Professional Fieldbus Connections

FieldTalk Modbus Master Library for .NET

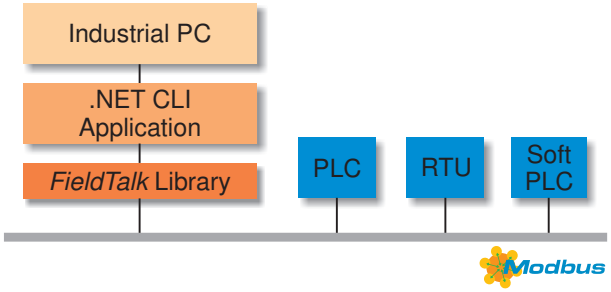
Software manual

Library version 2.10.0

Contents

1 Introduction

This *FieldTalk*™ Modbus® Master Library for .NET provides connectivity to Modbus slave compatible devices for VB.net and C# applications.



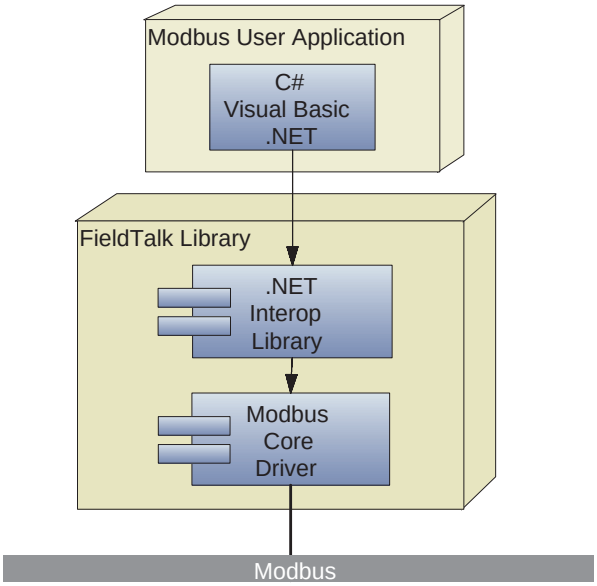
Typical applications are Modbus based Supervisory Control and Data Acquisition Systems (SCADA), Modbus data concentrators, Modbus gateways, User Interfaces and Factory Information Systems (FIS).

Features:

- Robust design suitable for real-time and industrial applications
- Full implementation of Bit Access and 16 Bits Access Function Codes as well as a subset of the most commonly used Diagnostics Function Codes
- Standard Modbus bit and 16-bit integer data types (coils, discretes & registers)
- Support for 32-bit integer, modulo-10000 and float data types, including Daniel/Enron protocol extensions
- Configurable word alignment for 32-bit types (big-endian, little-endian)
- Support of Broadcasting
- Failure and transmission counters
- Transmission and connection time-out supervision
- Detailed transmission and protocol failure reporting using error codes

1.1 Library Structure

The FieldTalk .NET library consists of two components: a *.NET Interop Library* with MS↩ IL code (managed code) and a *Modbus Core Driver* written in native code (unmanaged code). This architecture has significant performance benefits for .NET applications because the time critical communication code is executed as native code.



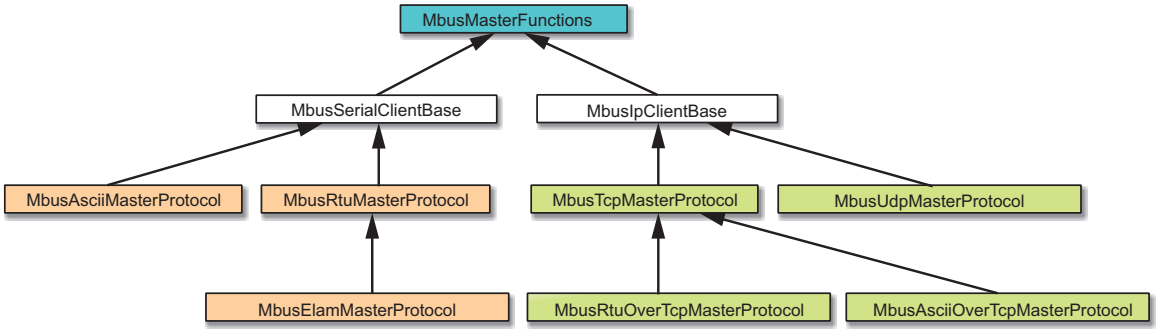
The two components are contained in separate files: `FieldTalk.Modbus.Master.dll` contains the *.NET Interop Library* and `libmbusmaster.dll` the native *Modbus Core Driver*. These two library files have to be deployed with your application.

For .NET Core, the deployment and loading of the native code library is taken care of by .NET core's package management. This works for Windows as well as for Linux platforms.

For classic .NET Framework 4.0 and 2.0, the deployment of the native code library is taken care of by the `FieldTalk.Modbus.Master.prop` file which the NutGet package manager will automatically add to your project. With that file, when compiling, a `bin/x86` and a `bin/x64` subfolder is created in the bin directory and the native libraries copied into these subfolders. When the .NET assembly is loaded, the correct native library will be loaded from one of these two sub-folders.

For the .NET Compact Framework (Windows Embedded Compact, Windows CE and Windows Mobile) the deployment of the native libray must be done manually.

The library's API is organised into one class for each Modbus protocol flavour and a common base class, which applies to all Modbus protocol flavours. Because the two serial-line protocols Modbus ASCII and Modbus RTU share some common code, an intermediate base class implements the functions specific to the serial protocols.



The base class MbusMasterFunctions contains all protocol unspecific functions, in particular the data and control functions defined by Modbus. All Modbus protocol flavours inherit from this base class.

The class MbusAsciiMasterProtocol implements the Modbus ASCII protocol, the class MbusRtuMasterProtocol implements the Modbus RTU protocol. The class MbusTcp↩ MasterProtocol implements the MODBUS/TCP protocol and the class MbusRtuOverTcp↩ MasterProtocol the Encapsulated Modbus RTU master protocol (also known as RTU over TCP or RTU/IP).

In order to use one of the four Modbus protocols, the desired Modbus protocol flavour class has to be instantiated:

C#

```
MbusRtuMasterProtocol mbusProtocol = new MbusRtuMasterProtocol();
```

VB.net

```
Dim mbusProtocol As MbusTcpMasterProtocol = New MbusTcpMasterProtocol
```

After a protocol object has been declared and opened, data and control functions can be used:

C#

```
mbusProtocol.writeSingleRegister(slaveId, startRef, 1234);
```

VB.net

```
mbusProtocol.writeSingleRegister(slaveId, startRef, 1234)
```

2 What You should know about Modbus

2.1 Some Background

The Modbus protocol family was originally developed by Schneider Automation Inc. as an industrial network for their Modicon programmable controllers.

Since then the Modbus protocol family has been established as vendor-neutral and open communication protocols, suitable for supervision and control of automation equipment.

2.2 Technical Information

Modbus is a master/slave protocol with half-duplex transmission.

One master and up to 247 slave devices can exist per network.

The protocol defines framing and message transfer as well as data and control functions.

The protocol does not define a physical network layer. Modbus works on different physical network layers. The ASCII and RTU protocol operate on RS-232, RS-422 and RS-485 physical networks. The Modbus/TCP protocol operates on all physical network layers supporting TCP/IP. This compromises 10BASE-T and 100BASE-T LANs as well as serial PPP and SLIP network layers.

Note

To utilise the multi-drop feature of Modbus, you need a multi-point network like RS-485. In order to access a RS-485 network, you will need a protocol converter which automatically switches between sending and transmitting operation. However some industrial hardware platforms have an embedded RS-485 line driver and support enabling and disabling of the RS-485 transmitter via the RTS signal. FieldTalk supports this RTS driven RS-485 mode.

2.2.1 The Protocol Functions

Modbus defines a set of data and control functions to perform data transfer, slave diagnostic and PLC program download.

FieldTalk implements the most commonly used functions for data transfer as well as some diagnostic functions. The functions to perform PLC program download and other device specific functions are outside the scope of FieldTalk.

All Bit Access and 16 Bits Access Modbus Function Codes have been implemented. In addition the most frequently used Diagnostics Function Codes have been implemented. This rich function set enables a user to solve nearly every Modbus data transfer problem.

The following table lists the available Modbus Function Codes in this library:

Function Code	Current Terminology	Classic Terminology
Bit Access		
1	Read Coils	Read Coil Status

Function Code	Current Terminology	Classic Terminology
2	Read Discrete Inputs	Read Input Status
5	Write Single Coil	Force Single Coil
15 (0F hex)	Write Multiple Coils	Force Multiple Coils
16 Bits Access		
3	Read Multiple Registers	Read Holding Registers
4	Read Input Registers	Read Input Registers
6	Write Single Register	Preset Single Register
16 (10 Hex)	Write Multiple Registers	Preset Multiple Registers
22 (16 hex)	Mask Write Register	Mask Write 4X Register
23 (17 hex)	Read/Write Multiple Registers	Read/Write 4X Registers
Diagnostics		
7	Read Exception Status	Read Exception Status
8 subcode 00	Diagnostics - Return Query Data	Diagnostics - Return Query Data
8 subcode 01	Diagnostics - Restart Communications Option	Diagnostics - Restart Communications Option
Vendor Specific		
Advantech	Send/Receive ADAM 5000/6000 ASCII commands	

2.2.2 How Slave Devices are identified

A slave device is identified with its unique address identifier. Valid address identifiers supported are 1 to 247. Some library functions also extend the slave ID to 255, please check the individual function's documentation.

Some Modbus functions support broadcasting. With functions supporting broadcasting, a master can send broadcasts to all slave devices of a network by using address identifier 0. Broadcasts are unconfirmed, there is no guarantee of message delivery. Therefore broadcasts should only be used for uncritical data like time synchronisation.

2.2.3 The Register Model and Data Tables

The Modbus data functions are based on a register model. A register is the smallest addressable entity with Modbus.

The register model is based on a series of tables which have distinguishing characteristics. The four tables are:

Table	Classic Terminology	Modicon Register Table	Characteristics
Discrete outputs	Coils	0:00000	Single bit, alterable by an application program, read-write

Table	Classic Terminology	Modicon Register Table	Characteristics
Discrete inputs	Inputs	1:00000	Single bit, provided by an I/O system, read-only
Input registers	Input registers	3:00000	16-bit quantity, provided by an I/O system, read-only
Output registers	Holding registers	4:00000	16-bit quantity, alterable by an application program, read-write

The Modbus protocol defines these areas very loose. The distinction between inputs and outputs and bit-addressable and register-addressable data items does not imply any slave specific behaviour. It is very common that slave devices implement all tables as overlapping memory area.

For each of those tables, the protocol allows a maximum of 65536 data items to be accessed. It is slave dependant, which data items are accessible by a master. Typically a slave implements only a small memory area, for example of 1024 bytes, to be accessed.

2.2.4 Data Encoding

Classic Modbus defines only two elementary data types. The discrete type and the register type. A discrete type represents a bit value and is typically used to address output coils and digital inputs of a PLC. A register type represents a 16-bit integer value. Some manufacturers offer a special protocol flavour with the option of a single register representing one 32-bit value.

All Modbus data function are based on the two elementary data types. These elementary data types are transferred in big-endian byte order.

Based on the elementary 16-bit register, any bulk information of any type can be exchanged as long as that information can be represented as a contiguous block of 16-bit registers. The protocol itself does not specify how 32-bit data and bulk data like strings is structured. Data representation depends on the slave's implementation and varies from device to device.

It is very common to transfer 32-bit float values and 32-bit integer values as pairs of two consecutive 16-bit registers in little-endian word order. However some manufacturers like Daniel and Enron implement an enhanced flavour of Modbus which supports 32-bit wide register transfers. FielTalk supports Daniel/Enron 32-bit wide register transfers.

The FieldTalk Modbus Master Library defines functions for the most common tasks like:

- Reading and Writing bit values
- Reading and Writing 16-bit integers
- Reading and Writing 32-bit integers as two consective registers
- Reading and Writing 32-bit floats as two consective registers

- Reading and Writing 32-bit integers using Daniel/Enron single register transfers
- Reading and Writing 32-bit floats using Daniel/Enron single register transfers
- Configuring the word order and representation for 32-bit values

2.2.5 Register and Discrete Numbering Scheme

Modicon PLC registers and discretes are addressed by a memory type and a register number or a discrete number, e.g. 4:00001 would be the first reference of the output registers.

The type offset which selects the Modicon register table must not be passed to the FieldTalk functions. The register table is selected by choosing the corresponding function call as the following table illustrates.

Master Function Call	Modicon Register Table
readCoils(), writeCoil(), forceMultipleCoils()	0:00000
readInputDiscretes	1:00000
readInputRegisters()	3:00000
writeMultipleRegisters(), readMultipleRegisters(), writeSingleRegister(), maskWriteRegister(), readWriteRegisters()	4:00000

Modbus registers are numbered starting from 1. This is different to the conventional programming logic where the first reference is addressed by 0.

Modbus discretes are numbered starting from 1 which addresses the most significant bit in a 16-bit word. This is very different to the conventional programming logic where the first reference is addressed by 0 and the least significant bit is bit 0.

The following table shows the correlation between Discrete Numbers and Bit Numbers:

Modbus Discrete Number	Bit Number	Modbus Discrete Number	Bit Number
1	15 (hex 0x8000)	9	7 (hex 0x0080)
2	14 (hex 0x4000)	10	6 (hex 0x0040)
3	13 (hex 0x2000)	11	5 (hex 0x0020)
4	12 (hex 0x1000)	12	4 (hex 0x0010)
5	11 (hex 0x0800)	13	3 (hex 0x0008)
6	10 (hex 0x0400)	14	2 (hex 0x0004)
7	9 (hex 0x0200)	15	1 (hex 0x0002)
8	8 (hex 0x0100)	16	0 (hex 0x0001)

When exchanging register number and discrete number parameters with FieldTalk functions and methdos you have to use the Modbus register and discrete numbering scheme. (Internally the functions will deduct 1 from the start register value before transmitting the value to the slave device.)

2.2.6 The ASCII Protocol

The ASCII protocol uses an hexadecimal ASCII encoding of data and a 8 bit checksum. The message frames are delimited with a ':' character at the beginning and a carriage return/linefeed sequence at the end.

The ASCII messaging is less efficient and less secure than the RTU messaging and therefore it should only be used to talk to devices which don't support RTU. Another application of the ASCII protocol are communication networks where the RTU messaging is not applicable because characters cannot be transmitted as a continuos stream to the slave device.

The ASCII messaging is state-less. There is no need to open or close connections to a particular slave device or special error recovery procedures.

A transmission failure is indicated by not receiving a reply from the slave. In case of a transmission failure, a master simply repeats the message. A slave which detects a transmission failure will discard the message without sending a reply to the master.

2.2.7 The RTU Protocol

The RTU protocol uses binary encoding of data and a 16 bit CRC check for detection of transmission errors. The message frames are delimited by a silent interval of at least 3.5 character transmission times before and after the transmission of the message.

When using RTU protocol it is very important that messages are sent as continuous character stream without gaps. If there is a gap of more than 3.5 character times while receiving the message, a slave device will interpret this as end of frame and discard the bytes received.

The RTU messaging is state-less. There is no need to open or close connections to a particular slave device or special error recovery procedures.

A transmission failure is indicated by not receiving a reply from the slave. In case of a transmission failure, a master simply repeats the message. A slave which detects a transmission failure will discard the message without sending a reply to the master.

2.2.8 The MODBUS/TCP Protocol

MODBUS/TCP is a TCP/IP based variant of the Modbus RTU protocol. It covers the use of Modbus messaging in an 'Intranet' or 'Internet' environment.

The MODBUS/TCP protocol uses binary encoding of data and TCP/IP's error detection mechanism for detection of transmission errors.

In contrast to the ASCII and RTU protocols MODBUS/TCP is a connection oriented protocol. It allows concurrent connections to the same slave as well as concurrent connections to multiple slave devices.

In case of a TCP/IP time-out or a protocol failure, a master shall close and re-open the connection and then repeat the message.

3 Design Background

FieldTalk is based on a programming language neutral but object oriented design model.

This design approach enables us to offer the protocol stack for the languages C++, C#, Visual Basic .NET, Java and Object Pascal while maintaining similar functionality.

During the course of implementation, the usability in automation, control and other industrial environments was always kept in mind.

4 Module Documentation

4.1 Data and Control Functions for all Modbus Protocol Flavours

This Modbus protocol library implements the most commonly used data functions as well as some control functions. The functions to perform PLC program download and other device specific functions are outside the scope of this library.

All Bit Access and 16 Bits Access Modbus Function Codes have been implemented. In addition the most frequently used Diagnostics Function Codes have been implemented. This rich function set enables a user to solve nearly every Modbus data transfer problem.

The following table lists the supported Modbus function codes:

Function Code	Current Terminology	Classic Terminology
Bit Access		
1	Read Coils	Read Coil Status
2	Read Discrete Inputs	Read Input Status
5	Write Single Coil	Force Single Coil
15 (0F hex)	Write Multiple Coils	Force Multiple Coils
16 Bits Access		
3	Read Multiple Registers	Read Holding Registers
4	Read Input Registers	Read Input Registers
6	Write Single Register	Preset Single Register
16 (10 Hex)	Write Multiple Registers	Preset Multiple Registers
22 (16 hex)	Mask Write Register	Mask Write 4X Register
23 (17 hex)	Read/Write Multiple Registers	Read/Write 4X Registers
Diagnostics		
7	Read Exception Status	Read Exception Status
8 subcode 00	Diagnostics - Return Query Data	Diagnostics - Return Query Data
8 subcode 01	Diagnostics - Restart Communi-cations Option	Diagnostics - Restart Communi-cations Option
Vendor Specific		
Advantech	Send/Receive ADAM 5000/6000 ASCII commands	

Remarks

When passing register numbers and discrete numbers to FieldTalk library functions you have to use the the Modbus register and discrete numbering scheme. See Register and Discrete Numbering Scheme. (Internally the functions will deduct 1 from the start register value before transmitting the value to the slave device.)

Using multiple instances of a MbusMaster... class enables concurrent protocol transfer on different communication channels (e.g. multiple TCP/IP sessions in separate threads or multiple COM ports in separate threads).

4.2 Serial Protocols

Classes

- class MbusRtuMasterProtocol

  Modbus RTU Master Protocol class
- class MbusAsciiMasterProtocol

  Modbus ASCII Master Protocol class
- class MbusElamMasterProtocol

  Extended Lufkin Automation Modbus Master Protocol

4.2.1 Detailed Description

The two classic serial Modbus protocol flavours RTU and ASCII are implemented in the MbusRtuMasterProtocol and MbusAsciiMasterProtocol classes.

The popular vendor specific Extended Lufkin Automation Modbus Master (ELAM) protocol is also available as class MbusElamMasterProtocol. This proprietary Modbus extension allows adressing of up to 2295 slave units and the retrieval of up to 2500 registers for Modbus functions 3 and 4.

These classes provide functions to open and to close serial port as well as data and control functions which can be used at any time after a protocol has been opened. The data and control functions are organized into different conformance classes. For a more detailed description of the data and control functions see section Data and Control Functions for all Modbus Protocol Flavours.

Using multiple instances of a MbusRtuMasterProtocol or MbusAsciiMasterProtocol class enables concurrent protocol transfers on multiple COM ports (they should be executed in separate threads).

See sections The RTU Protocol and The ASCII Protocol for some background information about the two serial Modbus protocols.

4.3 IP based Protocols

Classes

- class MbusUdpMasterProtocol

  MODBUS/UDP Master Protocol class

4.3.1 Detailed Description

The library provides several flavours of IP based Modbus protocols.

The MODBUS/TCP master protocol is implemented in the class MbusTcpMasterProtocol and is the only IP based protocol officially specified by the Modbus organisation.

In addition to MODBUS/TCP, the library offers implementations of both serial protocols RTU and ASCII transported over TCP streams. These are implemented in the classes MbusRtuOverTcpMasterProtocol and MbusAsciiOverTcpMasterProtocol.

Also an implementation for MODBUS/TCP packets transported via UDP is available in form of the class MbusUdpMasterProtocol.

All classes provide functions to establish and to close a TCP/IP connection to the slave as well as data and control functions which can be used after a connection to a slave device has been established successfully. For a more detailed description of the data and control functions see section Data and Control Functions for all Modbus Protocol Flavours.

Using multiple instances of a MbusTcpMasterProtocol class enables concurrent protocol transfers using multiple TCP/IP sessions. They should be executed in separate threads.

See section The MODBUS/TCP Protocol for some background information about MODBUS/TCP.

4.4 Error Management

Classes

- class BusProtocolErrors

  Protocol Errors and Modbus exceptions codes

4.4.1 Detailed Description

This module documents all the exception classes, error and return codes reported by the various library functions.

4.5 Device and Vendor Specific Modbus Functions

Functions

- Int32 adamSendReceiveAsciiCmd (string command, out string response)

  Send/Receive ADAM 5000/6000 ASCII command.
- Int32 adamSendReceiveAsciiCmd (string command, out string response)

  Send/Receive ADAM 5000/6000 ASCII command.

4.5.1 Detailed Description

Some device specific or vendor specific functions and enhancements are supported.

4.5.2 Function Documentation

adamSendReceiveAsciiCmd() [1/2] `Int32 adamSendReceiveAsciiCmd (`
`string` *command*`,`
`out string` *response* `)` `[inline]`

Sends an ADAM 5000/6000 ASCII command to the device and receives the reply as ASCII string. (e.g. "$01M" to retrieve the module name)

Note

No broadcast supported

Parameters

command	Command string. Must not be longer than 255 characters.
response	Response string. A possible trailing CR is removed.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

adamSendReceiveAsciiCmd() [2/2] `Int32 adamSendReceiveAsciiCmd (`
`string` *command*`,`
`out string` *response* `)` `[inline]`

Sends an ADAM 5000/6000 ASCII command to the device and receives the reply as ASCII string. (e.g. "$01M" to retrieve the module name)

Note

No broadcast supported

Parameters

command	Command string. Must not be longer than 255 characters.
response	Response string. A possible trailing CR is removed.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

5 Class Documentation

5.1 MbusRtuMasterProtocol Class Reference

Modbus RTU Master Protocol class

Public Member Functions

- MbusRtuMasterProtocol ()

  Creates new instance
- Int32 openProtocol ()

  Opens a serial Modbus protocol and the associated serial port with the port parameters configured via properties
- Int32 openProtocol (string portName, Int32 baudRate, Int32 dataBits, Int32 stopBits, Int32 parity)

  Opens a serial Modbus protocol and the associated serial port with the port parameters configured via properties
- Int32 enableRs485Mode (Int32 rtsDelay)

  Enables RS485 mode
- bool isOpen ()

  Returns whether the protocol is open or not.
- void closeProtocol ()

  Closes an open protocol including any associated communication resources (COM ports or sockets).

Static Public Member Functions

- static string getPackageVersion ()

  Returns the package version number.

Public Attributes

- const Int32 SER_DATABITS_7 = 7

  7 data bits
- const Int32 SER_DATABITS_8 = 8

  8 data bits
- const Int32 SER_STOPBITS_1 = 1

  1 stop bit
- const Int32 SER_STOPBITS_2 = 2

  2 stop bits
- const Int32 SER_PARITY_NONE = 0

  No parity
- const Int32 SER_PARITY_ODD = 1

Odd parity

- const Int32 SER_PARITY_EVEN = 2

  Even parity

Properties

- string portName `[get, set]`

  Serial port identifier property
- Int32 baudRate `[get, set]`

  Baud rate property in bps
- Int32 dataBits `[get, set]`

  Data bits property
- Int32 stopBits `[get, set]`

  Stop bits property
- Int32 parity `[get, set]`

  Parity property
- Int32 timeout `[get, set]`

  Time-out port property
- Int32 pollDelay `[get, set]`

  Poll delay property
- Int32 retryCnt `[get, set]`

  Retry count property

Bit Access

Table 0:00000 (Coils) and Table 1:0000 (Input Status)

- Int32 readCoils (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr)

  Modbus function 1 (01 hex), Read Coil Status/Read Coils.
- Int32 readCoils (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr, Int32 numCoils)

  Modbus function 1 (01 hex), Read Coil Status/Read Coils.
- Int32 readInputDiscretes (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr)

  Modbus function 2 (02 hex), Read Inputs Status/Read Input Discretes.
- Int32 readInputDiscretes (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr, Int32 numDiscretes)

  Modbus function 2 (02 hex), Read Inputs Status/Read Input Discretes.
- Int32 writeCoil (Int32 slaveAddr, Int32 bitAddr, bool bitVal)

  Modbus function 5 (05 hex), Force Single Coil/Write Coil.
- Int32 forceMultipleCoils (Int32 slaveAddr, Int32 startRef, bool[ ] bitArr)

  Modbus function 15 (0F hex), Force Multiple Coils.
- Int32 forceMultipleCoils (Int32 slaveAddr, Int32 startRef, bool[ ] bitArr, Int32 numCoils)

  Modbus function 15 (0F hex), Force Multiple Coils.

Register Access (16-bit, 32-bit and floating point)

Table 4:00000 (Holding Registers) and Table 3:00000 (Input Registers)

- Int32 readMultipleRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr)

  Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers (16-bit, 32-bit and floating point).
- Int32 readMultipleRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr, Int32 numRegs)

  Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers (16-bit, 32-bit and floating point).
- Int32 readInputRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr)

  Modbus function 4 (04 hex), Read Input Registers (16-bit, 32-bit and floating point).
- Int32 readInputRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr, Int32 numRegs)

  Modbus function 4 (04 hex), Read Input Registers (16-bit, 32-bit and floating point).
- Int32 writeSingleRegister (Int32 slaveAddr, Int32 regAddr, Int16 regVal)

  Modbus function 6 (06 hex), Preset Single Register/Write Single Register.
- Int32 writeSingleRegister (Int32 slaveAddr, Int32 regAddr, UInt16 regVal)

  Modbus function 6 (06 hex), Preset Single Register/Write Single Register.
- Int32 writeMultipleRegisters (Int32 slaveAddr, Int32 startRef, System.Array regArr)

  Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers (16-bit, 32-bit and floating point).
- Int32 writeMultipleRegisters (Int32 slaveAddr, Int32 startRef, System.Array regArr, Int32 numRegs)

  Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers (16-bit, 32-bit and floating point).
- Int32 maskWriteRegister (Int32 slaveAddr, Int32 regAddr, Int16 andMask, Int16 orMask)

  Modbus function 22 (16 hex), Mask Write Register.
- Int32 readWriteRegisters (Int32 slaveAddr, Int32 readRef, [In, Out] Int16[ ] readArr, Int32 writeRef, Int16[ ] writeArr)

  Modbus function 23 (17 hex), Read/Write Registers.
- Int32 readWriteRegisters (Int32 slaveAddr, Int32 readRef, [In, Out] Int16[ ] readArr, Int32 readCnt, Int32 writeRef, Int16[ ] writeArr, Int32 writeCnt)

  Modbus function 23 (17 hex), Read/Write Registers.

Modulo-10000 long integer Access

Table 4:00000 (Holding Registers) and Table 3:00000 (Input Registers)

- Int32 readMultipleMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32Arr)

  Modbus function 3 (03 hex) for 32-bit modulo-10000 long int data types, Read Holding Registers/Read Multiple Registers as modulo-10000 long int data.

- Int32 readMultipleMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32↩ Arr, Int32 numRegs)

  Modbus function 3 (03 hex) for 32-bit modulo-10000 long int data types, Read Holding Registers/Read Multiple Registers as modulo-10000 long int data.
- Int32 readInputMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32Arr)

  Modbus function 4 (04 hex) for 32-bit modulo-10000 long int data types, Read Input Registers as modulo-10000 long int data.
- Int32 readInputMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32Arr, Int32 numRegs)

  Modbus function 4 (04 hex) for 32-bit modulo-10000 long int data types, Read Input Registers as modulo-10000 long int data.
- Int32 writeMultipleMod10000 (Int32 slaveAddr, Int32 startRef, Int32[ ] int32Arr)

  Modbus function 16 (10 hex) for 32-bit modulo-10000 long int data types, Preset Multiple Registers/Write Multiple Registers with modulo-10000 long int data.
- Int32 writeMultipleMod10000 (Int32 slaveAddr, Int32 startRef, Int32[ ] int32Arr, Int32 numRegs)

  Modbus function 16 (10 hex) for 32-bit modulo-10000 long int data types, Preset Multiple Registers/Write Multiple Registers with modulo-10000 long int data.

