlotClosed funcPtr);
00222
00224 MODBUS EXPORT void cCpoConnectTx(cModbusClientPort clientPort, pfSlotTx funcPtr);
00225
00227 MODBUS EXPORT void cCpoConnectRx(cModbusClientPort clientPort, pfSlotRx funcPtr);
00228
00230 MODBUS_EXPORT void cCpoConnectError(cModbusClientPort clientPort, pfSlotError funcPtr);
00231
00233 MODBUS_EXPORT void cCpoDisconnectFunc(cModbusClientPort clientPort, void *funcPtr);
00234
00235
00236 //
00237 // ---
                                 ----- ModbusClient
00238 //
00239
00241 MODBUS_EXPORT cModbusClient cCliCreate(uint8_t unit, ProtocolType type, const void *settings, bool
00242
00244 MODBUS_EXPORT cModbusClient cCliCreateForClientPort(uint8_t unit, cModbusClientPort clientPort);
00245
00247 MODBUS_EXPORT void cCliDelete(cModbusClient client);
00248
00250 MODBUS_EXPORT const Char *cCliGetObjectName(cModbusClient client);
00251
00253 MODBUS_EXPORT void cCliSetObjectName(cModbusClient client, const Char *name);
00254
00256 MODBUS EXPORT ProtocolType cCliGetType(cModbusClient client);
00259 MODBUS EXPORT uint8 t cCliGetUnit (cModbusClient client);
00260
00262 MODBUS_EXPORT void cCliSetUnit(cModbusClient client, uint8_t unit);
00263
00265 MODBUS EXPORT bool cClilsOpen(cModbusClient client):
00266
00268 MODBUS EXPORT cModbusClientPort cCliGetPort(cModbusClient client);
00269
00271 MODBUS_EXPORT StatusCode cReadCoils(cModbusClient client, uint16_t offset, uint16_t count, void
      *values);
00272
00274 MODBUS_EXPORT StatusCode cReadDiscreteInputs(cModbusClient client, uint16_t offset, uint16_t count,
      void *values);
00275
00277 MODBUS_EXPORT StatusCode cReadHoldingRegisters(cModbusClient client, uint16_t offset, uint16_t count,
     uint16_t *values);
00278
00280 MODBUS EXPORT StatusCode cReadInputRegisters(cModbusClient client, uint16 t offset, uint16 t count,
     uint16_t *values);
00281
00283 MODBUS_EXPORT StatusCode cWriteSingleCoil(cModbusClient client, uint16_t offset, bool value);
00284
00286 MODBUS EXPORT StatusCode cWriteSingleRegister(cModbusClient client, uint16_t offset, uint16_t value);
00287
00289 MODBUS_EXPORT StatusCode cReadExceptionStatus(cModbusClient client, uint8_t *value);
00290
00292 MODBUS_EXPORT StatusCode cWriteMultipleCoils(cModbusClient client, uint16_t offset, uint16_t count,
      const void *values);
00293
00295 MODBUS EXPORT StatusCode cWriteMultipleRegisters(cModbusClient client, uint16 t offset, uint16 t
      count, const uint16 t *values);
00296
00298 MODBUS_EXPORT StatusCode cReadCoilsAsBoolArray(cModbusClient client, uint16_t offset, uint16_t count,
      bool *values);
00299
00301 MODBUS_EXPORT StatusCode cReadDiscreteInputsAsBoolArray(cModbusClient client, uint16_t offset,
      uint16_t count, bool *values);
00302
00304 MODBUS_EXPORT StatusCode cWriteMultipleCoilsAsBoolArray(cModbusClient client, uint16_t offset,
      uint16_t count, const bool *values);
00305
00307 MODBUS EXPORT StatusCode cCliGetLastPortStatus(cModbusClient client):
00308
00310 MODBUS_EXPORT StatusCode cCliGetLastPortErrorStatus(cModbusClient client);
00311
00313 MODBUS_EXPORT const Char *cCliGetLastPortErrorText(cModbusClient client);
00314
00315
00316 //
```

```
00317 // -----
00318 //
00319
00321 MODBUS_EXPORT cModbusServerPort cSpoCreate(cModbusInterface device, ProtocolType type, const void
      *settings, bool blocking);
00322
00324 MODBUS_EXPORT void cSpoDelete(cModbusServerPort serverPort);
00325
00327 MODBUS_EXPORT const Char *cSpoGetObjectName(cModbusServerPort serverPort);
00328
00330 MODBUS_EXPORT void cSpoSetObjectName(cModbusServerPort serverPort, const Char *name);
00331
00333 MODBUS_EXPORT ProtocolType cSpoGetType(cModbusServerPort serverPort);
00334
00336 MODBUS EXPORT bool cSpoIsTcpServer(cModbusServerPort serverPort);
00339 MODBUS_EXPORT cModbusInterface cSpoGetDevice(cModbusServerPort serverPort);
00340
00342 MODBUS_EXPORT bool cSpoIsOpen(cModbusServerPort serverPort);
00343
00345 MODBUS EXPORT StatusCode cSpoOpen(cModbusServerPort serverPort);
00346
00348 MODBUS_EXPORT StatusCode cSpoClose(cModbusServerPort serverPort);
00349
00351 MODBUS_EXPORT StatusCode cSpoProcess(cModbusServerPort serverPort);
00352
00354 MODBUS_EXPORT void cSpoConnectOpened(cModbusServerPort serverPort, pfSlotOpened funcPtr);
00355
00357 MODBUS_EXPORT void cSpoConnectClosed(cModbusServerPort serverPort, pfSlotClosed funcPtr);
00358
00360 MODBUS_EXPORT void cSpoConnectTx(cModbusServerPort serverPort, pfSlotTx funcPtr);
00361
00363 MODBUS_EXPORT void cSpoConnectRx(cModbusServerPort serverPort, pfSlotRx funcPtr);
00364
00366 MODBUS_EXPORT void cSpoConnectError(cModbusServerPort serverPort, pfSlotError funcPtr);
{\tt 00369\ MODBUS\_EXPORT\ void\ cSpoConnectNewConnection} ({\tt cModbusServerPort\ serverPort,\ pfSlotNewConnection})
      funcPtr);
00370
{\tt 00372\ MODBUS\_EXPORT\ void\ cSpoConnectCloseConnection(cModbusServerPort\ serverPort,\ pfSlotCloseConnection)}
00375 MODBUS_EXPORT void cSpoDisconnectFunc(cModbusServerPort serverPort, void *funcPtr);
00376
00377
00378 #ifdef __cplusplus
00379 } // extern "C"
00380 #endif
00381
00382 #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**

- 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 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)

## 8.3.1 Detailed Description

Contains general definitions of the Modbus protocol.

Author

serhmarch

Date

May 2024

8.4 Modbus.h 139

# 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"
00015
00016 class ModbusPort;
00017 class ModbusClientPort;
00018 class ModbusServerPort;
00019
00020 //
00021 // ----- Modbus interface
00022 //
00023
00040 class MODBUS EXPORT ModbusInterface
00041 {
00042 public:
00049
          virtual Modbus::StatusCode readCoils(uint8_t unit, uint16_t offset, uint16_t count, void *values);
00050
00057
         virtual Modbus::StatusCode readDiscreteInputs(uint8_t unit, uint16_t offset, uint16_t count, void
     *values);
00058
00065
          virtual Modbus::StatusCode readHoldingRegisters(uint8_t unit, uint16_t offset, uint16_t count,
     uint16_t *values);
00066
00073
          virtual Modbus::StatusCode readInputRegisters(uint8_t unit, uint16_t offset, uint16_t count,
     uint16_t *values);
00074
08000
          virtual Modbus::StatusCode writeSingleCoil(uint8_t unit, uint16_t offset, bool value);
00081
00087
          virtual Modbus::StatusCode writeSingleRegister(uint8_t unit, uint16_t offset, uint16_t value);
00088
00093
         virtual Modbus::StatusCode readExceptionStatus(uint8_t unit, uint8_t *status);
00094
00101
          virtual Modbus::StatusCode writeMultipleCoils(uint8_t unit, uint16_t offset, uint16_t count, const
      void *values);
00102
00109
         virtual Modbus::StatusCode writeMultipleRegisters(uint8_t unit, uint16_t offset, uint16_t count,
      const uint16_t *values);
00110 };
00111
00112 //
00113 // ---
                                    ----- Modbus namespace
00114 //
00115
00117 namespace Modbus {
00118
00120 typedef std::string String;
00121
00123 template <class T>
00124 using List = std::list<T>;
00125
00128 inline String toModbusString(int val) { return std::to_string(val); }
00129
00131 MODBUS EXPORT String bytesToString(const uint8 t* buff, uint32 t count);
00132
00134 MODBUS_EXPORT String asciiToString(const uint8_t* buff, uint32_t count);
00135
00137 MODBUS_EXPORT List<String> availableSerialPorts();
00138
00143 MODBUS_EXPORT ModbusPort *createPort(ProtocolType type, const void *settings, bool blocking);
00144
00149 MODBUS_EXPORT ModbusClientPort *createClientPort (ProtocolType type, const void *settings, bool
00150
00156 MODBUS_EXPORT ModbusServerPort *createServerPort (ModbusInterface *device, ProtocolType type, const
     void *settings, bool blocking);
00157
00158 } //namespace Modbus
00159
00160 #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_PATCH 0
00006 #define MODBUSLIB_VERSION_PATCH 0
00007
00008 #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

```
00008 #ifndef MODBUSASCPORT_H
00009 #define MODBUSASCPORT_H
00010
00011 #include "ModbusSerialPort.h"
00012
00019 class MODBUS_EXPORT ModbusAscPort : public ModbusSerialPort
00021 public:
00023
         ModbusAscPort(bool blocking = false);
00024
00026
          ~ModbusAscPort();
00027
00028 public:
00030
         Modbus::ProtocolType type() const override { return Modbus::ASC; }
00031
00032 protected:
         Modbus::StatusCode writeBuffer(uint8_t unit, uint8_t func, uint8_t *buff, uint16_t szInBuff)
00033
     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 };
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"
```

#### 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

```
00001
00008 #ifndef MODBUSCLIENT_H
00009 #define MODBUSCLIENT_H
00010
00011 #include "ModbusObject.h"
00012
00013 class ModbusClientPort;
00024 class MODBUS_EXPORT ModbusClient : public ModbusObject
00025 {
00026 public:
          ModbusClient(uint8_t unit, ModbusClientPort *port);
00030
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:
          Modbus::StatusCode readCoils(uint16_t offset, uint16_t count, void *values);
00050
00051
00053
          Modbus::StatusCode readDiscreteInputs(uint16_t offset, uint16_t count, void *values);
00054
00056
          Modbus::StatusCode readHoldingRegisters(uint16_t offset, uint16_t count, uint16_t *values);
00057
00059
          Modbus::StatusCode readInputRegisters(uint16_t offset, uint16_t count, uint16_t *values);
00060
00062
          Modbus::StatusCode writeSingleCoil(uint16_t offset, bool value);
```

```
00065
          Modbus::StatusCode writeSingleRegister(uint16_t offset, uint16_t value);
00066
          Modbus::StatusCode readExceptionStatus(uint8_t *value);
00068
00069
00071
          Modbus::StatusCode writeMultipleCoils(uint16_t offset, uint16_t count, const void *values);
00072
00074
          Modbus::StatusCode writeMultipleRegisters(uint16_t offset, uint16_t count, const uint16_t
00075
00077
          Modbus::StatusCode readCoilsAsBoolArray(uint16_t offset, uint16_t count, bool *values);
00078
08000
          Modbus::StatusCode readDiscreteInputsAsBoolArray(uint16_t offset, uint16_t count, bool *values);
00081
00083
          Modbus::StatusCode writeMultipleCoilsAsBoolArray(uint16_t offset, uint16_t count, const bool
00084
00085 public:
00087
          Modbus::StatusCode lastPortStatus() const;
00088
00090
          Modbus::StatusCode lastPortErrorStatus() const;
00091
          const Modbus::Char *lastPortErrorText() const;
00093
00094
00095 protected:
00097
         using ModbusObject::ModbusObject;
00099 };
00100
00101 #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"
```

#### 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** 

march

Date

May 2024

8.11 ModbusClientPort.h 143

# 8.11 ModbusClientPort.h

```
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:
          ModbusClientPort(ModbusPort *port);
00069
00070
00071 public:
00073
          Modbus::ProtocolType type() const;
00074
00076
          ModbusPort *port() const;
00077
00079
          Modbus::StatusCode close();
08000
00082
          bool isOpen() const;
00083
00085
          uint32_t repeatCount() const;
00086
00088
          void setRepeatCount(uint32_t v);
00089
00090 public: // Main interface
          Modbus::StatusCode readCoils(ModbusObject *client, uint8_t unit, uint16_t offset, uint16_t count,
00092
      void *values);
00093
00095
          Modbus::StatusCode readDiscreteInputs(ModbusObject *client, uint8 t unit, uint16 t offset,
      uint16_t count, void *values);
00096
00098
          Modbus::StatusCode readHoldingRegisters(ModbusObject *client, uint8_t unit, uint16_t offset,
      uint16_t count, uint16_t *values);
00099
00101
          Modbus::StatusCode readInputRegisters(ModbusObject *client, uint8_t unit, uint16_t offset,
      uint16_t count, uint16_t *values);
00102
          Modbus::StatusCode writeSingleCoil(ModbusObject *client, uint8_t unit, uint16_t offset, bool
00104
00105
00107
          Modbus::StatusCode writeSingleRegister(ModbusObject *client, uint8_t unit, uint16_t offset,
      uint16_t value);
00108
00110
          Modbus::StatusCode readExceptionStatus(ModbusObject *client, uint8_t unit, uint8_t *value);
00111
00113
          Modbus::StatusCode writeMultipleCoils(ModbusObject *client, uint8_t unit, uint16_t offset,
      uint16_t count, const void *values);
00114
00116
          Modbus::StatusCode writeMultipleRegisters(ModbusObject *client, uint8 t unit, uint16 t offset,
      uint16_t count, const uint16_t *values);
00117
          Modbus::StatusCode readCoilsAsBoolArray(ModbusObject *client, uint8_t unit, uint16_t offset,
00119
      uint16_t count, bool *values);
00120
          Modbus::StatusCode readDiscreteInputsAsBoolArray(ModbusObject *client, uint8 t unit, uint16 t
00122
      offset, uint16_t count, bool *values);
00123
          Modbus::StatusCode writeMultipleCoilsAsBoolArray(ModbusObject *client, uint8_t unit, uint16_t
00125
      offset, uint16_t count, const bool *values);
00126
00127 public: // Modbus Interface
00128
          Modbus::StatusCode readCoils(uint8 t unit, uint16 t offset, uint16 t count, void *values)
      override;
00129
          Modbus::StatusCode readDiscreteInputs(uint8_t unit, uint16_t offset, uint16_t count, void *values)
00130
          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
00131
      *values) override;
00132
          Modbus::StatusCode writeSingleCoil(uint8_t unit, uint16_t offset, bool value) override;
00133
          Modbus::StatusCode writeSingleRegister(uint8_t unit, uint16_t offset, uint16_t value) override;
00134
          Modbus::StatusCode readExceptionStatus(uint8_t unit, uint8_t *value) override;
```

```
00135
          Modbus::StatusCode writeMultipleCoils(uint8_t unit, uint16_t offset, uint16_t count, const void
      *values) override;
00136
         Modbus::StatusCode writeMultipleRegisters(uint8_t unit, uint16_t offset, uint16_t count, const
      uint16_t *values) override;
00137
00138 public:
00140
          inline Modbus::StatusCode readCoilsAsBoolArray(uint8_t unit, uint16_t offset, uint16_t count, bool
      *values) { return readCoilsAsBoolArray(this, unit, offset, count, values); }
00141
00143
         inline Modbus::StatusCode readDiscreteInputsAsBoolArray(uint8_t unit, uint16_t offset, uint16_t
      count, bool *values) { return readDiscreteInputsAsBoolArray(this, unit, offset, count, values); }
00144
          inline Modbus::StatusCode writeMultipleCoilsAsBoolArray(uint8_t unit, uint16_t offset, uint16_t
00146
      count, const bool *values) { return writeMultipleCoilsAsBoolArray(this, unit, offset, count, values);
00147
00148 public:
00150
          Modbus::StatusCode lastStatus() const;
00153
          Modbus::StatusCode lastErrorStatus() const;
00154
00156
          const Modbus::Char *lastErrorText() const;
00157
00158 public:
          const ModbusObject *currentClient() const;
00160
00161
00167
          RequestStatus getRequestStatus(ModbusObject *client);
00168
00170
          void cancelRequest(ModbusObject *client);
00171
00172 public: // SIGNALS
00174
          void signalOpened(const Modbus::Char *source);
00175
00177
          void signalClosed(const Modbus::Char *source);
00178
00180
          void signalTx(const Modbus::Char *source, const uint8_t* buff, uint16_t size);
00181
00183
          void signalRx(const Modbus::Char *source, const uint8_t* buff, uint16_t size);
00184
00186
          void signalError(const Modbus::Char *source, Modbus::StatusCode status, const Modbus::Char *text);
00187
00188 private:
          Modbus::StatusCode request(uint8_t unit, uint8_t func, uint8_t *buff, uint16_t szInBuff, uint16_t
00189
     maxSzBuff, uint16_t *szOutBuff);
00190
         Modbus::StatusCode process();
00191
          friend class ModbusClient;
00192 };
00193
00194 #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**

 #define MODBUSLIB\_VERSION ((MODBUSLIB\_VERSION\_MAJOR<<16)|(MODBUSLIB\_VERSION\_← MINOR<<8)|(MODBUSLIB VERSION PATCH))</li>

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) cstr

Macro for creating string literal, must be used like: StringLiteral ("Some string")

· #define CharLiteral(cchar) cchar

Macro for creating char literal, must be used like: 'CharLiteral('A')'.

• #define GET\_BIT(bitBuff, bitNum) ((((const uint8\_t\*)(bitBuff))[(bitNum)/8] & (1<<((bitNum)%8))) != 0)

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\_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\_VALUE\_BUFF\_SZ** 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\_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)

## **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 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 {

 $Modbus:: Memory\_Unknown = 0xFFFF \ , \ Modbus:: Memory\_0x = 0 \ , \ Modbus:: Memory\_Coils = Memory\_0x \ , \\ Modbus:: Me$ 

 $Modbus::Memory_1x = 1$ ,

 $Modbus:: Memory\_DiscreteInputs = Memory\_1x \ , \ Modbus:: Memory\_3x = 3 \ , \ Modbus:: Memory\_InputRegisters \ , \ Modbus:: Memory\_1x \ , \ Modb$ 

= 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 BadSerialOpen = Status Bad | 0x201 ,
 Modbus::Status BadSerialWrite , Modbus::Status BadSerialRead , Modbus::Status BadAscMissColon =
 Status Bad | 0x301, Modbus::Status BadAscMissCrLf,
 Modbus::Status BadAscChar, 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_BadTcpAccept, 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

 }
     FlowControl Parity for serial port.
```