Diagnostics

- Int32 readExceptionStatus (Int32 slaveAddr, out byte statusByte)

  Modbus function 7 (07 hex), Read Exception Status.
- Int32 returnQueryData (Int32 slaveAddr, byte[ ] queryArr, out byte[ ] echoArr)

  Modbus function code 8, sub-function 00, Return Query Data.
- Int32 restartCommunicationsOption (Int32 slaveAddr, Int32 clearEventLog)

  Modbus function code 8, sub-function 01, Restart Communications Option

User Defined Function Codes

- Int32 customFunction (Int32 slaveAddr, Int32 functionCode, byte[ ] requestData, [In, Out] byte[ ] responseData)

  User Defined Function Code

Protocol Configuration

- Int32 setTimeout (Int32 timeOut)

  Configures time-out
- Int32 getTimeout ()

  Returns the current time-out setting
- Int32 setPollDelay (Int32 pollDelay)

  Poll delay property
- Int32 getPollDelay ()

  Returns the poll delay time

- Int32 setRetryCnt (Int32 retryCnt)

  Configures the automatic retry setting
- Int32 getRetryCnt ()

  Returns the automatic retry count

Transmission Statistic Functions

- Int32 getTotalCounter ()

  Returns how often a message transfer has been executed
- void resetTotalCounter ()

  Resets total message transfer counter
- Int32 getSuccessCounter ()

  Returns how often a message transfer was successful
- void resetSuccessCounter ()

  Resets successful message transfer counter

Slave Configuration

- void configureStandard32BitMode ()

  Configures all slaves for Standard 32-bit Mode.
- Int32 configureStandard32BitMode (Int32 slaveAddr)

  Configures a slave for Standard 32-bit Mode.
- void configureEnron32BitMode ()

  Configures all slaves for Daniel/ENRON 32-bit Mode.
- Int32 configureEnron32BitMode (Int32 slaveAddr)

  Configures a slave for Daniel/ENRON 32-bit Mode.
- void configureCountFromOne ()

  Configures the reference counting scheme to start with one for all slaves.
- Int32 configureCountFromOne (Int32 slaveAddr)

  Configures the reference counting scheme to start with one for a slave.
- void configureCountFromZero ()

  Configures the reference counting scheme to start with zero for all slaves.
- void configureBigEndianInts ()

  Configures 32-bit int data type functions to do a word swap
- Int32 configureBigEndianInts (Int32 slaveAddr)

  Configures 32-bit int data type functions to do a word swap on a per slave basis
- void configureSwappedFloats ()

  Configures float data type functions to do a word swap
- Int32 configureSwappedFloats (Int32 slaveAddr)

  Configures float data type functions to do a word swap on a per slave basis
- void configureLittleEndianInts ()

  Configures 32-bit int data type functions NOT to do a word swap
- Int32 configureLittleEndianInts (Int32 slaveAddr)

Configures 32-bit int data type functions NOT to do a word swap on a per slave basis

- void configureIeeeFloats ()

  Configures float data type functions NOT to do a word swap

- Int32 configureIeeeFloats (Int32 slaveAddr)

  Configures float data type functions NOT to do a word swap on a per slave basis

- Int32 onfigureCountFromZero (Int32 slaveAddr)

  Configures the reference counting scheme to start with zero for a slave.

5.1.1 Detailed Description

This class realizes the Modbus RTU master protocol. It provides functions to open and to close serial port as well as data and control functions which can be used at any time after the protocol has been opened. The data and control functions are organized into different conformance classes.

It is possible to instantiate multiple instances of this class for establishing multiple connections on different serial ports (They should be executed in separate threads).

5.1.2 Constructor & Destructor Documentation

MbusRtuMasterProtocol() `MbusRtuMasterProtocol ( ) [inline]`

Exceptions

OutOfMemoryException	Creation of class failed

5.1.3 Member Function Documentation

openProtocol() `[1/2] Int32 openProtocol ( ) [inline], [inherited]`

This function opens the serial port. After a port has been opened, data and control functions can be used.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

openProtocol() [2/2]

```
Int32 openProtocol (
    string portName,
    Int32 baudRate,
    Int32 dataBits,
    Int32 stopBits,
    Int32 parity )  [inline], [inherited]
```

This function opens the serial port with a specific port settings. After a port has been opened, data and control functions can be used.

Note

The Modbus standard requires two stop bits if no parity is chosen. This library is not enforcing this but it is a recommended configuration.

Parameters

portName	Serial port identifier (eg "COM1")
baudRate	The port baud rate in bps (1200 - 115200, higher on some platforms)
dataBits	SER_DATABITS_7: 7 data bits (ASCII protocol only), SER_DATABITS_8: 8 data bits
stopBits	SER_STOPBITS_1: 1 stop bit, SER_STOPBITS_2: 2 stop bits
parity	SER_PARITY_NONE: no parity, SER_PARITY_ODD: odd parity, SER_PARITY_EVEN: even parity

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

enableRs485Mode()

```
Int32 enableRs485Mode (
    Int32 rtsDelay )  [inline], [inherited]
```

In RS485 mode the RTS signal can be used to enable and disable the transmitter of a RS232/RS485 converter. The RTS signal is asserted before sending data. It is cleared after the transmit buffer has been emptied and in addition the specified delay time has elapsed. The delay time is necessary because even the transmit buffer is already empty, the UART's FIFO will still contain unsent characters.

The use of RTS controlled RS232/RS485 converters should be avoided if possible. It is difficult to determine the exact time when to switch off the transmitter with non real-time operating systems like Windows and Linux. If it is switched off to early characters might still sit in the FIFO or the transmit register of the UART and these characters will be lost. Hence the slave will not recognize the message. On the other hand if it is switched off too late then the slave's message is corrupted and the master will not recognize the message.

Note

The delay value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.
A protocol must be closed in order to configure it.

Parameters

rtsDelay	Delay time in ms (Range as 0 - 100000) which applies after the transmit buffer is empty. 0 disables this mode.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readCoils() [1/2]

```
Int32 readCoils (
        Int32 slaveAddr,
        Int32 startRef,
        [In,Out] bool [] bitArr )  [inline], [inherited]
```

Reads the contents of the discrete outputs (coils, 0:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read. The length of the array determines how many coils are read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readCoils() [2/2]

```
Int32 readCoils (
        Int32 slaveAddr,
        Int32 startRef,
```

```
[In,Out] bool [] bitArr,
Int32 numCoils )  [inline], [inherited]
```

Reads the contents of the discrete outputs (coils, 0:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read.
numCoils	Number of coils to be read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputDiscretes() `[1/2]`

```
Int32 readInputDiscretes (
Int32 slaveAddr,
Int32 startRef,
[In,Out] bool [] bitArr )  [inline], [inherited]
```

Reads the contents of the discrete inputs (input status, 1:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read. The length of the array determines how many inputs are read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputDiscretes() [2/2]

```
Int32 readInputDiscretes (
        Int32 slaveAddr,
        Int32 startRef,
        [In,Out] bool [] bitArr,
        Int32 numDiscretes ) [inline], [inherited]
```

Reads the contents of the discrete inputs (input status, 1:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read.
num↩ Discretes	Number of inputs to be read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeCoil()

```
Int32 writeCoil (
        Int32 slaveAddr,
        Int32 bitAddr,
        bool bitVal ) [inline], [inherited]
```

Sets a single discrete output variable (coil, 0:00000 table) to either ON or OFF.

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
bitAddr	Coil address (Range: 1 - 65536)
bitVal	true sets, false clears discrete output variable

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

forceMultipleCoils() `[1/2]`

```
Int32 forceMultipleCoils (
    Int32 slaveAddr,
    Int32 startRef,
    bool [] bitArr )  [inline], [inherited]
```

Writes binary values into a sequence of discrete outputs (coils, 0:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which contains the data to be sent. The length of the array determines how many coils are written (Range: 1-1968).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

forceMultipleCoils() `[2/2]`

```
Int32 forceMultipleCoils (
    Int32 slaveAddr,
    Int32 startRef,
    bool [] bitArr,
    Int32 numCoils )  [inline], [inherited]
```

Writes binary values into a sequence of discrete outputs (coils, 0:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which contains the data to be sent.
numCoils	Number of coils to be written (Range: 1-1968).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleRegisters() `[1/2] Int32 readMultipleRegisters (`
`Int32 slaveAddr,`
`Int32 startRef,`
`[In,Out] System.Array regArr ) [inline], [inherited]`

Reads the contents of the output registers (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[]. The length of the array determines how many registers are read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleRegisters() `[2/2] Int32 readMultipleRegisters (`
`Int32 slaveAddr,`
`Int32 startRef,`
`[In,Out] System.Array regArr,`
`Int32 numRegs ) [inline], [inherited]`

Reads the contents of the output registers (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)

Parameters

startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[].
numRegs	Number of values to be read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputRegisters() `[1/2]`

```
Int32 readInputRegisters (
    Int32 slaveAddr,
    Int32 startRef,
    [In, Out] System.Array regArr )  [inline], [inherited]
```

Read the contents of the input registers (3:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[]. The length of the array determines how many registers are read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputRegisters() `[2/2]`

```
Int32 readInputRegisters (
    Int32 slaveAddr,
    Int32 startRef,
    [In, Out] System.Array regArr,
    Int32 numRegs )  [inline], [inherited]
```

Read the contents of the input registers (3:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[].
numRegs	Number of values to be read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeSingleRegister() `[1/2]` `Int32 writeSingleRegister (`
`Int32 slaveAddr,`
`Int32 regAddr,`
`Int16 regVal )` `[inline], [inherited]`

Writes a value into a single output register (holding register, 4:00000 reference).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
regAddr	Register address (Range: 1 - 65536)
regVal	Data to be sent as signed 16-bit value

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeSingleRegister() `[2/2]` `Int32 writeSingleRegister (`

```
Int32 slaveAddr,
Int32 regAddr,
UInt16 regVal )  [inline], [inherited]
```

Writes a value into a single output register (holding register, 4:00000 reference).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
regAddr	Register address (Range: 1 - 65536)
regVal	Data to be sent as unsigned 16-bit value

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleRegisters()

```
[1/2]  Int32 writeMultipleRegisters (
Int32 slaveAddr,
Int32 startRef,
System.Array regArr )  [inline], [inherited]
```

Writes values into a sequence of output registers (holding registers, 4:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array with data to be sent. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[]. The length of the array determines how many registers are written (Range: 1-123 for 16-bit, 1-61 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleRegisters() `[2/2] Int32 writeMultipleRegisters (`
`Int32 slaveAddr,`
`Int32 startRef,`
`System.Array regArr,`
`Int32 numRegs ) [inline], [inherited]`

Writes values into a sequence of output registers (holding registers, 4:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array with data to be sent. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[].
numRegs	Number of values to be written (Range: 1-123 for 16-bit, 1-61 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

maskWriteRegister() `Int32 maskWriteRegister (`
`Int32 slaveAddr,`
`Int32 regAddr,`
`Int16 andMask,`
`Int16 orMask ) [inline], [inherited]`

Masks bits according to an AND and an OR mask into a single output register (holding register, 4:00000 reference). Masking is done as follows: retVal = (currentVal AND andMask) OR (orMask AND (NOT andMask))

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
regAddr	Register address (Range: 1 - 65536)
andMask	Mask to be applied as a logic AND to the register
orMask	Mask to be applied as a logic OR to the register

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readWriteRegisters() `[1/2]`

```
Int32 readWriteRegisters (
    Int32 slaveAddr,
    Int32 readRef,
    [In,Out] Int16 [] readArr,
    Int32 writeRef,
    Int16 [] writeArr )  [inline], [inherited]
```

Combines reading and writing of the output registers in one transaction (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
readRef	Start registers for reading (Range: 1 - 65536)
readArr	Buffer which will contain the data read. The length of the array determines how many registers are read (Range: 1-125).
writeRef	Start registers for writing (Range: 1 - 65536)
writeArr	Buffer with data to be sent. The length of the array determines how many registers are written (Range: 1-121).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readWriteRegisters() `[2/2]`

```
Int32 readWriteRegisters (
    Int32 slaveAddr,
```

```
Int32 readRef,
[In,Out] Int16 [] readArr,
Int32 readCnt,
Int32 writeRef,
Int16 [] writeArr,
Int32 writeCnt )  [inline], [inherited]
```

Combines reading and writing of the output registers in one transaction (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
readRef	Start register for reading (Range: 1 - 65536)
readArr	Buffer which will contain the data read.
writeRef	Start register for writing (Range: 1 - 65536)
writeArr	Buffer with data to be sent.
readCnt	Number of registers to be read (Range: 1-125).
writeCnt	Number of registers to be written (Range: 1-121).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleMod10000() `[1/2]  Int32 readMultipleMod10000 (`

```
Int32 slaveAddr,
Int32 startRef,
[In,Out] Int32 [] int32Arr )  [inline], [inherited]
```

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value into 32-bit int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleMod10000() `[2/2]` `Int32 readMultipleMod10000 (`

```
Int32 slaveAddr,
Int32 startRef,
[In,Out] Int32 [] int32Arr,
Int32 numRegs )  [inline], [inherited]
```

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value into 32-bit int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read.
numRegs	Number of values to be read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputMod10000() `[1/2]` `Int32 readInputMod10000 (`

```
Int32 slaveAddr,
Int32 startRef,
[In,Out] Int32 [] int32Arr )  [inline], [inherited]
```

Reads the contents of pairs of consecutive input registers (3:00000 table) representing a modulo-10000 long int value into 32-bit long int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputMod10000() `[2/2]`

```
Int32 readInputMod10000 (
Int32 slaveAddr,
Int32 startRef,
[In,Out] Int32 [] int32Arr,
Int32 numRegs )  [inline], [inherited]
```

Reads the contents of pairs of consecutive input registers (3:00000 table) representing a modulo-10000 long int value into 32-bit long int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read.
numRegs	Number of values to be read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleMod10000() `[1/2] Int32 writeMultipleMod10000 (`

```
Int32 slaveAddr,
Int32 startRef,
Int32 [] int32Arr )  [inline], [inherited]
```

Writes long int values into pairs of output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function. Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer with the data to be sent. The length of the array determines how many values are written (Range: 1-61).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleMod10000() `[2/2] Int32 writeMultipleMod10000 (`

```
Int32 slaveAddr,
```

```
Int32 startRef,
Int32 [] int32Arr,
Int32 numRegs )  [inline], [inherited]
```

Writes long int values into pairs of output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function. Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer with the data to be sent.
numRegs	Number of values to be sent (Range: 1-61).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readExceptionStatus()

```
Int32 readExceptionStatus (
 Int32 slaveAddr,
 out byte statusByte )  [inline], [inherited]
```

Reads the eight exception status coils within the slave device.

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
statusByte	Slave status byte. The meaning of this status byte is slave specific and varies from device to device.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

returnQueryData() `Int32 returnQueryData (`
`Int32 slaveAddr,`
`byte [] queryArr,`
`out byte [] echoArr ) [inline], [inherited]`

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255 for serial, 0 - 255 for TCP)
queryArr	Buffer with data to be sent. The length of the array determines how many bytes are sent and returned
echoArr	Buffer which will contain the data read

Returns

BusProtocolErrors.FTALK_SUCCESS on success, FTALK_INVALID_REPLY_ERROR if reply does not match query data or error code. See BusProtocolErrors for more error codes.

References BusProtocolErrors.FTALK_INVALID_REPLY_ERROR, and BusProtocolErrors.FTALK_SUCCESS.

restartCommunicationsOption() `Int32 restartCommunicationsOption (`
`Int32 slaveAddr,`
`Int32 clearEventLog ) [inline], [inherited]`

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255 for serial, 0 - 255 for TCP)
clearEventLog	Flag when set to one clears in addition the slave's communication even log.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

customFunction() `Int32 customFunction (`
`Int32 slaveAddr,`
`Int32 functionCode,`
`byte [] requestData,`
`[In,Out] byte [] responseData ) [inline], [inherited]`

This method can be used to implement User Defined Function Codes. The caller has only to pass the user data to this function. The assembly of the Modbus frame (the so called ADU) including checksums, slave address and function code and subsequentially the transmission, is taken care of by this method.

The modbus specification reserves function codes 65-72 and 100-110 for user defined functions.

Note

Modbus functions usually have an implied response length and therefore the number of bytes expected to be received is known at the time when sending the request. In case of a custom Modbus function with an open or unknown response length, this function can not be used.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255 for serial, 0 - 255 for TCP)
functionCode	Custom function code to be used for Modbus transaction (Range: 1-127)
requestData	Array with data to be sent as request (not including slave address or function code). The length of the array determines how many request bytes are sent (Range: 0-252).
responseData	Buffer which will be filled with the response data received. The length of the array determines how many bytes are read (Range: 0-252).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

setTimeout() `Int32 setTimeout (`
`Int32 timeOut ) [inline], [inherited]`

Configures operation or socket time-out.

The time-out value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Parameters

timeOut	Timeout value in ms (Range: 1 - 100000)

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

getTimeout()

```
Int32 getTimeout ( )  [inline], [inherited]
```

Returns

Timeout value in ms

setPollDelay()

```
Int32 setPollDelay (
        Int32 pollDelay )  [inline], [inherited]
```

This function sets the delay time which applies between two consecutive Modbus read/write. A value of 0 disables the poll delay.

The delay value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Parameters

pollDelay	Delay value in ms (Range: 0 - 100000), 0 disables poll delay

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

getPollDelay() `Int32 getPollDelay ( ) [inline], [inherited]`

Returns

Delay time in ms, 0 if poll delay is switched off

setRetryCnt() `Int32 setRetryCnt ( Int32 retryCnt ) [inline], [inherited]`

Configures the automatic retry setting. A value of 0 disables any automatic retries.

Note

A protocol must be closed in order to configure it.

Parameters

retryCnt	Retry count (Range: 0 - 10), 0 disables retries

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

getRetryCnt() `Int32 getRetryCnt ( ) [inline], [inherited]`

Returns

Retry count

getTotalCounter() `Int32 getTotalCounter ( ) [inline], [inherited]`

Returns

Counter value

resetTotalCounter() `void resetTotalCounter ( ) [inline], [inherited]`

getSuccessCounter() `Int32 getSuccessCounter ( ) [inline], [inherited]`

Returns

Counter value

resetSuccessCounter() `void resetSuccessCounter ( ) [inline], [inherited]`

configureStandard32BitMode() `[1/2] void configureStandard32BitMode ( ) [inline], [inherited]`

In Standard 32-bit Register Mode a 32-bit value is transmitted as two consecutive 16-bit Modbus registers.

Note

This function call also re-configures the endianess to default little-endian for 32-bit values!

Note

This is the default mode.

configureStandard32BitMode() `[2/2] Int32 configureStandard32BitMode ( Int32 slaveAddr ) [inline], [inherited]`

In Standard 32-bit Register Mode a 32-bit value is transmitted as two consecutive 16-bit Modbus registers.

Note

This function call also re-configures the endianess to default little-endian for 32-bit values!

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureEnron32BitMode() `[1/2]` `void configureEnron32BitMode ( ) [inline], [inherited]`

Some Modbus flavours like the Daniel/Enron protocol represent a 32-bit value using one 32-bit Modbus register instead of two 16-bit registers.

Note

This function call also re-configures the endianess to big-endian for 32-bit values as defined by the Daniel/ENRON protocol!

configureEnron32BitMode() `[2/2]` `Int32 configureEnron32BitMode ( Int32 slaveAddr ) [inline], [inherited]`

Some Modbus flavours like the Daniel/Enron protocol represent a 32-bit value using one 32-bit Modbus register instead of two 16-bit registers.

Note

This function call also re-configures the endianess to big-endian for 32-bit values as defined by the Daniel/ENRON protocol!

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureCountFromOne() `[1/2]` `void configureCountFromOne ( ) [inline], [inherited]`

This renders the reference range to be 1 to 65536 (0x10000) and register #0 is an illegal register.

Note

This is the default mode.

configureCountFromOne() `[2/2]` `Int32 configureCountFromOne ( Int32 slaveAddr ) [inline], [inherited]`

This renders the reference range to be 1 to 65536 (0x10000) and register #0 is an illegal register.

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureCountFromZero() `void configureCountFromZero ( ) [inline], [inherited]`

This renders the valid reference range to be 0 to 65535 (0xFFFF).

This renders the first register to be #0 for all slaves.

configureBigEndianInts() `[1/2]` `void configureBigEndianInts ( ) [inline], [inherited]`

Modbus is using little-endian word order for 32-bit values. The data transfer functions operating upon 32-bit int data types can be configured to do a word swap which enables them to read 32-bit data correctly from a big-endian slave.

configureBigEndianInts() `[2/2]` `Int32 configureBigEndianInts ( Int32 slaveAddr ) [inline], [inherited]`

Modbus is using little-endian word order for 32-bit values. The data transfer functions operating upon 32-bit int data types can be configured to do a word swap which enables them to read 32-bit data correctly from a big-endian slave.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureSwappedFloats() [1/2] `void configureSwappedFloats ( ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Most platforms store floats in IEEE 754 little-endian order which does not need a word swap.

configureSwappedFloats() [2/2] `Int32 configureSwappedFloats ( Int32 slaveAddr ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Most platforms store floats in IEEE 754 little-endian order which does not need a word swap.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureLittleEndianInts() [1/2] `void configureLittleEndianInts ( ) [inline], [inherited]`

Note

This is the default mode.

configureLittleEndianInts() `[2/2] Int32 configureLittleEndianInts ( Int32 slaveAddr ) [inline], [inherited]`

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureIeeeFloats() `[1/2] void configureIeeeFloats ( ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Note

This is the default mode.

configureIeeeFloats() `[2/2] Int32 configureIeeeFloats ( Int32 slaveAddr ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

onfigureCountFromZero() `Int32 onfigureCountFromZero ( Int32 slaveAddr ) [inline], [inherited]`

This renders the valid reference range to be 0 to 65535 (0xFFFF).

This renders the first register to be #0 for all slaves.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

isOpen() `bool isOpen ( ) [inline], [inherited]`

Returns

True = open, False = closed

closeProtocol() `void closeProtocol ( ) [inline], [inherited]`

getPackageVersion() `static string getPackageVersion ( ) [inline], [static], [inherited]`

Returns

Package version string

5.1.4 Member Data Documentation

SER_DATABITS_7 `const Int32 SER_DATABITS_7 = 7 [inherited]`

SER_DATABITS_8 `const Int32 SER_DATABITS_8 = 8 [inherited]`

SER_STOPBITS_1 `const Int32 SER_STOPBITS_1 = 1 [inherited]`

SER_STOPBITS_2 `const Int32 SER_STOPBITS_2 = 2 [inherited]`

SER_PARITY_NONE `const Int32 SER_PARITY_NONE = 0 [inherited]`

SER_PARITY_ODD `const Int32 SER_PARITY_ODD = 1 [inherited]`

SER_PARITY_EVEN `const Int32 SER_PARITY_EVEN = 2 [inherited]`

5.1.5 Property Documentation

portName `string portName [get], [set], [inherited]`

Note

A protocol must be closed in order to configure it.

Serial port identifier (eg "COM1")

baudRate `Int32 baudRate [get], [set], [inherited]`

Note

A protocol must be closed in order to configure it.

Typically 1200 - 115200, maximum value depends on UART hardware

dataBits `Int32 dataBits [get], [set], [inherited]`

Note

A protocol must be closed in order to configure it.

SER_DATABITS_7 as 7 data bits (ASCII protocol only), SER_DATABITS_8 as data bits

stopBits `Int32 stopBits [get], [set], [inherited]`

Note

The Modbus standard requires two stop bits if no parity is chosen. This library is not enforcing this but it is a recommended configuration.
A protocol must be closed in order to configure it.

SER_STOPBITS_1 as 1 stop bit, SER_STOPBITS_2 as 2 stop bits

parity `Int32 parity [get], [set], [inherited]`

Note

The Modbus standard requires two stop bits if no parity is chosen. This library is not enforcing this but it is a recommended configuration.
A protocol must be closed in order to configure it.

SER_PARITY_NONE as no parity, SER_PARITY_ODD as odd parity, SER_PARITY_EVEN as even parity

timeout `Int32 timeout [get], [set], [inherited]`

Configures operation or socket time-out.

The time-out value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Timeout value in ms (Range: 1 - 100000)

pollDelay `Int32 pollDelay [get], [set], [inherited]`

This property sets the delay time which applies between two consecutive Modbus read/write. A value of 0 disables the poll delay.

The delay value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Delay value in ms (Range: 0 - 100000), 0 disables poll delay

retryCnt `Int32 retryCnt [get], [set], [inherited]`

Configures the automatic retry setting. A value of 0 disables any automatic retries.

Note

A protocol must be closed in order to configure it.

Retry count (Range: 0 - 10), 0 disables retries

5.2 MbusAsciiMasterProtocol Class Reference

Modbus ASCII Master Protocol class

Public Member Functions

- MbusAsciiMasterProtocol ()

  Creates new instance
- Int32 openProtocol ()

  Opens a serial Modbus protocol and the associated serial port with the port parameters configured via properties
- Int32 openProtocol (string portName, Int32 baudRate, Int32 dataBits, Int32 stopBits, Int32 parity)

  Opens a serial Modbus protocol and the associated serial port with the port parameters configured via properties
- Int32 enableRs485Mode (Int32 rtsDelay)

  Enables RS485 mode
- bool isOpen ()

  Returns whether the protocol is open or not.
- void closeProtocol ()

  Closes an open protocol including any associated communication resources (COM ports or sockets).

Static Public Member Functions

- static string getPackageVersion ()

  Returns the package version number.

Public Attributes

- const Int32 SER_DATABITS_7 = 7

  7 data bits
- const Int32 SER_DATABITS_8 = 8

  8 data bits
- const Int32 SER_STOPBITS_1 = 1

1 stop bit

- const Int32 SER_STOPBITS_2 = 2

  2 stop bits
- const Int32 SER_PARITY_NONE = 0

  No parity
- const Int32 SER_PARITY_ODD = 1

  Odd parity
- const Int32 SER_PARITY_EVEN = 2

  Even parity

Properties

- string portName `[get, set]`

  Serial port identifier property
- Int32 baudRate `[get, set]`

  Baud rate property in bps
- Int32 dataBits `[get, set]`

  Data bits property
- Int32 stopBits `[get, set]`

  Stop bits property
- Int32 parity `[get, set]`

  Parity property
- Int32 timeout `[get, set]`

  Time-out port property
- Int32 pollDelay `[get, set]`

  Poll delay property
- Int32 retryCnt `[get, set]`

  Retry count property

Bit Access

Table 0:00000 (Coils) and Table 1:0000 (Input Status)

- Int32 readCoils (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr)

  Modbus function 1 (01 hex), Read Coil Status/Read Coils.
- Int32 readCoils (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr, Int32 numCoils)

  Modbus function 1 (01 hex), Read Coil Status/Read Coils.
- Int32 readInputDiscretes (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr)

  Modbus function 2 (02 hex), Read Inputs Status/Read Input Discretes.
- Int32 readInputDiscretes (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr, Int32 numDiscretes)

  Modbus function 2 (02 hex), Read Inputs Status/Read Input Discretes.
- Int32 writeCoil (Int32 slaveAddr, Int32 bitAddr, bool bitVal)

Modbus function 5 (05 hex), Force Single Coil/Write Coil.

- Int32 forceMultipleCoils (Int32 slaveAddr, Int32 startRef, bool[ ] bitArr)

  Modbus function 15 (0F hex), Force Multiple Coils.

- Int32 forceMultipleCoils (Int32 slaveAddr, Int32 startRef, bool[ ] bitArr, Int32 numCoils)

  Modbus function 15 (0F hex), Force Multiple Coils.

Register Access (16-bit, 32-bit and floating point)

Table 4:00000 (Holding Registers) and Table 3:00000 (Input Registers)

- Int32 readMultipleRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr)

  Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers (16-bit, 32-bit and floating point).

- Int32 readMultipleRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr, Int32 numRegs)

  Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers (16-bit, 32-bit and floating point).

- Int32 readInputRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr)

  Modbus function 4 (04 hex), Read Input Registers (16-bit, 32-bit and floating point).

- Int32 readInputRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr, Int32 numRegs)

  Modbus function 4 (04 hex), Read Input Registers (16-bit, 32-bit and floating point).

- Int32 writeSingleRegister (Int32 slaveAddr, Int32 regAddr, Int16 regVal)

  Modbus function 6 (06 hex), Preset Single Register/Write Single Register.

- Int32 writeSingleRegister (Int32 slaveAddr, Int32 regAddr, UInt16 regVal)

  Modbus function 6 (06 hex), Preset Single Register/Write Single Register.

- Int32 writeMultipleRegisters (Int32 slaveAddr, Int32 startRef, System.Array regArr)

  Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers (16-bit, 32-bit and floating point).

- Int32 writeMultipleRegisters (Int32 slaveAddr, Int32 startRef, System.Array regArr, Int32 numRegs)

  Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers (16-bit, 32-bit and floating point).

- Int32 maskWriteRegister (Int32 slaveAddr, Int32 regAddr, Int16 andMask, Int16 or↩ Mask)

  Modbus function 22 (16 hex), Mask Write Register.

- Int32 readWriteRegisters (Int32 slaveAddr, Int32 readRef, [In, Out] Int16[ ] readArr, Int32 writeRef, Int16[ ] writeArr)

  Modbus function 23 (17 hex), Read/Write Registers.

- Int32 readWriteRegisters (Int32 slaveAddr, Int32 readRef, [In, Out] Int16[ ] readArr, Int32 readCnt, Int32 writeRef, Int16[ ] writeArr, Int32 writeCnt)

  Modbus function 23 (17 hex), Read/Write Registers.

Modulo-10000 long integer Access

Table 4:00000 (Holding Registers) and Table 3:00000 (Input Registers)

- Int32 readMultipleMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32↩ Arr)

  Modbus function 3 (03 hex) for 32-bit modulo-10000 long int data types, Read Holding Registers/Read Multiple Registers as modulo-10000 long int data.

- Int32 readMultipleMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32↩ Arr, Int32 numRegs)

  Modbus function 3 (03 hex) for 32-bit modulo-10000 long int data types, Read Holding Registers/Read Multiple Registers as modulo-10000 long int data.