# **Functions**

- bool Modbus::StatusIsProcessing (StatusCode status)
- bool Modbus::StatusIsGood (StatusCode status)
- bool Modbus::StatusIsBad (StatusCode status)
- 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 uint16 t Modbus::crc16 (const uint8 t \*byteArr, uint32 t count)
- MODBUS\_EXPORT uint8\_t Modbus::lrc (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)
- MODBUS\_EXPORT StatusCode Modbus::writeMemRegs (uint32\_t offset, uint32\_t count, const void \*values, void \*memBuff, uint32\_t memRegCount)
- MODBUS\_EXPORT StatusCode Modbus::readMemBits (uint32\_t offset, uint32\_t count, void \*values, const void \*memBuff, uint32\_t memBitCount)
- MODBUS\_EXPORT StatusCode Modbus::writeMemBits (uint32\_t offset, uint32\_t count, const void \*values, void \*memBuff, uint32\_t memBitCount)

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 strmaxlen)
- MODBUS EXPORT Char \* Modbus::sascii (const uint8 t \*buff, uint32 t count, Char \*str, uint32 t strmaxlen)
- MODBUS EXPORT Timer Modbus::timer ()
- 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 GET BITS

#define GET\_BITS(

```
bitBuff,
bitNum,
bitCount,
boolBuff )
Value:
for (uint16_t __i_ = 0; __i_ < bitCount; __i_++)
```

Macro for get bits begins with number bitNum with count from input bit array bitBuff to output bool array boolBuff.

boolBuff[\_\_i\_\_] = (((const uint8\_t\*)(bitBuff))[((bitNum)+\_\_i\_\_)/8] & (1«(((bitNum)+\_\_i\_\_)%8))) != 0;

# 8.12.2.2 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)

8.13 ModbusGlobal.h

## 8.12.2.3 SET\_BIT

Macro for set bit value with number bitNum to array bitBuff.

#### 8.12.2.4 SET BITS

Macro for set bits begins with number bitNum with count from input bool array boolBuffto output bit array bitBuff.

# 8.13 ModbusGlobal.h

```
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
\tt 00027 \texttt{ \#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
```

```
00034 #if defined(MODBUS_EXPORTS) && defined(MB_DECL_EXPORT)
00035 #define MODBUS_EXPORT MB_DECL_EXPORT
00036 #elif defined(MB_DECL_IMPORT)
00037 #define MODBUS_EXPORT MB_DECL_IMPORT
00038 #else
00039 #define MODBUS_EXPORT
00040 #endif
00041
00043 #define StringLiteral(cstr) cstr
00044
00046 #define CharLiteral(cchar) cchar
00047
00048 //
00049 // ----- Helper macros
00050 //
00051
00053 \# define \ GET\_BIT(bitBuff, \ bitNum) \ ((((const \ uint8\_t*)(bitBuff))[(bitNum)/8] \& (1 \\ ((bitNum) \\ \$8))) \ != \ 0)
00054
00056 #define SET_BIT(bitBuff, bitNum, value)
00057
          if (value)
              ((uint8_t*)(bitBuff))[(bitNum)/8] |= (1«((bitNum)%8));
00059
00060
              ((uint8 t*) (bitBuff)) [(bitNum) /8] &= (~(1 ((bitNum) %8)));
00061
00063 #define GET_BITS(bitBuff, bitNum, bitCount, boolBuff)
00064
          for (uint16_t __i_ = 0; __i_ < bitCount; __i_++)
              boolBuff[__i__] = (((const uint8_t*)(bitBuff))[((bitNum)+__i__)/8] & (1«(((bitNum)+__i__)%8)))
00065
      != 0;
00066
00068
      #define SET_BITS(bitBuff, bitNum, bitCount, boolBuff)
00069
          for (uint16_t __i_ = 0; __i_ < bitCount; __i_++)
00070
              if (boolBuff[ i 1)
00071
                  ((uint8_t*)(bitBuff))[((bitNum)+__i__)/8] |= (1«(((bitNum)+__i__)%8));
00072
00073
                 ((uint8_t*)(bitBuff))[((bitNum)+__i__)/8] &= (~(1«(((bitNum)+__i__)%8)));
00074
00075
00076 //
00077 // ----- Modbus function codes
00078 //
00079
00083 #define MBF_READ_COILS
00084 #define MBF_READ_DISCRETE_INPUTS
00085 #define MBF_READ_HOLDING_REGISTERS
00086 #define MBF_READ_INPUT_REGISTERS
00087 #define MBF_WRITE_SINGLE_COIL
00088 #define MBF_WRITE_SINGLE_REGISTER
00089 #define MBF_READ_EXCEPTION_STATUS
00090 #define MBF_DIAGNOSTICS
00091 #define MBF_GET_COMM_EVENT_COUNTER 00092 #define MBF_GET_COMM_EVENT_LOG
                                                       11
12
00093 #define MBF_WRITE_MULTIPLE_COILS
                                                       15
00094 #define MBF_WRITE_MULTIPLE_REGISTERS
00095 #define MBF_REPORT_SERVER_ID
                                                       17
00096 #define MBF_READ_FILE_RECORD
00097 #define MBF_WRITE_FILE_RECORD
00098 #define MBF_MASK_WRITE_REGISTER
                                                       22
00099 #define MBF_READ_WRITE_MULTIPLE_REGISTERS
                                                       23
00100 #define MBF_READ_FIFO_QUEUE
00101 #define MBF_ENCAPSULATED_INTERFACE_TRANSPORT
                                                       43
00102 #define MBF_ILLEGAL_FUNCTION
00103 #define MBF_EXCEPTION
00105
00106
00107 //
               ----- Modbus count constants
00108 // ---
00109 //
```

8.13 ModbusGlobal.h

```
00110
00112 #define MB_BYTE_SZ_BITES 8
00113
00115 #define MB_REGE_SZ_BITES 16
00116
00118 #define MB_REGE_SZ_BYTES 2
00119
00121 #define MB_VALUE_BUFF_SZ 255
00122
00124 #define MB MAX REGISTERS 127
00125
00127 #define MB_MAX_DISCRETS 2040
00128
00131
00133 #define MB_RTU_IO_BUFF_SZ 264
00134
00136 #define MB ASC IO BUFF SZ 529
00137
00139 #define MB_TCP_IO_BUFF_SZ 268
00140
00141 #ifdef __cplusplus
00142
00143 namespace Modbus {
00144
00145 #ifdef QT_CORE_LIB
00146 Q_NAMESPACE
00147 #endif
00148
00149 #endif // __cplusplus
00150
00152 typedef void* Handle;
00153
00155 typedef char Char;
00156
00158 typedef uint32_t Timer;
00159
00161 enum Constants
00162 {
00163
          VALID_MODBUS_ADDRESS_BEGIN = 1
00164
         VALID_MODBUS_ADDRESS_END = 247,
STANDARD_TCP_PORT = 502
00165
         STANDARD_TCP_PORT
00166 };
00167
00168 //====== Modbus protocol types =======
00171 typedef enum _MemoryType
00172 {
00173
          Memory_Unknown = 0xFFFF,
00174
         Memory_0x = 0,
         Memory_Coils = Memory_0x,
00175
00176
         Memory_1x = 1,
00177
         Memory_DiscreteInputs = Memory_1x,
00178
         Memory_3x = 3,
00179
         Memory_InputRegisters = Memory_3x,
00180
         Memory_4x = 4,
         Memory_HoldingRegisters = Memory_4x,
00181
00182 } MemoryType;
00183
00185 #ifdef __cplusplus // Note: for Qt/moc support
00186 enum StatusCode
00187 #else
00188 typedef enum _StatusCode
00189 #endif
00190 {
00191
          Status_Processing
                                         = 0x80000000
00192
         Status_Good
                                          = 0 \times 000000000
00193
         Status Bad
                                          = 0 \times 01000000.
         Status_Uncertain
00194
                                          = 0 \times 02000000
00195
00196
         //---- Modbus standart errors begin -----
00197
          // from 0 to 255 \,
00198
          Status_BadIllegalFunction
                                                       = Status_Bad | 0x01,
00199
          Status_BadIllegalDataAddress
                                                       = Status_Bad |
                                                                        0x02,
00200
                                                       = Status Bad |
          Status_BadIllegalDataValue
                                                                        0x03,
00201
          Status_BadServerDeviceFailure
                                                       = Status_Bad |
                                                                        0x04,
00202
          Status_BadAcknowledge
                                                       = Status_Bad |
00203
          Status_BadServerDeviceBusy
                                                       = Status_Bad |
                                                                        0x06,
                                                      = Status_Bad |
= Status_Bad |
00204
          Status_BadNegativeAcknowledge
                                                                        0x07,
00205
          Status_BadMemoryParityError
                                                                        0x08.
          Status_BadGatewayPathUnavailable
                                                       = Status_Bad |
00206
                                                                        0 \times 0 A.
          Status_BadGatewayTargetDeviceFailedToRespond = Status_Bad | 0x0B,
00207
00208
          //---- Modbus standart errors end ---
00209
00210
          //---- Modbus common errors begin -----
                                      = Status_Bad | 0x101,
00211
          Status_BadEmptyResponse
          Status_BadNotCorrectRequest
00212
00213
          Status_BadNotCorrectResponse
```

```
00214
           Status_BadWriteBufferOverflow
00215
          Status_BadReadBufferOverflow
00216
00217
           //---- Modbus common errors end -----
00218
00219
           //-- Modbus serial specified errors begin --
                                    = Status_Bad | 0x201,
00220
           Status_BadSerialOpen
00221
           Status_BadSerialWrite
00222
           Status_BadSerialRead
00223
          //--- Modbus serial specified errors end ---
00224
00225
          //--- Modbus ASC specified errors begin ----
00226
          Status_BadAscMissColon = Status_Bad | 0x301,
00227
           Status_BadAscMissCrLf
00228
           Status_BadAscChar
00229
          Status_BadLrc
          //--- Modbus ASC specified errors end ----
00230
00231
          //--- Modbus RTU specified errors begin ---
00233
          Status_BadCrc
                                            = Status_Bad | 0x401,
          //---- Modbus RTU specified errors end ---
00234
00235
00236
           //--_ Modbus TCP specified errors begin --
                                          = Status_Bad | 0x501,
00237
          Status_BadTcpCreate
00238
          Status_BadTcpConnect,
00239
          Status_BadTcpWrite,
00240
          Status_BadTcpRead,
00241
          Status_BadTcpBind,
00242
          Status_BadTcpListen,
          Status_BadTcpAccept,
00243
00244
          Status_BadTcpDisconnect,
00245
          //--- Modbus TCP specified errors end ---
00246 }
00247 #ifdef __cplusplus
00248 ;
00249 #else
00250 StatusCode;
00251 #endif
00252
00254 #ifdef __cplusplus // Note: for Qt/moc support
00255 enum ProtocolType
00256 #else
00257 typedef enum _ProtocolType
00258 #endif
00259 {
00260
00261
          RTU,
00262
          TCP
00263 }
00264 #ifdef __cplusplus
00265;
00266 #else
00267 ProtocolType;
00268 #endif
00269
00270
00272 #ifdef __cplusplus // Note: for Qt/moc support
00273 enum Parity
00274 #else
00275 typedef enum _Parity
00276 #endif
00277 {
0027/ (
00278 NoParity,
00279 EvenParity,
00280 OddParity,
00281 SpaceParity,
00282 MarkParity
00284 #ifdef __cplusplus
00285;
00286 #else
00287 Parity;
00288 #endif
00289
00290
00292 #ifdef __cplusplus // Note: for Qt/moc support
00293 enum StopBits
00294 #else
00295 typedef enum _StopBits
00296 #endif
00297 {
00298
          OneStop,
00299
          OneAndHalfStop,
00300
          TwoStop
00301 }
00302 #ifdef __cplusplus
00303;
```

8.13 ModbusGlobal.h

```
00304 #else
00305 StopBits;
00306 #endif
00307
00309 #ifdef
               cplusplus // Note: for Ot/moc support
00310 enum FlowControl
00311 #else
00312 typedef enum _FlowControl
00313 #endif
00314 {
          NoFlowControl.
00315
00316
          HardwareControl.
00317
          SoftwareControl
00318 }
00319 #ifdef __cplusplus
00320 ;
00321 #else
00322 FlowControl;
00323 #endif
00324
00325 #ifdef OT CORE LIB
00326 Q_ENUM_NS(StatusCode)
00327 Q_ENUM_NS(ProtocolType)
00328 O ENUM NS(Parity)
00329 Q_ENUM_NS(StopBits)
00330 Q_ENUM_NS(FlowControl)
00331 #endif
00332
00334 typedef struct
00335 {
00336
          const Char *portName
00337
          int32_t baudRate
00338
                      dataBits
          int8_t
00339
          Parity
                      parity
00340
          StopBits
                       stopBits
          FlowControl flowControl
00341
                     timeoutFirstByte;
timeoutInterByte;
00342
          uint32 t
00343
          uint32_t
00344 } SerialSettings;
00345
00347 typedef struct
00348 {
          const Char *host
00349
00350
          uint16_t
                      port
          uint16_t
00351
                       timeout;
00352 } TcpSettings;
00353
00356 #endif
00359 inline bool StatusIsProcessing(StatusCode status) { return status == Status_Processing; }
00360
00362 inline bool StatusIsGood(StatusCode status) { return status == Status_Good; }
00363
00365 inline bool StatusIsBad(StatusCode status) { return (status & Status Bad) != 0; }
00368 inline bool StatusIsUncertain(StatusCode status) { return (status & Status Uncertain) != 0; }
00369
00371 inline bool StatusIsStandardError(StatusCode status) { return (status & Status_Bad) && ((status &
      0xFF00) == 0);}
00372
00374 inline bool getBit(const void *bitBuff, uint16_t bitNum) { return GET_BIT (bitBuff, bitNum); }
00375
00377 inline bool getBitS(const void *bitBuff, uint16_t bitNum, uint16_t maxBitCount) { return (bitNum <
     maxBitCount) ? getBit(bitBuff, bitNum) : false; }
00378
00380 inline void setBit(void *bitBuff, uint16 t bitNum, bool value) { SET BIT (bitBuff, bitNum, value) }
00381
00383 inline void setBitS(void *bitBuff, uint16_t bitNum, bool value, uint16_t maxBitCount) { if (bitNum <
      maxBitCount) setBit(bitBuff, bitNum, value); }
00384
00388 inline bool *getBits(const void *bitBuff, uint16_t bitNum, uint16_t bitCount, bool *boolBuff) {
      GET_BITS(bitBuff, bitNum, bitCount, boolBuff) return boolBuff; }
00389
00392 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; }
00393
00397 inline void *setBits(void *bitBuff, uint16_t bitNum, uint16_t bitCount, const bool *boolBuff) {
    SET_BITS(bitBuff, bitNum, bitCount, boolBuff) return bitBuff; }
00401 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; }
00402
00405 MODBUS EXPORT uint16 t crc16(const uint8 t *byteArr, uint32 t count);
```

```
00406
00409 MODBUS_EXPORT uint8_t lrc(const uint8_t *byteArr, uint32_t count);
00410
00417 MODBUS_EXPORT StatusCode readMemRegs(uint32_t offset, uint32_t count, void *values, const void
      *memBuff, uint32_t memRegCount);
00418
00425 MODBUS_EXPORT StatusCode writeMemRegs (uint32_t offset, uint32_t count, const void *values, void
      *memBuff, uint32_t memRegCount);
00426
00433 MODBUS_EXPORT StatusCode readMemBits(uint32_t offset, uint32_t count, void *values, const void
      *memBuff, uint32_t memBitCount);
00434
00441 MODBUS_EXPORT StatusCode writeMemBits(uint32_t offset, uint32_t count, const void *values, void
      *memBuff, uint32_t memBitCount);
00442
00450 MODBUS_EXPORT uint32_t bytesToAscii(const uint8_t* bytesBuff, uint8_t* asciiBuff, uint32_t count);
00451
00459 MODBUS_EXPORT uint32_t asciiToBytes(const uint8_t* asciiBuff, uint8_t* bytesBuff, uint32_t count);
00460
00462 MODBUS_EXPORT Char *sbytes(const uint8_t* buff, uint32_t count, Char *str, uint32_t strmaxlen);
00463
00465 MODBUS_EXPORT Char *sascii(const uint8_t* buff, uint32_t count, Char *str, uint32_t strmaxlen);
00466
00468 MODBUS EXPORT Timer timer();
00469
00471 MODBUS_EXPORT void msleep(uint32_t msec);
00472
00473 #ifdef __cplusplus
00474 } //extern "C"
00475 #endif
00476
00477 #ifdef __cplusplus
00478 } //namespace Modbus
00479 #endif
00480
00481 #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

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

# **Typedefs**

```
    template < class T , class ReturnType , class ... Args > using ModbusMethodPointer = ReturnType(T::*)(Args...)
```

ModbusMethodPointer-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** 

march

Date

May 2024

# 8.15 ModbusObject.h

```
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** 

march

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
          bool isNonBlocking() const;
00064
00065
00066 public: // errors
00068
          Modbus::StatusCode lastErrorStatus() const;
00069
00071
          const Modbus::Char *lastErrorText() const;
00072
00073 public:
00075
          virtual Modbus::StatusCode writeBuffer(uint8_t unit, uint8_t func, uint8_t *buff, uint16_t
      szInBuff) = 0;
00076
00078
          virtual Modbus::StatusCode readBuffer(uint8_t &unit, uint8_t &func, uint8_t *buff, uint16_t
     maxSzBuff, uint16_t *szOutBuff) = 0;
00079
00081
          virtual Modbus::StatusCode write() = 0;
00082
00084
          virtual Modbus::StatusCode read() = 0;
00085
00086 public: // buffer
          virtual const uint8_t *readBufferData() const = 0;
00088
00089
00091
          virtual uint16_t readBufferSize() const = 0;
00092
00094
          virtual const uint8_t *writeBufferData() const = 0;
00095
00097
          virtual uint16 t writeBufferSize() const = 0;
00098
00099 protected:
         Modbus::StatusCode setError(Modbus::StatusCode status, const Modbus::Char *text);
00102
00103 protected:
00105
         ModbusPortPrivate *d_ptr;
          ModbusPort(ModbusPortPrivate *d);
00106
00108 };
00109
00110 #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

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

## **Functions**

- 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)

template < class EnumType >

EnumType Modbus::enumValue (const QVariant &value, bool \*ok)

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 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)

8.21 ModbusQt.h 161

- 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)
- 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 <OMetaEnum>
00014 #include <QHash>
00015 #include <QVariant>
00016
00017 namespace Modbus {
00018
00020 typedef QHash<QString, QVariant> Settings;
00021
00024 class MODBUS_EXPORT Strings
00025 {
00026 public:
00027
        const QString unit
00028
         const QString type
00029
         const OString host
00030
         const QString port
         const QString timeout
00031
00032
         const QString serialPortName
00033
         const QString baudRate
         const QString dataBits const QString parity
00034
00035
00036
         const QString stopBits
00037
         const QString flowControl
00038
          const QString timeoutFirstByte;
00039
          const QString timeoutInterByte;
00040
00042
          Strings():
00043
00045
          static const Strings &instance();
```

```
00046 };
00047
00050 class MODBUS_EXPORT Defaults
00051 {
00052 public:
00053
         const uint8_t
                             unit
          const ProtocolType type
00055
         const QString
                             port
00056
         const uint16_t
00057
         const uint32_t
                             timeout
00058
                             serialPortName
         const QString
00059
         const int32 t
                             baudRate
00060
         const int8_t
                             dataBits
                             parity
00061
         const Parity
00062
          const StopBits
                             stopBits
00063
          const FlowControl flowControl
                             timeoutFirstByte:
00064
          const uint32 t
00065
                             timeoutInterByte;
         const uint32 t
00066
00068
         Defaults();
00069
00071
          static const Defaults &instance();
00072 };
00073
00076 class MODBUS_EXPORT Address
00077 {
00078 public:
00080
         Address();
00081
00083
         Address(Modbus::MemoryType, quint16 offset);
00084
00087
         Address (quint 32 adr);
00088
00089 public:
00091
         inline bool isValid() const { return m_type != Memory_Unknown; }
00092
00094
          inline MemoryType type() const { return static cast<MemoryType>(m type); }
00097
          inline quint16 offset() const { return m_offset; }
00098
00100
          inline quint32 number() const { return m_offset+1; }
00101
          OString toString() const:
00104
00105
          inline operator quint32 () const { return number() | (m_type«16); }
00108
00109
00111
         Address & operator = (quint32 v);
00112
00113 private:
00114
         quint16 m_type;
00115
         quint16 m_offset;
00116 };
00117
00119 inline Address addressFromString(const QString &s) { return Address(s.toUInt()); }
00120
00122 template <class EnumType>
00123 inline QString enumKey(int value)
00124 {
00125
          const QMetaEnum me = QMetaEnum::fromType<EnumType>();
00126
          return QString(me.valueToKey(value));
00127 }
00128
00130 template <class EnumType>
00131 inline QString enumKey(EnumType value, const QString &byDef = QString())
00132 {
00133
          const QMetaEnum me = QMetaEnum::fromType<EnumType>();
00134
         const char *key = me.valueToKey(value);
00135
          if (key)
00136
             return OString(me.valueToKev(value));
00137
          else
00138
             return byDef;
00139 }
00140
00142 template <class EnumType>
00143 inline EnumType enumValue(const QString& key, bool* ok = nullptr)
00144 {
00145
          const QMetaEnum me = QMetaEnum::fromType<EnumType>();
00146
         return static_cast<EnumType>(me.keyToValue(key.toLatin1().constData(), ok));
00147
00148 }
00149
00152 template <class EnumType>
00153 inline EnumType enumValue(const QVariant& value, bool *ok)
00154 {
00155
         bool okInner;
00156
         int v = value.toInt(&okInner);
00157
         if (okInner)
```

```
00158
         {
00159
              const QMetaEnum me = QMetaEnum::fromType<EnumType>();
00160
              if (me.valueToKey(v)) // check value exists
00161
              {
00162
                  if (ok)
00163
                      *ok = true;
00164
                  return static_cast<EnumType>(v);
00165
00166
                  *ok = false;
00167
00168
              return static_cast<EnumType>(-1);
00169
00170
         return enumValue<EnumType>(value.toString(), ok);
00171 }
00172
00175 template <class EnumType>
00176 inline EnumType enumValue(const QVariant& value, EnumType defaultValue)
00177 {
          bool okInner;
00179
         EnumType v = enumValue<EnumType>(value, &okInner);
00180
         if (okInner)
              return v;
00181
         return defaultValue;
00182
00183 }
00184
00186 template <class EnumType>
00187 inline EnumType enumValue(const QVariant& value)
00188 {
00189
          return enumValue<EnumType>(value, nullptr);
00190 }
00191
00194 MODBUS_EXPORT ProtocolType toProtocolType(const QString &s, bool *ok = nullptr);
00195
00198 MODBUS_EXPORT ProtocolType toProtocolType (const QVariant &v, bool \starok = nullptr);
00199
00202 MODBUS_EXPORT Parity toParity(const QString &s, bool *ok = nullptr);
00203
00206 MODBUS_EXPORT Parity toParity (const QVariant &v, bool *ok = nullptr);
00207
00210 MODBUS_EXPORT StopBits toStopBits(const QString &s, bool *ok = nullptr);
00211
00214 MODBUS EXPORT StopBits toStopBits (const OVariant &v, bool *ok = nullptr);
00215
00218 MODBUS_EXPORT FlowControl toFlowControl (const QString &s, bool *ok = nullptr);
00222 MODBUS_EXPORT FlowControl toFlowControl (const QVariant &v, bool *ok = nullptr);
00223
00225 MODBUS_EXPORT QString toString(StatusCode v);
00226
00228 MODBUS EXPORT OString toString(ProtocolType v);
00231 MODBUS_EXPORT QString toString(Parity v);
00232
00234 MODBUS_EXPORT QString toString(StopBits v);
00235
00237 MODBUS_EXPORT QString toString(FlowControl v);
00240 MODBUS_EXPORT QStringList availableSerialPortList();
00241
00244 MODBUS_EXPORT ModbusPort *createPort(const Settings &settings, bool blocking = false);
00245
00248 MODBUS EXPORT ModbusClientPort *createClientPort(const Settings &settings, bool blocking = false);
00252 MODBUS_EXPORT ModbusServerPort *createServerPort (ModbusInterface *device, const Settings &settings,
      bool blocking = false);
00253
00254 } // namespace Modbus
00255
00256 #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.

```
00001
00008 #ifndef MODBUSRTUPORT_H
00009 #define MODBUSRTUPORT_H
00011 #include "ModbusSerialPort.h"
00012
00019 class MODBUS_EXPORT ModbusRtuPort : public ModbusSerialPort
00020 {
00021 public:
         ModbusRtuPort(bool blocking = false);
00024
00026
         ~ModbusRtuPort();
00027
00028 public:
         Modbus::ProtocolType type() const override { return Modbus::RTU; }
00030
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;
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"
```

8.25 ModbusSerialPort.h 165

#### 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.

#### 8.24.1 Detailed Description

Contains definition of base serial port class.

**Author** 

serhmarch

Date

May 2024

## 8.25 ModbusSerialPort.h

```
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
00028
             const int32_t
                                       baudRate
00029
             const int8_t
                                       dataBits
                                      parity
00030
             const Modbus::Parity
00031
             const Modbus::StopBits
                                        stopBits
00032
             const Modbus::FlowControl flowControl
00033
              const uint32 t
                                        timeoutFirstBvte:
00034
             const uint32_t
                                        timeoutInterByte;
00035
00037
             Defaults();
00038
00040
              static const Defaults &instance();
00041
         } ;
00042
00043 public:
00045
         Modbus::Handle handle() const override;
00046
00048
         Modbus::StatusCode open() override;
00049
00051
         Modbus::StatusCode close() override;
00052
00054
         bool isOpen() const override;
00055
00056 public: // settings
00058
         const Modbus::Char *portName() const;
00059
00061
         void setPortName(const Modbus::Char *portName);
00062
00064
         int32_t baudRate() const;
00065
00067
         void setBaudRate(int32_t baudRate);
00068
00070
         int8 t dataBits() const;
00071
         void setDataBits(int8_t dataBits);
```

166 File Documentation

```
00074
00076
          Modbus::Parity parity() const;
00077
00079
          void setParity(Modbus::Parity parity);
00080
00082
          Modbus::StopBits stopBits() const;
00085
          void setStopBits(Modbus::StopBits stopBits);
00086
00088
          Modbus::FlowControl flowControl() const;
00089
00091
          void setFlowControl(Modbus::FlowControl flowControl);
00092
00094
          uint32_t timeoutFirstByte() const;
00095
00097
          void setTimeoutFirstByte(uint32_t timeout);
00098
00100
          uint32 t timeoutInterByte() const;
00101
00103
          void setTimeoutInterByte(uint32_t timeout);
00104
00105 public:
00106
          const uint8 t *readBufferData() const override;
00107
          uint16 t readBufferSize() const override;
00108
          const uint8_t *writeBufferData() const override;
          uint16_t writeBufferSize() const override;
00110
00111 protected:
00112
         Modbus::StatusCode write() override;
00113
          Modbus::StatusCode read() override;
00114
00115 protected:
00117
         using ModbusPort::ModbusPort;
00119 };
00120
00121 #endif // MODBUSSERIALPORT_H
```

#### 8.26 ModbusServerPort.h

```
00001
00008 #ifndef MODBUSSERVERPORT_H
00009 #define MODBUSSERVERPORT_H
00010
00011 #include "ModbusObject.h"
00021 class MODBUS_EXPORT ModbusServerPort : public ModbusObject
00022 {
00023 public:
00026
         ModbusInterface *device() const;
00027
00028 public: // server port interface
         virtual Modbus::ProtocolType type() const = 0;
00031
00033
          virtual bool isTcpServer() const;
00034
00037
         virtual Modbus::StatusCode open() = 0;
00038
00040
         virtual Modbus::StatusCode close() = 0;
00041
00043
          virtual bool isOpen() const = 0;
00044
00047
          virtual Modbus::StatusCode process() = 0;
00048
00049 public:
00051
         bool isStateClosed() const;
00052
00053 public: // SIGNALS
00055
         void signalOpened(const Modbus::Char *source);
00056
00058
          void signalClosed(const Modbus::Char *source);
00059
00062
          void signalTx(const Modbus::Char *source, const uint8_t* buff, uint16_t size);
00063
00066
          void signalRx(const Modbus::Char *source, const uint8_t* buff, uint16_t size);
00067
00069
          void signalError(const Modbus::Char *source, Modbus::StatusCode status, const Modbus::Char *text);
00071 protected:
00072
          using ModbusObject::ModbusObject;
00073 1:
00074
00075 #endif // MODBUSSERVERPORT_H
```

## 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

march

Date

May 2024

### 8.28 ModbusServerResource.h

```
00001
00008 #ifndef MODBUSSERVERRESOURCE_H
00009 #define MODBUSSERVERRESOURCE_H
00010
00011 #include "ModbusServerPort.h"
00012
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
00046
         Modbus::StatusCode process() override;
00047
00048 protected:
00050
         virtual Modbus::StatusCode processInputData(const uint8_t *buff, uint16_t sz);
00051
00053
         virtual Modbus::StatusCode processDevice();
00054
00055
00057
         virtual Modbus::StatusCode processOutputData(uint8_t *buff, uint16_t &sz);
00058
00059 protected:
00060
          using ModbusServerPort::ModbusServerPort;
00061 };
00062
00063 #endif // MODBUSSERVERRESOURCE_H
```

168 File Documentation

## 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

Defaults class constain default settings values for ModbusTcpPort.