- Int32 readInputMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32Arr)

  Modbus function 4 (04 hex) for 32-bit modulo-10000 long int data types, Read Input Registers as modulo-10000 long int data.

- Int32 readInputMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32Arr, Int32 numRegs)

  Modbus function 4 (04 hex) for 32-bit modulo-10000 long int data types, Read Input Registers as modulo-10000 long int data.

- Int32 writeMultipleMod10000 (Int32 slaveAddr, Int32 startRef, Int32[ ] int32Arr)

  Modbus function 16 (10 hex) for 32-bit modulo-10000 long int data types, Preset Multiple Registers/Write Multiple Registers with modulo-10000 long int data.

- Int32 writeMultipleMod10000 (Int32 slaveAddr, Int32 startRef, Int32[ ] int32Arr, Int32 numRegs)

  Modbus function 16 (10 hex) for 32-bit modulo-10000 long int data types, Preset Multiple Registers/Write Multiple Registers with modulo-10000 long int data.

Diagnostics

- Int32 readExceptionStatus (Int32 slaveAddr, out byte statusByte)

  Modbus function 7 (07 hex), Read Exception Status.

- Int32 returnQueryData (Int32 slaveAddr, byte[ ] queryArr, out byte[ ] echoArr)

  Modbus function code 8, sub-function 00, Return Query Data.

- Int32 restartCommunicationsOption (Int32 slaveAddr, Int32 clearEventLog)

  Modbus function code 8, sub-function 01, Restart Communications Option

User Defined Function Codes

- Int32 customFunction (Int32 slaveAddr, Int32 functionCode, byte[ ] requestData, [In, Out] byte[ ] responseData)

  User Defined Function Code

Protocol Configuration

- Int32 setTimeout (Int32 timeOut)

  Configures time-out
- Int32 getTimeout ()

  Returns the current time-out setting
- Int32 setPollDelay (Int32 pollDelay)

  Poll delay property
- Int32 getPollDelay ()

  Returns the poll delay time
- Int32 setRetryCnt (Int32 retryCnt)

  Configures the automatic retry setting
- Int32 getRetryCnt ()

  Returns the automatic retry count

Transmission Statistic Functions

- Int32 getTotalCounter ()

  Returns how often a message transfer has been executed
- void resetTotalCounter ()

  Resets total message transfer counter
- Int32 getSuccessCounter ()

  Returns how often a message transfer was successful
- void resetSuccessCounter ()

  Resets successful message transfer counter

Slave Configuration

- void configureStandard32BitMode ()

  Configures all slaves for Standard 32-bit Mode.
- Int32 configureStandard32BitMode (Int32 slaveAddr)

  Configures a slave for Standard 32-bit Mode.
- void configureEnron32BitMode ()

  Configures all slaves for Daniel/ENRON 32-bit Mode.
- Int32 configureEnron32BitMode (Int32 slaveAddr)

  Configures a slave for Daniel/ENRON 32-bit Mode.
- void configureCountFromOne ()

  Configures the reference counting scheme to start with one for all slaves.
- Int32 configureCountFromOne (Int32 slaveAddr)

  Configures the reference counting scheme to start with one for a slave.
- void configureCountFromZero ()

  Configures the reference counting scheme to start with zero for all slaves.
- void configureBigEndianInts ()

Configures 32-bit int data type functions to do a word swap

- Int32 configureBigEndianInts (Int32 slaveAddr)

  Configures 32-bit int data type functions to do a word swap on a per slave basis
- void configureSwappedFloats ()

  Configures float data type functions to do a word swap
- Int32 configureSwappedFloats (Int32 slaveAddr)

  Configures float data type functions to do a word swap on a per slave basis
- void configureLittleEndianInts ()

  Configures 32-bit int data type functions NOT to do a word swap
- Int32 configureLittleEndianInts (Int32 slaveAddr)

  Configures 32-bit int data type functions NOT to do a word swap on a per slave basis
- void configureIeeeFloats ()

  Configures float data type functions NOT to do a word swap
- Int32 configureIeeeFloats (Int32 slaveAddr)

  Configures float data type functions NOT to do a word swap on a per slave basis
- Int32 onfigureCountFromZero (Int32 slaveAddr)

  Configures the reference counting scheme to start with zero for a slave.

5.2.1 Detailed Description

This class realizes the Modbus ASCII master protocol. It provides functions to open and to close serial port as well as data and control functions which can be used at any time after the protocol has been opened. The data and control functions are organized into different conformance classes.

It is possible to instantiate multiple instances of this class for establishing multiple connections on different serial ports (They should be executed in separate threads).

5.2.2 Constructor & Destructor Documentation

MbusAsciiMasterProtocol() `MbusAsciiMasterProtocol ( ) [inline]`

Exceptions

OutOfMemoryException	Creation of class failed

5.2.3 Member Function Documentation

openProtocol() `[1/2] Int32 openProtocol ( ) [inline], [inherited]`

This function opens the serial port. After a port has been opened, data and control functions can be used.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

openProtocol() [2/2]

```
Int32 openProtocol (
    string portName,
    Int32 baudRate,
    Int32 dataBits,
    Int32 stopBits,
    Int32 parity )  [inline], [inherited]
```

This function opens the serial port with a specific port settings. After a port has been opened, data and control functions can be used.

Note

The Modbus standard requires two stop bits if no parity is chosen. This library is not enforcing this but it is a recommended configuration.

Parameters

portName	Serial port identifier (eg "COM1")
baudRate	The port baud rate in bps (1200 - 115200, higher on some platforms)
dataBits	SER_DATABITS_7: 7 data bits (ASCII protocol only), SER_DATABITS_8: 8 data bits
stopBits	SER_STOPBITS_1: 1 stop bit, SER_STOPBITS_2: 2 stop bits
parity	SER_PARITY_NONE: no parity, SER_PARITY_ODD: odd parity, SER_PARITY_EVEN: even parity

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

enableRs485Mode()

```
Int32 enableRs485Mode (
    Int32 rtsDelay )  [inline], [inherited]
```

In RS485 mode the RTS signal can be used to enable and disable the transmitter of a RS232/RS485 converter. The RTS signal is asserted before sending data. It is cleared after the transmit buffer has been emptied and in addition the specified delay time has elapsed.

The delay time is necessary because even the transmit buffer is already empty, the UART's FIFO will still contain unsent characters.

The use of RTS controlled RS232/RS485 converters should be avoided if possible. It is difficult to determine the exact time when to switch off the transmitter with non real-time operating systems like Windows and Linux. If it is switched off to early characters might still sit in the FIFO or the transmit register of the UART and these characters will be lost. Hence the slave will not recognize the message. On the other hand if it is switched off too late then the slave's message is corrupted and the master will not recognize the message.

Note

The delay value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.
A protocol must be closed in order to configure it.

Parameters

rtsDelay	Delay time in ms (Range as 0 - 100000) which applies after the transmit buffer is empty. 0 disables this mode.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readCoils()

```
[1/2]  Int32 readCoils (
           Int32 slaveAddr,
           Int32 startRef,
           [In,Out] bool [] bitArr )  [inline], [inherited]
```

Reads the contents of the discrete outputs (coils, 0:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read. The length of the array determines how many coils are read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readCoils() [2/2]

```
Int32 readCoils (
        Int32 slaveAddr,
        Int32 startRef,
        [In,Out] bool [] bitArr,
        Int32 numCoils )  [inline], [inherited]
```

Reads the contents of the discrete outputs (coils, 0:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read.
numCoils	Number of coils to be read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputDiscretes() [1/2]

```
Int32 readInputDiscretes (
        Int32 slaveAddr,
        Int32 startRef,
        [In,Out] bool [] bitArr )  [inline], [inherited]
```

Reads the contents of the discrete inputs (input status, 1:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read. The length of the array determines how many inputs are read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputDiscretes() [2/2]

```
Int32 readInputDiscretes (
    Int32 slaveAddr,
    Int32 startRef,
    [In,Out] bool [] bitArr,
    Int32 numDiscretes ) [inline], [inherited]
```

Reads the contents of the discrete inputs (input status, 1:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read.
num↩ Discretes	Number of inputs to be read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeCoil()

```
Int32 writeCoil (
    Int32 slaveAddr,
    Int32 bitAddr,
    bool bitVal ) [inline], [inherited]
```

Sets a single discrete output variable (coil, 0:00000 table) to either ON or OFF.

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
bitAddr	Coil address (Range: 1 - 65536)
bitVal	true sets, false clears discrete output variable

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

forceMultipleCoils() `[1/2]`

```
Int32 forceMultipleCoils (
    Int32 slaveAddr,
    Int32 startRef,
    bool [] bitArr )  [inline], [inherited]
```

Writes binary values into a sequence of discrete outputs (coils, 0:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which contains the data to be sent. The length of the array determines how many coils are written (Range: 1-1968).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

forceMultipleCoils() `[2/2]`

```
Int32 forceMultipleCoils (
    Int32 slaveAddr,
    Int32 startRef,
    bool [] bitArr,
    Int32 numCoils )  [inline], [inherited]
```

Writes binary values into a sequence of discrete outputs (coils, 0:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which contains the data to be sent.
numCoils	Number of coils to be written (Range: 1-1968).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleRegisters() `[1/2] Int32 readMultipleRegisters (`
`Int32 slaveAddr,`
`Int32 startRef,`
`[In,Out] System.Array regArr ) [inline], [inherited]`

Reads the contents of the output registers (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[]. The length of the array determines how many registers are read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleRegisters() `[2/2] Int32 readMultipleRegisters (`
`Int32 slaveAddr,`
`Int32 startRef,`
`[In,Out] System.Array regArr,`
`Int32 numRegs ) [inline], [inherited]`

Reads the contents of the output registers (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)

Parameters

startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[].
numRegs	Number of values to be read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputRegisters() `[1/2]`

```
Int32 readInputRegisters (
    Int32 slaveAddr,
    Int32 startRef,
    [In, Out] System.Array regArr )  [inline], [inherited]
```

Read the contents of the input registers (3:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[]. The length of the array determines how many registers are read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputRegisters() `[2/2]`

```
Int32 readInputRegisters (
    Int32 slaveAddr,
    Int32 startRef,
    [In, Out] System.Array regArr,
    Int32 numRegs )  [inline], [inherited]
```

Read the contents of the input registers (3:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[].
numRegs	Number of values to be read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeSingleRegister() `[1/2]` `Int32 writeSingleRegister (`
`Int32 slaveAddr,`
`Int32 regAddr,`
`Int16 regVal )` `[inline], [inherited]`

Writes a value into a single output register (holding register, 4:00000 reference).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
regAddr	Register address (Range: 1 - 65536)
regVal	Data to be sent as signed 16-bit value

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeSingleRegister() `[2/2]` `Int32 writeSingleRegister (`

```
Int32 slaveAddr,
Int32 regAddr,
UInt16 regVal )  [inline], [inherited]
```

Writes a value into a single output register (holding register, 4:00000 reference).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
regAddr	Register address (Range: 1 - 65536)
regVal	Data to be sent as unsigned 16-bit value

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleRegisters() `[1/2]  Int32 writeMultipleRegisters (`

```
Int32 slaveAddr,
Int32 startRef,
System.Array regArr )  [inline], [inherited]
```

Writes values into a sequence of output registers (holding registers, 4:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array with data to be sent. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[]. The length of the array determines how many registers are written (Range: 1-123 for 16-bit, 1-61 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleRegisters() `[2/2] Int32 writeMultipleRegisters (`
`Int32 slaveAddr,`
`Int32 startRef,`
`System.Array regArr,`
`Int32 numRegs ) [inline], [inherited]`

Writes values into a sequence of output registers (holding registers, 4:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array with data to be sent. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[].
numRegs	Number of values to be written (Range: 1-123 for 16-bit, 1-61 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

maskWriteRegister() `Int32 maskWriteRegister (`
`Int32 slaveAddr,`
`Int32 regAddr,`
`Int16 andMask,`
`Int16 orMask ) [inline], [inherited]`

Masks bits according to an AND and an OR mask into a single output register (holding register, 4:00000 reference). Masking is done as follows: retVal = (currentVal AND andMask) OR (orMask AND (NOT andMask))

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
regAddr	Register address (Range: 1 - 65536)
andMask	Mask to be applied as a logic AND to the register
orMask	Mask to be applied as a logic OR to the register

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readWriteRegisters() `[1/2]`

```
Int32 readWriteRegisters (
    Int32 slaveAddr,
    Int32 readRef,
    [In,Out] Int16 [] readArr,
    Int32 writeRef,
    Int16 [] writeArr )  [inline], [inherited]
```

Combines reading and writing of the output registers in one transaction (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
readRef	Start registers for reading (Range: 1 - 65536)
readArr	Buffer which will contain the data read. The length of the array determines how many registers are read (Range: 1-125).
writeRef	Start registers for writing (Range: 1 - 65536)
writeArr	Buffer with data to be sent. The length of the array determines how many registers are written (Range: 1-121).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readWriteRegisters() `[2/2]`

```
Int32 readWriteRegisters (
    Int32 slaveAddr,
```

```
Int32 readRef,
[In,Out] Int16 [] readArr,
Int32 readCnt,
Int32 writeRef,
Int16 [] writeArr,
Int32 writeCnt )  [inline], [inherited]
```

Combines reading and writing of the output registers in one transaction (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
readRef	Start register for reading (Range: 1 - 65536)
readArr	Buffer which will contain the data read.
writeRef	Start register for writing (Range: 1 - 65536)
writeArr	Buffer with data to be sent.
readCnt	Number of registers to be read (Range: 1-125).
writeCnt	Number of registers to be written (Range: 1-121).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleMod10000() `[1/2]`

```
Int32 readMultipleMod10000 (
Int32 slaveAddr,
Int32 startRef,
[In,Out] Int32 [] int32Arr )  [inline], [inherited]
```

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value into 32-bit int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleMod10000() `[2/2] Int32 readMultipleMod10000 (`

```
Int32 slaveAddr,
Int32 startRef,
[In,Out] Int32 [] int32Arr,
Int32 numRegs ) [inline], [inherited]
```

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value into 32-bit int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read.
numRegs	Number of values to be read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputMod10000() `[1/2] Int32 readInputMod10000 (`

```
Int32 slaveAddr,
Int32 startRef,
[In,Out] Int32 [] int32Arr )  [inline], [inherited]
```

Reads the contents of pairs of consecutive input registers (3:00000 table) representing a modulo-10000 long int value into 32-bit long int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputMod10000() `[2/2]`

```
Int32 readInputMod10000 (
Int32 slaveAddr,
Int32 startRef,
[In,Out] Int32 [] int32Arr,
Int32 numRegs )  [inline], [inherited]
```

Reads the contents of pairs of consecutive input registers (3:00000 table) representing a modulo-10000 long int value into 32-bit long int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read.
numRegs	Number of values to be read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleMod10000() `[1/2] Int32 writeMultipleMod10000 (`

```
Int32 slaveAddr,
Int32 startRef,
Int32 [] int32Arr )  [inline], [inherited]
```

Writes long int values into pairs of output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function. Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer with the data to be sent. The length of the array determines how many values are written (Range: 1-61).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleMod10000() `[2/2] Int32 writeMultipleMod10000 (`

```
Int32 slaveAddr,
```

```
Int32 startRef,
Int32 [] int32Arr,
Int32 numRegs )  [inline], [inherited]
```

Writes long int values into pairs of output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function. Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer with the data to be sent.
numRegs	Number of values to be sent (Range: 1-61).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readExceptionStatus()

```
Int32 readExceptionStatus (
 Int32 slaveAddr,
 out byte statusByte )  [inline], [inherited]
```

Reads the eight exception status coils within the slave device.

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
statusByte	Slave status byte. The meaning of this status byte is slave specific and varies from device to device.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

returnQueryData() `Int32 returnQueryData (`
`Int32 slaveAddr,`
`byte [] queryArr,`
`out byte [] echoArr ) [inline], [inherited]`

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255 for serial, 0 - 255 for TCP)
queryArr	Buffer with data to be sent. The length of the array determines how many bytes are sent and returned
echoArr	Buffer which will contain the data read

Returns

BusProtocolErrors.FTALK_SUCCESS on success, FTALK_INVALID_REPLY_ERROR if reply does not match query data or error code. See BusProtocolErrors for more error codes.

References BusProtocolErrors.FTALK_INVALID_REPLY_ERROR, and BusProtocolErrors.FTALK_SUCCESS.

restartCommunicationsOption() `Int32 restartCommunicationsOption (`
`Int32 slaveAddr,`
`Int32 clearEventLog ) [inline], [inherited]`

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255 for serial, 0 - 255 for TCP)
clearEventLog	Flag when set to one clears in addition the slave's communication even log.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

customFunction() `Int32 customFunction (`
`Int32 slaveAddr,`
`Int32 functionCode,`
`byte [] requestData,`
`[In,Out] byte [] responseData ) [inline], [inherited]`

This method can be used to implement User Defined Function Codes. The caller has only to pass the user data to this function. The assembly of the Modbus frame (the so called ADU) including checksums, slave address and function code and subsequentially the transmission, is taken care of by this method.

The modbus specification reserves function codes 65-72 and 100-110 for user defined functions.

Note

Modbus functions usually have an implied response length and therefore the number of bytes expected to be received is known at the time when sending the request. In case of a custom Modbus function with an open or unknown response length, this function can not be used.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255 for serial, 0 - 255 for TCP)
functionCode	Custom function code to be used for Modbus transaction (Range: 1-127)
requestData	Array with data to be sent as request (not including slave address or function code). The length of the array determines how many request bytes are sent (Range: 0-252).
responseData	Buffer which will be filled with the response data received. The length of the array determines how many bytes are read (Range: 0-252).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

setTimeout() `Int32 setTimeout (`
`Int32 timeOut ) [inline], [inherited]`

Configures operation or socket time-out.

The time-out value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Parameters

timeOut	Timeout value in ms (Range: 1 - 100000)

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

getTimeout()

```
Int32 getTimeout ( )  [inline], [inherited]
```

Returns

Timeout value in ms

setPollDelay()

```
Int32 setPollDelay (
        Int32 pollDelay )  [inline], [inherited]
```

This function sets the delay time which applies between two consecutive Modbus read/write. A value of 0 disables the poll delay.

The delay value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Parameters

pollDelay	Delay value in ms (Range: 0 - 100000), 0 disables poll delay

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

getPollDelay() `Int32 getPollDelay ( ) [inline], [inherited]`

Returns

Delay time in ms, 0 if poll delay is switched off

setRetryCnt() `Int32 setRetryCnt ( Int32 retryCnt ) [inline], [inherited]`

Configures the automatic retry setting. A value of 0 disables any automatic retries.

Note

A protocol must be closed in order to configure it.

Parameters

retryCnt	Retry count (Range: 0 - 10), 0 disables retries

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

getRetryCnt() `Int32 getRetryCnt ( ) [inline], [inherited]`

Returns

Retry count

getTotalCounter() `Int32 getTotalCounter ( ) [inline], [inherited]`

Returns

Counter value

resetTotalCounter() `void resetTotalCounter ( ) [inline], [inherited]`

getSuccessCounter() `Int32 getSuccessCounter ( ) [inline], [inherited]`

Returns

Counter value

resetSuccessCounter() `void resetSuccessCounter ( ) [inline], [inherited]`

configureStandard32BitMode() `[1/2] void configureStandard32BitMode ( ) [inline], [inherited]`

In Standard 32-bit Register Mode a 32-bit value is transmitted as two consecutive 16-bit Modbus registers.

Note

This function call also re-configures the endianess to default little-endian for 32-bit values!

Note

This is the default mode.

configureStandard32BitMode() `[2/2] Int32 configureStandard32BitMode ( Int32 slaveAddr ) [inline], [inherited]`

In Standard 32-bit Register Mode a 32-bit value is transmitted as two consecutive 16-bit Modbus registers.

Note

This function call also re-configures the endianess to default little-endian for 32-bit values!

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureEnron32BitMode() `[1/2]` `void configureEnron32BitMode ( ) [inline], [inherited]`

Some Modbus flavours like the Daniel/Enron protocol represent a 32-bit value using one 32-bit Modbus register instead of two 16-bit registers.

Note

This function call also re-configures the endianess to big-endian for 32-bit values as defined by the Daniel/ENRON protocol!

configureEnron32BitMode() `[2/2]` `Int32 configureEnron32BitMode ( Int32 slaveAddr ) [inline], [inherited]`

Some Modbus flavours like the Daniel/Enron protocol represent a 32-bit value using one 32-bit Modbus register instead of two 16-bit registers.

Note

This function call also re-configures the endianess to big-endian for 32-bit values as defined by the Daniel/ENRON protocol!

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureCountFromOne() `[1/2]` `void configureCountFromOne ( ) [inline], [inherited]`

This renders the reference range to be 1 to 65536 (0x10000) and register #0 is an illegal register.

Note

This is the default mode.

configureCountFromOne() `[2/2]` `Int32 configureCountFromOne ( Int32 slaveAddr ) [inline], [inherited]`

This renders the reference range to be 1 to 65536 (0x10000) and register #0 is an illegal register.

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureCountFromZero() `void configureCountFromZero ( ) [inline], [inherited]`

This renders the valid reference range to be 0 to 65535 (0xFFFF).

This renders the first register to be #0 for all slaves.

configureBigEndianInts() `[1/2]` `void configureBigEndianInts ( ) [inline], [inherited]`

Modbus is using little-endian word order for 32-bit values. The data transfer functions operating upon 32-bit int data types can be configured to do a word swap which enables them to read 32-bit data correctly from a big-endian slave.

configureBigEndianInts() `[2/2]` `Int32 configureBigEndianInts ( Int32 slaveAddr ) [inline], [inherited]`

Modbus is using little-endian word order for 32-bit values. The data transfer functions operating upon 32-bit int data types can be configured to do a word swap which enables them to read 32-bit data correctly from a big-endian slave.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureSwappedFloats() `[1/2]` `void configureSwappedFloats ( ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Most platforms store floats in IEEE 754 little-endian order which does not need a word swap.

configureSwappedFloats() `[2/2]` `Int32 configureSwappedFloats ( Int32 slaveAddr ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Most platforms store floats in IEEE 754 little-endian order which does not need a word swap.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureLittleEndianInts() `[1/2]` `void configureLittleEndianInts ( ) [inline], [inherited]`

Note

This is the default mode.

configureLittleEndianInts() `[2/2] Int32 configureLittleEndianInts ( Int32 slaveAddr ) [inline], [inherited]`

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureIeeeFloats() `[1/2] void configureIeeeFloats ( ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Note

This is the default mode.

configureIeeeFloats() `[2/2] Int32 configureIeeeFloats ( Int32 slaveAddr ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

onfigureCountFromZero() `Int32 onfigureCountFromZero ( Int32 slaveAddr ) [inline], [inherited]`

This renders the valid reference range to be 0 to 65535 (0xFFFF).

This renders the first register to be #0 for all slaves.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

isOpen() `bool isOpen ( ) [inline], [inherited]`

Returns

True = open, False = closed

closeProtocol() `void closeProtocol ( ) [inline], [inherited]`

getPackageVersion() `static string getPackageVersion ( ) [inline], [static], [inherited]`

Returns

Package version string

5.2.4 Member Data Documentation

SER_DATABITS_7 `const Int32 SER_DATABITS_7 = 7 [inherited]`

SER_DATABITS_8 `const Int32 SER_DATABITS_8 = 8 [inherited]`

SER_STOPBITS_1 `const Int32 SER_STOPBITS_1 = 1 [inherited]`

SER_STOPBITS_2 `const Int32 SER_STOPBITS_2 = 2 [inherited]`

SER_PARITY_NONE `const Int32 SER_PARITY_NONE = 0 [inherited]`

SER_PARITY_ODD `const Int32 SER_PARITY_ODD = 1 [inherited]`

SER_PARITY_EVEN `const Int32 SER_PARITY_EVEN = 2 [inherited]`

5.2.5 Property Documentation

portName `string portName [get], [set], [inherited]`

Note

A protocol must be closed in order to configure it.

Serial port identifier (eg "COM1")

baudRate `Int32 baudRate [get], [set], [inherited]`

Note

A protocol must be closed in order to configure it.

Typically 1200 - 115200, maximum value depends on UART hardware

dataBits `Int32 dataBits [get], [set], [inherited]`

Note

A protocol must be closed in order to configure it.

SER_DATABITS_7 as 7 data bits (ASCII protocol only), SER_DATABITS_8 as data bits

stopBits `Int32 stopBits [get], [set], [inherited]`

Note

The Modbus standard requires two stop bits if no parity is chosen. This library is not enforcing this but it is a recommended configuration.
A protocol must be closed in order to configure it.

SER_STOPBITS_1 as 1 stop bit, SER_STOPBITS_2 as 2 stop bits

parity `Int32 parity [get], [set], [inherited]`

Note

The Modbus standard requires two stop bits if no parity is chosen. This library is not enforcing this but it is a recommended configuration.
A protocol must be closed in order to configure it.

SER_PARITY_NONE as no parity, SER_PARITY_ODD as odd parity, SER_PARITY_EVEN as even parity

timeout `Int32 timeout [get], [set], [inherited]`

Configures operation or socket time-out.

The time-out value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Timeout value in ms (Range: 1 - 100000)

pollDelay `Int32 pollDelay [get], [set], [inherited]`

This property sets the delay time which applies between two consecutive Modbus read/write. A value of 0 disables the poll delay.

The delay value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Delay value in ms (Range: 0 - 100000), 0 disables poll delay

retryCnt `Int32 retryCnt [get], [set], [inherited]`

Configures the automatic retry setting. A value of 0 disables any automatic retries.

Note

A protocol must be closed in order to configure it.

Retry count (Range: 0 - 10), 0 disables retries

5.3 MbusElamMasterProtocol Class Reference

Extended Lufkin Automation Modbus Master Protocol

Public Member Functions

- MbusElamMasterProtocol ()

  Creates new instance
- Int32 openProtocol ()

  Opens a serial Modbus protocol and the associated serial port with the port parameters configured via properties
- Int32 openProtocol (string portName, Int32 baudRate, Int32 dataBits, Int32 stopBits, Int32 parity)

  Opens a serial Modbus protocol and the associated serial port with the port parameters configured via properties
- Int32 enableRs485Mode (Int32 rtsDelay)

  Enables RS485 mode
- bool isOpen ()

  Returns whether the protocol is open or not.
- void closeProtocol ()

  Closes an open protocol including any associated communication resources (COM ports or sockets).

Static Public Member Functions

- static string getPackageVersion ()

  Returns the package version number.

Public Attributes

- const Int32 SER_DATABITS_7 = 7

  7 data bits
- const Int32 SER_DATABITS_8 = 8

  8 data bits
- const Int32 SER_STOPBITS_1 = 1

1 stop bit

- const Int32 SER_STOPBITS_2 = 2

  2 stop bits
- const Int32 SER_PARITY_NONE = 0

  No parity
- const Int32 SER_PARITY_ODD = 1

  Odd parity
- const Int32 SER_PARITY_EVEN = 2

  Even parity

Properties

- string portName `[get, set]`

  Serial port identifier property
- Int32 baudRate `[get, set]`

  Baud rate property in bps
- Int32 dataBits `[get, set]`

  Data bits property
- Int32 stopBits `[get, set]`

  Stop bits property
- Int32 parity `[get, set]`

  Parity property
- Int32 timeout `[get, set]`

  Time-out port property
- Int32 pollDelay `[get, set]`

  Poll delay property
- Int32 retryCnt `[get, set]`

  Retry count property

Bit Access

Table 0:00000 (Coils) and Table 1:0000 (Input Status)

- Int32 readCoils (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr)

  Modbus function 1 (01 hex), Read Coil Status/Read Coils.
- Int32 readCoils (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr, Int32 numCoils)

  Modbus function 1 (01 hex), Read Coil Status/Read Coils.
- Int32 readInputDiscretes (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr)

  Modbus function 2 (02 hex), Read Inputs Status/Read Input Discretes.
- Int32 readInputDiscretes (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr, Int32 numDiscretes)

  Modbus function 2 (02 hex), Read Inputs Status/Read Input Discretes.
- Int32 writeCoil (Int32 slaveAddr, Int32 bitAddr, bool bitVal)

Modbus function 5 (05 hex), Force Single Coil/Write Coil.

- Int32 forceMultipleCoils (Int32 slaveAddr, Int32 startRef, bool[ ] bitArr)

  Modbus function 15 (0F hex), Force Multiple Coils.

- Int32 forceMultipleCoils (Int32 slaveAddr, Int32 startRef, bool[ ] bitArr, Int32 numCoils)

  Modbus function 15 (0F hex), Force Multiple Coils.

Register Access (16-bit, 32-bit and floating point)

Table 4:00000 (Holding Registers) and Table 3:00000 (Input Registers)

- Int32 readMultipleRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr)

  Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers (16-bit, 32-bit and floating point).

- Int32 readMultipleRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr, Int32 numRegs)

  Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers (16-bit, 32-bit and floating point).

- Int32 readInputRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr)

  Modbus function 4 (04 hex), Read Input Registers (16-bit, 32-bit and floating point).

- Int32 readInputRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr, Int32 numRegs)

  Modbus function 4 (04 hex), Read Input Registers (16-bit, 32-bit and floating point).

- Int32 writeSingleRegister (Int32 slaveAddr, Int32 regAddr, Int16 regVal)

  Modbus function 6 (06 hex), Preset Single Register/Write Single Register.