### 8.29.1 Detailed Description

Header file of class ModbusTcpPort.

**Author** 

march

Date

April 2024

## 8.30 ModbusTcpPort.h

```
00001
00008 #ifndef MODBUSTCPPORT H
00009 #define MODBUSTCPPORT_H
00010
00011 #include "ModbusPort.h"
00012
00013 class ModbusTcpSocket;
00014
00021 class MODBUS_EXPORT ModbusTcpPort : public ModbusPort
00022 {
00023 public:
      struct MODBUS_EXPORT Defaults
00026
00027
            00028
00029
                                 port
00030
             const uint32_t
                                 timeout;
00031
00033
00034
00036
             static const Defaults &instance();
00037
         };
00038
00039 public:
00041
         ModbusTcpPort (ModbusTcpSocket *socket, bool blocking = false);
00042
00044
         ModbusTcpPort(bool blocking = false);
00045
00046 public:
00048
         Modbus::ProtocolType type() const override { return Modbus::TCP; }
```

```
00051
         Modbus::Handle handle() const override;
00052
00053
         Modbus::StatusCode open() override;
00054
         Modbus::StatusCode close() override;
00055
         bool isOpen() const override;
00056
00057 public:
00059
         const Modbus::Char *host() const;
00060
00062
         void setHost(const Modbus::Char *host);
00063
00065
         uint16_t port() const;
00066
00068
         void setPort(uint16_t port);
00069
00071
         uint32_t timeout() const;
00072
00074
         void setTimeout(uint32 t timeout);
00076
         void setNextRequestRepeated(bool v) override;
00077
00079
         bool autoIncrement() const;
08000
00081 public:
00082
         const uint8_t *readBufferData() const override;
         uint16_t readBufferSize() const override;
00084
         const uint8_t *writeBufferData() const override;
00085
         uint16_t writeBufferSize() const override;
00086
00087 protected:
       Modbus::StatusCode write() override;
00088
00089
         Modbus::StatusCode read() override;
         Modbus::StatusCode writeBuffer(uint8_t unit, uint8_t func, uint8_t *buff, uint16_t szInBuff)
00091
         Modbus::StatusCode readBuffer(uint8_t &unit, uint8_t &func, uint8_t *buff, uint16_t maxSzBuff,
     uint16_t *szOutBuff) override;
00092
00093 protected:
00094
         using ModbusPort::ModbusPort;
00095 };
00096
00097 #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

170 File Documentation

## 8.32 ModbusTcpServer.h

```
00001
00008 #ifndef MODBUSSERVERTCP H
00009 #define MODBUSSERVERTCP_H
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:
          ModbusTcpServer(ModbusInterface *device);
00041
00042 public:
00044
          uint16_t port() const;
00045
00047
          void setPort(uint16 t port);
00050
          int timeout() const;
00051
00053
          void setTimeout(int timeout);
00054
00055 public:
00057
          Modbus::ProtocolType type() const override { return Modbus::TCP; }
00058
00060
          bool isTcpServer() const override { return true; }
00061
00068
          Modbus::StatusCode open() override;
00069
          Modbus::StatusCode close() override;
00074
00076
          bool isOpen() const override;
00077
00079
          Modbus::StatusCode process() override;
08000
00081 public:
          virtual ModbusServerPort *createTcpPort(ModbusTcpSocket *socket);
00084
00085 public: // SIGNALS
00087
          void signalNewConnection(const Modbus::Char *source);
00088
00090
          void signalCloseConnection(const Modbus::Char *source);
00091
00092 protected:
00094
         ModbusTcpSocket *nextPendingConnection();
00095
00097
          void clearConnections();
00098
00099 protected:
00100
          using ModbusServerPort::ModbusServerPort;
00101 };
00102
00103 #endif // MODBUSSERVERTCP_H
```

## Index

```
_MemoryType
                                                    c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusGlobal.h,
    Modbus, 20
                                                             144, 149
\simModbusAscPort
                                                    c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusObject.h,
    ModbusAscPort, 46
                                                             154, 155
                                                    c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusPlatform.h,
\simModbusObject
    ModbusObject, 71
                                                             157, 158
\simModbusPort
                                                    c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusPort.h,
    ModbusPort, 74
                                                             158, 159
\simModbusRtuPort
                                                    c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusQt.h,
    ModbusRtuPort, 80
                                                             160, 161
\simModbusSlotBase
                                                    c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusRtuPort.h,
    ModbusSlotBase< ReturnType, Args >, 95
                                                             163, 164
                                                    c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusSerialPort.h,
Address
                                                             164, 165
    Modbus::Address, 37, 38
                                                    c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusServerPort.h,
addressFromString
    Modbus, 23
                                                    c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusServerResource.h,
ASC
    Modbus, 21
                                                    c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusTcpPort.h,
asciiToBytes
    Modbus, 23
                                                    c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusTcpServer.h,
asciiToString
                                                             169, 170
    Modbus, 23
                                                    cancelRequest
autoIncrement
                                                        ModbusClientPort, 55
    ModbusTcpPort, 101
                                                    cCliCreate
availableSerialPortList
                                                        cModbus.h, 120
    Modbus, 23
                                                    cCliCreateForClientPort
availableSerialPorts
                                                        cModbus.h, 120
    Modbus, 23
                                                    cCliDelete
                                                        cModbus.h, 120
baudRate
                                                    cCliGetLastPortErrorStatus
    ModbusSerialPort, 83
                                                        cModbus.h. 120
bytesToAscii
                                                    cCliGetLastPortErrorText
    Modbus, 23
                                                        cModbus.h. 120
bytesToString
                                                    cCliGetLastPortStatus
    Modbus, 24
                                                        cModbus.h, 120
                                                    cCliGetObjectName
c:/Users/march/Dropbox/PRJ/ModbusLib/src/cModbus.h,
                                                        cModbus.h, 121
         113.134
                                                    cCliGetPort
c:/Users/march/Dropbox/PRJ/ModbusLib/src/Modbus.h,
                                                        cModbus.h, 121
         137, 139
c:/Users/march/Dropbox/PRJ/ModbusLib/src/Modbus_config.h,
                                                        cModbus.h, 121
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusAscPort.n,____
                                                        'cModbus.h, 121
cModbus.h, 121
c:/Users/march/Dropbox/PRJ/ModbusLib/src/ModbusClientPort.n
cModbus.h, 121
         142, 143
```

cCliSetUnit

M II 1 404	M II 1 100
cModbus.h, 121	cModbus.h, 126
cCpoClose	cCpoWriteSingleCoil
cModbus.h, 122	cModbus.h, 127
cCpoConnectClosed	cCpoWriteSingleRegister
cModbus.h, 122	cModbus.h, 127
cCpoConnectError	cCreateModbusDevice
cModbus.h, 122	cModbus.h, 127
cCpoConnectOpened	cDeleteModbusDevice
cModbus.h, 122	cModbus.h, 127
cCpoConnectRx	clearConnections
cModbus.h, 122	ModbusTcpServer, 107 close
cCpoConnectTx cModbus.h, 122	ModbusClientPort, 55
cCpoCreate	ModbusPort, 75
•	
cModbus.h, 123	ModbusSerialPort, 83
cCpoCreateForPort	ModbusServerPoret, 88
cModbus.h, 123 cCpoDelete	ModbusServerResource, 93
•	ModbusTcpPort, 101 ModbusTcpServer, 107
cModbus.h, 123	•
cCpoDisconnectFunc	cModbus.h
cModbus.h, 123 cCpoGetLastErrorStatus	cCliCreateFarClientPart 120
cModbus.h, 123	cCliCreateForClientPort, 120 cCliDelete, 120
cCpoGetLastErrorText	
•	cCliGetLastPortErrorStatus, 120
cModbus.h, 123	cCliGetLastPortErrorText, 120
cCpoGetLastStatus	cCliGetLastPortStatus, 120
cModbus.h, 124 cCpoGetObjectName	cCliGetObjectName, 121 cCliGetPort, 121
cModbus.h, 124	cCliGetType, 121
cCpoGetRepeatCount	cCliGetUnit, 121
cModbus.h, 124	cClilsOpen, 121
cCpoGetType	cCliSetObjectName, 121
cModbus.h, 124	cCliSetUnit, 121
cCpolsOpen	cCpoClose, 122
cModbus.h, 124	cCpoConnectClosed, 122
cCpoReadCoils	cCpoConnectError, 122
cModbus.h, 124	cCpoConnectOpened, 122
cCpoReadCoilsAsBoolArray	cCpoConnectRx, 122
cModbus.h, 124	cCpoConnectTx, 122
cCpoReadDiscreteInputs	cCpoCreate, 123
cModbus.h, 125	cCpoCreateForPort, 123
cCpoReadDiscreteInputsAsBoolArray	cCpoDelete, 123
cModbus.h, 125	cCpoDisconnectFunc, 123
cCpoReadExceptionStatus	cCpoGetLastErrorStatus, 123
cModbus.h, 125	cCpoGetLastErrorText, 123
cCpoReadHoldingRegisters	cCpoGetLastStatus, 124
cModbus.h, 125	cCpoGetObjectName, 124
cCpoReadInputRegisters	cCpoGetRepeatCount, 124
cModbus.h, 125	cCpoGetType, 124
cCpoSetObjectName	cCpolsOpen, 124
cModbus.h, 126	cCpoReadCoils, 124
cCpoSetRepeatCount	cCpoReadCoilsAsBoolArray, 124
cModbus.h, 126	cCpoReadDiscreteInputs, 125
cCpoWriteMultipleCoils	cCpoReadDiscreteInputsAsBoolArray, 125
cModbus.h, 126	cCpoReadExceptionStatus, 125
cCpoWriteMultipleCoilsAsBoolArray	cCpoReadHoldingRegisters, 125
cModbus.h, 126	cCpoReadInputRegisters, 125
cCpoWriteMultipleRegisters	cCpoSetObjectName, 126
	Separation tallo, The

cCpoSetRepeatCount, 126	ModbusObject, 71
cCpoWriteMultipleCoils, 126	Constants
cCpoWriteMultipleCoilsAsBoolArray, 126	Modbus, 20
cCpoWriteMultipleRegisters, 126	cPortCreate
cCpoWriteSingleCoil, 127	cModbus.h, 128
cCpoWriteSingleRegister, 127	cPortDelete
cCreateModbusDevice, 127	cModbus.h, 128
cDeleteModbusDevice, 127	crc16
cPortCreate, 128	Modbus, 24
cPortDelete, 128	cReadCoils
cReadCoils, 128	cModbus.h, 128
cReadCoilsAsBoolArray, 128	cReadCoilsAsBoolArray
cReadDiscreteInputs, 128	cModbus.h, 128
cReadDiscreteInputsAsBoolArray, 129	cReadDiscreteInputs
cReadExceptionStatus, 129	cModbus.h, 128
cReadHoldingRegisters, 129	cReadDiscreteInputsAsBoolArray
cReadInputRegisters, 129	cModbus.h, 129
cSpoClose, 129	cReadExceptionStatus
cSpoConnectCloseConnection, 130	cModbus.h, 129
cSpoConnectClosed, 130	cReadHoldingRegisters
cSpoConnectError, 130	cModbus.h, 129
cSpoConnectNewConnection, 130	cReadInputRegisters
cSpoConnectOpened, 130	cModbus.h, 129
cSpoConnectRx, 130	createClientPort
cSpoConnectTx, 131	Modbus, 24
cSpoCreate, 131	createPort
cSpoDelete, 131	Modbus, 25
cSpoDisconnectFunc, 131	createServerPort
cSpoGetDevice, 131	Modbus, 25
cSpoGetObjectName, 131	createTcpPort
cSpoGetType, 132	ModbusTcpServer, 107
cSpolsOpen, 132	cSpoClose
cSpolsTcpServer, 132	cModbus.h, 129
cSpoOpen, 132	cSpoConnectCloseConnection
cSpoProcess, 132	cModbus.h, 130
cSpoSetObjectName, 132	cSpoConnectClosed
cWriteMultipleCoils, 132	cModbus.h, 130
cWriteMultipleCoilsAsBoolArray, 133	cSpoConnectError
cWriteMultipleRegisters, 133	cModbus.h, 130
cWriteSingleCoil, 133	cSpoConnectNewConnection
cWriteSingleRegister, 133	cModbus.h, 130
pfReadCoils, 116	cSpoConnectOpened
pfReadDiscreteInputs, 116	cModbus.h, 130
pfReadExceptionStatus, 116	cSpoConnectRx
pfReadHoldingRegisters, 117	cModbus.h, 130
pfReadInputRegisters, 117	cSpoConnectTx
pfSlotCloseConnection, 117	cModbus.h, 131
pfSlotClosed, 117	cSpoCreate
pfSlotError, 117	•
pfSlotNewConnection, 118	cModbus.h, 131
•	cSpoDelete
pfSlotOpened, 118	cModbus.h, 131
pfSlotRx, 118	cSpoDisconnectFunc
pfSlotTx, 118	cModbus.h, 131
pfWriteMultipleCoils, 118	cSpoGetDevice
pfWriteMultipleRegisters, 119	cModbus.h, 131
pfWriteSingleCoil, 119	cSpoGetObjectName
pfWriteSingleRegister, 119	cModbus.h, 131
connect	cSpoGetType

cModbus.h, 132	Modbus, 28
cSpolsOpen	getBitS
cModbus.h, 132	Modbus, 28
cSpolsTcpServer	getBits
cModbus.h, 132	Modbus, 28
cSpoOpen	getBitsS
cModbus.h, 132	Modbus, 28
cSpoProcess	getRequestStatus
cModbus.h, 132	ModbusClientPort, 55
cSpoSetObjectName	handle
cModbus.h, 132	
currentClient	ModbusPort, 75 ModbusSerialPort, 83
ModbusClientPort, 55	ModbusTcpPort, 101
cWriteMultipleCoils	HardwareControl
cModbus.h, 132	Modbus, 20
cWriteMultipleCoilsAsBoolArray	host
cModbus.h, 133	ModbusTcpPort, 101
cWriteMultipleRegisters	Wodbus repriort, 101
cModbus.h, 133	instance
cWriteSingleCoil	Modbus::Defaults, 40
cModbus.h, 133 cWriteSingleRegister	Modbus::Strings, 111
cModbus.h, 133	ModbusSerialPort::Defaults, 42
Civioabus.ii, 133	ModbusTcpPort::Defaults, 43
dataBits	ModbusTcpServer::Defaults, 44
ModbusSerialPort, 83	isBlocking
Defaults	ModbusPort, 75
Modbus::Defaults, 40	isChanged
ModbusSerialPort::Defaults, 41	ModbusPort, 75
ModbusTcpPort::Defaults, 42	isNonBlocking
ModbusTcpServer::Defaults, 43	ModbusPort, 75
device	isOpen
ModbusServerPort, 88	ModbusClient, 48
disconnect	ModbusClientPort, 56
ModbusObject, 72	ModbusPort, 75
disconnectFunc	ModbusSerialPort, 83
ModbusObject, 72	ModbusServerPort, 89
•	ModbusServerResource, 93
emitSignal	ModbusTcpPort, 101
ModbusObject, 72	ModbusTcpServer, 107
enumKey	isServerMode
Modbus, 27	ModbusPort, 75
enumValue	isStateClosed
Modbus, 27	ModbusServerPort, 89
EvenParity	isTcpServer
Modbus, 21	ModbusServerPort, 89
exec	ModbusTcpServer, 107
ModbusSlotBase< ReturnType, Args >, 95	isValid
ModbusSlotFunction< ReturnType, Args >, 97	Modbus::Address, 38
ModbusSlotMethod< T, ReturnType, Args >, 98	la at Francis Chatrica
FlourControl	lastErrorStatus
FlowControl Modbus 20	ModbusPort, 76
Modbus, 20 flowControl	ModbusPort, 76
	lastErrorText  ModbusClientPort 56
ModbusSerialPort, 83	ModbusPort, 76
GET BITS	ModbusPort, 76 lastPortErrorStatus
ModbusGlobal.h, 148	
getBit	ModbusClient, 48 lastPortErrorText
90	iastruiteitui text

ModbusClient, 49 IastPortStatus ModbusClient, 49 IastStatus ModbusClient, 49 Immory_0, 20 Modbus, 28 ModbusClientPort, 56 Irc Modbus, 28 ModbusClientPort, 56 Irc Modbus, 28 MarkParity Modbus, 21 MarkParity, 21 Modbus, 21 MarkParity, 21 Modbus, 21 MarkParity, 21 Modbus, 20 MarkParity, 21 Modbus, 20 Mornory_DiscreteInputs, 20 Mornory_DiscreteInputs Modbus, 20 Mornory_OliscreteInputs Modbus, 20 Mornory_DiscreteInputs Mornory_DiscreteInputs, 20 Doeslop, 22 Parity, 21 ProtocoTippe, 21 Sassilia, 30 SetBits, 31 SetBits, 30 SetBits, 30 SetBits, 30 SetBits, 30 SetBits, 31 SetBits, 30 SetBits,		
Memory_Ix, 20	ModbusClient, 49	MarkParity, 21
lastStatus Modbus CilentPort, 56 Irc Modbus, 21 MarkParity Modbus, 21 MB_RTU_IO_BUFF_SZ ModbusGlobalh, 148 Memory_Dx Modbus, 20 Memory_Dx Modbus, 20 Memory_Sx Modbus, 20 Memory_Lrival Sylval	lastPortStatus	Memory_0x, 20
Modbus, 28  MarkParity Modbus, 21  MartU_IO_BUFF_SZ ModbusGlobal.h, 148  Memory J. Northegisters, 20 Memory_Unknown, 20 msleep, 29 Modbus, 20 Memory_Ix Modbus, 20 Memory_Ix Modbus, 20 Memory_Sx Modbus, 20 Memory_Sx Modbus, 20 Memory_Loking Memory_Sx Modbus, 20 Memory_Loking Memory_Coils Modbus, 20 Memory_Loking Memory_Coils Memory_Loking Memory_Loking Memory_Loking Memory_Loking Memory_Loking Memory_Loking Memory_Loking Memory_Loking Memory_Loking Memory_Unknown, 20 Memory_Loking Memory_Doking Memory_Loking	ModbusClient, 49	Memory_1x, 20
Inc Modbus, 28  MarkParity Modbus, 21  MB RTU IO BUFF SZ ModbusGlobal In, 148  Memory Ox Modbus, 20  Memory_1x Modbus, 20  Memory_1x Modbus, 20  Memory_Lax Modbus, 20  Memory_Lax Modbus, 20  Memory_Discretelinputs Modbus, 20	lastStatus	Memory_3x, 20
Morbus, 28  MarkParily Morbus, 21  Memory, DiscreteInputs, 20 Morbus, 20 Memory, DiscreteInputs, 21 Morbus, 20 Memory, Andobus, 20 Memory, Andobus, 20 Memory, Andobus, 20 Memory, Andobus, 20 Memory, Lotis Morbus, 20 Memory, Lotis Morbus, 20 Memory, DiscreteInputs Morbus, 20 Memory, DiscreteInputs, 20 AndHalfStop, 22 Parily, 21 Sascii, 30 SetBits, 30 S	ModbusClientPort, 56	Memory_4x, 20
MarkParily Modbus, 21  MB_RTU_IO_BUFF_SZ ModbusGlobal.h, 148 Memory_Ox Modbus, 20 Memory_Ix Modbus, 20 Memory_Ix Modbus, 20 Memory_Ax Modbus, 20 Memory_Borrelelinputs Modbus, 20 Memory_HoldingRegisters Modbus, 20 Memory_Discretelinputs Modbus, 20 Memory_Linknown Modbus, 20 Memory_Linknown Modbus, 20 Memory_InpulRegisters Modbus,	Irc	Memory_Coils, 20
MarRParily Modbus, 21  MB_RTU_IO_BUFF_SZ ModbusGlobal.h, 148 Memory_Ox Modbus, 20 Memory_1x Modbus, 20 Memory_Ax Modbus, 20 Memory_Loblis Modbus, 20 Memory_Discretelinputs Modbus, 20 Memory_Discretelinputs Modbus, 20 Memory_InputRegisters Modbus, 20 Memory_Unknown Modbus, 20 Memory_Unknown Modbus, 20 Memory_Unknown ModbusSiolfMerthod< T, ReturnType, Args >, 95 ModbusSiofffunction	Modbus, 28	Memory_DiscreteInputs, 20