- Int32 writeSingleRegister (Int32 slaveAddr, Int32 regAddr, UInt16 regVal)

  Modbus function 6 (06 hex), Preset Single Register/Write Single Register.

- Int32 writeMultipleRegisters (Int32 slaveAddr, Int32 startRef, System.Array regArr)

  Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers (16-bit, 32-bit and floating point).

- Int32 writeMultipleRegisters (Int32 slaveAddr, Int32 startRef, System.Array regArr, Int32 numRegs)

  Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers (16-bit, 32-bit and floating point).

- Int32 maskWriteRegister (Int32 slaveAddr, Int32 regAddr, Int16 andMask, Int16 or↩ Mask)

  Modbus function 22 (16 hex), Mask Write Register.

- Int32 readWriteRegisters (Int32 slaveAddr, Int32 readRef, [In, Out] Int16[ ] readArr, Int32 writeRef, Int16[ ] writeArr)

  Modbus function 23 (17 hex), Read/Write Registers.

- Int32 readWriteRegisters (Int32 slaveAddr, Int32 readRef, [In, Out] Int16[ ] readArr, Int32 readCnt, Int32 writeRef, Int16[ ] writeArr, Int32 writeCnt)

  Modbus function 23 (17 hex), Read/Write Registers.

Modulo-10000 long integer Access

Table 4:00000 (Holding Registers) and Table 3:00000 (Input Registers)

- Int32 readMultipleMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32↩ Arr)

  Modbus function 3 (03 hex) for 32-bit modulo-10000 long int data types, Read Holding Registers/Read Multiple Registers as modulo-10000 long int data.

- Int32 readMultipleMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32↩ Arr, Int32 numRegs)

  Modbus function 3 (03 hex) for 32-bit modulo-10000 long int data types, Read Holding Registers/Read Multiple Registers as modulo-10000 long int data.

- Int32 readInputMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32Arr)

  Modbus function 4 (04 hex) for 32-bit modulo-10000 long int data types, Read Input Registers as modulo-10000 long int data.

- Int32 readInputMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32Arr, Int32 numRegs)

  Modbus function 4 (04 hex) for 32-bit modulo-10000 long int data types, Read Input Registers as modulo-10000 long int data.

- Int32 writeMultipleMod10000 (Int32 slaveAddr, Int32 startRef, Int32[ ] int32Arr)

  Modbus function 16 (10 hex) for 32-bit modulo-10000 long int data types, Preset Multiple Registers/Write Multiple Registers with modulo-10000 long int data.

- Int32 writeMultipleMod10000 (Int32 slaveAddr, Int32 startRef, Int32[ ] int32Arr, Int32 numRegs)

  Modbus function 16 (10 hex) for 32-bit modulo-10000 long int data types, Preset Multiple Registers/Write Multiple Registers with modulo-10000 long int data.

Diagnostics

- Int32 readExceptionStatus (Int32 slaveAddr, out byte statusByte)

  Modbus function 7 (07 hex), Read Exception Status.

- Int32 returnQueryData (Int32 slaveAddr, byte[ ] queryArr, out byte[ ] echoArr)

  Modbus function code 8, sub-function 00, Return Query Data.

- Int32 restartCommunicationsOption (Int32 slaveAddr, Int32 clearEventLog)

  Modbus function code 8, sub-function 01, Restart Communications Option

User Defined Function Codes

- Int32 customFunction (Int32 slaveAddr, Int32 functionCode, byte[ ] requestData, [In, Out] byte[ ] responseData)

  User Defined Function Code

Protocol Configuration

- Int32 setTimeout (Int32 timeOut)

  Configures time-out
- Int32 getTimeout ()

  Returns the current time-out setting
- Int32 setPollDelay (Int32 pollDelay)

  Poll delay property
- Int32 getPollDelay ()

  Returns the poll delay time
- Int32 setRetryCnt (Int32 retryCnt)

  Configures the automatic retry setting
- Int32 getRetryCnt ()

  Returns the automatic retry count

Transmission Statistic Functions

- Int32 getTotalCounter ()

  Returns how often a message transfer has been executed
- void resetTotalCounter ()

  Resets total message transfer counter
- Int32 getSuccessCounter ()

  Returns how often a message transfer was successful
- void resetSuccessCounter ()

  Resets successful message transfer counter

Slave Configuration

- void configureStandard32BitMode ()

  Configures all slaves for Standard 32-bit Mode.
- Int32 configureStandard32BitMode (Int32 slaveAddr)

  Configures a slave for Standard 32-bit Mode.
- void configureEnron32BitMode ()

  Configures all slaves for Daniel/ENRON 32-bit Mode.
- Int32 configureEnron32BitMode (Int32 slaveAddr)

  Configures a slave for Daniel/ENRON 32-bit Mode.
- void configureCountFromOne ()

  Configures the reference counting scheme to start with one for all slaves.
- Int32 configureCountFromOne (Int32 slaveAddr)

  Configures the reference counting scheme to start with one for a slave.
- void configureCountFromZero ()

  Configures the reference counting scheme to start with zero for all slaves.
- void configureBigEndianInts ()

Configures 32-bit int data type functions to do a word swap

- Int32 configureBigEndianInts (Int32 slaveAddr)

  Configures 32-bit int data type functions to do a word swap on a per slave basis
- void configureSwappedFloats ()

  Configures float data type functions to do a word swap
- Int32 configureSwappedFloats (Int32 slaveAddr)

  Configures float data type functions to do a word swap on a per slave basis
- void configureLittleEndianInts ()

  Configures 32-bit int data type functions NOT to do a word swap
- Int32 configureLittleEndianInts (Int32 slaveAddr)

  Configures 32-bit int data type functions NOT to do a word swap on a per slave basis
- void configureIeeeFloats ()

  Configures float data type functions NOT to do a word swap
- Int32 configureIeeeFloats (Int32 slaveAddr)

  Configures float data type functions NOT to do a word swap on a per slave basis
- Int32 onfigureCountFromZero (Int32 slaveAddr)

  Configures the reference counting scheme to start with zero for a slave.

5.3.1 Detailed Description

This class realizes the Extended Lufkin Automation (ELAM) Modbus protocol. This proprietary Modbus extension allows adressing of up to 2295 slave units and the retrieval of up to 2500 registers for Modbus functions 3 and 4.

It's implementation is based on the specification "ELAM Extended Lufkin Automation Modbus Version 1.01" published by LUFKIN Automation. The ELAM multiple instruction requests extensions are not implemented.

Tests showed the following size limits with a LUFKIN SAM Well Manager device:

Coils: 1992 for read Registers: 2500 to read, 60 for write

It is possible to instantiate multiple instances of this class for establishing multiple connections on different serial ports (They should be executed in separate threads).

5.3.2 Constructor & Destructor Documentation

MbusElamMasterProtocol() `MbusElamMasterProtocol ( ) [inline]`

Exceptions

OutOfMemoryException	Creation of class failed

5.3.3 Member Function Documentation

openProtocol() [1/2] `Int32 openProtocol ( ) [inline], [inherited]`

This function opens the serial port. After a port has been opened, data and control functions can be used.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

openProtocol() [2/2]

```
Int32 openProtocol (
    string portName,
    Int32 baudRate,
    Int32 dataBits,
    Int32 stopBits,
    Int32 parity ) [inline], [inherited]
```

This function opens the serial port with a specific port settings. After a port has been opened, data and control functions can be used.

Note

The Modbus standard requires two stop bits if no parity is chosen. This library is not enforcing this but it is a recommended configuration.

Parameters

portName	Serial port identifier (eg "COM1")
baudRate	The port baud rate in bps (1200 - 115200, higher on some platforms)
dataBits	SER_DATABITS_7: 7 data bits (ASCII protocol only), SER_DATABITS_8: 8 data bits
stopBits	SER_STOPBITS_1: 1 stop bit, SER_STOPBITS_2: 2 stop bits
parity	SER_PARITY_NONE: no parity, SER_PARITY_ODD: odd parity, SER_PARITY_EVEN: even parity

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

enableRs485Mode() `Int32 enableRs485Mode (`
`Int32 rtsDelay ) [inline], [inherited]`

In RS485 mode the RTS signal can be used to enable and disable the transmitter of a RS232/RS485 converter. The RTS signal is asserted before sending data. It is cleared after the transmit buffer has been emptied and in addition the specified delay time has elapsed. The delay time is necessary because even the transmit buffer is already empty, the UART's FIFO will still contain unsent characters.

The use of RTS controlled RS232/RS485 converters should be avoided if possible. It is difficult to determine the exact time when to switch off the transmitter with non real-time operating systems like Windows and Linux. If it is switched off to early characters might still sit in the FIFO or the transmit register of the UART and these characters will be lost. Hence the slave will not recognize the message. On the other hand if it is switched off too late then the slave's message is corrupted and the master will not recognize the message.

Note

The delay value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.
A protocol must be closed in order to configure it.

Parameters

rtsDelay	Delay time in ms (Range as 0 - 100000) which applies after the transmit buffer is empty. 0 disables this mode.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readCoils() `[1/2] Int32 readCoils (`
`Int32 slaveAddr,`
`Int32 startRef,`
`[In,Out] bool [] bitArr ) [inline], [inherited]`

Reads the contents of the discrete outputs (coils, 0:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)

Parameters

startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read. The length of the array determines how many coils are read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readCoils() [2/2]

```
Int32 readCoils (
        Int32 slaveAddr,
        Int32 startRef,
        [In,Out] bool [] bitArr,
        Int32 numCoils )  [inline], [inherited]
```

Reads the contents of the discrete outputs (coils, 0:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read.
numCoils	Number of coils to be read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputDiscretes() [1/2]

```
Int32 readInputDiscretes (
        Int32 slaveAddr,
        Int32 startRef,
        [In,Out] bool [] bitArr )  [inline], [inherited]
```

Reads the contents of the discrete inputs (input status, 1:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read. The length of the array determines how many inputs are read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputDiscretes() `[2/2]`

```
Int32 readInputDiscretes (
    Int32 slaveAddr,
    Int32 startRef,
    [In,Out] bool [] bitArr,
    Int32 numDiscretes )  [inline], [inherited]
```

Reads the contents of the discrete inputs (input status, 1:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read.
num↩ Discretes	Number of inputs to be read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeCoil()

```
Int32 writeCoil (
    Int32 slaveAddr,
```

```
Int32 bitAddr,
bool bitVal )  [inline], [inherited]
```

Sets a single discrete output variable (coil, 0:00000 table) to either ON or OFF.

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
bitAddr	Coil address (Range: 1 - 65536)
bitVal	true sets, false clears discrete output variable

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

forceMultipleCoils() `[1/2]`

```
Int32 forceMultipleCoils (
Int32 slaveAddr,
Int32 startRef,
bool [] bitArr )  [inline], [inherited]
```

Writes binary values into a sequence of discrete outputs (coils, 0:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which contains the data to be sent. The length of the array determines how many coils are written (Range: 1-1968).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

forceMultipleCoils() `[2/2]`

```
Int32 forceMultipleCoils (
    Int32 slaveAddr,
    Int32 startRef,
    bool [] bitArr,
    Int32 numCoils )  [inline], [inherited]
```

Writes binary values into a sequence of discrete outputs (coils, 0:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which contains the data to be sent.
numCoils	Number of coils to be written (Range: 1-1968).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleRegisters() `[1/2]`

```
Int32 readMultipleRegisters (
    Int32 slaveAddr,
    Int32 startRef,
    [In,Out] System.Array regArr )  [inline], [inherited]
```

Reads the contents of the output registers (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[]. The length of the array determines how many registers are read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleRegisters() `[2/2] Int32 readMultipleRegisters (`
`Int32 slaveAddr,`
`Int32 startRef,`
`[In,Out] System.Array regArr,`
`Int32 numRegs ) [inline], [inherited]`

Reads the contents of the output registers (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[].
numRegs	Number of values to be read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputRegisters() `[1/2] Int32 readInputRegisters (`
`Int32 slaveAddr,`
`Int32 startRef,`
`[In, Out] System.Array regArr ) [inline], [inherited]`

Read the contents of the input registers (3:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)

Parameters

startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[]. The length of the array determines how many registers are read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputRegisters() `[2/2]`

```
Int32 readInputRegisters (
    Int32 slaveAddr,
    Int32 startRef,
    [In, Out] System.Array regArr,
    Int32 numRegs )  [inline], [inherited]
```

Read the contents of the input registers (3:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[].
numRegs	Number of values to be read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeSingleRegister() `[1/2]`

```
Int32 writeSingleRegister (
    Int32 slaveAddr,
    Int32 regAddr,
    Int16 regVal )  [inline], [inherited]
```

Writes a value into a single output register (holding register, 4:00000 reference).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
regAddr	Register address (Range: 1 - 65536)
regVal	Data to be sent as signed 16-bit value

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeSingleRegister() `[2/2] Int32 writeSingleRegister (`
`Int32 slaveAddr,`
`Int32 regAddr,`
`UInt16 regVal ) [inline], [inherited]`

Writes a value into a single output register (holding register, 4:00000 reference).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
regAddr	Register address (Range: 1 - 65536)
regVal	Data to be sent as unsigned 16-bit value

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleRegisters() `[1/2] Int32 writeMultipleRegisters (`
`Int32 slaveAddr,`
`Int32 startRef,`
`System.Array regArr ) [inline], [inherited]`

Writes values into a sequence of output registers (holding registers, 4:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array with data to be sent. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[]. The length of the array determines how many registers are written (Range: 1-123 for 16-bit, 1-61 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleRegisters() `[2/2] Int32 writeMultipleRegisters ( Int32 slaveAddr, Int32 startRef, System.Array regArr, Int32 numRegs ) [inline], [inherited]`

Writes values into a sequence of output registers (holding registers, 4:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array with data to be sent. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[].
numRegs	Number of values to be written (Range: 1-123 for 16-bit, 1-61 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

maskWriteRegister()

```
Int32 maskWriteRegister (
    Int32 slaveAddr,
    Int32 regAddr,
    Int16 andMask,
    Int16 orMask )  [inline], [inherited]
```

Masks bits according to an AND and an OR mask into a single output register (holding register, 4:00000 reference). Masking is done as follows: retVal = (currentVal AND andMask) OR (orMask AND (NOT andMask))

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
regAddr	Register address (Range: 1 - 65536)
andMask	Mask to be applied as a logic AND to the register
orMask	Mask to be applied as a logic OR to the register

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readWriteRegisters()

```
[1/2]  Int32 readWriteRegisters (
    Int32 slaveAddr,
    Int32 readRef,
    [In,Out] Int16 [] readArr,
    Int32 writeRef,
    Int16 [] writeArr )  [inline], [inherited]
```

Combines reading and writing of the output registers in one transaction (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
readRef	Start registers for reading (Range: 1 - 65536)

Parameters

readArr	Buffer which will contain the data read. The length of the array determines how many registers are read (Range: 1-125).
writeRef	Start registers for writing (Range: 1 - 65536)
writeArr	Buffer with data to be sent. The length of the array determines how many registers are written (Range: 1-121).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readWriteRegisters() `[2/2]`

```
Int32 readWriteRegisters (
    Int32 slaveAddr,
    Int32 readRef,
    [In,Out] Int16 [] readArr,
    Int32 readCnt,
    Int32 writeRef,
    Int16 [] writeArr,
    Int32 writeCnt )  [inline], [inherited]
```

Combines reading and writing of the output registers in one transaction (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
readRef	Start register for reading (Range: 1 - 65536)
readArr	Buffer which will contain the data read.
writeRef	Start register for writing (Range: 1 - 65536)
writeArr	Buffer with data to be sent.
readCnt	Number of registers to be read (Range: 1-125).
writeCnt	Number of registers to be written (Range: 1-121).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleMod10000() `[1/2] Int32 readMultipleMod10000 (`

```
Int32 slaveAddr,
Int32 startRef,
[In,Out] Int32 [] int32Arr ) [inline], [inherited]
```

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value into 32-bit int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleMod10000() `[2/2] Int32 readMultipleMod10000 (`

```
Int32 slaveAddr,
Int32 startRef,
[In,Out] Int32 [] int32Arr,
Int32 numRegs ) [inline], [inherited]
```

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value into 32-bit int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read.
numRegs	Number of values to be read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputMod10000() `[1/2]` `Int32 readInputMod10000 (`
`Int32 slaveAddr,`
`Int32 startRef,`
`[In,Out] Int32 [] int32Arr )` `[inline], [inherited]`

Reads the contents of pairs of consecutive input registers (3:00000 table) representing a modulo-10000 long int value into 32-bit long int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputMod10000() `[2/2]` `Int32 readInputMod10000 (`
`Int32 slaveAddr,`

```
Int32 startRef,
[In,Out] Int32 [] int32Arr,
Int32 numRegs )  [inline], [inherited]
```

Reads the contents of pairs of consecutive input registers (3:00000 table) representing a modulo-10000 long int value into 32-bit long int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read.
numRegs	Number of values to be read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleMod10000() `[1/2]  Int32 writeMultipleMod10000 (`

```
Int32 slaveAddr,
Int32 startRef,
Int32 [] int32Arr )  [inline], [inherited]
```

Writes long int values into pairs of output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)

Parameters

startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer with the data to be sent. The length of the array determines how many values are written (Range: 1-61).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleMod10000() `[2/2]` `Int32 writeMultipleMod10000 (`
`Int32 slaveAddr,`
`Int32 startRef,`
`Int32 [] int32Arr,`
`Int32 numRegs )` `[inline], [inherited]`

Writes long int values into pairs of output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function. Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer with the data to be sent.
numRegs	Number of values to be sent (Range: 1-61).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readExceptionStatus() `Int32 readExceptionStatus (`
`Int32 slaveAddr,`
`out byte statusByte )` `[inline], [inherited]`

Reads the eight exception status coils within the slave device.

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
statusByte	Slave status byte. The meaning of this status byte is slave specific and varies from device to device.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

returnQueryData() `Int32 returnQueryData (`
`Int32 slaveAddr,`
`byte [] queryArr,`
`out byte [] echoArr ) [inline], [inherited]`

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255 for serial, 0 - 255 for TCP)
queryArr	Buffer with data to be sent. The length of the array determines how many bytes are sent and returned
echoArr	Buffer which will contain the data read

Returns

BusProtocolErrors.FTALK_SUCCESS on success, FTALK_INVALID_REPLY_ERROR if reply does not match query data or error code. See BusProtocolErrors for more error codes.

References BusProtocolErrors.FTALK_INVALID_REPLY_ERROR, and BusProtocolErrors.FT↩ALK_SUCCESS.

restartCommunicationsOption() `Int32 restartCommunicationsOption (`
`Int32 slaveAddr,`
`Int32 clearEventLog ) [inline], [inherited]`

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255 for serial, 0 - 255 for TCP)
clearEventLog	Flag when set to one clears in addition the slave's communication even log.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

customFunction() `Int32 customFunction (`
`Int32 slaveAddr,`
`Int32 functionCode,`
`byte [] requestData,`
`[In,Out] byte [] responseData ) [inline], [inherited]`

This method can be used to implement User Defined Function Codes. The caller has only to pass the user data to this function. The assembly of the Modbus frame (the so called ADU) including checksums, slave address and function code and subsequentially the transmission, is taken care of by this method.

The modbus specification reserves function codes 65-72 and 100-110 for user defined functions.

Note

Modbus functions usually have an implied response length and therefore the number of bytes expected to be received is known at the time when sending the request. In case of a custom Modbus function with an open or unknown response length, this function can not be used.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255 for serial, 0 - 255 for TCP)
functionCode	Custom function code to be used for Modbus transaction (Range: 1-127)

Parameters

requestData	Array with data to be sent as request (not including slave address or function code). The length of the array determines how many request bytes are sent (Range: 0-252).
responseData	Buffer which will be filled with the response data received. The length of the array determines how many bytes are read (Range: 0-252).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

setTimeout() `Int32 setTimeout ( Int32 timeOut ) [inline], [inherited]`

Configures operation or socket time-out.

The time-out value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Parameters

timeOut	Timeout value in ms (Range: 1 - 100000)

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

getTimeout() `Int32 getTimeout ( ) [inline], [inherited]`

Returns

Timeout value in ms

setPollDelay() `Int32 setPollDelay ( Int32 pollDelay ) [inline], [inherited]`

This function sets the delay time which applies between two consecutive Modbus read/write. A value of 0 disables the poll delay.

The delay value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Parameters

pollDelay	Delay value in ms (Range: 0 - 100000), 0 disables poll delay

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

getPollDelay() `Int32 getPollDelay ( ) [inline], [inherited]`

Returns

Delay time in ms, 0 if poll delay is switched off

setRetryCnt() `Int32 setRetryCnt ( Int32 retryCnt ) [inline], [inherited]`

Configures the automatic retry setting. A value of 0 disables any automatic retries.

Note

A protocol must be closed in order to configure it.

Parameters

retryCnt	Retry count (Range: 0 - 10), 0 disables retries

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

getRetryCnt() `Int32 getRetryCnt ( ) [inline], [inherited]`

Returns

Retry count

getTotalCounter() `Int32 getTotalCounter ( ) [inline], [inherited]`

Returns

Counter value

resetTotalCounter() `void resetTotalCounter ( ) [inline], [inherited]`

getSuccessCounter() `Int32 getSuccessCounter ( ) [inline], [inherited]`

Returns

Counter value

resetSuccessCounter() `void resetSuccessCounter ( ) [inline], [inherited]`

configureStandard32BitMode() `[1/2] void configureStandard32BitMode ( ) [inline], [inherited]`

In Standard 32-bit Register Mode a 32-bit value is transmitted as two consecutive 16-bit Modbus registers.

Note

This function call also re-configures the endianess to default little-endian for 32-bit values!

Note

This is the default mode.

configureStandard32BitMode() [2/2] `Int32 configureStandard32BitMode ( Int32 slaveAddr ) [inline], [inherited]`

In Standard 32-bit Register Mode a 32-bit value is transmitted as two consecutive 16-bit Modbus registers.

Note

This function call also re-configures the endianess to default little-endian for 32-bit values!

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureEnron32BitMode() [1/2] `void configureEnron32BitMode ( ) [inline], [inherited]`

Some Modbus flavours like the Daniel/Enron protocol represent a 32-bit value using one 32-bit Modbus register instead of two 16-bit registers.

Note

This function call also re-configures the endianess to big-endian for 32-bit values as defined by the Daniel/ENRON protocol!

configureEnron32BitMode() [2/2] `Int32 configureEnron32BitMode ( Int32 slaveAddr ) [inline], [inherited]`

Some Modbus flavours like the Daniel/Enron protocol represent a 32-bit value using one 32-bit Modbus register instead of two 16-bit registers.

Note

This function call also re-configures the endianess to big-endian for 32-bit values as defined by the Daniel/ENRON protocol!

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureCountFromOne() `[1/2] void configureCountFromOne ( ) [inline], [inherited]`

This renders the reference range to be 1 to 65536 (0x10000) and register #0 is an illegal register.

Note

This is the default mode.

configureCountFromOne() `[2/2] Int32 configureCountFromOne (`
`Int32 slaveAddr ) [inline], [inherited]`

This renders the reference range to be 1 to 65536 (0x10000) and register #0 is an illegal register.

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureCountFromZero() `void configureCountFromZero ( ) [inline], [inherited]`

This renders the valid reference range to be 0 to 65535 (0xFFFF).

This renders the first register to be #0 for all slaves.

configureBigEndianInts() `[1/2] void configureBigEndianInts ( ) [inline], [inherited]`

Modbus is using little-endian word order for 32-bit values. The data transfer functions operating upon 32-bit int data types can be configured to do a word swap which enables them to read 32-bit data correctly from a big-endian slave.

configureBigEndianInts() `[2/2] Int32 configureBigEndianInts ( Int32 slaveAddr ) [inline], [inherited]`

Modbus is using little-endian word order for 32-bit values. The data transfer functions operating upon 32-bit int data types can be configured to do a word swap which enables them to read 32-bit data correctly from a big-endian slave.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureSwappedFloats() `[1/2] void configureSwappedFloats ( ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Most platforms store floats in IEEE 754 little-endian order which does not need a word swap.

configureSwappedFloats() `[2/2] Int32 configureSwappedFloats ( Int32 slaveAddr ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Most platforms store floats in IEEE 754 little-endian order which does not need a word swap.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureLittleEndianInts() `[1/2]` `void configureLittleEndianInts ( ) [inline], [inherited]`

Note

This is the default mode.

configureLittleEndianInts() `[2/2]` `Int32 configureLittleEndianInts ( Int32 slaveAddr ) [inline], [inherited]`

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureIeeeFloats() `[1/2]` `void configureIeeeFloats ( ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Note

This is the default mode.

configureIeeeFloats() `[2/2]` `Int32 configureIeeeFloats ( Int32 slaveAddr ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

onfigureCountFromZero() `Int32 onfigureCountFromZero ( Int32 slaveAddr ) [inline], [inherited]`

This renders the valid reference range to be 0 to 65535 (0xFFFF).

This renders the first register to be #0 for all slaves.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

isOpen() `bool isOpen ( ) [inline], [inherited]`

Returns

True = open, False = closed

closeProtocol() `void closeProtocol ( ) [inline], [inherited]`

getPackageVersion() `static string getPackageVersion ( ) [inline], [static], [inherited]`

Returns

Package version string

5.3.4 Member Data Documentation

SER_DATABITS_7 `const Int32 SER_DATABITS_7 = 7 [inherited]`

SER_DATABITS_8 `const Int32 SER_DATABITS_8 = 8 [inherited]`

SER_STOPBITS_1 `const Int32 SER_STOPBITS_1 = 1 [inherited]`

SER_STOPBITS_2 `const Int32 SER_STOPBITS_2 = 2 [inherited]`

SER_PARITY_NONE `const Int32 SER_PARITY_NONE = 0 [inherited]`

SER_PARITY_ODD `const Int32 SER_PARITY_ODD = 1 [inherited]`

SER_PARITY_EVEN `const Int32 SER_PARITY_EVEN = 2 [inherited]`

5.3.5 Property Documentation

portName `string portName [get], [set], [inherited]`

Note

A protocol must be closed in order to configure it.

Serial port identifier (eg "COM1")

baudRate `Int32 baudRate [get], [set], [inherited]`

Note

A protocol must be closed in order to configure it.

Typically 1200 - 115200, maximum value depends on UART hardware

dataBits `Int32 dataBits [get], [set], [inherited]`

Note

A protocol must be closed in order to configure it.

SER_DATABITS_7 as 7 data bits (ASCII protocol only), SER_DATABITS_8 as data bits

stopBits `Int32 stopBits [get], [set], [inherited]`

Note

The Modbus standard requires two stop bits if no parity is chosen. This library is not enforcing this but it is a recommended configuration.
A protocol must be closed in order to configure it.

SER_STOPBITS_1 as 1 stop bit, SER_STOPBITS_2 as 2 stop bits

parity `Int32 parity [get], [set], [inherited]`

Note

The Modbus standard requires two stop bits if no parity is chosen. This library is not enforcing this but it is a recommended configuration.
A protocol must be closed in order to configure it.

SER_PARITY_NONE as no parity, SER_PARITY_ODD as odd parity, SER_PARITY_EVEN as even parity

timeout `Int32 timeout [get], [set], [inherited]`

Configures operation or socket time-out.

The time-out value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Timeout value in ms (Range: 1 - 100000)

pollDelay `Int32 pollDelay [get], [set], [inherited]`

This property sets the delay time which applies between two consecutive Modbus read/write. A value of 0 disables the poll delay.

The delay value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Delay value in ms (Range: 0 - 100000), 0 disables poll delay

retryCnt `Int32 retryCnt [get], [set], [inherited]`

Configures the automatic retry setting. A value of 0 disables any automatic retries.

Note

A protocol must be closed in order to configure it.

Retry count (Range: 0 - 10), 0 disables retries

5.4 MbusTcpMasterProtocol Class Reference

MODBUS/TCP Master Protocol class

Public Member Functions

- MbusTcpMasterProtocol ()

  Creates new instance
- override Int32 setPort (Int16 portNo)

  Sets the TCP port number of the Modbus slave device.
- Int32 adamSendReceiveAsciiCmd (string command, out string response)

  Send/Receive ADAM 5000/6000 ASCII command.
- Int32 openProtocol ()

  Connects to a TCP slave.
- Int32 openProtocol (string hostName)

  Connects to a TCP slave.
- Int16 getPort ()

  Returns the TCP port number used by the protocol.
- bool isOpen ()

  Returns whether the protocol is open or not.
- void closeProtocol ()

  Closes an open protocol including any associated communication resources (COM ports or sockets).

Static Public Member Functions

- static string getPackageVersion ()
  Returns the package version number.