Modbus, 21  MB_RTU_IO_BUFF_SZ ModbusGlobal.h, 148  Memory_Ox Modbus, 20  Memory_1x Modbus, 20  Memory_Ax Modbus, 20  Memory_Coils Modbus, 20  Memory_Discretelipouts Modbus, 20  Memory_Discretelipouts Modbus, 20  Memory_HoldingRegisters Modbus, 20  Memory_Unknown Modbus, 20  Memory_HoldingRegisters Modbus, 20  Memory_Unknown Modbus, 20  Memory_Unknown Modbus, 20  Memory_Unknown Modbus, 20  Memory_Indegisters Modbus, 20  Memory_Unknown Modbus, 20  Memory_Indegisters Modbus, 20  Memory_Discretelipouts Modbus, 20  Memory_Indegisters Modbus, 20  Memory_Discretelipouts Modbus, 20		Memory_HoldingRegisters, 20
MB_RTU_IO_BUFF_SZ ModbusGlobalh, 1,48 Memory_Ox Modbus, 20 Memory_1x Modbus, 20 Memory_Ax Modbus, 20 Memory_Lobic Retelliputs Modbus, 20 Memory_Discretelinputs Modbus, 20 Memory_InputRegisters Modbus, 20 Memory_Unknown ModbusSlotBase <returntype, args="">, 95 ModbusSlotIbase\ReturnType, Args &gt;, 97 ModbusSlotIbase\ReturnType, Args &gt;, 98 Modbus, 17 </returntype,>		Memory_InputRegisters, 20
MB_RTU_IO_BUFF_SZ Modbus(Global, 1, 148 Memory_0x Modbus, 20 Memory_1x Modbus, 20 Memory_4x Modbus, 20 Memory_Coils Modbus, 20 Memory_Coils Modbus, 20 Memory_Coils Modbus, 20 Memory_Discretelnputs Modbus, 20 Memory_Discretelnputs Modbus, 20 Memory_Discretelnputs Modbus, 20 Memory_Discretelnputs Modbus, 20 Memory_Inheligisters Modbus, 20 Memory_Inhe		Memory Unknown, 20
ModbusGlobal.h, 148 Memory Ox Memory Ox Modbus, 20 Memory Jax Modbus, 20 Memory Jax Modbus, 20 Memory Lax Modbus, 20 Memory Lax Modbus, 20 Memory Lax Modbus, 20 Memory Discretelinputs Modbus, 20 Memory Discretelinputs Modbus, 20 Memory HoldingRegisters Modbus, 20 Memory HoldingRegisters Modbus, 20 Memory InputRegisters Modbus, 2		msleep, 29
Memory_Ox Modbus, 20 Memory_1x Modbus, 20 Memory_4x Modbus, 20 Memory_Ax Modbus, 20 Memory_Coils Modbus, 20 Memory_Discretelinputs Modbus, 20 Memory_Discretelinputs Modbus, 20 Memory_InputRegisters Modbus, 20 Memory_Unknown Modbus, 20 methodOrFunction ModbusSlotFunction< Modbus_20 methodOrFunction Modbus_20 methodOrFunction Modbus_20 methodOrFunction Modbus_20 methodOrFunction Andbus_20 methodOrFunction Modbus_20 methodOrFunc	ModbusGlobal.h, 148	•
Modbus, 20 Memory Jax Modbus, 20 Memory_dx Modbus, 20 Memory_Colls Modbus, 20 Memory_Colls Modbus, 20 Memory_Dolls Modbus, 20 Memory_Dolls Modbus, 20 Memory_InputRegisters Modbus, 20 Memory_InputRegisters Modbus, 20 Memory_Unknown Modbus, 20 Memory_Unknown Modbus, 20 Memory_Unknown ModbusSlotBase< ReturnType, Args >, 95 ModbusSlotTenction< ReturnType, Args >, 97 ModbusSlotTenction< ReturnType, Args >, 98 Modbus, 17 MemoryType, 20 addressFromString, 23 ASC, 21asaiToBytes, 23asaiToBytes, 23availableSerialPortList, 23availableSerialPortList, 23availableSerialPort, 24createClientPort, 24createClientPort, 24createClientPort, 25createServerPort, 25enumNey, 27enumValue, 27EvenParity, 21FlowControl, 20getBits, 28getBits, 2	Memory_0x	NoParity, 21
Memory_Tx Modbus, 20 Memory_dx Modbus, 20 Memory_dx Modbus, 20 Memory_Lex Modbus, 20 Memory_Discretelinputs Modbus, 20 Memory_Discretelinputs Modbus, 20 Memory_HoldingRegisters Modbus, 20 Memory_HoldingRegisters Modbus, 20 Memory_InputRegisters Modbus, 20 Memory_Unknown Modbus, 20 Memory_Unknown ModbusStoftFunction <a href="ReturnType">ReturnType</a> , 4rgs >, 95 Modbus, 17 _Memory Type, 20 addressFromString, 23 ASC, 21 asciiToBytes, 23 asciiToString, 23 availableSerialPortList, 23 availableSerialPortList, 23 availableSerialPortList, 23 availableSerialPortList, 23 availableSerialPortList, 23 availableSerialPortList, 23 bytesToString, 24 Constants, 20 crc16, 24 createClientPort, 24 createClientPort, 24 createClientPort, 24 createClientPort, 24 createClientPort, 25 enumKey, 27 enumValue, 27 exemparity, 21 FlowControl, 20 getBit, 28 getBits, 28 getBi	Modbus, 20	•
Modbus, 20 Memory_ax Modbus, 20 Memory_colls Modbus, 20 Memory_Colls Modbus, 20 Memory_Discretelinputs Modbus, 20 Memory_InjutRegisters Modbus, 20 Memory_InjutRegister Modbus, 20 Memory_InjutRegister Modbus, 20 Memory_InjutRegister Modbus, 20 Memory_InjutRegister	Memory_1x	•
Memory_ax Modbus, 20 Memory_dx Modbus, 20 Memory_Colls Modbus, 20 Memory_Discretelnputs Modbus, 20 Memory_HoldingRegisters Modbus, 20 Memory_InputRegisters Modbus, 20 Memory_InputRegister Modbus, 20 Memory_InputRegister Modbus, 20	Modbus, 20	•
Modbus, 20 Memory_4x Modbus, 20 Memory_Colis Modbus, 20 Memory_DiscreteInputs Modbus, 20 Memory_DiscreteInputs Modbus, 20 Memory_InputRegisters Modbus, 20 Memory_InputRegisters Modbus, 20 Memory_InputRegisters Modbus, 20 Memory_Unknown Modbus, 20 Memory_InputRegisters Modbus, 20 SetBits, 30 SetBi	Memory_3x	•
Memory_Ax Modbus, 20 Memory_Coils Modbus, 20 Memory_Discretelinputs Modbus, 20 Memory_HoldingRegisters Modbus, 20 Memory_InputRegisters Modbus, 20 Memory_Unknown Modbus, 20 Memory_Unknown Modbus, 20 Memory_Unknown ModbusSlotBase< ReturnType, Args >, 95 ModbusSlotFunction         setBits, 30 setBits, 20 setBits, 30 setBits, 30 setBits, 30 setBits, 20 setBits, 30 setBits, 30 setBits, 20 setBits, 30 setBits, 20 setBits, 30 setBits, 30 setBits, 20 setBi	Modbus, 20	•
Modbus, 20  Memory_Coils Modbus, 20  Memory_Discretelinputs Modbus, 20  Memory_HoldingRegisters Modbus, 20  Memory_InputRegisters Modbus, 20  Memory_InputRegisters Modbus, 20  Memory_InputRegisters Modbus, 20  Memory_Unknown Modbus, 20  methodOrFunction ModbusSlotBase         setBits, 30	Memory_4x	
Memory Coils Modbus, 20 Memory Discretelnputs Modbus, 20 Memory HoldingRegisters Modbus, 20 Memory InputRegisters Modbus, 20 Memory Linknown Modbus, 20 Memory Linknown Modbus, 20 methodOrFunction ModbusSlottBase< ReturnType, Args >, 95 ModbusSlottBuse< ReturnType, Args >, 97 ModbusSlottBuse Memory Type, 20 addressFromString, 23 ASC, 21 ascilToBytes, 23 ascilToBytes, 23 availableSerialPortList, 23 availableSerialPorts, 23 bytesToAscii, 23 bytesToAscii, 23 bytesToAscii, 23 bytesToAscii, 23 createServerPort, 25 createServerPort, 25 createServerPort, 25 enumKey, 27 enumWalue, 27 EvenParity, 21 FlowControl, 20 gerBits, 28 getBits, 28 g	Modbus, 20	
Modbus, 20  Memory_Discretelnputs Modbus, 20  Memory_HoldingRegisters Modbus, 20  Memory_InpuRegisters Modbus, 20  Memory_Unknown Modbus, 20  methodOrFunction ModbusSlotBase< ReturnType, Args >, 95 ModbusSlotFunction< ReturnType, Args >, 97 ModbusSlotMethod < T, ReturnType, Args >, 97 ModbusSlotMethod < T, ReturnType, Args >, 98  Modbus, 17 Memory_Type, 20 addressFromString, 23 ASC, 21 asciiToBytes, 23 asciiToBytes, 23 availableSerialPortList, 23 availableSerialPortList, 23 bytesToAscii, 23 bytesToAscii, 23 bytesToAscii, 24 Constants, 20 crate_ClientPort, 24 createPort, 25 createServerPort, 25 enumKey, 27 enumValue, 27 EvenParity, 21 FlowControl, 20 getBits, 28 HardwareControl, 20 setBits, 30 setBits, 28 setBits, 30 setBits, 28 setBits, 30 setBits, 29 setBits, 31 softwareControl, 20 SpaceParity, 21 Status_BadAscMscControl, 20 Status_BadAscMscControl, 20 Status_BadAscMisscOton, 22 Status_BadGatewayPathUnavailable, 22 Status_Ba	Memory_Coils	
Memory_DiscreteInputs Modbus, 20  Memory_HoldingRegisters Modbus, 20  Memory_InputRegisters Modbus, 20  Memory_Unknown Modbus, 20  Memory_Unknown Modbus, 20  Memory_Unknown ModbusSlotBase< ReturnType, Args >, 95 ModbusSlotIbunction < ReturnType, Args >, 97 ModbusSlotIbunction < ReturnType, Args >, 97 ModbusSlotMethod< T, ReturnType, Args >, 98  Modbus, 17 MemoryType, 20 addressFromString, 23 ASC, 21 asciiToBytes, 23 asciiToBytes, 23 availableSerialPortList, 23 availableSerialPortList, 23 bytesToAscii, 23 bytesToAscii, 23 bytesToString, 24 Constants, 20 crc16, 24 createClientPort, 24 createClientPort, 25 createServerPort, 25 enumKey, 27 enumValue, 27 EvenParity, 21 FlowControl, 20 getBits, 28 getBits, 2	Modbus, 20	
Modbus, 20  Memory_HoldingRegisters Modbus, 20  Memory_Unknown Modbus, 20  Memory_Unknown Modbus, 20  methodOrFunction  ModbusSlotBase< ReturnType, Args >, 95 ModbusSlotBuse	Memory_DiscreteInputs	
Memory HoldingRegisters Modbus, 20 Memory UnptrRegisters Modbus, 20 Memory Unknown Modbus, 20 methodOrFunction ModbusSlotBase< ReturnType, Args >, 95 ModbusSlotFunction         SpaceParity, 21 STANDARD_TCP_PORT, 20 Status_BadAscknowledge, 21 Status_BadAscKnissColon, 22 Status_BadAscKnissColon, 22 Status_BadAscKnissCrl.f, 22 Status_BadAscMissCrl.f, 22 Status_BadCemptyResponse, 22 Status_BadCatewayPathUnavailable, 22 Status_BadCatewayPathUnavailable, 22 Status_BadCatewayPathUnavailable, 22 Status_BadCatewayPathUnavailable, 22 Status_BadIllegalDataAddress, 21 Status_BadIllegalDataAddress, 21 Status_BadIllegalDataValue, 21 Status_BadIllegalDataValue, 21 Status_BadNemoryParityError, 22 Status_BadNemory	Modbus, 20	-
Modbus, 20  Memory _InputRegisters     Modbus, 20  Memory _Unknown     Modbus, 20  methodOrFunction  ModbusSlotBase< ReturnType, Args >, 95  ModbusSlotFunction < ReturnType, Args >, 97  ModbusSlotMethod < T, ReturnType, Args >, 97  ModbusSlotFunction < ReturnType, Args >, 98  Modbus, 17 MemoryType, 20  addressFromString, 23  ASC, 21 asciiToBytes, 23  availableSerialPortList, 23  availableSerialPortList, 23  availableSerialPorts, 23  bytesToAscii, 23  bytesToAscii, 23  bytesToString, 24  Constants, 20  createPort, 25  createPort, 25  createPort, 25  createPort, 25  enumKey, 27  enumValue, 27  EvenParity, 21  FlowControl, 20  getBits, 28  getBits, 28  getBits, 28  getBits, 28  getBits, 28  HardwareControl, 20  ssetBits, 30  setBits, 30  setBits, 30  setBits, 30  setBits, 31  SoftwareControl, 20  scatWispack Shad, 21  Status Bad, 21  Status BadAscMissCotlor, 22  Status BadAscMissCotlon, 22  Status BadGreveyTeviceFailedToRespond, 22  Status BadCre, 22  Status BadCre, 22  Status BadIllegalDataAddress, 21  Status BadIllegalDataAddress, 21  Status BadIllegalDataAddress, 21  Status BadIllegalPottaAddress, 22  Status BadIllegalPotte,	Memory HoldingRegisters	
Memory_InputRegisters    Modbus, 20 Memory_Unknown    Modbus, 20 methodOrFunction    ModbusSlotBase< ReturnType, Args >, 95    ModbusSlotBase\SeturnType, Args >, 97    ModbusSlotMethod< T, ReturnType, Args >, 98 Modbus, 17   MemoryType, 20    addressFromString, 23    ASC, 21    asciiToSytes, 23    availableSerialPortList, 23    availableSerialPorts, 23    bytesToAscii, 23    bytesToAscii, 23    bytesToString, 24    createClientPort, 24    createClientPort, 25    createServerPort, 25    enumKey, 27    enumValue, 27    EvenParity, 21    FlowControl, 20    SpaceParity, 21    Status_Bad, 21    Status_BadAscMissColon, 22    Status_BadAscMissColon, 22    Status_BadAscMissColon, 22    Status_BadGatewayPathUnavailable, 22    Status_BadGatewayPathUnavailable, 22    Status_BadGatewayTargetDeviceFailedToRespond, 22    Status_BadIllegalDataAddress, 21    Status_BadIllegalDataValue, 21    Status_BadIllegalDataValue, 21    Status_BadNegativeAcknowledge, 22    Status_BadNegativeAcknowledge, 22    Status_BadNotCorrectRequest, 22    Status_BadNotCorrectRequest, 22    Status_BadSerialOpen, 22    Status_BadSerialPride, 21    Status_BadSerialPride, 22    Status_BadSerialPrid	· - · · ·	
Modbus, 20  Memory_Unknown Modbus, 20  methodOrFunction ModbusSlotBase < ReturnType, Args >, 95 ModbusSlotBase < ReturnType, Args >, 97 ModbusSlotMethod < T, ReturnType, Args >, 98  Modbus, 17 MemoryType, 20 addressFromString, 23 ASC, 21 asciiToBytes, 23 availableSerialPortList, 23 availableSerialPorts, 23 bytesToAscii, 23 bytesToAscii, 23 bytesToString, 24 Constants, 20 crc16, 24 createClientPort, 24 createClientPort, 25 enumKey, 27 enumValue, 27 EvenParity, 21 FlowControl, 20 getBit, 28 getBits, 28 HardwareControl, 20 SoftwareControl, 20 SpaceParity, 21 STANDARD TCP_PORT, 20 Status_BadAcknowledge, 21 Status_BadAscChar, 22 Status_BadAscMissColon, 22 Status_BadAscMissColon, 22 Status_BadSerisCrt_f, 22 Status_BadSerisptyResponse, 22 Status_BadGatewayPathUnavailable, 22 Status_BadGatewayTargetDeviceFailedToRespond, 22 Status_BadGatewayTargetDeviceFailedToRespond, 22 Status_BadBlllegalDataAddress, 21 Status_BadlllegalDataAddress, 21 Status_BadlllegalDataValue, 21 Status_BadMemoryParityError, 22 Status_BadNegativeAcknowledge, 22 Status_BadNegativeAck	Memory InputRegisters	
Memory_Unknown Modbus, 20 methodOrFunction ModbusSlotBase< ReturnType, Args >, 95 ModbusSlotBase< ReturnType, Args >, 97 ModbusSlotMethod< T, ReturnType, Args >, 98 Modbus, 17MemoryType, 20 addressFromString, 23 ASC, 21 asciiToString, 23 availableSerialPortList, 23 availableSerialPorts, 23 bytesToAscii, 23 bytesToAscii, 23 createClientPort, 24 createPort, 25 createServerPort, 25 enumKey, 27 enumValue, 27 EvenParity, 21 FlowControl, 20 getBits, 28 HardwareControl, 20  SagaceParity, 21 Status_Badler SpaceParity, 21 Status_BadAcknowledge, 21 Status_BadAcknowledge, 21 Status_BadAscMissColon, 22 Status_BadGerievayPathUnavailable, 22 Status_BadGetewayPathUnavailable, 22 Status_BadGlatewayPathUnavailable, 22 Status_BadGlatewayPathUnavailable, 22 Status_BadIlegalDataAddress, 21 Status_BadIllegalDataAddress, 21 Status_BadIllegalDataValue, 21 Status_BadIllegalDataValue, 21 Status_BadIllegalDataValue, 21 Status_BadNotCorrectRequest, 22 Status_BadNotCorrectRequest, 22 Status_BadNotCorrectRequest, 22 Status_BadSerialPoor, 22 Status_BadSer		
Modbus, 20 methodOrFunction ModbusSlotBase< ReturnType, Args >, 95 ModbusSlotFunction< ReturnType, Args >, 97 ModbusSlotMethod< T, ReturnType, Args >, 98 Modbus, 17MemoryType, 20 addressFromString, 23 ASC, 21 asciiToString, 23 availableSerialPortList, 23 availableSerialPortList, 23 bytesToAscii, 23 bytesToAscii, 20 crc16, 24 createClientPort, 25 createServerPort, 25 enumKey, 27 enumValue, 27 EvenParity, 21 FlowControl, 20 getBits, 28 HardwareControl, 20  Status_BadCknowledge, 21 Status_BadAscMissColon, 22 Status_BadAscMissColon, 22 Status_BadSemissColon, 22 Status_BadSemissColon, 22 Status_BadCrc, 22 Status_BadCrc, 22 Status_BadGatewayPathUnavailable, 22 Status_BadGatewayPathUnavailable, 22 Status_BadGatewayPathUnavailable, 22 Status_BadGatewayPathUnavailable, 22 Status_BadIllegalDataAddress, 21 Status_BadIllegalDataAddress, 21 Status_BadIllegalDataValue, 21 Status_BadMemoryParityError, 22 Status_BadNegativeAcknowledge, 22 Status_BadNegativeAcknow		