Properties

- override Int16 port `[get, set]`
  TCP port property
- string hostName `[get, set]`
  Host name
- Int32 timeout `[get, set]`
  Time-out port property
- Int32 pollDelay `[get, set]`
  Poll delay property
- Int32 retryCnt `[get, set]`
  Retry count property

Bit Access

Table 0:00000 (Coils) and Table 1:0000 (Input Status)

- Int32 readCoils (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr)
  Modbus function 1 (01 hex), Read Coil Status/Read Coils.
- Int32 readCoils (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr, Int32 numCoils)
  Modbus function 1 (01 hex), Read Coil Status/Read Coils.
- Int32 readInputDiscretes (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr)
  Modbus function 2 (02 hex), Read Inputs Status/Read Input Discretes.
- Int32 readInputDiscretes (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr, Int32 numDiscretes)
  Modbus function 2 (02 hex), Read Inputs Status/Read Input Discretes.
- Int32 writeCoil (Int32 slaveAddr, Int32 bitAddr, bool bitVal)
  Modbus function 5 (05 hex), Force Single Coil/Write Coil.
- Int32 forceMultipleCoils (Int32 slaveAddr, Int32 startRef, bool[ ] bitArr)
  Modbus function 15 (0F hex), Force Multiple Coils.
- Int32 forceMultipleCoils (Int32 slaveAddr, Int32 startRef, bool[ ] bitArr, Int32 numCoils)
  Modbus function 15 (0F hex), Force Multiple Coils.

Register Access (16-bit, 32-bit and floating point)

Table 4:00000 (Holding Registers) and Table 3:00000 (Input Registers)

- Int32 readMultipleRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr)

Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers (16-bit, 32-bit and floating point).

- Int32 readMultipleRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr, Int32 numRegs)

  Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers (16-bit, 32-bit and floating point).
- Int32 readInputRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr)

  Modbus function 4 (04 hex), Read Input Registers (16-bit, 32-bit and floating point).
- Int32 readInputRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr, Int32 numRegs)

  Modbus function 4 (04 hex), Read Input Registers (16-bit, 32-bit and floating point).
- Int32 writeSingleRegister (Int32 slaveAddr, Int32 regAddr, Int16 regVal)

  Modbus function 6 (06 hex), Preset Single Register/Write Single Register.
- Int32 writeSingleRegister (Int32 slaveAddr, Int32 regAddr, UInt16 regVal)

  Modbus function 6 (06 hex), Preset Single Register/Write Single Register.
- Int32 writeMultipleRegisters (Int32 slaveAddr, Int32 startRef, System.Array regArr)

  Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers (16-bit, 32-bit and floating point).
- Int32 writeMultipleRegisters (Int32 slaveAddr, Int32 startRef, System.Array regArr, Int32 numRegs)

  Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers (16-bit, 32-bit and floating point).
- Int32 maskWriteRegister (Int32 slaveAddr, Int32 regAddr, Int16 andMask, Int16 or↩ Mask)

  Modbus function 22 (16 hex), Mask Write Register.
- Int32 readWriteRegisters (Int32 slaveAddr, Int32 readRef, [In, Out] Int16[ ] readArr, Int32 writeRef, Int16[ ] writeArr)

  Modbus function 23 (17 hex), Read/Write Registers.
- Int32 readWriteRegisters (Int32 slaveAddr, Int32 readRef, [In, Out] Int16[ ] readArr, Int32 readCnt, Int32 writeRef, Int16[ ] writeArr, Int32 writeCnt)

  Modbus function 23 (17 hex), Read/Write Registers.

Modulo-10000 long integer Access

Table 4:00000 (Holding Registers) and Table 3:00000 (Input Registers)

- Int32 readMultipleMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32↩ Arr)

  Modbus function 3 (03 hex) for 32-bit modulo-10000 long int data types, Read Holding Registers/Read Multiple Registers as modulo-10000 long int data.
- Int32 readMultipleMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32↩ Arr, Int32 numRegs)

  Modbus function 3 (03 hex) for 32-bit modulo-10000 long int data types, Read Holding Registers/Read Multiple Registers as modulo-10000 long int data.
- Int32 readInputMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32Arr)

  Modbus function 4 (04 hex) for 32-bit modulo-10000 long int data types, Read Input Registers as modulo-10000 long int data.

- Int32 readInputMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32Arr, Int32 numRegs)

  Modbus function 4 (04 hex) for 32-bit modulo-10000 long int data types, Read Input Registers as modulo-10000 long int data.
- Int32 writeMultipleMod10000 (Int32 slaveAddr, Int32 startRef, Int32[ ] int32Arr)

  Modbus function 16 (10 hex) for 32-bit modulo-10000 long int data types, Preset Multiple Registers/Write Multiple Registers with modulo-10000 long int data.
- Int32 writeMultipleMod10000 (Int32 slaveAddr, Int32 startRef, Int32[ ] int32Arr, Int32 numRegs)

  Modbus function 16 (10 hex) for 32-bit modulo-10000 long int data types, Preset Multiple Registers/Write Multiple Registers with modulo-10000 long int data.

Diagnostics

- Int32 readExceptionStatus (Int32 slaveAddr, out byte statusByte)

  Modbus function 7 (07 hex), Read Exception Status.
- Int32 returnQueryData (Int32 slaveAddr, byte[ ] queryArr, out byte[ ] echoArr)

  Modbus function code 8, sub-function 00, Return Query Data.
- Int32 restartCommunicationsOption (Int32 slaveAddr, Int32 clearEventLog)

  Modbus function code 8, sub-function 01, Restart Communications Option

User Defined Function Codes

- Int32 customFunction (Int32 slaveAddr, Int32 functionCode, byte[ ] requestData, [In, Out] byte[ ] responseData)

  User Defined Function Code

Protocol Configuration

- Int32 setTimeout (Int32 timeOut)

  Configures time-out
- Int32 getTimeout ()

  Returns the current time-out setting
- Int32 setPollDelay (Int32 pollDelay)

  Poll delay property
- Int32 getPollDelay ()

  Returns the poll delay time
- Int32 setRetryCnt (Int32 retryCnt)

  Configures the automatic retry setting
- Int32 getRetryCnt ()

  Returns the automatic retry count

Transmission Statistic Functions

- Int32 getTotalCounter ()
  Returns how often a message transfer has been executed
- void resetTotalCounter ()
  Resets total message transfer counter
- Int32 getSuccessCounter ()
  Returns how often a message transfer was successful
- void resetSuccessCounter ()
  Resets successful message transfer counter

Slave Configuration

- void configureStandard32BitMode ()
  Configures all slaves for Standard 32-bit Mode.
- Int32 configureStandard32BitMode (Int32 slaveAddr)
  Configures a slave for Standard 32-bit Mode.
- void configureEnron32BitMode ()
  Configures all slaves for Daniel/ENRON 32-bit Mode.
- Int32 configureEnron32BitMode (Int32 slaveAddr)
  Configures a slave for Daniel/ENRON 32-bit Mode.
- void configureCountFromOne ()
  Configures the reference counting scheme to start with one for all slaves.
- Int32 configureCountFromOne (Int32 slaveAddr)
  Configures the reference counting scheme to start with one for a slave.
- void configureCountFromZero ()
  Configures the reference counting scheme to start with zero for all slaves.
- void configureBigEndianInts ()
  Configures 32-bit int data type functions to do a word swap
- Int32 configureBigEndianInts (Int32 slaveAddr)
  Configures 32-bit int data type functions to do a word swap on a per slave basis
- void configureSwappedFloats ()
  Configures float data type functions to do a word swap
- Int32 configureSwappedFloats (Int32 slaveAddr)
  Configures float data type functions to do a word swap on a per slave basis
- void configureLittleEndianInts ()
  Configures 32-bit int data type functions NOT to do a word swap
- Int32 configureLittleEndianInts (Int32 slaveAddr)
  Configures 32-bit int data type functions NOT to do a word swap on a per slave basis
- void configureIeeeFloats ()
  Configures float data type functions NOT to do a word swap
- Int32 configureIeeeFloats (Int32 slaveAddr)
  Configures float data type functions NOT to do a word swap on a per slave basis
- Int32 onfigureCountFromZero (Int32 slaveAddr)
  Configures the reference counting scheme to start with zero for a slave.

5.4.1 Detailed Description

This class realises the MODBUS/TCP master protocol. It provides functions to establish and to close a TCP/IP connection to the slave as well as data and control functions which can be used after a connection to a slave device has been established successfully. The data and control functions are organized into different conformance classes.

It is possible to instantiate multiple instances of this class for establishing multiple connections to either the same or different hosts.

5.4.2 Constructor & Destructor Documentation

MbusTcpMasterProtocol() `MbusTcpMasterProtocol ( ) [inline]`

Exceptions

OutOfMemoryException	Creation of class failed

5.4.3 Member Function Documentation

setPort() `override Int32 setPort ( Int16 portNo ) [inline], [virtual]`

Usually the port number remains unchanged and defaults to 502 for Modbus/TCP and 1100 for RTU over TCP. However if the port number has to be different this function must be called before opening the connection with openProtocol().

Note

A protocol must be closed in order to configure it.

Parameters

portNo	Port number to be used when opening the connection

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

Reimplemented from MbusIpClientBase.

openProtocol() `[1/2]` `Int32 openProtocol ( ) [inline], [inherited]`

This function establishes a logical network connection between master and slave. After a connection has been established data and control functions can be used. A TCP/IP connection should be closed if it is no longer needed.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

openProtocol() `[2/2]` `Int32 openProtocol ( string hostName ) [inline], [inherited]`

This function establishes a logical network connection between master and slave. After a connection has been established data and control functions can be used. A TCP/IP connection should be closed if it is no longer needed.

Parameters

hostName	String with IP address or host name (eg "127.0.0.1")

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

getPort() `Int16 getPort ( ) [inline], [inherited]`

Returns

Currently set port number

readCoils() `[1/2]`

```
Int32 readCoils (
        Int32 slaveAddr,
        Int32 startRef,
        [In,Out] bool [] bitArr ) [inline], [inherited]
```

Reads the contents of the discrete outputs (coils, 0:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read. The length of the array determines how many coils are read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readCoils() [2/2]

```
Int32 readCoils (
        Int32 slaveAddr,
        Int32 startRef,
        [In,Out] bool [] bitArr,
        Int32 numCoils )  [inline], [inherited]
```

Reads the contents of the discrete outputs (coils, 0:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read.
numCoils	Number of coils to be read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputDiscretes() [1/2]

```
Int32 readInputDiscretes (
        Int32 slaveAddr,
```

```
Int32 startRef,
[In,Out] bool [] bitArr )  [inline], [inherited]
```

Reads the contents of the discrete inputs (input status, 1:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read. The length of the array determines how many inputs are read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputDiscretes() `[2/2]`

```
Int32 readInputDiscretes (
Int32 slaveAddr,
Int32 startRef,
[In,Out] bool [] bitArr,
Int32 numDiscretes )  [inline], [inherited]
```

Reads the contents of the discrete inputs (input status, 1:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read.
num↩ Discretes	Number of inputs to be read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeCoil() `Int32 writeCoil ( Int32 slaveAddr, Int32 bitAddr, bool bitVal ) [inline], [inherited]`

Sets a single discrete output variable (coil, 0:00000 table) to either ON or OFF.

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
bitAddr	Coil address (Range: 1 - 65536)
bitVal	true sets, false clears discrete output variable

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

forceMultipleCoils() `[1/2] Int32 forceMultipleCoils ( Int32 slaveAddr, Int32 startRef, bool [] bitArr ) [inline], [inherited]`

Writes binary values into a sequence of discrete outputs (coils, 0:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which contains the data to be sent. The length of the array determines how many coils are written (Range: 1-1968).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

forceMultipleCoils() `[2/2]`

```
Int32 forceMultipleCoils (
    Int32 slaveAddr,
    Int32 startRef,
    bool [] bitArr,
    Int32 numCoils )  [inline], [inherited]
```

Writes binary values into a sequence of discrete outputs (coils, 0:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which contains the data to be sent.
numCoils	Number of coils to be written (Range: 1-1968).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleRegisters() `[1/2]`

```
Int32 readMultipleRegisters (
    Int32 slaveAddr,
    Int32 startRef,
    [In,Out] System.Array regArr )  [inline], [inherited]
```

Reads the contents of the output registers (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)

Parameters

regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[]. The length of the array determines how many registers are read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleRegisters() `[2/2] Int32 readMultipleRegisters ( Int32 slaveAddr, Int32 startRef, [In,Out] System.Array regArr, Int32 numRegs ) [inline], [inherited]`

Reads the contents of the output registers (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[].
numRegs	Number of values to be read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputRegisters() `[1/2] Int32 readInputRegisters ( Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr ) [inline], [inherited]`

Read the contents of the input registers (3:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[]. The length of the array determines how many registers are read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputRegisters() `[2/2]`

```
Int32 readInputRegisters (
    Int32 slaveAddr,
    Int32 startRef,
    [In, Out] System.Array regArr,
    Int32 numRegs )  [inline], [inherited]
```

Read the contents of the input registers (3:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[].
numRegs	Number of values to be read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeSingleRegister() `[1/2]` `Int32 writeSingleRegister (`
`Int32 slaveAddr,`
`Int32 regAddr,`
`Int16 regVal )` `[inline], [inherited]`

Writes a value into a single output register (holding register, 4:00000 reference).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
regAddr	Register address (Range: 1 - 65536)
regVal	Data to be sent as signed 16-bit value

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeSingleRegister() `[2/2]` `Int32 writeSingleRegister (`
`Int32 slaveAddr,`
`Int32 regAddr,`
`UInt16 regVal )` `[inline], [inherited]`

Writes a value into a single output register (holding register, 4:00000 reference).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
regAddr	Register address (Range: 1 - 65536)
regVal	Data to be sent as unsigned 16-bit value

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleRegisters() `[1/2] Int32 writeMultipleRegisters (`
`Int32 slaveAddr,`
`Int32 startRef,`
`System.Array regArr ) [inline], [inherited]`

Writes values into a sequence of output registers (holding registers, 4:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array with data to be sent. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[]. The length of the array determines how many registers are written (Range: 1-123 for 16-bit, 1-61 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleRegisters() `[2/2] Int32 writeMultipleRegisters (`
`Int32 slaveAddr,`
`Int32 startRef,`
`System.Array regArr,`
`Int32 numRegs ) [inline], [inherited]`

Writes values into a sequence of output registers (holding registers, 4:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array with data to be sent. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[].
numRegs	Number of values to be written (Range: 1-123 for 16-bit, 1-61 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

maskWriteRegister() `Int32 maskWriteRegister ( Int32 slaveAddr, Int32 regAddr, Int16 andMask, Int16 orMask ) [inline], [inherited]`

Masks bits according to an AND and an OR mask into a single output register (holding register, 4:00000 reference). Masking is done as follows: retVal = (currentVal AND andMask) OR (orMask AND (NOT andMask))

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
regAddr	Register address (Range: 1 - 65536)
andMask	Mask to be applied as a logic AND to the register
orMask	Mask to be applied as a logic OR to the register

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readWriteRegisters() `[1/2] Int32 readWriteRegisters ( Int32 slaveAddr, Int32 readRef, [In,Out] Int16 [] readArr, Int32 writeRef, Int16 [] writeArr ) [inline], [inherited]`

Combines reading and writing of the output registers in one transaction (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
readRef	Start registers for reading (Range: 1 - 65536)
readArr	Buffer which will contain the data read. The length of the array determines how many registers are read (Range: 1-125).
writeRef	Start registers for writing (Range: 1 - 65536)
writeArr	Buffer with data to be sent. The length of the array determines how many registers are written (Range: 1-121).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readWriteRegisters() `[2/2]`

```
Int32 readWriteRegisters (
    Int32 slaveAddr,
    Int32 readRef,
    [In,Out] Int16 [] readArr,
    Int32 readCnt,
    Int32 writeRef,
    Int16 [] writeArr,
    Int32 writeCnt )  [inline], [inherited]
```

Combines reading and writing of the output registers in one transaction (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
readRef	Start register for reading (Range: 1 - 65536)
readArr	Buffer which will contain the data read.
writeRef	Start register for writing (Range: 1 - 65536)
writeArr	Buffer with data to be sent.
readCnt	Number of registers to be read (Range: 1-125).
writeCnt	Number of registers to be written (Range: 1-121).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleMod10000() [1/2] Int32 readMultipleMod10000 (
Int32 *slaveAddr*,
Int32 *startRef*,
[In,Out] Int32 [] *int32Arr*) [inline], [inherited]

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value into 32-bit int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleMod10000() [2/2] Int32 readMultipleMod10000 (
Int32 *slaveAddr*,
Int32 *startRef*,
[In,Out] Int32 [] *int32Arr*,
Int32 *numRegs*) [inline], [inherited]

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value into 32-bit int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read.
numRegs	Number of values to be read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputMod10000() `[1/2] Int32 readInputMod10000 (`
`Int32 slaveAddr,`
`Int32 startRef,`
`[In,Out] Int32 [] int32Arr ) [inline], [inherited]`

Reads the contents of pairs of consecutive input registers (3:00000 table) representing a modulo-10000 long int value into 32-bit long int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputMod10000() [2/2] Int32 readInputMod10000 (
Int32 *slaveAddr*,
Int32 *startRef*,
[In,Out] Int32 [] *int32Arr*,
Int32 *numRegs*) [inline], [inherited]

Reads the contents of pairs of consecutive input registers (3:00000 table) representing a modulo-10000 long int value into 32-bit long int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read.
numRegs	Number of values to be read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleMod10000() [1/2] Int32 writeMultipleMod10000 (
Int32 *slaveAddr*,
Int32 *startRef*,
Int32 [] *int32Arr*) [inline], [inherited]

Writes long int values into pairs of output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer with the data to be sent. The length of the array determines how many values are written (Range: 1-61).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleMod10000() `[2/2] Int32 writeMultipleMod10000 (`

```
Int32 slaveAddr,
Int32 startRef,
Int32 [] int32Arr,
Int32 numRegs ) [inline], [inherited]
```

Writes long int values into pairs of output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer with the data to be sent.
numRegs	Number of values to be sent (Range: 1-61).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readExceptionStatus() `Int32 readExceptionStatus ( Int32 slaveAddr, out byte statusByte ) [inline], [inherited]`

Reads the eight exception status coils within the slave device.

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
statusByte	Slave status byte. The meaning of this status byte is slave specific and varies from device to device.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

returnQueryData() `Int32 returnQueryData ( Int32 slaveAddr, byte [] queryArr, out byte [] echoArr ) [inline], [inherited]`

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255 for serial, 0 - 255 for TCP)
queryArr	Buffer with data to be sent. The length of the array determines how many bytes are sent and returned
echoArr	Buffer which will contain the data read

Returns

BusProtocolErrors.FTALK_SUCCESS on success, FTALK_INVALID_REPLY_ERROR if reply does not match query data or error code. See BusProtocolErrors for more error codes.

References BusProtocolErrors.FTALK_INVALID_REPLY_ERROR, and BusProtocolErrors.FTALK_SUCCESS.

restartCommunicationsOption() `Int32 restartCommunicationsOption (`
`Int32 slaveAddr,`
`Int32 clearEventLog ) [inline], [inherited]`

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255 for serial, 0 - 255 for TCP)
clearEventLog	Flag when set to one clears in addition the slave's communication even log.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

customFunction() `Int32 customFunction (`
`Int32 slaveAddr,`
`Int32 functionCode,`
`byte [] requestData,`
`[In,Out] byte [] responseData ) [inline], [inherited]`

This method can be used to implement User Defined Function Codes. The caller has only to pass the user data to this function. The assembly of the Modbus frame (the so called ADU) including checksums, slave address and function code and subsequentially the transmission, is taken care of by this method.

The modbus specification reserves function codes 65-72 and 100-110 for user defined functions.

Note

Modbus functions usually have an implied response length and therefore the number of bytes expected to be received is known at the time when sending the request. In case of a custom Modbus function with an open or unknown response length, this function can not be used.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255 for serial, 0 - 255 for TCP)
functionCode	Custom function code to be used for Modbus transaction (Range: 1-127)
requestData	Array with data to be sent as request (not including slave address or function code). The length of the array determines how many request bytes are sent (Range: 0-252).
responseData	Buffer which will be filled with the response data received. The length of the array determines how many bytes are read (Range: 0-252).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

setTimeout() `Int32 setTimeout ( Int32 timeOut ) [inline], [inherited]`

Configures operation or socket time-out.

The time-out value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Parameters

timeOut	Timeout value in ms (Range: 1 - 100000)

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

getTimeout() `Int32 getTimeout ( ) [inline], [inherited]`

Returns

Timeout value in ms

setPollDelay() `Int32 setPollDelay ( Int32 pollDelay ) [inline], [inherited]`

This function sets the delay time which applies between two consecutive Modbus read/write. A value of 0 disables the poll delay.

The delay value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Parameters

pollDelay	Delay value in ms (Range: 0 - 100000), 0 disables poll delay

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

getPollDelay() `Int32 getPollDelay ( ) [inline], [inherited]`

Returns

Delay time in ms, 0 if poll delay is switched off

setRetryCnt() `Int32 setRetryCnt ( Int32 retryCnt ) [inline], [inherited]`

Configures the automatic retry setting. A value of 0 disables any automatic retries.

Note

A protocol must be closed in order to configure it.

Parameters

retryCnt	Retry count (Range: 0 - 10), 0 disables retries

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

getRetryCnt() `Int32 getRetryCnt ( ) [inline], [inherited]`

Returns

Retry count

getTotalCounter() `Int32 getTotalCounter ( ) [inline], [inherited]`

Returns

Counter value

resetTotalCounter() `void resetTotalCounter ( ) [inline], [inherited]`

getSuccessCounter() `Int32 getSuccessCounter ( ) [inline], [inherited]`

Returns

Counter value

resetSuccessCounter() `void resetSuccessCounter ( ) [inline], [inherited]`

configureStandard32BitMode() `[1/2] void configureStandard32BitMode ( ) [inline], [inherited]`

In Standard 32-bit Register Mode a 32-bit value is transmitted as two consecutive 16-bit Modbus registers.

Note

This function call also re-configures the endianess to default little-endian for 32-bit values!

Note

This is the default mode.

configureStandard32BitMode() `[2/2] Int32 configureStandard32BitMode ( Int32 slaveAddr ) [inline], [inherited]`

In Standard 32-bit Register Mode a 32-bit value is transmitted as two consecutive 16-bit Modbus registers.

Note

This function call also re-configures the endianess to default little-endian for 32-bit values!

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureEnron32BitMode() `[1/2] void configureEnron32BitMode ( ) [inline], [inherited]`

Some Modbus flavours like the Daniel/Enron protocol represent a 32-bit value using one 32-bit Modbus register instead of two 16-bit registers.

Note

This function call also re-configures the endianess to big-endian for 32-bit values as defined by the Daniel/ENRON protocol!

configureEnron32BitMode() `[2/2]` `Int32 configureEnron32BitMode ( Int32 slaveAddr ) [inline], [inherited]`

Some Modbus flavours like the Daniel/Enron protocol represent a 32-bit value using one 32-bit Modbus register instead of two 16-bit registers.

Note

This function call also re-configures the endianess to big-endian for 32-bit values as defined by the Daniel/ENRON protocol!

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureCountFromOne() `[1/2]` `void configureCountFromOne ( ) [inline], [inherited]`

This renders the reference range to be 1 to 65536 (0x10000) and register #0 is an illegal register.

Note

This is the default mode.

configureCountFromOne() `[2/2]` `Int32 configureCountFromOne ( Int32 slaveAddr ) [inline], [inherited]`

This renders the reference range to be 1 to 65536 (0x10000) and register #0 is an illegal register.

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureCountFromZero() `void configureCountFromZero ( ) [inline], [inherited]`

This renders the valid reference range to be 0 to 65535 (0xFFFF).

This renders the first register to be #0 for all slaves.

configureBigEndianInts() `[1/2] void configureBigEndianInts ( ) [inline], [inherited]`

Modbus is using little-endian word order for 32-bit values. The data transfer functions operating upon 32-bit int data types can be configured to do a word swap which enables them to read 32-bit data correctly from a big-endian slave.

configureBigEndianInts() `[2/2] Int32 configureBigEndianInts ( Int32 slaveAddr ) [inline], [inherited]`

Modbus is using little-endian word order for 32-bit values. The data transfer functions operating upon 32-bit int data types can be configured to do a word swap which enables them to read 32-bit data correctly from a big-endian slave.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureSwappedFloats() `[1/2] void configureSwappedFloats ( ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Most platforms store floats in IEEE 754 little-endian order which does not need a word swap.

configureSwappedFloats() `[2/2]` `Int32 configureSwappedFloats ( Int32 slaveAddr ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Most platforms store floats in IEEE 754 little-endian order which does not need a word swap.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureLittleEndianInts() `[1/2]` `void configureLittleEndianInts ( ) [inline], [inherited]`

Note

This is the default mode.

configureLittleEndianInts() `[2/2]` `Int32 configureLittleEndianInts ( Int32 slaveAddr ) [inline], [inherited]`

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureIeeeFloats() `[1/2] void configureIeeeFloats ( ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Note

This is the default mode.

configureIeeeFloats() `[2/2] Int32 configureIeeeFloats ( Int32 slaveAddr ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

onfigureCountFromZero() `Int32 onfigureCountFromZero ( Int32 slaveAddr ) [inline], [inherited]`

This renders the valid reference range to be 0 to 65535 (0xFFFF).

This renders the first register to be #0 for all slaves.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

isOpen() `bool isOpen ( ) [inline], [inherited]`

Returns

True = open, False = closed

closeProtocol() `void closeProtocol ( ) [inline], [inherited]`

getPackageVersion() `static string getPackageVersion ( ) [inline], [static], [inherited]`

Returns

Package version string

5.4.4 Property Documentation

port `override Int16 port [get], [set]`

Usually the port number remains unchanged and defaults to 502. However if the port number has to be different from 502 this property must be set before opening the connection with openProtocol().

Note

A protocol must be closed in order to configure it.

TCP Port number of slave device. Default value is 502.

hostName `string hostName [get], [set], [inherited]`

A protocol must be closed in order to configure it.

String with IP address or host name (eg "127.0.0.1")

timeout `Int32 timeout [get], [set], [inherited]`

Configures operation or socket time-out.

The time-out value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Timeout value in ms (Range: 1 - 100000)

pollDelay `Int32 pollDelay [get], [set], [inherited]`

This property sets the delay time which applies between two consecutive Modbus read/write. A value of 0 disables the poll delay.

The delay value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Delay value in ms (Range: 0 - 100000), 0 disables poll delay

retryCnt `Int32 retryCnt [get], [set], [inherited]`

Configures the automatic retry setting. A value of 0 disables any automatic retries.

Note

A protocol must be closed in order to configure it.

Retry count (Range: 0 - 10), 0 disables retries

5.5 MbusRtuOverTcpMasterProtocol Class Reference

MODBUS/Encapsulated RTU (RTU emulated on TCP) Master Protocol class

Public Member Functions

- MbusRtuOverTcpMasterProtocol ()

  Creates new instance
- override Int32 setPort (Int16 portNo)

  Sets the TCP port number used to connect to the Modbus RTU slave device.
- Int32 openProtocol ()

Connects to a TCP slave.

- Int32 openProtocol (string hostName)

  Connects to a TCP slave.
- Int16 getPort ()

  Returns the TCP port number used by the protocol.
- bool isOpen ()

  Returns whether the protocol is open or not.
- void closeProtocol ()

  Closes an open protocol including any associated communication resources (COM ports or sockets).

Static Public Member Functions

- static string getPackageVersion ()

  Returns the package version number.

Properties

- override Int16 port `[get, set]`

  TCP port property
- string hostName `[get, set]`

  Host name
- Int32 timeout `[get, set]`

  Time-out port property
- Int32 pollDelay `[get, set]`

  Poll delay property
- Int32 retryCnt `[get, set]`

  Retry count property

Bit Access

Table 0:00000 (Coils) and Table 1:0000 (Input Status)

- Int32 readCoils (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr)

  Modbus function 1 (01 hex), Read Coil Status/Read Coils.
- Int32 readCoils (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr, Int32 numCoils)

  Modbus function 1 (01 hex), Read Coil Status/Read Coils.
- Int32 readInputDiscretes (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr)

  Modbus function 2 (02 hex), Read Inputs Status/Read Input Discretes.
- Int32 readInputDiscretes (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr, Int32 numDiscretes)

  Modbus function 2 (02 hex), Read Inputs Status/Read Input Discretes.
- Int32 writeCoil (Int32 slaveAddr, Int32 bitAddr, bool bitVal)

Modbus function 5 (05 hex), Force Single Coil/Write Coil.

- Int32 forceMultipleCoils (Int32 slaveAddr, Int32 startRef, bool[ ] bitArr)

  Modbus function 15 (0F hex), Force Multiple Coils.

- Int32 forceMultipleCoils (Int32 slaveAddr, Int32 startRef, bool[ ] bitArr, Int32 numCoils)

  Modbus function 15 (0F hex), Force Multiple Coils.

Register Access (16-bit, 32-bit and floating point)

Table 4:00000 (Holding Registers) and Table 3:00000 (Input Registers)

- Int32 readMultipleRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr)

  Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers (16-bit, 32-bit and floating point).

- Int32 readMultipleRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr, Int32 numRegs)

  Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers (16-bit, 32-bit and floating point).

- Int32 readInputRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr)

  Modbus function 4 (04 hex), Read Input Registers (16-bit, 32-bit and floating point).

- Int32 readInputRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr, Int32 numRegs)

  Modbus function 4 (04 hex), Read Input Registers (16-bit, 32-bit and floating point).

- Int32 writeSingleRegister (Int32 slaveAddr, Int32 regAddr, Int16 regVal)

  Modbus function 6 (06 hex), Preset Single Register/Write Single Register.

- Int32 writeSingleRegister (Int32 slaveAddr, Int32 regAddr, UInt16 regVal)

  Modbus function 6 (06 hex), Preset Single Register/Write Single Register.

- Int32 writeMultipleRegisters (Int32 slaveAddr, Int32 startRef, System.Array regArr)

  Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers (16-bit, 32-bit and floating point).

- Int32 writeMultipleRegisters (Int32 slaveAddr, Int32 startRef, System.Array regArr, Int32 numRegs)

  Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers (16-bit, 32-bit and floating point).

- Int32 maskWriteRegister (Int32 slaveAddr, Int32 regAddr, Int16 andMask, Int16 or↩ Mask)

  Modbus function 22 (16 hex), Mask Write Register.

- Int32 readWriteRegisters (Int32 slaveAddr, Int32 readRef, [In, Out] Int16[ ] readArr, Int32 writeRef, Int16[ ] writeArr)

  Modbus function 23 (17 hex), Read/Write Registers.

- Int32 readWriteRegisters (Int32 slaveAddr, Int32 readRef, [In, Out] Int16[ ] readArr, Int32 readCnt, Int32 writeRef, Int16[ ] writeArr, Int32 writeCnt)

  Modbus function 23 (17 hex), Read/Write Registers.

Modulo-10000 long integer Access

Table 4:00000 (Holding Registers) and Table 3:00000 (Input Registers)

- Int32 readMultipleMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32↩ Arr)

  Modbus function 3 (03 hex) for 32-bit modulo-10000 long int data types, Read Holding Registers/Read Multiple Registers as modulo-10000 long int data.
- Int32 readMultipleMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32↩ Arr, Int32 numRegs)

  Modbus function 3 (03 hex) for 32-bit modulo-10000 long int data types, Read Holding Registers/Read Multiple Registers as modulo-10000 long int data.
- Int32 readInputMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32Arr)

  Modbus function 4 (04 hex) for 32-bit modulo-10000 long int data types, Read Input Registers as modulo-10000 long int data.
- Int32 readInputMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32Arr, Int32 numRegs)

  Modbus function 4 (04 hex) for 32-bit modulo-10000 long int data types, Read Input Registers as modulo-10000 long int data.
- Int32 writeMultipleMod10000 (Int32 slaveAddr, Int32 startRef, Int32[ ] int32Arr)

  Modbus function 16 (10 hex) for 32-bit modulo-10000 long int data types, Preset Multiple Registers/Write Multiple Registers with modulo-10000 long int data.
- Int32 writeMultipleMod10000 (Int32 slaveAddr, Int32 startRef, Int32[ ] int32Arr, Int32 numRegs)

  Modbus function 16 (10 hex) for 32-bit modulo-10000 long int data types, Preset Multiple Registers/Write Multiple Registers with modulo-10000 long int data.

Diagnostics

- Int32 readExceptionStatus (Int32 slaveAddr, out byte statusByte)

  Modbus function 7 (07 hex), Read Exception Status.
- Int32 returnQueryData (Int32 slaveAddr, byte[ ] queryArr, out byte[ ] echoArr)

  Modbus function code 8, sub-function 00, Return Query Data.
- Int32 restartCommunicationsOption (Int32 slaveAddr, Int32 clearEventLog)

  Modbus function code 8, sub-function 01, Restart Communications Option

User Defined Function Codes

- Int32 customFunction (Int32 slaveAddr, Int32 functionCode, byte[ ] requestData, [In, Out] byte[ ] responseData)

  User Defined Function Code

Protocol Configuration

- Int32 setTimeout (Int32 timeOut)

  Configures time-out
- Int32 getTimeout ()

  Returns the current time-out setting
- Int32 setPollDelay (Int32 pollDelay)

  Poll delay property
- Int32 getPollDelay ()

  Returns the poll delay time
- Int32 setRetryCnt (Int32 retryCnt)

  Configures the automatic retry setting
- Int32 getRetryCnt ()

  Returns the automatic retry count

Transmission Statistic Functions

- Int32 getTotalCounter ()

  Returns how often a message transfer has been executed
- void resetTotalCounter ()

  Resets total message transfer counter
- Int32 getSuccessCounter ()

  Returns how often a message transfer was successful
- void resetSuccessCounter ()

  Resets successful message transfer counter

Slave Configuration

- void configureStandard32BitMode ()

  Configures all slaves for Standard 32-bit Mode.
- Int32 configureStandard32BitMode (Int32 slaveAddr)

  Configures a slave for Standard 32-bit Mode.
- void configureEnron32BitMode ()

  Configures all slaves for Daniel/ENRON 32-bit Mode.
- Int32 configureEnron32BitMode (Int32 slaveAddr)

  Configures a slave for Daniel/ENRON 32-bit Mode.
- void configureCountFromOne ()

  Configures the reference counting scheme to start with one for all slaves.
- Int32 configureCountFromOne (Int32 slaveAddr)

  Configures the reference counting scheme to start with one for a slave.
- void configureCountFromZero ()

  Configures the reference counting scheme to start with zero for all slaves.
- void configureBigEndianInts ()

Configures 32-bit int data type functions to do a word swap

- Int32 configureBigEndianInts (Int32 slaveAddr)

  Configures 32-bit int data type functions to do a word swap on a per slave basis
- void configureSwappedFloats ()

  Configures float data type functions to do a word swap
- Int32 configureSwappedFloats (Int32 slaveAddr)

  Configures float data type functions to do a word swap on a per slave basis
- void configureLittleEndianInts ()

  Configures 32-bit int data type functions NOT to do a word swap
- Int32 configureLittleEndianInts (Int32 slaveAddr)

  Configures 32-bit int data type functions NOT to do a word swap on a per slave basis
- void configureIeeeFloats ()

  Configures float data type functions NOT to do a word swap
- Int32 configureIeeeFloats (Int32 slaveAddr)

  Configures float data type functions NOT to do a word swap on a per slave basis
- Int32 onfigureCountFromZero (Int32 slaveAddr)

  Configures the reference counting scheme to start with zero for a slave.

5.5.1 Detailed Description

This class realises the encapsulated (emulated) RTU over TCP master protocol. It provides functions to establish and to close a TCP/IP connection to the slave as well as data and control functions which can be used after a connection to a slave device has been established successfully. The data and control functions are organized different conformance classes.

Using multiple instances of a MbusRtuOverTcpMasterProtocol class enables concurrent protocol transfers using multiple TCP/IP sessions (They should be executed in separate threads).

5.5.2 Constructor & Destructor Documentation

MbusRtuOverTcpMasterProtocol() `MbusRtuOverTcpMasterProtocol ( ) [inline]`

Exceptions

OutOfMemoryException	Creation of class failed

5.5.3 Member Function Documentation

setPort() `override Int32 setPort ( Int16 portNo ) [inline], [virtual]`

Defaults to 1100. Must be set before opening the connection with openProtocol().

Note

A protocol must be closed in order to configure it.

Parameters

portNo	Port number to be used when opening the connection

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

Reimplemented from MbusIpClientBase.

openProtocol() `[1/2] Int32 openProtocol ( ) [inline], [inherited]`

This function establishes a logical network connection between master and slave. After a connection has been established data and control functions can be used. A TCP/IP connection should be closed if it is no longer needed.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

openProtocol() `[2/2] Int32 openProtocol ( string hostName ) [inline], [inherited]`

This function establishes a logical network connection between master and slave. After a connection has been established data and control functions can be used. A TCP/IP connection should be closed if it is no longer needed.

Parameters

hostName	String with IP address or host name (eg "127.0.0.1")

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

getPort() `Int16 getPort ( ) [inline], [inherited]`

Returns

Currently set port number

readCoils() `[1/2]`

```
Int32 readCoils (
        Int32 slaveAddr,
        Int32 startRef,
        [In,Out] bool [] bitArr ) [inline], [inherited]
```

Reads the contents of the discrete outputs (coils, 0:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read. The length of the array determines how many coils are read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readCoils() `[2/2]`

```
Int32 readCoils (
        Int32 slaveAddr,
        Int32 startRef,
        [In,Out] bool [] bitArr,
        Int32 numCoils ) [inline], [inherited]
```

Reads the contents of the discrete outputs (coils, 0:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read.
numCoils	Number of coils to be read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputDiscretes() `[1/2]`

```
Int32 readInputDiscretes (
    Int32 slaveAddr,
    Int32 startRef,
    [In,Out] bool [] bitArr )  [inline], [inherited]
```

Reads the contents of the discrete inputs (input status, 1:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read. The length of the array determines how many inputs are read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputDiscretes() `[2/2]`

```
Int32 readInputDiscretes (
    Int32 slaveAddr,
    Int32 startRef,
```

```
[In,Out] bool [] bitArr,
Int32 numDiscretes )  [inline], [inherited]
```

Reads the contents of the discrete inputs (input status, 1:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read.
num↩ Discretes	Number of inputs to be read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeCoil() `Int32 writeCoil (`

```
Int32 slaveAddr,
Int32 bitAddr,
bool bitVal )  [inline], [inherited]
```

Sets a single discrete output variable (coil, 0:00000 table) to either ON or OFF.

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
bitAddr	Coil address (Range: 1 - 65536)
bitVal	true sets, false clears discrete output variable

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

forceMultipleCoils() [1/2]

```
Int32 forceMultipleCoils (
    Int32 slaveAddr,
    Int32 startRef,
    bool [] bitArr )  [inline], [inherited]
```

Writes binary values into a sequence of discrete outputs (coils, 0:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which contains the data to be sent. The length of the array determines how many coils are written (Range: 1-1968).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

forceMultipleCoils() [2/2]

```
Int32 forceMultipleCoils (
    Int32 slaveAddr,
    Int32 startRef,
    bool [] bitArr,
    Int32 numCoils )  [inline], [inherited]
```

Writes binary values into a sequence of discrete outputs (coils, 0:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which contains the data to be sent.
numCoils	Number of coils to be written (Range: 1-1968).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleRegisters() `[1/2] Int32 readMultipleRegisters (`
`Int32 slaveAddr,`
`Int32 startRef,`
`[In,Out] System.Array regArr ) [inline], [inherited]`

Reads the contents of the output registers (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[]. The length of the array determines how many registers are read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleRegisters() `[2/2] Int32 readMultipleRegisters (`
`Int32 slaveAddr,`
`Int32 startRef,`
`[In,Out] System.Array regArr,`
`Int32 numRegs ) [inline], [inherited]`

Reads the contents of the output registers (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[].
numRegs	Number of values to be read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputRegisters() `[1/2]`

```
Int32 readInputRegisters (
    Int32 slaveAddr,
    Int32 startRef,
    [In, Out] System.Array regArr )  [inline], [inherited]
```

Read the contents of the input registers (3:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[]. The length of the array determines how many registers are read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputRegisters() `[2/2]`

```
Int32 readInputRegisters (
    Int32 slaveAddr,
    Int32 startRef,
    [In, Out] System.Array regArr,
    Int32 numRegs )  [inline], [inherited]
```

Read the contents of the input registers (3:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)

Parameters

startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[].
numRegs	Number of values to be read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeSingleRegister() `[1/2]` `Int32 writeSingleRegister (`
`Int32 slaveAddr,`
`Int32 regAddr,`
`Int16 regVal )` `[inline], [inherited]`

Writes a value into a single output register (holding register, 4:00000 reference).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
regAddr	Register address (Range: 1 - 65536)
regVal	Data to be sent as signed 16-bit value

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeSingleRegister() `[2/2]` `Int32 writeSingleRegister (`
`Int32 slaveAddr,`
`Int32 regAddr,`
`UInt16 regVal )` `[inline], [inherited]`

Writes a value into a single output register (holding register, 4:00000 reference).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
regAddr	Register address (Range: 1 - 65536)
regVal	Data to be sent as unsigned 16-bit value

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleRegisters() `[1/2] Int32 writeMultipleRegisters (`
`Int32 slaveAddr,`
`Int32 startRef,`
`System.Array regArr ) [inline], [inherited]`

Writes values into a sequence of output registers (holding registers, 4:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array with data to be sent. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[]. The length of the array determines how many registers are written (Range: 1-123 for 16-bit, 1-61 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleRegisters() `[2/2] Int32 writeMultipleRegisters (`
`Int32 slaveAddr,`
`Int32 startRef,`

```
System.Array regArr,
Int32 numRegs )  [inline], [inherited]
```

Writes values into a sequence of output registers (holding registers, 4:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array with data to be sent. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[].
numRegs	Number of values to be written (Range: 1-123 for 16-bit, 1-61 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

maskWriteRegister()

```
Int32 maskWriteRegister (
    Int32 slaveAddr,
    Int32 regAddr,
    Int16 andMask,
    Int16 orMask )  [inline], [inherited]
```

Masks bits according to an AND and an OR mask into a single output register (holding register, 4:00000 reference). Masking is done as follows: retVal = (currentVal AND andMask) OR (orMask AND (NOT andMask))

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
regAddr	Register address (Range: 1 - 65536)
andMask	Mask to be applied as a logic AND to the register
orMask	Mask to be applied as a logic OR to the register

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readWriteRegisters() [1/2]

```
Int32 readWriteRegisters (
    Int32 slaveAddr,
    Int32 readRef,
    [In,Out] Int16 [] readArr,
    Int32 writeRef,
    Int16 [] writeArr )  [inline], [inherited]
```

Combines reading and writing of the output registers in one transaction (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
readRef	Start registers for reading (Range: 1 - 65536)
readArr	Buffer which will contain the data read. The length of the array determines how many registers are read (Range: 1-125).
writeRef	Start registers for writing (Range: 1 - 65536)
writeArr	Buffer with data to be sent. The length of the array determines how many registers are written (Range: 1-121).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readWriteRegisters() [2/2]

```
Int32 readWriteRegisters (
    Int32 slaveAddr,
    Int32 readRef,
    [In,Out] Int16 [] readArr,
    Int32 readCnt,
    Int32 writeRef,
    Int16 [] writeArr,
    Int32 writeCnt )  [inline], [inherited]
```

Combines reading and writing of the output registers in one transaction (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
readRef	Start register for reading (Range: 1 - 65536)
readArr	Buffer which will contain the data read.
writeRef	Start register for writing (Range: 1 - 65536)
writeArr	Buffer with data to be sent.
readCnt	Number of registers to be read (Range: 1-125).
writeCnt	Number of registers to be written (Range: 1-121).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleMod10000() `[1/2] Int32 readMultipleMod10000 (`
`Int32 slaveAddr,`
`Int32 startRef,`
`[In,Out] Int32 [] int32Arr ) [inline], [inherited]`

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value into 32-bit int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleMod10000() `[2/2] Int32 readMultipleMod10000 (`

```
Int32 slaveAddr,
Int32 startRef,
[In,Out] Int32 [] int32Arr,
Int32 numRegs )  [inline], [inherited]
```

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value into 32-bit int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read.
numRegs	Number of values to be read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputMod10000() `[1/2] Int32 readInputMod10000 (`

```
Int32 slaveAddr,
Int32 startRef,
[In,Out] Int32 [] int32Arr )  [inline], [inherited]
```

Reads the contents of pairs of consecutive input registers (3:00000 table) representing a modulo-10000 long int value into 32-bit long int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputMod10000() `[2/2] Int32 readInputMod10000 (`

```
Int32 slaveAddr,
Int32 startRef,
[In,Out] Int32 [] int32Arr,
Int32 numRegs ) [inline], [inherited]
```

Reads the contents of pairs of consecutive input registers (3:00000 table) representing a modulo-10000 long int value into 32-bit long int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read.
numRegs	Number of values to be read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleMod10000() `[1/2]` `Int32 writeMultipleMod10000 (`
`Int32 slaveAddr,`
`Int32 startRef,`
`Int32 [] int32Arr )` `[inline], [inherited]`

Writes long int values into pairs of output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer with the data to be sent. The length of the array determines how many values are written (Range: 1-61).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleMod10000() `[2/2]` `Int32 writeMultipleMod10000 (`
`Int32 slaveAddr,`
`Int32 startRef,`
`Int32 [] int32Arr,`
`Int32 numRegs )` `[inline], [inherited]`

Writes long int values into pairs of output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function. Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer with the data to be sent.
numRegs	Number of values to be sent (Range: 1-61).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readExceptionStatus() `Int32 readExceptionStatus (`
`Int32 slaveAddr,`
`out byte statusByte ) [inline], [inherited]`

Reads the eight exception status coils within the slave device.

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
statusByte	Slave status byte. The meaning of this status byte is slave specific and varies from device to device.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

returnQueryData() `Int32 returnQueryData (`
`Int32 slaveAddr,`

```
byte [] queryArr,
out byte [] echoArr )  [inline], [inherited]
```

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255 for serial, 0 - 255 for TCP)
queryArr	Buffer with data to be sent. The length of the array determines how many bytes are sent and returned
echoArr	Buffer which will contain the data read

Returns

BusProtocolErrors.FTALK_SUCCESS on success, FTALK_INVALID_REPLY_ERROR if reply does not match query data or error code. See BusProtocolErrors for more error codes.

References BusProtocolErrors.FTALK_INVALID_REPLY_ERROR, and BusProtocolErrors.FTALK_SUCCESS.

restartCommunicationsOption() `Int32 restartCommunicationsOption (`

```
Int32 slaveAddr,
Int32 clearEventLog )  [inline], [inherited]
```

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255 for serial, 0 - 255 for TCP)
clearEventLog	Flag when set to one clears in addition the slave's communication even log.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

customFunction() `Int32 customFunction (`
`Int32 slaveAddr,`
`Int32 functionCode,`
`byte [] requestData,`
`[In,Out] byte [] responseData ) [inline], [inherited]`

This method can be used to implement User Defined Function Codes. The caller has only to pass the user data to this function. The assembly of the Modbus frame (the so called ADU) including checksums, slave address and function code and subsequentially the transmission, is taken care of by this method.

The modbus specification reserves function codes 65-72 and 100-110 for user defined functions.

Note

Modbus functions usually have an implied response length and therefore the number of bytes expected to be received is known at the time when sending the request. In case of a custom Modbus function with an open or unknown response length, this function can not be used.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255 for serial, 0 - 255 for TCP)
functionCode	Custom function code to be used for Modbus transaction (Range: 1-127)
requestData	Array with data to be sent as request (not including slave address or function code). The length of the array determines how many request bytes are sent (Range: 0-252).
responseData	Buffer which will be filled with the response data received. The length of the array determines how many bytes are read (Range: 0-252).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

setTimeout() `Int32 setTimeout (`
`Int32 timeOut ) [inline], [inherited]`

Configures operation or socket time-out.

The time-out value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Parameters

timeOut	Timeout value in ms (Range: 1 - 100000)

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

getTimeout()

```
Int32 getTimeout ( )  [inline], [inherited]
```

Returns

Timeout value in ms

setPollDelay()

```
Int32 setPollDelay (
        Int32 pollDelay )  [inline], [inherited]
```

This function sets the delay time which applies between two consecutive Modbus read/write. A value of 0 disables the poll delay.

The delay value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Parameters

pollDelay	Delay value in ms (Range: 0 - 100000), 0 disables poll delay

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

getPollDelay() `Int32 getPollDelay ( ) [inline], [inherited]`

Returns

Delay time in ms, 0 if poll delay is switched off

setRetryCnt() `Int32 setRetryCnt ( Int32 retryCnt ) [inline], [inherited]`

Configures the automatic retry setting. A value of 0 disables any automatic retries.

Note

A protocol must be closed in order to configure it.

Parameters

retryCnt	Retry count (Range: 0 - 10), 0 disables retries

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

getRetryCnt() `Int32 getRetryCnt ( ) [inline], [inherited]`

Returns

Retry count

getTotalCounter() `Int32 getTotalCounter ( ) [inline], [inherited]`

Returns

Counter value

resetTotalCounter() `void resetTotalCounter ( ) [inline], [inherited]`

getSuccessCounter() `Int32 getSuccessCounter ( ) [inline], [inherited]`

Returns

Counter value

resetSuccessCounter() `void resetSuccessCounter ( ) [inline], [inherited]`

configureStandard32BitMode() `[1/2] void configureStandard32BitMode ( ) [inline], [inherited]`

In Standard 32-bit Register Mode a 32-bit value is transmitted as two consecutive 16-bit Modbus registers.

Note

This function call also re-configures the endianess to default little-endian for 32-bit values!

Note

This is the default mode.

configureStandard32BitMode() `[2/2] Int32 configureStandard32BitMode ( Int32 slaveAddr ) [inline], [inherited]`

In Standard 32-bit Register Mode a 32-bit value is transmitted as two consecutive 16-bit Modbus registers.

Note

This function call also re-configures the endianess to default little-endian for 32-bit values!

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureEnron32BitMode() [1/2] `void configureEnron32BitMode ( ) [inline], [inherited]`

Some Modbus flavours like the Daniel/Enron protocol represent a 32-bit value using one 32-bit Modbus register instead of two 16-bit registers.

Note

This function call also re-configures the endianess to big-endian for 32-bit values as defined by the Daniel/ENRON protocol!

configureEnron32BitMode() [2/2] `Int32 configureEnron32BitMode ( Int32 slaveAddr ) [inline], [inherited]`

Some Modbus flavours like the Daniel/Enron protocol represent a 32-bit value using one 32-bit Modbus register instead of two 16-bit registers.

Note

This function call also re-configures the endianess to big-endian for 32-bit values as defined by the Daniel/ENRON protocol!

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureCountFromOne() [1/2] `void configureCountFromOne ( ) [inline], [inherited]`

This renders the reference range to be 1 to 65536 (0x10000) and register #0 is an illegal register.

Note

This is the default mode.

configureCountFromOne() [2/2] `Int32 configureCountFromOne ( Int32 slaveAddr ) [inline], [inherited]`

This renders the reference range to be 1 to 65536 (0x10000) and register #0 is an illegal register.

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureCountFromZero() `void configureCountFromZero ( ) [inline], [inherited]`

This renders the valid reference range to be 0 to 65535 (0xFFFF).

This renders the first register to be #0 for all slaves.

configureBigEndianInts() [1/2] `void configureBigEndianInts ( ) [inline], [inherited]`

Modbus is using little-endian word order for 32-bit values. The data transfer functions operating upon 32-bit int data types can be configured to do a word swap which enables them to read 32-bit data correctly from a big-endian slave.

configureBigEndianInts() [2/2] `Int32 configureBigEndianInts ( Int32 slaveAddr ) [inline], [inherited]`

Modbus is using little-endian word order for 32-bit values. The data transfer functions operating upon 32-bit int data types can be configured to do a word swap which enables them to read 32-bit data correctly from a big-endian slave.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureSwappedFloats() `[1/2]` `void configureSwappedFloats ( ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Most platforms store floats in IEEE 754 little-endian order which does not need a word swap.

configureSwappedFloats() `[2/2]` `Int32 configureSwappedFloats ( Int32 slaveAddr ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Most platforms store floats in IEEE 754 little-endian order which does not need a word swap.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureLittleEndianInts() `[1/2]` `void configureLittleEndianInts ( ) [inline], [inherited]`

Note

This is the default mode.

configureLittleEndianInts() `[2/2] Int32 configureLittleEndianInts ( Int32 slaveAddr ) [inline], [inherited]`

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureIeeeFloats() `[1/2] void configureIeeeFloats ( ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Note

This is the default mode.

configureIeeeFloats() `[2/2] Int32 configureIeeeFloats ( Int32 slaveAddr ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

onfigureCountFromZero() `Int32 onfigureCountFromZero ( Int32 slaveAddr ) [inline], [inherited]`

This renders the valid reference range to be 0 to 65535 (0xFFFF).

This renders the first register to be #0 for all slaves.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

isOpen() `bool isOpen ( ) [inline], [inherited]`

Returns

True = open, False = closed

closeProtocol() `void closeProtocol ( ) [inline], [inherited]`

getPackageVersion() `static string getPackageVersion ( ) [inline], [static], [inherited]`

Returns

Package version string

5.5.4 Property Documentation

port `override Int16 port [get], [set]`

Defaults to 1100. Must be set before opening the connection with openProtocol().

Note

A protocol must be closed in order to configure it.

TCP Port number of slave device. Default value is 1100.

hostName `string hostName [get], [set], [inherited]`

A protocol must be closed in order to configure it.

String with IP address or host name (eg "127.0.0.1")

timeout `Int32 timeout [get], [set], [inherited]`

Configures operation or socket time-out.

The time-out value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Timeout value in ms (Range: 1 - 100000)

pollDelay `Int32 pollDelay [get], [set], [inherited]`

This property sets the delay time which applies between two consecutive Modbus read/write. A value of 0 disables the poll delay.

The delay value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Delay value in ms (Range: 0 - 100000), 0 disables poll delay

retryCnt `Int32 retryCnt [get], [set], [inherited]`

Configures the automatic retry setting. A value of 0 disables any automatic retries.

Note

A protocol must be closed in order to configure it.

Retry count (Range: 0 - 10), 0 disables retries

5.6 MbusAsciiOverTcpMasterProtocol Class Reference

MODBUS ASCII over TCP Master Protocol class

Public Member Functions

- MbusAsciiOverTcpMasterProtocol ()

  Creates new instance
- override Int32 setPort (Int16 portNo)

  Sets the TCP port number used to connect to the Modbus ASCII slave device.
- Int32 openProtocol ()

  Connects to a TCP slave.
- Int32 openProtocol (string hostName)

  Connects to a TCP slave.
- Int16 getPort ()

  Returns the TCP port number used by the protocol.
- bool isOpen ()

  Returns whether the protocol is open or not.
- void closeProtocol ()

  Closes an open protocol including any associated communication resources (COM ports or sockets).

Static Public Member Functions

- static string getPackageVersion ()

  Returns the package version number.

Properties

- override Int16 port `[get, set]`

  TCP port property
- string hostName `[get, set]`

  Host name
- Int32 timeout `[get, set]`

  Time-out port property
- Int32 pollDelay `[get, set]`

  Poll delay property
- Int32 retryCnt `[get, set]`

  Retry count property

Bit Access

Table 0:00000 (Coils) and Table 1:0000 (Input Status)

- Int32 readCoils (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr)

  Modbus function 1 (01 hex), Read Coil Status/Read Coils.
- Int32 readCoils (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr, Int32 numCoils)

Modbus function 1 (01 hex), Read Coil Status/Read Coils.

- Int32 readInputDiscretes (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr)

  Modbus function 2 (02 hex), Read Inputs Status/Read Input Discretes.
- Int32 readInputDiscretes (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr, Int32 numDiscretes)

  Modbus function 2 (02 hex), Read Inputs Status/Read Input Discretes.
- Int32 writeCoil (Int32 slaveAddr, Int32 bitAddr, bool bitVal)

  Modbus function 5 (05 hex), Force Single Coil/Write Coil.
- Int32 forceMultipleCoils (Int32 slaveAddr, Int32 startRef, bool[ ] bitArr)

  Modbus function 15 (0F hex), Force Multiple Coils.
- Int32 forceMultipleCoils (Int32 slaveAddr, Int32 startRef, bool[ ] bitArr, Int32 numCoils)

  Modbus function 15 (0F hex), Force Multiple Coils.

Register Access (16-bit, 32-bit and floating point)

Table 4:00000 (Holding Registers) and Table 3:00000 (Input Registers)

- Int32 readMultipleRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr)

  Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers (16-bit, 32-bit and floating point).
- Int32 readMultipleRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr, Int32 numRegs)

  Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers (16-bit, 32-bit and floating point).
- Int32 readInputRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr)

  Modbus function 4 (04 hex), Read Input Registers (16-bit, 32-bit and floating point).
- Int32 readInputRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr, Int32 numRegs)

  Modbus function 4 (04 hex), Read Input Registers (16-bit, 32-bit and floating point).
- Int32 writeSingleRegister (Int32 slaveAddr, Int32 regAddr, Int16 regVal)

  Modbus function 6 (06 hex), Preset Single Register/Write Single Register.
- Int32 writeSingleRegister (Int32 slaveAddr, Int32 regAddr, UInt16 regVal)

  Modbus function 6 (06 hex), Preset Single Register/Write Single Register.
- Int32 writeMultipleRegisters (Int32 slaveAddr, Int32 startRef, System.Array regArr)

  Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers (16-bit, 32-bit and floating point).
- Int32 writeMultipleRegisters (Int32 slaveAddr, Int32 startRef, System.Array regArr, Int32 numRegs)

  Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers (16-bit, 32-bit and floating point).
- Int32 maskWriteRegister (Int32 slaveAddr, Int32 regAddr, Int16 andMask, Int16 or↩ Mask)

  Modbus function 22 (16 hex), Mask Write Register.
- Int32 readWriteRegisters (Int32 slaveAddr, Int32 readRef, [In, Out] Int16[ ] readArr, Int32 writeRef, Int16[ ] writeArr)

Modbus function 23 (17 hex), Read/Write Registers.

- Int32 readWriteRegisters (Int32 slaveAddr, Int32 readRef, [In, Out] Int16[ ] readArr, Int32 readCnt, Int32 writeRef, Int16[ ] writeArr, Int32 writeCnt)

  Modbus function 23 (17 hex), Read/Write Registers.