methodOrFunction  ModbusSlotBase < ReturnType, Args >, 95  ModbusSlotFunction < ReturnType, Args >, 97  ModbusSlotMethod < T, ReturnType, Args >, 98  Modbus, 17 MemoryType, 20     addressFromString, 23  ASC, 21     asciiToBytes, 23     availableSerialPortList, 23     availableSerialPortLst, 23     availableSerialPortString, 24     Constants, 20     crc16, 24     createClientPort, 25     createServerPort, 25     createServerPort, 25     enumKey, 27     enumValue, 27     EvenParity, 21     getBits, 28     getBits, 28     getBits, 28     getBits, 28     HardwareControl, 20  ModbusSlotMethod < T, ReturnType, Args >, 97     ModbusSlotMethod < T, ReturnType, Args >, 98     Status_BadAscMissColon, 22     Status_BadAscMissColon, 22     Status_BadAscMissColon, 22     Status_BadAscMissColon, 22     Status_BadGatewayPathUnavailable, 22     Status_BadGatewayPathUnavailable, 22     Status_BadGatewayPathUnavailable, 22     Status_BadGatewayTargetDeviceFailedToRespond, 22     Status_BadGatewayTargetDeviceFailedToRespond, 22     Status_BadlllegalDataAddress, 21     Status_BadlllegalDataAddress, 21     Status_BadlllegalDataValue, 21     Status_BadlllegalDataValue, 21     Status_BadNeroryParityError, 22     Status_BadNeroryParityError, 22     Status_BadNotCorrectRequest, 22     Status_BadNotCorrectRequest, 22     Status_BadSerialPore, 22     Status_BadSerialPend, 22     Status_BadSerialPend, 22     Status_BadSerialPred, 22     Status_BadSerialPred, 22     Status_BadSerialPred, 22     Status_BadSerialPred, 22     Status_BadSerialPred, 22     Status_BadTopConnect, 22     Status_BadTopConnect, 22     Status_BadTopConnect, 22     Status_BadTopConnect, 22     Status_BadTopConnect, 22	• —	•
ModbusSlotBase < ReturnType, Args >, 95 ModbusSlotFunction < ReturnType, Args >, 97 ModbusSlotMethod < T, ReturnType, Args >, 98 Modbus, 17 MemoryType, 20 addressFromString, 23 ASC, 21 asciiToBytes, 23 availableSerialPortList, 23 availableSerialPortList, 23 bytesToAscii, 23 bytesToAscii, 23 bytesToString, 24 Constants, 20 crc16, 24 createClientPort, 25 createServerPort, 25 enumKey, 27 enumValue, 27 EvenParity, 21 FlowControl, 20 getBit, 28 getBits, 28 getBits, 28 getBits, 28 HardwareControl, 20  MemoryType, Args >, 97 Status_BadAcknowledge, 21 Status_BadAscChirs, 22 Status_BadAscChirs, 22 Status_BadAscMissColon, 22 Status_BadAcro, 22 Status_BadAcknowledge, 22 Status_BadCrc, 22 Status_BadGre, 22 Status_BadGre, 22 Status_BadGatewayPathUnavailable, 22 Status_BadGre, 22 Status_BadGre, 22 Status_BadGre, 22 Status_BadGre, 22 Status_BadNegatevayPathUnavailable, 22 Status_BadCrc, 22 Status		
ModbusSlotFunction ReturnType, Args >, 97		
ModbusSlotMethod < T, ReturnType, Args >, 98  Modbus, 17  _MemoryType, 20 addressFromString, 23 ASC, 21 asciiToBytes, 23 availableSerialPortList, 23 availableSerialPorts, 23 bytesToString, 24 Constants, 20 crc16, 24 createClientPort, 24 createPort, 25 createServerPort, 25 enumKey, 27 enumValue, 27 EvenParity, 21 FlowControl, 20 getBits, 28 HardwareControl, 20  addressFromString, 22 Status_BadAscMissColon, 22 Status_BadCrc, 22 Status_BadCrc, 22 Status_BadGatewayPathUnavailable, 22 Status_BadGatewayPathUnavailable, 22 Status_BadGatewayTargetDeviceFailedToRespond, 22 Status_BadGatewayTargetDeviceFailedToRespond, 22 Status_BadGatewayTargetDeviceFailedToRespond, 22 Status_BadGatewayTargetDeviceFailedToRespond, 22 Status_BadGatewayTargetDeviceFailedToRespond, 22 Status_BadGatewayPathUnavailable, 22 Status_BadGatewayTargetDeviceFailedToRespond, 22 Status_BadGatewayTargetDeviceFailedToRespond, 22 Status_BadIllegalDataAddress, 21 Status_BadlllegalPunction, 21 Status_BadHegalPunction, 21 Status_BadNegativeAcknowledge, 22 Status_BadNotCorrectRequest, 22 Status_BadNotCorrectRequest, 22 Status_BadNotCorrectRequest, 22 Status_BadNotCorrectRequest, 22 Status_BadSerialPoverflow, 22 Status_BadSerialPoverflow, 22 Status_BadSerialPoverflow, 22 Status_BadSerialPoverflow, 22 Status_BadSerialPoverflow, 22 Status_BadSerialPoverflow, 22 Status_BadSerialPoperdered, 22 Status_BadTcpAccept, 22 Status_BadTcpAccept, 22 Status_BadTcpConnect, 22 Status_BadTcpConnect, 22 Status_BadTcpCreate, 22	· · · · · · · · · · · · · · · · · · ·	_
Modbus, 17 MemoryType, 20 addressFromString, 23 ASC, 21 asciiToBytes, 23 availableSerialPortList, 23 availableSerialPortLst, 23 bytesToAscii, 23 bytesToString, 24 Constants, 20 crc16, 24 createPort, 25 createServerPort, 25 enumKey, 27 enumValue, 27 EvenParity, 21 FlowControl, 20 getBits, 28 HardwareControl, 20  Status BadTcpCreate, 22 Status_BadTopCreate, 22		
MemoryType, 20 addressFromString, 23 ASC, 21 asciiToBytes, 23 asciiToString, 23 availableSerialPortList, 23 availableSerialPorts, 23 bytesToAscii, 23 bytesToString, 24 Constants, 20 cre16, 24 createPort, 25 createServerPort, 25 enumKey, 27 enumValue, 27 EvenParity, 21 FlowControl, 20 getBits, 28 ASC, 21 Status_BadlEmptyResponse, 22 Status_BadGatewayPathUnavailable, 22 Status_BadGatewayPathUnavailable, 22 Status_BadGatewayPathUnavailable, 22 Status_BadGatewayPathUnavailable, 22 Status_BadIllegalDataAddress, 21 Status_BadIllegalDataValue, 21 Status_BadIllegalDataValue, 21 Status_BadllegalPunction, 21 Status_BadllegalPunction, 21 Status_BadllegalPunction, 21 Status_BadMemoryParityError, 22 Status_BadMemoryParityError, 22 Status_BadNotCorrectRequest, 22 Status_BadNotCorrectRequest, 22 Status_BadNotCorrectRequest, 22 Status_BadSerialOpen, 22 Status_BadSerialOpen, 22 Status_BadSerialOpen, 22 Status_BadSerialWrite, 22 Status_BadSerialWrite, 22 Status_BadSerialWrite, 22 Status_BadSerialWrite, 22 Status_BadSerialPorticeBusy, 22 Status_BadTepConnect, 22 Status_BadTepConnect, 22 Status_BadTepConnect, 22 Status_BadTepConnect, 22 Status_BadTepConnect, 22 Status_BadTepConnect, 22	2,	
addressFromString, 23 ASC, 21 ASC, 21 asciiToBytes, 23 asciiToBytes, 23 availableSerialPortList, 23 availableSerialPorts, 23 bytesToAscii, 23 bytesToString, 24 Constants, 20 crc16, 24 createClientPort, 24 createServerPort, 25 createServerPort, 25 enumKey, 27 enumValue, 27 EvenParity, 21 FlowControl, 20 getBit, 28 getBits, 28 HardwareControl, 20  Status BadTopCereate, 22 Status BadTopCreate, 22 Status BadTopCreate, 22 Status BadSerounce, 22 Status BadTopConnect, 22 Status BadTopConnect, 22 Status BadTopConnect, 22	•	— · · · · · · · · · · · · · · · · · · ·
ASC, 21 asciiToBytes, 23 asciiToBytes, 23 asciiToString, 23 availableSerialPortList, 23 availableSerialPorts, 23 bytesToAscii, 23 bytesToString, 24 Constants, 20 crc16, 24 createClientPort, 24 createServerPort, 25 enumKey, 27 enumValue, 27 EvenParity, 21 FlowControl, 20 getBit, 28 getBits, 28 HardwareControl, 20 Status, BadGatewayPathUnavailable, 22 Status, BadGatewayPathUnavailable, 22 Status, BadGatewayPathUnavailable, 22 Status, BadGltewayPathUnavailable, 22 Status, BadGatewayPathUnavailable, 22 Status, BadTcpConnect, 22 Status, BadTcpConnect, 22 Status, BadTcpCoreate, 22		
asciiToBytes, 23 asciiToString, 23 availableSerialPortList, 23 availableSerialPorts, 23 bytesToAscii, 23 bytesToString, 24 Constants, 20 crc16, 24 createClientPort, 25 createServerPort, 25 enumKey, 27 enumValue, 27 EvenParity, 21 FlowControl, 20 getBit, 28 getBits, 28 gavailableSerialPorts, 23 Status_BadGatewayTargetDeviceFailedToRespond, 22 Status_BadGatewayTargetDeviceFailedToRespond, 22 Status_BadIllegalDataAddress, 21 Status_BadIllegalDataValue, 21 Status_BadIllegalDataValue, 21 Status_BadIllegalDataValue, 21 Status_BadHeadres, 21 Status_BadHeadrey, 22 Status_BadMemoryParityError, 22 Status_BadMemoryParityError, 22 Status_BadNotCorrectRequest, 22 Status_BadNotCorrectRequest, 22 Status_BadNotCorrectResponse, 22 Status_BadSerialPoore, 22 Status_BadSerialPoore, 22 Status_BadSerialPoore, 22 Status_BadSerialPoore, 22 Status_BadSerialPoore, 22 Status_BadTcpConnect, 22 Status_BadTcpConnect, 22 Status_BadTcpConnect, 22	<u>.</u>	
asciiToString, 23 availableSerialPortList, 23 availableSerialPorts, 23 bytesToAscii, 23 bytesToString, 24 Constants, 20 crc16, 24 createClientPort, 24 createPort, 25 createServerPort, 25 enumKey, 27 enumValue, 27 EvenParity, 21 FlowControl, 20 getBits, 28 getBits, 28 getBits, 28 getBitsS, 28 HardwareControl, 20  status_BadllegalDataValue, 21 status_BadlllegalFunction, 21 status_BadlllegalFunction, 21 status_BadllegalFunction, 21 status_BadllegalFunction, 21 status_BadllegalFunction, 21 status_Badlerc, 22 status_Badlegalfunction, 21 status_Badlerc, 22 status_Badlegalfunction, 21 status_BadlegalPataValue, 21 status_BadlegalPatevaler status_BadlegalPataValue, 21 status_BadlegalPadlevaler status_BadlegalPadlevaler status_BadlegalPadlevaler status_Badlerover status_Badlerover status_Badlerover status_Ba		
availableSerialPortList, 23 availableSerialPorts, 23 bytesToAscii, 23 bytesToString, 24 Constants, 20 crc16, 24 createClientPort, 25 createServerPort, 25 enumKey, 27 enumValue, 27 EvenParity, 21 FlowControl, 20 getBit, 28 getBits, 28	-	_ ,
availableSerialPorts, 23 bytesToAscii, 23 bytesToString, 24 Constants, 20 crc16, 24 createClientPort, 24 createPort, 25 enumKey, 27 enumValue, 27 EvenParity, 21 EvenParity, 21 FlowControl, 20 getBit, 28 getBits, 28 getBits, 28 getBitsS, 28 HardwareControl, 20  Status_BadlllegalDataValue, 21 Status_BadlllegalFunction, 21 Status_BadlllegalFunction, 21 Status_BadlllegalFunction, 21 Status_BadllegalFunction, 21 Status_BadNotCorrectReapore, 22 Status_BadNotCorrectReapore, 22 Status_BadReadBufferOverflow, 22 Status_BadReadBufferOverflow, 22 Status_BadSerialOpen, 22 Status_BadSerialOpen, 22 Status_BadSerialWrite, 22 Status_BadServerDeviceBusy, 22 Status_BadServerDeviceFailure, 21 Status_BadTcpAccept, 22 Status_BadTcpConnect, 22 Status_BadTcpConnect, 22 Status_BadTcpCreate, 22		
bytesToAscii, 23 bytesToString, 24 Constants, 20 crc16, 24 createClientPort, 24 createPort, 25 createServerPort, 25 enumKey, 27 enumValue, 27 EvenParity, 21 FlowControl, 20 getBit, 28 getBits, 28 getBits, 28 getBits, 28 bytesToAscii, 23 Status_BadIllegalFunction, 21 Status_BadMemoryParityError, 22 Status_BadMemoryParityError, 22 Status_BadNotCorrectRequest, 22 Status_BadNotCorrectRequest, 22 Status_BadNotCorrectResponse, 22 Status_BadNotCorrectResponse, 22 Status_BadReadBufferOverflow, 22 Status_BadSerialOpen, 22 Status_BadSerialOpen, 22 Status_BadSerialWrite, 22 Status_BadSerialWrite, 22 Status_BadServerDeviceBusy, 22 Status_BadServerDeviceFailure, 21 Status_BadServerDeviceFailure, 21 Status_BadTcpAccept, 22 Status_BadTcpConnect, 22 Status_BadTcpConnect, 22 Status_BadTcpConnect, 22 Status_BadTcpCreate, 22	, , , , , , , , , , , , , , , , , , ,	_ · ·
bytesToString, 24 Constants, 20 crc16, 24 createClientPort, 24 createServerPort, 25 enumKey, 27 enumValue, 27 EvenParity, 21 FlowControl, 20 getBit, 28 getBits, 28 getBits, 28 getBits, 28 HardwareControl, 20  Status_BadLrc, 22 Status_BadMemoryParityError, 22 Status_BadNotCorrectRequest, 22 Status_BadNotCorrectResponse, 22 Status_BadReadBufferOverflow, 22 Status_BadSerialOpen, 22 Status_BadSerialOpen, 22 Status_BadSerialWrite, 22 Status_BadSerialWrite, 22 Status_BadServerDeviceBusy, 22 Status_BadServerDeviceFailure, 21 Status_BadServerDeviceFailure, 21 Status_BadTcpConnect, 22 Status_BadTcpConnect, 22 Status_BadTcpConect, 22 Status_BadTcpCreate, 22		
Constants, 20 crc16, 24 createClientPort, 24 createPort, 25 createServerPort, 25 enumKey, 27 enumValue, 27 EvenParity, 21 FlowControl, 20 getBit, 28 getBitS, 28 getBitS, 28 HardwareControl, 20 Crc16, 24 Status_BadMemoryParityError, 22 Status_BadNotCorrectRequest, 22 Status_BadNotCorrectResponse, 22 Status_BadReadBufferOverflow, 22 Status_BadSerialOpen, 22 Status_BadSerialRead, 22 Status_BadSerialWrite, 22 Status_BadSerialWrite, 22 Status_BadServerDeviceBusy, 22 Status_BadServerDeviceFailure, 21 Status_BadServerDeviceFailure, 21 Status_BadTcpConnect, 22 Status_BadTcpConnect, 22 Status_BadTcpConnect, 22 Status_BadTcpCreate, 22		_ •
crc16, 24 createClientPort, 24 createPort, 25 createServerPort, 25 enumKey, 27 enumValue, 27 EvenParity, 21 FlowControl, 20 getBits, 28 getBits, 28 getBits, 28 getBitsS, 28 HardwareControl, 20  createClientPort, 24 Status_BadNotCorrectRequest, 22 Status_BadNotCorrectResponse, 22 Status_BadReadBufferOverflow, 22 Status_BadReadBufferOverflow, 22 Status_BadSerialOpen, 22 Status_BadSerialRead, 22 Status_BadSerialWrite, 22 Status_BadServerDeviceBusy, 22 Status_BadServerDeviceFailure, 21 Status_BadTcpAccept, 22 Status_BadTcpAccept, 22 Status_BadTcpConnect, 22		
createClientPort, 24 createPort, 25 createServerPort, 25 enumKey, 27 enumValue, 27 EvenParity, 21 FlowControl, 20 getBit, 28 getBitS, 28 getBitS, 28 HardwareControl, 20  Status_BadNotCorrectResponse, 22 Status_BadReadBufferOverflow, 22 Status_BadSerialOpen, 22 Status_BadSerialRead, 22 Status_BadSerialWrite, 22 Status_BadServerDeviceBusy, 22 Status_BadServerDeviceBusy, 22 Status_BadServerDeviceFailure, 21 Status_BadTcpAccept, 22 Status_BadTcpConnect, 22 Status_BadTcpConnect, 22 Status_BadTcpConnect, 22 Status_BadTcpConnect, 22 Status_BadTcpCreate, 22		
createPort, 25 createServerPort, 25 createServerPort, 25 enumKey, 27 enumValue, 27 EvenParity, 21 FlowControl, 20 getBit, 28 getBitS, 28 getBitS, 28 HardwareControl, 20  Status_BadNotCorrectResponse, 22 Status_BadReadBufferOverflow, 22 Status_BadSerialOpen, 22 Status_BadSerialRead, 22 Status_BadSerialWrite, 22 Status_BadServerDeviceBusy, 22 Status_BadServerDeviceFailure, 21 Status_BadServerDeviceFailure, 21 Status_BadTcpAccept, 22 Status_BadTcpConnect, 22 Status_BadTcpConnect, 22 Status_BadTcpConnect, 22 Status_BadTcpConnect, 22 Status_BadTcpConnect, 22 Status_BadTcpConnect, 22 Status_BadTcpCreate, 22		_ •