Modulo-10000 long integer Access

Table 4:00000 (Holding Registers) and Table 3:00000 (Input Registers)

- Int32 readMultipleMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32↩ Arr)

  Modbus function 3 (03 hex) for 32-bit modulo-10000 long int data types, Read Holding Registers/Read Multiple Registers as modulo-10000 long int data.
- Int32 readMultipleMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32↩ Arr, Int32 numRegs)

  Modbus function 3 (03 hex) for 32-bit modulo-10000 long int data types, Read Holding Registers/Read Multiple Registers as modulo-10000 long int data.
- Int32 readInputMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32Arr)

  Modbus function 4 (04 hex) for 32-bit modulo-10000 long int data types, Read Input Registers as modulo-10000 long int data.
- Int32 readInputMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32Arr, Int32 numRegs)

  Modbus function 4 (04 hex) for 32-bit modulo-10000 long int data types, Read Input Registers as modulo-10000 long int data.
- Int32 writeMultipleMod10000 (Int32 slaveAddr, Int32 startRef, Int32[ ] int32Arr)

  Modbus function 16 (10 hex) for 32-bit modulo-10000 long int data types, Preset Multiple Registers/Write Multiple Registers with modulo-10000 long int data.
- Int32 writeMultipleMod10000 (Int32 slaveAddr, Int32 startRef, Int32[ ] int32Arr, Int32 numRegs)

  Modbus function 16 (10 hex) for 32-bit modulo-10000 long int data types, Preset Multiple Registers/Write Multiple Registers with modulo-10000 long int data.

Diagnostics

- Int32 readExceptionStatus (Int32 slaveAddr, out byte statusByte)

  Modbus function 7 (07 hex), Read Exception Status.
- Int32 returnQueryData (Int32 slaveAddr, byte[ ] queryArr, out byte[ ] echoArr)

  Modbus function code 8, sub-function 00, Return Query Data.
- Int32 restartCommunicationsOption (Int32 slaveAddr, Int32 clearEventLog)

  Modbus function code 8, sub-function 01, Restart Communications Option

User Defined Function Codes

- Int32 customFunction (Int32 slaveAddr, Int32 functionCode, byte[ ] requestData, [In, Out] byte[ ] responseData)

  User Defined Function Code

Protocol Configuration

- Int32 setTimeout (Int32 timeOut)

  Configures time-out
- Int32 getTimeout ()

  Returns the current time-out setting
- Int32 setPollDelay (Int32 pollDelay)

  Poll delay property
- Int32 getPollDelay ()

  Returns the poll delay time
- Int32 setRetryCnt (Int32 retryCnt)

  Configures the automatic retry setting
- Int32 getRetryCnt ()

  Returns the automatic retry count

Transmission Statistic Functions

- Int32 getTotalCounter ()

  Returns how often a message transfer has been executed
- void resetTotalCounter ()

  Resets total message transfer counter
- Int32 getSuccessCounter ()

  Returns how often a message transfer was successful
- void resetSuccessCounter ()

  Resets successful message transfer counter

Slave Configuration

- void configureStandard32BitMode ()

  Configures all slaves for Standard 32-bit Mode.
- Int32 configureStandard32BitMode (Int32 slaveAddr)

  Configures a slave for Standard 32-bit Mode.
- void configureEnron32BitMode ()

  Configures all slaves for Daniel/ENRON 32-bit Mode.
- Int32 configureEnron32BitMode (Int32 slaveAddr)

  Configures a slave for Daniel/ENRON 32-bit Mode.
- void configureCountFromOne ()

  Configures the reference counting scheme to start with one for all slaves.
- Int32 configureCountFromOne (Int32 slaveAddr)

  Configures the reference counting scheme to start with one for a slave.
- void configureCountFromZero ()

  Configures the reference counting scheme to start with zero for all slaves.
- void configureBigEndianInts ()

Configures 32-bit int data type functions to do a word swap

- Int32 configureBigEndianInts (Int32 slaveAddr)

  Configures 32-bit int data type functions to do a word swap on a per slave basis
- void configureSwappedFloats ()

  Configures float data type functions to do a word swap
- Int32 configureSwappedFloats (Int32 slaveAddr)

  Configures float data type functions to do a word swap on a per slave basis
- void configureLittleEndianInts ()

  Configures 32-bit int data type functions NOT to do a word swap
- Int32 configureLittleEndianInts (Int32 slaveAddr)

  Configures 32-bit int data type functions NOT to do a word swap on a per slave basis
- void configureIeeeFloats ()

  Configures float data type functions NOT to do a word swap
- Int32 configureIeeeFloats (Int32 slaveAddr)

  Configures float data type functions NOT to do a word swap on a per slave basis
- Int32 onfigureCountFromZero (Int32 slaveAddr)

  Configures the reference counting scheme to start with zero for a slave.

5.6.1 Detailed Description

This class realises the Modbus ASCII protocol using TCP as transport layer. It provides functions to establish and to close a TCP/IP connection to the slave as well as data and control functions which can be used after a connection to a slave device has been established successfully. The data and control functions are organized different conformance classes.

Using multiple instances of a MbusAsciiOverTcpMasterProtocol class enables concurrent protocol transfers using multiple TCP/IP sessions (They should be executed in separate threads).

5.6.2 Constructor & Destructor Documentation

MbusAsciiOverTcpMasterProtocol() `MbusAsciiOverTcpMasterProtocol ( ) [inline]`

Exceptions

OutOfMemoryException	Creation of class failed

5.6.3 Member Function Documentation

setPort() `override Int32 setPort (`
`Int16 portNo ) [inline], [virtual]`

Defaults to 23 (Telnet port). Must be set before opening the connection with openProtocol().

Note

A protocol must be closed in order to configure it.

Parameters

portNo	Port number to be used when opening the connection

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

Reimplemented from MbusIpClientBase.

openProtocol() `[1/2] Int32 openProtocol ( ) [inline], [inherited]`

This function establishes a logical network connection between master and slave. After a connection has been established data and control functions can be used. A TCP/IP connection should be closed if it is no longer needed.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

openProtocol() `[2/2] Int32 openProtocol (`
`string hostName ) [inline], [inherited]`

This function establishes a logical network connection between master and slave. After a connection has been established data and control functions can be used. A TCP/IP connection should be closed if it is no longer needed.

Parameters

hostName	String with IP address or host name (eg "127.0.0.1")

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

getPort() `Int16 getPort ( ) [inline], [inherited]`

Returns

Currently set port number

readCoils() `[1/2]`

```
Int32 readCoils (
        Int32 slaveAddr,
        Int32 startRef,
        [In,Out] bool [] bitArr ) [inline], [inherited]
```

Reads the contents of the discrete outputs (coils, 0:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read. The length of the array determines how many coils are read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readCoils() `[2/2]`

```
Int32 readCoils (
        Int32 slaveAddr,
        Int32 startRef,
        [In,Out] bool [] bitArr,
        Int32 numCoils ) [inline], [inherited]
```

Reads the contents of the discrete outputs (coils, 0:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read.
numCoils	Number of coils to be read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputDiscretes() `[1/2]`

```
Int32 readInputDiscretes (
    Int32 slaveAddr,
    Int32 startRef,
    [In,Out] bool [] bitArr )  [inline], [inherited]
```

Reads the contents of the discrete inputs (input status, 1:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read. The length of the array determines how many inputs are read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputDiscretes() `[2/2]`

```
Int32 readInputDiscretes (
    Int32 slaveAddr,
    Int32 startRef,
```

```
[In,Out] bool [] bitArr,
Int32 numDiscretes )  [inline], [inherited]
```

Reads the contents of the discrete inputs (input status, 1:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read.
num↩ Discretes	Number of inputs to be read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeCoil() `Int32 writeCoil (`

```
Int32 slaveAddr,
Int32 bitAddr,
bool bitVal )  [inline], [inherited]
```

Sets a single discrete output variable (coil, 0:00000 table) to either ON or OFF.

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
bitAddr	Coil address (Range: 1 - 65536)
bitVal	true sets, false clears discrete output variable

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

forceMultipleCoils() [1/2]

```
Int32 forceMultipleCoils (
    Int32 slaveAddr,
    Int32 startRef,
    bool [] bitArr )  [inline], [inherited]
```

Writes binary values into a sequence of discrete outputs (coils, 0:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which contains the data to be sent. The length of the array determines how many coils are written (Range: 1-1968).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

forceMultipleCoils() [2/2]

```
Int32 forceMultipleCoils (
    Int32 slaveAddr,
    Int32 startRef,
    bool [] bitArr,
    Int32 numCoils )  [inline], [inherited]
```

Writes binary values into a sequence of discrete outputs (coils, 0:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which contains the data to be sent.
numCoils	Number of coils to be written (Range: 1-1968).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleRegisters() `[1/2] Int32 readMultipleRegisters (`
`Int32 slaveAddr,`
`Int32 startRef,`
`[In,Out] System.Array regArr ) [inline], [inherited]`

Reads the contents of the output registers (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[]. The length of the array determines how many registers are read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleRegisters() `[2/2] Int32 readMultipleRegisters (`
`Int32 slaveAddr,`
`Int32 startRef,`
`[In,Out] System.Array regArr,`
`Int32 numRegs ) [inline], [inherited]`

Reads the contents of the output registers (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[].
numRegs	Number of values to be read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputRegisters() `[1/2]`

```
Int32 readInputRegisters (
    Int32 slaveAddr,
    Int32 startRef,
    [In, Out] System.Array regArr )  [inline], [inherited]
```

Read the contents of the input registers (3:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[]. The length of the array determines how many registers are read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputRegisters() `[2/2]`

```
Int32 readInputRegisters (
    Int32 slaveAddr,
    Int32 startRef,
    [In, Out] System.Array regArr,
    Int32 numRegs )  [inline], [inherited]
```

Read the contents of the input registers (3:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)

Parameters

startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[].
numRegs	Number of values to be read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeSingleRegister() `[1/2]` `Int32 writeSingleRegister (`
`Int32 slaveAddr,`
`Int32 regAddr,`
`Int16 regVal )` `[inline], [inherited]`

Writes a value into a single output register (holding register, 4:00000 reference).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
regAddr	Register address (Range: 1 - 65536)
regVal	Data to be sent as signed 16-bit value

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeSingleRegister() `[2/2]` `Int32 writeSingleRegister (`
`Int32 slaveAddr,`
`Int32 regAddr,`
`UInt16 regVal )` `[inline], [inherited]`

Writes a value into a single output register (holding register, 4:00000 reference).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
regAddr	Register address (Range: 1 - 65536)
regVal	Data to be sent as unsigned 16-bit value

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleRegisters() `[1/2] Int32 writeMultipleRegisters (`
`Int32 slaveAddr,`
`Int32 startRef,`
`System.Array regArr ) [inline], [inherited]`

Writes values into a sequence of output registers (holding registers, 4:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array with data to be sent. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[]. The length of the array determines how many registers are written (Range: 1-123 for 16-bit, 1-61 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleRegisters() `[2/2] Int32 writeMultipleRegisters (`
`Int32 slaveAddr,`
`Int32 startRef,`

```
System.Array regArr,
Int32 numRegs )  [inline], [inherited]
```

Writes values into a sequence of output registers (holding registers, 4:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array with data to be sent. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[].
numRegs	Number of values to be written (Range: 1-123 for 16-bit, 1-61 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

maskWriteRegister()

```
Int32 maskWriteRegister (
    Int32 slaveAddr,
    Int32 regAddr,
    Int16 andMask,
    Int16 orMask )  [inline], [inherited]
```

Masks bits according to an AND and an OR mask into a single output register (holding register, 4:00000 reference). Masking is done as follows: retVal = (currentVal AND andMask) OR (orMask AND (NOT andMask))

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
regAddr	Register address (Range: 1 - 65536)
andMask	Mask to be applied as a logic AND to the register
orMask	Mask to be applied as a logic OR to the register

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readWriteRegisters() [1/2]

```
Int32 readWriteRegisters (
    Int32 slaveAddr,
    Int32 readRef,
    [In,Out] Int16 [] readArr,
    Int32 writeRef,
    Int16 [] writeArr )  [inline], [inherited]
```

Combines reading and writing of the output registers in one transaction (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
readRef	Start registers for reading (Range: 1 - 65536)
readArr	Buffer which will contain the data read. The length of the array determines how many registers are read (Range: 1-125).
writeRef	Start registers for writing (Range: 1 - 65536)
writeArr	Buffer with data to be sent. The length of the array determines how many registers are written (Range: 1-121).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readWriteRegisters() [2/2]

```
Int32 readWriteRegisters (
    Int32 slaveAddr,
    Int32 readRef,
    [In,Out] Int16 [] readArr,
    Int32 readCnt,
    Int32 writeRef,
    Int16 [] writeArr,
    Int32 writeCnt )  [inline], [inherited]
```

Combines reading and writing of the output registers in one transaction (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
readRef	Start register for reading (Range: 1 - 65536)
readArr	Buffer which will contain the data read.
writeRef	Start register for writing (Range: 1 - 65536)
writeArr	Buffer with data to be sent.
readCnt	Number of registers to be read (Range: 1-125).
writeCnt	Number of registers to be written (Range: 1-121).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleMod10000() `[1/2] Int32 readMultipleMod10000 (`
`Int32 slaveAddr,`
`Int32 startRef,`
`[In,Out] Int32 [] int32Arr ) [inline], [inherited]`

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value into 32-bit int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleMod10000() `[2/2] Int32 readMultipleMod10000 (`

```
Int32 slaveAddr,
Int32 startRef,
[In,Out] Int32 [] int32Arr,
Int32 numRegs ) [inline], [inherited]
```

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value into 32-bit int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read.
numRegs	Number of values to be read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputMod10000() `[1/2] Int32 readInputMod10000 (`

```
Int32 slaveAddr,
Int32 startRef,
[In,Out] Int32 [] int32Arr ) [inline], [inherited]
```

Reads the contents of pairs of consecutive input registers (3:00000 table) representing a modulo-10000 long int value into 32-bit long int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputMod10000() `[2/2] Int32 readInputMod10000 (`

```
Int32 slaveAddr,
Int32 startRef,
[In,Out] Int32 [] int32Arr,
Int32 numRegs ) [inline], [inherited]
```

Reads the contents of pairs of consecutive input registers (3:00000 table) representing a modulo-10000 long int value into 32-bit long int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read.
numRegs	Number of values to be read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleMod10000() [1/2] `Int32 writeMultipleMod10000 (`

```
Int32 slaveAddr,
Int32 startRef,
Int32 [] int32Arr ) [inline], [inherited]
```

Writes long int values into pairs of output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer with the data to be sent. The length of the array determines how many values are written (Range: 1-61).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleMod10000() [2/2] `Int32 writeMultipleMod10000 (`

```
Int32 slaveAddr,
Int32 startRef,
Int32 [] int32Arr,
Int32 numRegs ) [inline], [inherited]
```

Writes long int values into pairs of output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function. Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer with the data to be sent.
numRegs	Number of values to be sent (Range: 1-61).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readExceptionStatus() `Int32 readExceptionStatus (`
`Int32 slaveAddr,`
`out byte statusByte ) [inline], [inherited]`

Reads the eight exception status coils within the slave device.

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
statusByte	Slave status byte. The meaning of this status byte is slave specific and varies from device to device.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

returnQueryData() `Int32 returnQueryData (`
`Int32 slaveAddr,`

```
byte [] queryArr,
out byte [] echoArr )  [inline], [inherited]
```

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255 for serial, 0 - 255 for TCP)
queryArr	Buffer with data to be sent. The length of the array determines how many bytes are sent and returned
echoArr	Buffer which will contain the data read

Returns

BusProtocolErrors.FTALK_SUCCESS on success, FTALK_INVALID_REPLY_ERROR if reply does not match query data or error code. See BusProtocolErrors for more error codes.

References BusProtocolErrors.FTALK_INVALID_REPLY_ERROR, and BusProtocolErrors.FTALK_SUCCESS.

restartCommunicationsOption() `Int32 restartCommunicationsOption (`

```
Int32 slaveAddr,
Int32 clearEventLog )  [inline], [inherited]
```

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255 for serial, 0 - 255 for TCP)
clearEventLog	Flag when set to one clears in addition the slave's communication even log.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

customFunction() `Int32 customFunction (`
`Int32 slaveAddr,`
`Int32 functionCode,`
`byte [] requestData,`
`[In,Out] byte [] responseData ) [inline], [inherited]`

This method can be used to implement User Defined Function Codes. The caller has only to pass the user data to this function. The assembly of the Modbus frame (the so called ADU) including checksums, slave address and function code and subsequentially the transmission, is taken care of by this method.

The modbus specification reserves function codes 65-72 and 100-110 for user defined functions.

Note

Modbus functions usually have an implied response length and therefore the number of bytes expected to be received is known at the time when sending the request. In case of a custom Modbus function with an open or unknown response length, this function can not be used.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255 for serial, 0 - 255 for TCP)
functionCode	Custom function code to be used for Modbus transaction (Range: 1-127)
requestData	Array with data to be sent as request (not including slave address or function code). The length of the array determines how many request bytes are sent (Range: 0-252).
responseData	Buffer which will be filled with the response data received. The length of the array determines how many bytes are read (Range: 0-252).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

setTimeout() `Int32 setTimeout (`
`Int32 timeOut ) [inline], [inherited]`

Configures operation or socket time-out.

The time-out value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Parameters

timeOut	Timeout value in ms (Range: 1 - 100000)

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

getTimeout()

```
Int32 getTimeout ( )  [inline], [inherited]
```

Returns

Timeout value in ms

setPollDelay()

```
Int32 setPollDelay (
        Int32 pollDelay )  [inline], [inherited]
```

This function sets the delay time which applies between two consecutive Modbus read/write. A value of 0 disables the poll delay.

The delay value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Parameters

pollDelay	Delay value in ms (Range: 0 - 100000), 0 disables poll delay

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

getPollDelay() `Int32 getPollDelay ( ) [inline], [inherited]`

Returns

Delay time in ms, 0 if poll delay is switched off

setRetryCnt() `Int32 setRetryCnt ( Int32 retryCnt ) [inline], [inherited]`

Configures the automatic retry setting. A value of 0 disables any automatic retries.

Note

A protocol must be closed in order to configure it.

Parameters

retryCnt	Retry count (Range: 0 - 10), 0 disables retries

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

getRetryCnt() `Int32 getRetryCnt ( ) [inline], [inherited]`

Returns

Retry count

getTotalCounter() `Int32 getTotalCounter ( ) [inline], [inherited]`

Returns

Counter value

resetTotalCounter() `void resetTotalCounter ( ) [inline], [inherited]`

getSuccessCounter() `Int32 getSuccessCounter ( ) [inline], [inherited]`

Returns

Counter value

resetSuccessCounter() `void resetSuccessCounter ( ) [inline], [inherited]`

configureStandard32BitMode() `[1/2] void configureStandard32BitMode ( ) [inline], [inherited]`

In Standard 32-bit Register Mode a 32-bit value is transmitted as two consecutive 16-bit Modbus registers.

Note

This function call also re-configures the endianess to default little-endian for 32-bit values!

Note

This is the default mode.

configureStandard32BitMode() `[2/2] Int32 configureStandard32BitMode ( Int32 slaveAddr ) [inline], [inherited]`

In Standard 32-bit Register Mode a 32-bit value is transmitted as two consecutive 16-bit Modbus registers.

Note

This function call also re-configures the endianess to default little-endian for 32-bit values!

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureEnron32BitMode() [1/2] `void configureEnron32BitMode ( ) [inline], [inherited]`

Some Modbus flavours like the Daniel/Enron protocol represent a 32-bit value using one 32-bit Modbus register instead of two 16-bit registers.

Note

This function call also re-configures the endianess to big-endian for 32-bit values as defined by the Daniel/ENRON protocol!

configureEnron32BitMode() [2/2] `Int32 configureEnron32BitMode ( Int32 slaveAddr ) [inline], [inherited]`

Some Modbus flavours like the Daniel/Enron protocol represent a 32-bit value using one 32-bit Modbus register instead of two 16-bit registers.

Note

This function call also re-configures the endianess to big-endian for 32-bit values as defined by the Daniel/ENRON protocol!

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureCountFromOne() [1/2] `void configureCountFromOne ( ) [inline], [inherited]`

This renders the reference range to be 1 to 65536 (0x10000) and register #0 is an illegal register.

Note

This is the default mode.

configureCountFromOne() [2/2] `Int32 configureCountFromOne ( Int32 slaveAddr ) [inline], [inherited]`

This renders the reference range to be 1 to 65536 (0x10000) and register #0 is an illegal register.

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureCountFromZero() `void configureCountFromZero ( ) [inline], [inherited]`

This renders the valid reference range to be 0 to 65535 (0xFFFF).

This renders the first register to be #0 for all slaves.

configureBigEndianInts() [1/2] `void configureBigEndianInts ( ) [inline], [inherited]`

Modbus is using little-endian word order for 32-bit values. The data transfer functions operating upon 32-bit int data types can be configured to do a word swap which enables them to read 32-bit data correctly from a big-endian slave.

configureBigEndianInts() [2/2] `Int32 configureBigEndianInts ( Int32 slaveAddr ) [inline], [inherited]`

Modbus is using little-endian word order for 32-bit values. The data transfer functions operating upon 32-bit int data types can be configured to do a word swap which enables them to read 32-bit data correctly from a big-endian slave.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureSwappedFloats() `[1/2]` `void configureSwappedFloats ( ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Most platforms store floats in IEEE 754 little-endian order which does not need a word swap.

configureSwappedFloats() `[2/2]` `Int32 configureSwappedFloats ( Int32 slaveAddr ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Most platforms store floats in IEEE 754 little-endian order which does not need a word swap.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureLittleEndianInts() `[1/2]` `void configureLittleEndianInts ( ) [inline], [inherited]`

Note

This is the default mode.

configureLittleEndianInts() `[2/2] Int32 configureLittleEndianInts ( Int32 slaveAddr ) [inline], [inherited]`

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureIeeeFloats() `[1/2] void configureIeeeFloats ( ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Note

This is the default mode.

configureIeeeFloats() `[2/2] Int32 configureIeeeFloats ( Int32 slaveAddr ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

onfigureCountFromZero() `Int32 onfigureCountFromZero ( Int32 slaveAddr ) [inline], [inherited]`

This renders the valid reference range to be 0 to 65535 (0xFFFF).

This renders the first register to be #0 for all slaves.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

isOpen() `bool isOpen ( ) [inline], [inherited]`

Returns

True = open, False = closed

closeProtocol() `void closeProtocol ( ) [inline], [inherited]`

getPackageVersion() `static string getPackageVersion ( ) [inline], [static], [inherited]`

Returns

Package version string

5.6.4 Property Documentation

port `override Int16 port [get], [set]`

Defaults to 23 (Telnet port). Must be set before opening the connection with openProtocol().

Note

A protocol must be closed in order to configure it.

TCP Port number of slave device. Default value is 23.

hostName `string hostName [get], [set], [inherited]`

A protocol must be closed in order to configure it.

String with IP address or host name (eg "127.0.0.1")

timeout `Int32 timeout [get], [set], [inherited]`

Configures operation or socket time-out.

The time-out value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Timeout value in ms (Range: 1 - 100000)

pollDelay `Int32 pollDelay [get], [set], [inherited]`

This property sets the delay time which applies between two consecutive Modbus read/write. A value of 0 disables the poll delay.

The delay value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Delay value in ms (Range: 0 - 100000), 0 disables poll delay

retryCnt `Int32 retryCnt [get], [set], [inherited]`

Configures the automatic retry setting. A value of 0 disables any automatic retries.

Note

A protocol must be closed in order to configure it.

Retry count (Range: 0 - 10), 0 disables retries

5.7 MbusUdpMasterProtocol Class Reference

MODBUS/UDP Master Protocol class

Public Member Functions

- MbusUdpMasterProtocol ()

  Creates new instance
- override Int32 setPort (Int16 portNo)

  Sets the TCP port number of the Modbus slave device.
- Int32 adamSendReceiveAsciiCmd (string command, out string response)

  Send/Receive ADAM 5000/6000 ASCII command.
- Int32 openProtocol ()

  Connects to a TCP slave.
- Int32 openProtocol (string hostName)

  Connects to a TCP slave.
- Int16 getPort ()

  Returns the TCP port number used by the protocol.
- bool isOpen ()

  Returns whether the protocol is open or not.
- void closeProtocol ()

  Closes an open protocol including any associated communication resources (COM ports or sockets).

Static Public Member Functions

- static string getPackageVersion ()

  Returns the package version number.

Properties

- override Int16 port `[get, set]`

  TCP port property
- string hostName `[get, set]`

  Host name
- Int32 timeout `[get, set]`

  Time-out port property
- Int32 pollDelay `[get, set]`

  Poll delay property
- Int32 retryCnt `[get, set]`

  Retry count property

Bit Access

Table 0:00000 (Coils) and Table 1:0000 (Input Status)

- Int32 readCoils (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr)

Modbus function 1 (01 hex), Read Coil Status/Read Coils.

- Int32 readCoils (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr, Int32 numCoils)

  Modbus function 1 (01 hex), Read Coil Status/Read Coils.
- Int32 readInputDiscretes (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr)

  Modbus function 2 (02 hex), Read Inputs Status/Read Input Discretes.
- Int32 readInputDiscretes (Int32 slaveAddr, Int32 startRef, [In, Out] bool[ ] bitArr, Int32 numDiscretes)

  Modbus function 2 (02 hex), Read Inputs Status/Read Input Discretes.
- Int32 writeCoil (Int32 slaveAddr, Int32 bitAddr, bool bitVal)

  Modbus function 5 (05 hex), Force Single Coil/Write Coil.
- Int32 forceMultipleCoils (Int32 slaveAddr, Int32 startRef, bool[ ] bitArr)

  Modbus function 15 (0F hex), Force Multiple Coils.
- Int32 forceMultipleCoils (Int32 slaveAddr, Int32 startRef, bool[ ] bitArr, Int32 numCoils)

  Modbus function 15 (0F hex), Force Multiple Coils.

Register Access (16-bit, 32-bit and floating point)

Table 4:00000 (Holding Registers) and Table 3:00000 (Input Registers)

- Int32 readMultipleRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr)

  Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers (16-bit, 32-bit and floating point).
- Int32 readMultipleRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr, Int32 numRegs)

  Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers (16-bit, 32-bit and floating point).
- Int32 readInputRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr)

  Modbus function 4 (04 hex), Read Input Registers (16-bit, 32-bit and floating point).
- Int32 readInputRegisters (Int32 slaveAddr, Int32 startRef, [In, Out] System.Array regArr, Int32 numRegs)

  Modbus function 4 (04 hex), Read Input Registers (16-bit, 32-bit and floating point).
- Int32 writeSingleRegister (Int32 slaveAddr, Int32 regAddr, Int16 regVal)

  Modbus function 6 (06 hex), Preset Single Register/Write Single Register.
- Int32 writeSingleRegister (Int32 slaveAddr, Int32 regAddr, UInt16 regVal)

  Modbus function 6 (06 hex), Preset Single Register/Write Single Register.
- Int32 writeMultipleRegisters (Int32 slaveAddr, Int32 startRef, System.Array regArr)

  Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers (16-bit, 32-bit and floating point).
- Int32 writeMultipleRegisters (Int32 slaveAddr, Int32 startRef, System.Array regArr, Int32 numRegs)

  Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers (16-bit, 32-bit and floating point).
- Int32 maskWriteRegister (Int32 slaveAddr, Int32 regAddr, Int16 andMask, Int16 or↩ Mask)

Modbus function 22 (16 hex), Mask Write Register.

- Int32 readWriteRegisters (Int32 slaveAddr, Int32 readRef, [In, Out] Int16[ ] readArr, Int32 writeRef, Int16[ ] writeArr)

  Modbus function 23 (17 hex), Read/Write Registers.

- Int32 readWriteRegisters (Int32 slaveAddr, Int32 readRef, [In, Out] Int16[ ] readArr, Int32 readCnt, Int32 writeRef, Int16[ ] writeArr, Int32 writeCnt)

  Modbus function 23 (17 hex), Read/Write Registers.

Modulo-10000 long integer Access

Table 4:00000 (Holding Registers) and Table 3:00000 (Input Registers)

- Int32 readMultipleMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32↩ Arr)

  Modbus function 3 (03 hex) for 32-bit modulo-10000 long int data types, Read Holding Registers/Read Multiple Registers as modulo-10000 long int data.

- Int32 readMultipleMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32↩ Arr, Int32 numRegs)

  Modbus function 3 (03 hex) for 32-bit modulo-10000 long int data types, Read Holding Registers/Read Multiple Registers as modulo-10000 long int data.

- Int32 readInputMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32Arr)

  Modbus function 4 (04 hex) for 32-bit modulo-10000 long int data types, Read Input Registers as modulo-10000 long int data.

- Int32 readInputMod10000 (Int32 slaveAddr, Int32 startRef, [In, Out] Int32[ ] int32Arr, Int32 numRegs)

  Modbus function 4 (04 hex) for 32-bit modulo-10000 long int data types, Read Input Registers as modulo-10000 long int data.

- Int32 writeMultipleMod10000 (Int32 slaveAddr, Int32 startRef, Int32[ ] int32Arr)

  Modbus function 16 (10 hex) for 32-bit modulo-10000 long int data types, Preset Multiple Registers/Write Multiple Registers with modulo-10000 long int data.

- Int32 writeMultipleMod10000 (Int32 slaveAddr, Int32 startRef, Int32[ ] int32Arr, Int32 numRegs)

  Modbus function 16 (10 hex) for 32-bit modulo-10000 long int data types, Preset Multiple Registers/Write Multiple Registers with modulo-10000 long int data.