createServerPort, 25 enumKey, 27 enumValue, 27 EvenParity, 21 FlowControl, 20 getBit, 28 getBitS, 28 getBits, 28 getBitsS, 28 HardwareControl, 20 Status_BadTopCoreate, 22 Status_BadSerialPead, 22 Status_BadSerialWrite, 22 Status_BadServerDeviceBusy, 22 Status_BadServerDeviceFailure, 21 Status_BadTcpAccept, 22 Status_BadTcpConnect, 22		<del>-</del>
enumKey, 27 enumValue, 27 EvenParity, 21 FlowControl, 20 getBit, 28 getBitS, 28 Status_BadTcpAccept, 22 Status_BadTcpConnect, 22 Status_BadTcpConnect, 22 Status_BadTcpConnect, 22 Status_BadTcpConnect, 22 Status_BadTcpConnect, 22		<del>-</del>
enumValue, 27 EvenParity, 21 FlowControl, 20 getBit, 28 getBitS, 2		<del>-</del>
EvenParity, 21  FlowControl, 20  getBit, 28  getBitS, 28  Status_BadTcpBind, 22  getBits, 28  Status_BadTcpConnect, 22  Status_BadTcpConnect, 22  Status_BadTcpConnect, 22  Status_BadTcpCreate, 22	•	·
FlowControl, 20 getBit, 28 getBitS, 28 getBits, 28 getBits, 28 getBits, 28 getBits, 28 getBits, 28 getBitsOnaction Status_BadServerDeviceFailure, 21 Status_BadTcpAccept, 22 Status_BadTcpBind, 22 getBitsS, 28 HardwareControl, 20 Status_BadTcpConnect, 22 Status_BadTcpCreate, 22		
getBit, 28 Status_BadServerDeviceFailure, 21 getBitS, 28 Status_BadTcpAccept, 22 getBits, 28 Status_BadTcpBind, 22 getBitsS, 28 Status_BadTcpConnect, 22 HardwareControl, 20 Status_BadTcpCreate, 22		
getBitS, 28 Status_BadTcpAccept, 22 getBits, 28 Status_BadTcpBind, 22 getBitsS, 28 Status_BadTcpConnect, 22 HardwareControl, 20 Status_BadTcpCreate, 22		
getBits, 28 Status_BadTcpBind, 22 getBitsS, 28 Status_BadTcpConnect, 22 HardwareControl, 20 Status_BadTcpCreate, 22	-	
getBitsS, 28 Status_BadTcpConnect, 22 HardwareControl, 20 Status_BadTcpCreate, 22	-	
HardwareControl, 20 Status BadTcpCreate, 22	-	
Olalas Baaroporcalo, ZZ	<del>-</del>	
IFC, ∠ŏ		Status_BadTcpCreate, 22
	IIC, ∠8	

Status_BadTcpDisconnect, 22	readCoils, 49
Status_BadTcpListen, 22	readCoilsAsBoolArray, 49
Status_BadTcpRead, 22	readDiscreteInputs, 49
Status_BadTcpWrite, 22	readDiscreteInputsAsBoolArray, 50
Status_BadWriteBufferOverflow, 22	readExceptionStatus, 50
Status_Good, 21	readHoldingRegisters, 50
Status_Processing, 21	readInputRegisters, 50
Status_Uncertain, 21	setUnit, 50
StatusCode, 21	type, 51
StatusIsBad, 31	unit, 51
StatusIsGood, 31	writeMultipleCoils, 51
StatusIsProcessing, 31	writeMultipleCoilsAsBoolArray, 51
StatusIsStandardError, 31	writeMultipleRegisters, 51
StatusIsUncertain, 32	writeSingleCoil, 51
StopBits, 22	writeSingleRegister, 52
TCP, 21	ModbusClientPort, 52
timer, 32	cancelRequest, 55
toFlowControl, 32	close, 55
toModbusString, 32	currentClient, 55
toParity, 32, 33	getRequestStatus, 55
toProtocolType, 33	isOpen, 56
toStopBits, 33	lastErrorStatus, 56
toString, 33, 34	lastErrorText, 56
TwoStop, 22	lastStatus, 56
VALID MODBUS ADDRESS BEGIN, 20	ModbusClientPort, 55
VALID_MODBUS_ADDRESS_END, 20	port, 56
writeMemBits, 34	readCoils, 56
writeMemRegs, 35	readCoilsAsBoolArray, 57
Modbus::Address, 37	readDiscreteInputs, 57, 58
Address, 37, 38	readDiscreteInputsAsBoolArray, 58
isValid, 38	readExceptionStatus, 59
number, 38	readHoldingRegisters, 59
offset, 38	readInputRegisters, 60
operator quint32, 38	repeatCount, 61
operator=, 38	setRepeatCount, 61
toString, 39	signalClosed, 61
type, 39	signalError, 61
Modbus::Defaults, 39	signalOpened, 61
Defaults, 40	signalRx, 61
instance, 40	signalTx, 61
Modbus::SerialSettings, 110	type, 62
Modbus::Strings, 110	writeMultipleCoils, 62
instance, 111	writeMultipleCoilsAsBoolArray, 62, 63
Strings, 111	writeMultipleRegisters, 63
Modbus::TcpSettings, 112	writeSingleCoil, 64
ModbusAscPort, 44	writeSingleRegister, 64
~ModbusAscPort, 46	ModbusGlobal.h
ModbusAscPort, 46	GET BITS, 148
readBuffer, 46	MB_RTU_IO_BUFF_SZ, 148
type, 46	SET_BIT, 148
writeBuffer, 46	SET_BITS, 149
ModbusClient, 47	ModbusInterface, 65
isOpen, 48	readCoils, 66
lastPortErrorStatus, 48	readDiscreteInputs, 66
lastPortErrorText, 49	readExceptionStatus, 67
lastPortStatus, 49	readHoldingRegisters, 67
ModbusClient, 48	readInputRegisters, 67
port, 49	writeMultipleCoils, 68
port, To	wittervialtipie della, de

writeMultipleRegisters, 68	setBaudRate, 85
writeSingleCoil, 69	setDataBits, 85
writeSingleRegister, 69	setFlowControl, 85
ModbusLib, 1	setParity, 85
ModbusObject, 70	setPortName, 85
~ModbusObject, 71	setStopBits, 85
connect, 71	setTimeoutFirstByte, 85
disconnect, 72	setTimeoutInterByte, 86
disconnectFunc, 72	stopBits, 86
emitSignal, 72	timeoutFirstByte, 86
ModbusObject, 71	
The state of the s	timeoutInterByte, 86
ModbusServerPort, 89	write, 86
objectName, 73	writeBufferData, 86
sender, 73	writeBufferSize, 86
setObjectName, 73	ModbusSerialPort::Defaults, 41
ModbusPort, 73	Defaults, 41
$\sim$ ModbusPort, 74	instance, 42
close, 75	ModbusServerPort, 87
handle, 75	close, 88
isBlocking, 75	device, 88
isChanged, 75	isOpen, 89
isNonBlocking, 75	isStateClosed, 89
isOpen, 75	isTcpServer, 89
isServerMode, 75	ModbusObject, 89
lastErrorStatus, 76	open, 89
lastErrorText, 76	process, 89
	·
open, 76	signalClosed, 90
read, 76	signalError, 90
readBuffer, 76	signalOpened, 90
readBufferData, 76	signalRx, 90
readBufferSize, 77	signalTx, 90
setError, 77	type, 90
setNextRequestRepeated, 77	ModbusServerResource, 91
setServerMode, 77	close, 93
type, 77	isOpen, 93
write, 77	ModbusServerResource, 93
writeBuffer, 78	open, <mark>93</mark>
writeBufferData, 78	port, 93
writeBufferSize, 78	process, 93
ModbusRtuPort, 78	processDevice, 94
~ModbusRtuPort, 80	processInputData, 94
ModbusRtuPort, 80	processOutputData, 94
readBuffer, 80	type, 94
type, 80	ModbusSlotBase< ReturnType, Args >, 94
••	
writeBuffer, 81	~ModbusSlotBase, 95
ModbusSerialPort, 81	exec, 95
baudRate, 83	methodOrFunction, 95
close, 83	object, 95
dataBits, 83	ModbusSlotFunction
flowControl, 83	ModbusSlotFunction< ReturnType, Args >, 96
handle, 83	ModbusSlotFunction < ReturnType, Args >, 96
isOpen, 83	exec, 97
open, 84	methodOrFunction, 97
parity, 84	ModbusSlotFunction, 96
portName, 84	ModbusSlotMethod
read, 84	ModbusSlotMethod< T, ReturnType, Args >, 98
readBufferData, 84	ModbusSlotMethod< T, ReturnType, Args >, 97
readBufferSize, 84	exec, 98
.0405411070120, 01	5,00,00

methodOrFunction, 98	Modbus::Address, 38
ModbusSlotMethod, 98	
object, 99	object
ModbusTcpPort, 99	ModbusSlotBase< ReturnType, Args >, 95
autoIncrement, 101	ModbusSlotMethod< T, ReturnType, Args >, 99
close, 101	objectName
handle, 101	ModbusObject, 73
host, 101	OddParity
isOpen, 101	Modbus, 21
ModbusTcpPort, 101	offset
open, 102	Modbus::Address, 38
port, 102	OneAndHalfStop
read, 102	Modbus, 22
readBuffer, 102	OneStop
readBufferData, 102	Modbus, 22
readBufferSize, 102	open
setHost, 103	ModbusPort, 76
setNextRequestRepeated, 103	ModbusSerialPort, 84
	ModbusServerPort, 89
setPort, 103	ModbusServerResource, 93
setTimeout, 103	ModbusTcpPort, 102
timeout, 103	ModbusTcpServer, 108
type, 103	operator quint32
write, 104	·
writeBuffer, 104	Modbus::Address, 38
writeBufferData, 104	operator=
writeBufferSize, 104	Modbus::Address, 38
ModbusTcpPort::Defaults, 42	Parity
Defaults, 42	Modbus, 21
instance, 43	
ModbusTcpServer, 105	parity ModbusSerialPort, 84
clearConnections, 107	
close, 107	pfReadCoils
createTcpPort, 107	cModbus.h, 116
isOpen, 107	pfReadDiscreteInputs
isTcpServer, 107	cModbus.h, 116
ModbusTcpServer, 107	pfReadExceptionStatus
nextPendingConnection, 108	cModbus.h, 116
open, 108	pfReadHoldingRegisters
port, 108	cModbus.h, 117
process, 108	pfReadInputRegisters
setPort, 108	cModbus.h, 117
setTimeout, 109	pfSlotCloseConnection
signalCloseConnection, 109	cModbus.h, 117
signalNewConnection, 109	pfSlotClosed
timeout, 109	cModbus.h, 117
type, 109	pfSlotError
ModbusTcpServer::Defaults, 43	cModbus.h, 117
Defaults, 43	pfSlotNewConnection
instance, 44	cModbus.h, 118
msleep	pfSlotOpened
Modbus, 29	cModbus.h, 118
Wodous, 20	pfSlotRx
nextPendingConnection	cModbus.h, 118
ModbusTcpServer, 108	pfSlotTx
NoFlowControl	cModbus.h, 118
Modbus, 20	pfWriteMultipleCoils
NoParity	cModbus.h, 118
Modbus, 21	pfWriteMultipleRegisters
number	cModbus.h, 119
<del></del>	, -

pfWriteSingleCoil	ModbusInterface, 67
cModbus.h, 119	readHoldingRegisters
pfWriteSingleRegister	ModbusClient, 50
cModbus.h, 119	ModbusClientPort, 59
port	ModbusInterface, 67
ModbusClient, 49	readInputRegisters
ModbusClientPort, 56	ModbusClient, 50
ModbusServerResource, 93	ModbusClientPort, 60
ModbusTcpPort, 102	ModbusInterface, 67
ModbusTcpServer, 108	readMemBits
portName	Modbus, 29
ModbusSerialPort, 84	readMemRegs
process	Modbus, 29
ModbusServerPort, 89	repeatCount
ModbusServerResource, 93	ModbusClientPort, 61
	RTU
ModbusTcpServer, 108	
processDevice	Modbus, 21
ModbusServerResource, 94	sascii
processInputData	Modbus, 30
ModbusServerResource, 94	sbytes
processOutputData	Modbus, 30
ModbusServerResource, 94	sender
ProtocolType	
Modbus, 21	ModbusObject, 73
a d	SET_BIT
read	ModbusGlobal.h, 148
ModbusPort, 76	SET_BITS
ModbusSerialPort, 84	ModbusGlobal.h, 149
ModbusTcpPort, 102	setBaudRate
readBuffer	ModbusSerialPort, 85
ModbusAscPort, 46	setBit
ModbusPort, 76	Modbus, 30
ModbusRtuPort, 80	setBitS
ModbusTcpPort, 102	Modbus, 30
readBufferData	setBits
ModbusPort, 76	Modbus, 30
ModbusSerialPort, 84	setBitsS
ModbusTcpPort, 102	Modbus, 31
readBufferSize	setDataBits
ModbusPort, 77	ModbusSerialPort, 85
ModbusSerialPort, 84	setError
ModbusTcpPort, 102	ModbusPort, 77
readCoils	setFlowControl
ModbusClient, 49	ModbusSerialPort, 85
ModbusClientPort, 56	setHost
ModbusInterface, 66	ModbusTcpPort, 103
readCoilsAsBoolArray	setNextRequestRepeated
ModbusClient, 49	ModbusPort, 77
ModbusClientPort, 57	ModbusTcpPort, 103
readDiscreteInputs	setObjectName
ModbusClient, 49	ModbusObject, 73
ModbusClientPort, 57, 58	setParity
ModbusInterface, 66	ModbusSerialPort, 85
readDiscreteInputsAsBoolArray	setPort
•	
ModbusClient, 50	ModbusTcpPort, 103
ModbusClientPort, 58	ModbusTcpServer, 108
readExceptionStatus	setPortName
ModbusClient, 50	ModbusSerialPort, 85
ModbusClientPort, 59	setRepeatCount

ModbusClientPort, 61	Modbus 21
setServerMode	Modbus, 21 Status_BadlllegalDataValue
ModbusPort, 77	Modbus, 21
setStopBits	Status_BadIllegalFunction
ModbusSerialPort, 85	Modbus, 21
setTimeout	Status_BadLrc
ModbusTcpPort, 103	Modbus, 22
ModbusTcpServer, 109	Status_BadMemoryParityError
setTimeoutFirstByte	Modbus, 22
ModbusSerialPort, 85	Status BadNegativeAcknowledge
setTimeoutInterByte	Modbus. 22
ModbusSerialPort, 86	Status_BadNotCorrectRequest
setUnit	Modbus, 22
ModbusClient, 50	Status_BadNotCorrectResponse
signalCloseConnection	Modbus, 22
ModbusTcpServer, 109	Status BadReadBufferOverflow
signalClosed	Modbus, 22
ModbusClientPort, 61	Status_BadSerialOpen
ModbusServerPort, 90	Modbus, 22
signalError	Status BadSerialRead
ModbusClientPort, 61	Modbus, 22
ModbusServerPort, 90	Status BadSerialWrite
signalNewConnection	Modbus, 22
ModbusTcpServer, 109	Status_BadServerDeviceBusy
signalOpened	Modbus, 22
ModbusClientPort, 61	Status_BadServerDeviceFailure
ModbusServerPort, 90	Modbus, 21
signalRx	Status_BadTcpAccept
ModbusClientPort, 61	Modbus, 22
ModbusServerPort, 90	Status_BadTcpBind
signalTx	Modbus, 22
ModbusClientPort, 61	Status_BadTcpConnect
ModbusServerPort, 90	Modbus, 22
SoftwareControl	Status_BadTcpCreate
Modbus, 20	Modbus, 22
SpaceParity	Status_BadTcpDisconnect
Modbus, 21	Modbus, 22
STANDARD_TCP_PORT	Status_BadTcpListen
Modbus, 20	Modbus, 22
Status_Bad	Status_BadTcpRead
Modbus, 21	Modbus, 22
Status_BadAcknowledge	Status_BadTcpWrite
Modbus, 21	Modbus, 22
Status_BadAscChar	Status_BadWriteBufferOverflow
Modbus, 22	Modbus, 22
Status_BadAscMissColon	Status_Good
Modbus, 22	Modbus, 21
Status_BadAscMissCrLf	Status_Processing
Modbus, 22	Modbus, 21
Status_BadCrc	Status_Uncertain
Modbus, 22	Modbus, 21
Status_BadEmptyResponse	StatusCode
Modbus, 22	Modbus, 21
Status_BadGatewayPathUnavailable	StatusIsBad Modbus 21
Modbus, 22 Status_BadGatewayTargetDeviceFailedToRespond	Modbus, 31 StatusIsGood
Modbus, 22	Modbus, 31
Status_BadlllegalDataAddress	StatusIsProcessing
otatus_paulieyalpataAuuless	Olalusisi 1006ssiriy

Modbus, 31	ModbusPort, 77
StatusIsStandardError	ModbusSerialPort, 86
Modbus, 31	ModbusTcpPort, 104
StatusIsUncertain	writeBuffer
Modbus, 32	ModbusAscPort, 46
StopBits	ModbusPort, 78
Modbus, 22	ModbusRtuPort, 81
stopBits	ModbusTcpPort, 104
ModbusSerialPort, 86	writeBufferData
Strings	ModbusPort, 78
Modbus::Strings, 111	ModbusSerialPort, 86
<b>G</b> ,	ModbusTcpPort, 104
TCP	writeBufferSize
Modbus, 21	ModbusPort, 78
timeout	ModbusSerialPort, 86
ModbusTcpPort, 103	ModbusTcpPort, 104
ModbusTcpServer, 109	writeMemBits
timeoutFirstByte	Modbus, 34
ModbusSerialPort, 86	writeMemRegs
timeoutInterByte	Modbus, 35
ModbusSerialPort, 86	writeMultipleCoils
timer	ModbusClient, 51
Modbus, 32	ModbusClientPort, 62
toFlowControl	ModbusInterface, 68
Modbus, 32	writeMultipleCoilsAsBoolArray
toModbusString	ModbusClient, 51
Modbus, 32	ModbusClientPort, 62, 63
toParity	writeMultipleRegisters
Modbus, 32, 33	ModbusClient, 51
toProtocolType	ModbusClientPort, 63
Modbus, 33	ModbusInterface, 68
toStopBits	writeSingleCoil
Modbus, 33	ModbusClient, 51
toString	ModbusClientPort, 64
Modbus, 33, 34	ModbusInterface, 69
Modbus::Address, 39	writeSingleRegister
TwoStop	ModbusClient, 52
Modbus, 22	ModbusClientPort, 64
type	ModbusInterface, 69
Modbus::Address, 39	
ModbusAscPort, 46	
ModbusClient, 51	
ModbusClientPort, 62	
ModbusPort, 77	
ModbusRtuPort, 80	
ModbusServerPort, 90	
ModbusServerResource, 94	
ModbusTcpPort, 103	
ModbusTcpServer, 109	
unit	
unit ModbusClient, 51	
VALID_MODBUS_ADDRESS_BEGIN	
Modbus, 20	
VALID_MODBUS_ADDRESS_END	
Modbus, 20	
write	
WITE	