Diagnostics

- Int32 readExceptionStatus (Int32 slaveAddr, out byte statusByte)

  Modbus function 7 (07 hex), Read Exception Status.

- Int32 returnQueryData (Int32 slaveAddr, byte[ ] queryArr, out byte[ ] echoArr)

  Modbus function code 8, sub-function 00, Return Query Data.

- Int32 restartCommunicationsOption (Int32 slaveAddr, Int32 clearEventLog)

  Modbus function code 8, sub-function 01, Restart Communications Option

User Defined Function Codes

- Int32 customFunction (Int32 slaveAddr, Int32 functionCode, byte[ ] requestData, [In, Out] byte[ ] responseData)

  User Defined Function Code

Protocol Configuration

- Int32 setTimeout (Int32 timeOut)

  Configures time-out
- Int32 getTimeout ()

  Returns the current time-out setting
- Int32 setPollDelay (Int32 pollDelay)

  Poll delay property
- Int32 getPollDelay ()

  Returns the poll delay time
- Int32 setRetryCnt (Int32 retryCnt)

  Configures the automatic retry setting
- Int32 getRetryCnt ()

  Returns the automatic retry count

Transmission Statistic Functions

- Int32 getTotalCounter ()

  Returns how often a message transfer has been executed
- void resetTotalCounter ()

  Resets total message transfer counter
- Int32 getSuccessCounter ()

  Returns how often a message transfer was successful
- void resetSuccessCounter ()

  Resets successful message transfer counter

Slave Configuration

- void configureStandard32BitMode ()

  Configures all slaves for Standard 32-bit Mode.
- Int32 configureStandard32BitMode (Int32 slaveAddr)

  Configures a slave for Standard 32-bit Mode.
- void configureEnron32BitMode ()

  Configures all slaves for Daniel/ENRON 32-bit Mode.
- Int32 configureEnron32BitMode (Int32 slaveAddr)

  Configures a slave for Daniel/ENRON 32-bit Mode.
- void configureCountFromOne ()

Configures the reference counting scheme to start with one for all slaves.

- Int32 configureCountFromOne (Int32 slaveAddr)

  Configures the reference counting scheme to start with one for a slave.
- void configureCountFromZero ()

  Configures the reference counting scheme to start with zero for all slaves.
- void configureBigEndianInts ()

  Configures 32-bit int data type functions to do a word swap
- Int32 configureBigEndianInts (Int32 slaveAddr)

  Configures 32-bit int data type functions to do a word swap on a per slave basis
- void configureSwappedFloats ()

  Configures float data type functions to do a word swap
- Int32 configureSwappedFloats (Int32 slaveAddr)

  Configures float data type functions to do a word swap on a per slave basis
- void configureLittleEndianInts ()

  Configures 32-bit int data type functions NOT to do a word swap
- Int32 configureLittleEndianInts (Int32 slaveAddr)

  Configures 32-bit int data type functions NOT to do a word swap on a per slave basis
- void configureIeeeFloats ()

  Configures float data type functions NOT to do a word swap
- Int32 configureIeeeFloats (Int32 slaveAddr)

  Configures float data type functions NOT to do a word swap on a per slave basis
- Int32 onfigureCountFromZero (Int32 slaveAddr)

  Configures the reference counting scheme to start with zero for a slave.

5.7.1 Detailed Description

This class realises a Modbus client using MODBUS over UDP protocol variant. It provides functions to establish a UDP connection to the slave as well as data and control functions which can be used after a connection to a slave device has been established successfully. The data and control functions are organized into different conformance classes.

It is possible to instantiate multiple instances of this class for establishing multiple connections to either the same or different hosts.

5.7.2 Constructor & Destructor Documentation

MbusUdpMasterProtocol() `MbusUdpMasterProtocol ( ) [inline]`

Exceptions

OutOfMemoryException	Creation of class failed

5.7.3 Member Function Documentation

setPort() `override Int32 setPort ( Int16 portNo ) [inline], [virtual]`

Usually the port number remains unchanged and defaults to 502 for Modbus/TCP and 1100 for RTU over TCP. However if the port number has to be different this function must be called before opening the connection with openProtocol().

Note

A protocol must be closed in order to configure it.

Parameters

portNo	Port number to be used when opening the connection

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

Reimplemented from MbusIpClientBase.

openProtocol() `[1/2] Int32 openProtocol ( ) [inline], [inherited]`

This function establishes a logical network connection between master and slave. After a connection has been established data and control functions can be used. A TCP/IP connection should be closed if it is no longer needed.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

openProtocol() `[2/2] Int32 openProtocol ( string hostName ) [inline], [inherited]`

This function establishes a logical network connection between master and slave. After a connection has been established data and control functions can be used. A TCP/IP connection should be closed if it is no longer needed.

Parameters

hostName	String with IP address or host name (eg "127.0.0.1")

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

getPort() `Int16 getPort ( ) [inline], [inherited]`

Returns

Currently set port number

readCoils() `[1/2] Int32 readCoils (`
`Int32 slaveAddr,`
`Int32 startRef,`
`[In,Out] bool [] bitArr ) [inline], [inherited]`

Reads the contents of the discrete outputs (coils, 0:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read. The length of the array determines how many coils are read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readCoils() `[2/2] Int32 readCoils (`
`Int32 slaveAddr,`

```
Int32 startRef,
[In,Out] bool [] bitArr,
Int32 numCoils )  [inline], [inherited]
```

Reads the contents of the discrete outputs (coils, 0:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read.
numCoils	Number of coils to be read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputDiscretes() `[1/2]`

```
Int32 readInputDiscretes (
Int32 slaveAddr,
Int32 startRef,
[In,Out] bool [] bitArr )  [inline], [inherited]
```

Reads the contents of the discrete inputs (input status, 1:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read. The length of the array determines how many inputs are read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputDiscretes() [2/2]

```
Int32 readInputDiscretes (
        Int32 slaveAddr,
        Int32 startRef,
        [In,Out] bool [] bitArr,
        Int32 numDiscretes ) [inline], [inherited]
```

Reads the contents of the discrete inputs (input status, 1:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which will contain the data read.
num↩ Discretes	Number of inputs to be read (Range: 1-2000).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeCoil()

```
Int32 writeCoil (
        Int32 slaveAddr,
        Int32 bitAddr,
        bool bitVal ) [inline], [inherited]
```

Sets a single discrete output variable (coil, 0:00000 table) to either ON or OFF.

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
bitAddr	Coil address (Range: 1 - 65536)
bitVal	true sets, false clears discrete output variable

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

forceMultipleCoils() `[1/2]`

```
Int32 forceMultipleCoils (
    Int32 slaveAddr,
    Int32 startRef,
    bool [] bitArr )  [inline], [inherited]
```

Writes binary values into a sequence of discrete outputs (coils, 0:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which contains the data to be sent. The length of the array determines how many coils are written (Range: 1-1968).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

forceMultipleCoils() `[2/2]`

```
Int32 forceMultipleCoils (
    Int32 slaveAddr,
    Int32 startRef,
    bool [] bitArr,
    Int32 numCoils )  [inline], [inherited]
```

Writes binary values into a sequence of discrete outputs (coils, 0:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
bitArr	Buffer which contains the data to be sent.
numCoils	Number of coils to be written (Range: 1-1968).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleRegisters() `[1/2]` `Int32 readMultipleRegisters (`
`Int32 slaveAddr,`
`Int32 startRef,`
`[In,Out] System.Array regArr ) [inline], [inherited]`

Reads the contents of the output registers (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[]. The length of the array determines how many registers are read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleRegisters() `[2/2]` `Int32 readMultipleRegisters (`
`Int32 slaveAddr,`
`Int32 startRef,`
`[In,Out] System.Array regArr,`
`Int32 numRegs ) [inline], [inherited]`

Reads the contents of the output registers (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)

Parameters

startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[].
numRegs	Number of values to be read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputRegisters() `[1/2]`

```
Int32 readInputRegisters (
    Int32 slaveAddr,
    Int32 startRef,
    [In, Out] System.Array regArr )  [inline], [inherited]
```

Read the contents of the input registers (3:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[]. The length of the array determines how many registers are read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputRegisters() `[2/2]`

```
Int32 readInputRegisters (
    Int32 slaveAddr,
    Int32 startRef,
    [In, Out] System.Array regArr,
    Int32 numRegs )  [inline], [inherited]
```

Read the contents of the input registers (3:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array which will be filled with the data read. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[].
numRegs	Number of values to be read (Range: 1-125 for 16-bit, 1-62 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeSingleRegister() `[1/2]` `Int32 writeSingleRegister (`
`Int32 slaveAddr,`
`Int32 regAddr,`
`Int16 regVal )` `[inline], [inherited]`

Writes a value into a single output register (holding register, 4:00000 reference).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
regAddr	Register address (Range: 1 - 65536)
regVal	Data to be sent as signed 16-bit value

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeSingleRegister() `[2/2]` `Int32 writeSingleRegister (`

```
Int32 slaveAddr,
Int32 regAddr,
UInt16 regVal )  [inline], [inherited]
```

Writes a value into a single output register (holding register, 4:00000 reference).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
regAddr	Register address (Range: 1 - 65536)
regVal	Data to be sent as unsigned 16-bit value

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleRegisters() `[1/2]  Int32 writeMultipleRegisters (`

```
Int32 slaveAddr,
Int32 startRef,
System.Array regArr )  [inline], [inherited]
```

Writes values into a sequence of output registers (holding registers, 4:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array with data to be sent. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[]. The length of the array determines how many registers are written (Range: 1-123 for 16-bit, 1-61 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleRegisters() `[2/2] Int32 writeMultipleRegisters (`
`Int32 slaveAddr,`
`Int32 startRef,`
`System.Array regArr,`
`Int32 numRegs ) [inline], [inherited]`

Writes values into a sequence of output registers (holding registers, 4:00000 table).

Note

Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Array with data to be sent. Accepted array types are Int16[], UInt16[], Int32[], UInt32[] and float[].
numRegs	Number of values to be written (Range: 1-123 for 16-bit, 1-61 for 32-bit and floats).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

maskWriteRegister() `Int32 maskWriteRegister (`
`Int32 slaveAddr,`
`Int32 regAddr,`
`Int16 andMask,`
`Int16 orMask ) [inline], [inherited]`

Masks bits according to an AND and an OR mask into a single output register (holding register, 4:00000 reference). Masking is done as follows: retVal = (currentVal AND andMask) OR (orMask AND (NOT andMask))

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
regAddr	Register address (Range: 1 - 65536)
andMask	Mask to be applied as a logic AND to the register
orMask	Mask to be applied as a logic OR to the register

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readWriteRegisters() `[1/2]`

```
Int32 readWriteRegisters (
    Int32 slaveAddr,
    Int32 readRef,
    [In,Out] Int16 [] readArr,
    Int32 writeRef,
    Int16 [] writeArr )  [inline], [inherited]
```

Combines reading and writing of the output registers in one transaction (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
readRef	Start registers for reading (Range: 1 - 65536)
readArr	Buffer which will contain the data read. The length of the array determines how many registers are read (Range: 1-125).
writeRef	Start registers for writing (Range: 1 - 65536)
writeArr	Buffer with data to be sent. The length of the array determines how many registers are written (Range: 1-121).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readWriteRegisters() `[2/2]`

```
Int32 readWriteRegisters (
    Int32 slaveAddr,
```

```
Int32 readRef,
[In,Out] Int16 [] readArr,
Int32 readCnt,
Int32 writeRef,
Int16 [] writeArr,
Int32 writeCnt )  [inline], [inherited]
```

Combines reading and writing of the output registers in one transaction (holding registers, 4:00000 table).

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
readRef	Start register for reading (Range: 1 - 65536)
readArr	Buffer which will contain the data read.
writeRef	Start register for writing (Range: 1 - 65536)
writeArr	Buffer with data to be sent.
readCnt	Number of registers to be read (Range: 1-125).
writeCnt	Number of registers to be written (Range: 1-121).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleMod10000() `[1/2]`

```
Int32 readMultipleMod10000 (
Int32 slaveAddr,
Int32 startRef,
[In,Out] Int32 [] int32Arr )  [inline], [inherited]
```

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value into 32-bit int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readMultipleMod10000() `[2/2]`

```
Int32 readMultipleMod10000 (
    Int32 slaveAddr,
    Int32 startRef,
    [In,Out] Int32 [] int32Arr,
    Int32 numRegs )  [inline], [inherited]
```

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value into 32-bit int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read.
numRegs	Number of values to be read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputMod10000() `[1/2]`

```
Int32 readInputMod10000 (
```

```
Int32 slaveAddr,
Int32 startRef,
[In,Out] Int32 [] int32Arr )  [inline], [inherited]
```

Reads the contents of pairs of consecutive input registers (3:00000 table) representing a modulo-10000 long int value into 32-bit long int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readInputMod10000() `[2/2]`

```
Int32 readInputMod10000 (
Int32 slaveAddr,
Int32 startRef,
[In,Out] Int32 [] int32Arr,
Int32 numRegs )  [inline], [inherited]
```

Reads the contents of pairs of consecutive input registers (3:00000 table) representing a modulo-10000 long int value into 32-bit long int values and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer which will be filled with the data read.
numRegs	Number of values to be read (Range: 1-62).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleMod10000() `[1/2] Int32 writeMultipleMod10000 (`
`Int32 slaveAddr,`
`Int32 startRef,`
`Int32 [] int32Arr ) [inline], [inherited]`

Writes long int values into pairs of output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function.
Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer with the data to be sent. The length of the array determines how many values are written (Range: 1-61).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

writeMultipleMod10000() `[2/2] Int32 writeMultipleMod10000 (`
`Int32 slaveAddr,`

```
Int32 startRef,
Int32 [] int32Arr,
Int32 numRegs )  [inline], [inherited]
```

Writes long int values into pairs of output registers (holding registers, 4:00000 table) representing a modulo-10000 long int value and performs number format conversion.

Note

Modbus does not know about any other data type than discretes and 16-bit registers. Because a modulo-10000 value is of 32-bit length, it will be transferred as two consecutive 16-bit registers. This means that the amount of registers transferred with this function is twice the amount of int values passed to this function. Broadcast supported for serial protocols

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start reference (Range: 1 - 65536)
int32Arr	Buffer with the data to be sent.
numRegs	Number of values to be sent (Range: 1-61).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

readExceptionStatus()

```
Int32 readExceptionStatus (
 Int32 slaveAddr,
 out byte statusByte )  [inline], [inherited]
```

Reads the eight exception status coils within the slave device.

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
statusByte	Slave status byte. The meaning of this status byte is slave specific and varies from device to device.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

returnQueryData() `Int32 returnQueryData (`
`Int32 slaveAddr,`
`byte [] queryArr,`
`out byte [] echoArr ) [inline], [inherited]`

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255 for serial, 0 - 255 for TCP)
queryArr	Buffer with data to be sent. The length of the array determines how many bytes are sent and returned
echoArr	Buffer which will contain the data read

Returns

BusProtocolErrors.FTALK_SUCCESS on success, FTALK_INVALID_REPLY_ERROR if reply does not match query data or error code. See BusProtocolErrors for more error codes.

References BusProtocolErrors.FTALK_INVALID_REPLY_ERROR, and BusProtocolErrors.FTALK_SUCCESS.

restartCommunicationsOption() `Int32 restartCommunicationsOption (`
`Int32 slaveAddr,`
`Int32 clearEventLog ) [inline], [inherited]`

Note

No broadcast supported

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255 for serial, 0 - 255 for TCP)
clearEventLog	Flag when set to one clears in addition the slave's communication even log.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

customFunction() `Int32 customFunction (`
`Int32 slaveAddr,`
`Int32 functionCode,`
`byte [] requestData,`
`[In,Out] byte [] responseData )  [inline], [inherited]`

This method can be used to implement User Defined Function Codes. The caller has only to pass the user data to this function. The assembly of the Modbus frame (the so called ADU) including checksums, slave address and function code and subsequentially the transmission, is taken care of by this method.

The modbus specification reserves function codes 65-72 and 100-110 for user defined functions.

Note

Modbus functions usually have an implied response length and therefore the number of bytes expected to be received is known at the time when sending the request. In case of a custom Modbus function with an open or unknown response length, this function can not be used.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255 for serial, 0 - 255 for TCP)
functionCode	Custom function code to be used for Modbus transaction (Range: 1-127)
requestData	Array with data to be sent as request (not including slave address or function code). The length of the array determines how many request bytes are sent (Range: 0-252).
responseData	Buffer which will be filled with the response data received. The length of the array determines how many bytes are read (Range: 0-252).

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

setTimeout() `Int32 setTimeout (`
`Int32 timeOut )  [inline], [inherited]`

Configures operation or socket time-out.

The time-out value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Parameters

timeOut	Timeout value in ms (Range: 1 - 100000)

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

getTimeout()

```
Int32 getTimeout ( )  [inline], [inherited]
```

Returns

Timeout value in ms

setPollDelay()

```
Int32 setPollDelay (
        Int32 pollDelay )  [inline], [inherited]
```

This function sets the delay time which applies between two consecutive Modbus read/write. A value of 0 disables the poll delay.

The delay value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Parameters

pollDelay	Delay value in ms (Range: 0 - 100000), 0 disables poll delay

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

getPollDelay() `Int32 getPollDelay ( ) [inline], [inherited]`

Returns

Delay time in ms, 0 if poll delay is switched off

setRetryCnt() `Int32 setRetryCnt ( Int32 retryCnt ) [inline], [inherited]`

Configures the automatic retry setting. A value of 0 disables any automatic retries.

Note

A protocol must be closed in order to configure it.

Parameters

retryCnt	Retry count (Range: 0 - 10), 0 disables retries

Returns

BusProtocolErrors.FTALK_SUCCESS on success or error code. See BusProtocolErrors for possible error codes.

getRetryCnt() `Int32 getRetryCnt ( ) [inline], [inherited]`

Returns

Retry count

getTotalCounter() `Int32 getTotalCounter ( ) [inline], [inherited]`

Returns

Counter value

resetTotalCounter() `void resetTotalCounter ( ) [inline], [inherited]`

getSuccessCounter() `Int32 getSuccessCounter ( ) [inline], [inherited]`

Returns

Counter value

resetSuccessCounter() `void resetSuccessCounter ( ) [inline], [inherited]`

configureStandard32BitMode() `[1/2] void configureStandard32BitMode ( ) [inline], [inherited]`

In Standard 32-bit Register Mode a 32-bit value is transmitted as two consecutive 16-bit Modbus registers.

Note

This function call also re-configures the endianess to default little-endian for 32-bit values!

Note

This is the default mode.

configureStandard32BitMode() `[2/2] Int32 configureStandard32BitMode ( Int32 slaveAddr ) [inline], [inherited]`

In Standard 32-bit Register Mode a 32-bit value is transmitted as two consecutive 16-bit Modbus registers.

Note

This function call also re-configures the endianess to default little-endian for 32-bit values!

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureEnron32BitMode() `[1/2]` `void configureEnron32BitMode ( ) [inline], [inherited]`

Some Modbus flavours like the Daniel/Enron protocol represent a 32-bit value using one 32-bit Modbus register instead of two 16-bit registers.

Note

This function call also re-configures the endianess to big-endian for 32-bit values as defined by the Daniel/ENRON protocol!

configureEnron32BitMode() `[2/2]` `Int32 configureEnron32BitMode ( Int32 slaveAddr ) [inline], [inherited]`

Some Modbus flavours like the Daniel/Enron protocol represent a 32-bit value using one 32-bit Modbus register instead of two 16-bit registers.

Note

This function call also re-configures the endianess to big-endian for 32-bit values as defined by the Daniel/ENRON protocol!

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureCountFromOne() `[1/2]` `void configureCountFromOne ( ) [inline], [inherited]`

This renders the reference range to be 1 to 65536 (0x10000) and register #0 is an illegal register.

Note

This is the default mode.

configureCountFromOne() `[2/2]` `Int32 configureCountFromOne ( Int32 slaveAddr ) [inline], [inherited]`

This renders the reference range to be 1 to 65536 (0x10000) and register #0 is an illegal register.

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureCountFromZero() `void configureCountFromZero ( ) [inline], [inherited]`

This renders the valid reference range to be 0 to 65535 (0xFFFF).

This renders the first register to be #0 for all slaves.

configureBigEndianInts() `[1/2]` `void configureBigEndianInts ( ) [inline], [inherited]`

Modbus is using little-endian word order for 32-bit values. The data transfer functions operating upon 32-bit int data types can be configured to do a word swap which enables them to read 32-bit data correctly from a big-endian slave.

configureBigEndianInts() `[2/2]` `Int32 configureBigEndianInts ( Int32 slaveAddr ) [inline], [inherited]`

Modbus is using little-endian word order for 32-bit values. The data transfer functions operating upon 32-bit int data types can be configured to do a word swap which enables them to read 32-bit data correctly from a big-endian slave.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureSwappedFloats() `[1/2]` `void configureSwappedFloats ( ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Most platforms store floats in IEEE 754 little-endian order which does not need a word swap.

configureSwappedFloats() `[2/2]` `Int32 configureSwappedFloats ( Int32 slaveAddr ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Most platforms store floats in IEEE 754 little-endian order which does not need a word swap.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureLittleEndianInts() `[1/2]` `void configureLittleEndianInts ( ) [inline], [inherited]`

Note

This is the default mode.

configureLittleEndianInts() `[2/2] Int32 configureLittleEndianInts ( Int32 slaveAddr ) [inline], [inherited]`

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

configureIeeeFloats() `[1/2] void configureIeeeFloats ( ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Note

This is the default mode.

configureIeeeFloats() `[2/2] Int32 configureIeeeFloats ( Int32 slaveAddr ) [inline], [inherited]`

The data functions operating upon 32-bit float data types can be configured to do a word swap.

Note

This is the default mode.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

onfigureCountFromZero() `Int32 onfigureCountFromZero ( Int32 slaveAddr ) [inline], [inherited]`

This renders the valid reference range to be 0 to 65535 (0xFFFF).

This renders the first register to be #0 for all slaves.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255) A value of zero configures the behaviour for broadcasting.

Returns

BusProtocolErrors.FTALK_SUCCESS on success or BusProtocolErrors.FTALK_ILLEGAL_ARGUMENT if argument out of range

isOpen() `bool isOpen ( ) [inline], [inherited]`

Returns

True = open, False = closed

closeProtocol() `void closeProtocol ( ) [inline], [inherited]`

getPackageVersion() `static string getPackageVersion ( ) [inline], [static], [inherited]`

Returns

Package version string

5.7.4 Property Documentation

port `override Int16 port [get], [set]`

Usually the port number remains unchanged and defaults to 502. However if the port number has to be different from 502 this property must be set before opening the connection with openProtocol().

Note

A protocol must be closed in order to configure it.

TCP Port number of slave device. Default value is 502.

hostName `string hostName [get], [set], [inherited]`

A protocol must be closed in order to configure it.

String with IP address or host name (eg "127.0.0.1")

timeout `Int32 timeout [get], [set], [inherited]`

Configures operation or socket time-out.

The time-out value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Timeout value in ms (Range: 1 - 100000)

pollDelay `Int32 pollDelay [get], [set], [inherited]`

This property sets the delay time which applies between two consecutive Modbus read/write. A value of 0 disables the poll delay.

The delay value is indicative only and not guaranteed to be maintained. How precise it is followed depends on the operating system used, it's scheduling priority and it's system timer resolution.

Note

A protocol must be closed in order to configure it.

Delay value in ms (Range: 0 - 100000), 0 disables poll delay

retryCnt `Int32 retryCnt [get], [set], [inherited]`

Configures the automatic retry setting. A value of 0 disables any automatic retries.

Note

A protocol must be closed in order to configure it.

Retry count (Range: 0 - 10), 0 disables retries

5.8 BusProtocolErrors Class Reference

Protocol Errors and Modbus exceptions codes

Static Public Member Functions

- static string getBusProtocolErrorText (Int32 errCode)
  Returns string description of an error code

Public Attributes

- const Int32 FTALK_SUCCESS = 0
  Operation was successful
- const Int32 FTALK_ILLEGAL_ARGUMENT_ERROR = 1
  Illegal argument error
- const Int32 FTALK_ILLEGAL_STATE_ERROR = 2
  Illegal state error
- const Int32 FTALK_EVALUATION_EXPIRED = 3
  Evaluation expired
- const Int32 FTALK_IO_ERROR_CLASS = 0x40
  IO error class
- const Int32 FTALK_IO_ERROR = (FTALK_IO_ERROR_CLASS | 1)
  IO error
- const Int32 FTALK_OPEN_ERR = (FTALK_IO_ERROR_CLASS | 2)
  Port or socket open error
- const Int32 FTALK_PORT_ALREADY_OPEN = (FTALK_IO_ERROR_CLASS | 3)
  Serial port already open
- const Int32 FTALK_TCPIP_CONNECT_ERR = (FTALK_IO_ERROR_CLASS | 4)
  TCPIP connection error
- const Int32 FTALK_CONNECTION_WAS_CLOSED = (FTALK_IO_ERROR_CLASS | 5)
  Remote peer closed TCPIP connection
- const Int32 FTALK_SOCKET_LIB_ERROR = (FTALK_IO_ERROR_CLASS | 6)
  Socket library error
- const Int32 FTALK_PORT_ALREADY_BOUND = (FTALK_IO_ERROR_CLASS | 7)
  TCP port already bound
- const Int32 FTALK_LISTEN_FAILED = (FTALK_IO_ERROR_CLASS | 8)
  Listen failed
- const Int32 FTALK_FILEDES_EXCEEDED = (FTALK_IO_ERROR_CLASS | 9)
  File descriptors exceeded
- const Int32 FTALK_PORT_NO_ACCESS = (FTALK_IO_ERROR_CLASS | 10)
  No permission to access serial port or TCP port
- const Int32 FTALK_PORT_NOT_AVAIL = (FTALK_IO_ERROR_CLASS | 11)
  TCP port not available
- const Int32 FTALK_BUS_PROTOCOL_ERROR_CLASS = 0x80
  Fieldbus protocol error class
- const Int32 FTALK_CHECKSUM_ERROR = (FTALK_BUS_PROTOCOL_ERROR_CLASS | 1)
  Checksum error

- const Int32 FTALK_INVALID_FRAME_ERROR = (FTALK_BUS_PROTOCOL_ERROR_CLASS | 2)

  Invalid frame error
- const Int32 FTALK_INVALID_REPLY_ERROR = (FTALK_BUS_PROTOCOL_ERROR_CLASS | 3)

  Invalid reply error
- const Int32 FTALK_REPLY_TIMEOUT_ERROR = (FTALK_BUS_PROTOCOL_ERROR_CLASS | 4)

  Reply time-out
- const Int32 FTALK_SEND_TIMEOUT_ERROR = (FTALK_BUS_PROTOCOL_ERROR_CLASS | 5)

  Send time-out
- const Int32 FTALK_MBUS_EXCEPTION_RESPONSE = (FTALK_BUS_PROTOCOL_ERROR_CLASS | 0x20)

  Modbus exception response
- const Int32 FTALK_MBUS_ILLEGAL_FUNCTION_RESPONSE = (FTALK_MBUS_EXCEPTION_RESPONSE | 1)

  Illegal Function exception response
- const Int32 FTALK_MBUS_ILLEGAL_ESS_RESPONSE = (FTALK_MBUS_EXCEPTION_RESPONSE | 2)

  Illegal Data address exception response
- const Int32 FTALK_MBUS_ILLEGAL_VALUE_RESPONSE = (FTALK_MBUS_EXCEPTION_RESPONSE | 3)

  Illegal Data Value exception response
- const Int32 FTALK_MBUS_SLAVE_FAILURE_RESPONSE = (FTALK_MBUS_EXCEPTION_RESPONSE | 4)

  Slave Device Failure exception response

5.8.1 Detailed Description

Definition of error codes returned by the FieldTalk functions. The error code can be converted to a UNICODE error text using the function BusProtocolErrors.getBusProtocolErrorText.

5.8.2 Member Function Documentation

getBusProtocolErrorText() `static string getBusProtocolErrorText ( Int32 errCode ) [inline], [static]`

Parameters

errCode	FieldTalk error code

Returns

Error text string

5.8.3 Member Data Documentation

FTALK_SUCCESS `const Int32 FTALK_SUCCESS = 0`

This return codes indicates no error.

FTALK_ILLEGAL_ARGUMENT_ERROR `const Int32 FTALK_ILLEGAL_ARGUMENT_ERROR = 1`

A parameter passed to the function returning this error code is invalid or out of range.

FTALK_ILLEGAL_STATE_ERROR `const Int32 FTALK_ILLEGAL_STATE_ERROR = 2`

The function is called in a wrong state. This return code is returned by all functions if the protocol has not been opened succesfully yet.

FTALK_EVALUATION_EXPIRED `const Int32 FTALK_EVALUATION_EXPIRED = 3`

This version of the library is a function limited evaluation version and has now expired.

FTALK_IO_ERROR_CLASS `const Int32 FTALK_IO_ERROR_CLASS = 0x40`

Errors of this class signal a problem in conjunction with the IO system.

FTALK_IO_ERROR `const Int32 FTALK_IO_ERROR = (FTALK_IO_ERROR_CLASS | 1)`

The underlaying IO system reported an error.

FTALK_OPEN_ERR `const Int32 FTALK_OPEN_ERR = (FTALK_IO_ERROR_CLASS | 2)`

The TCPIP socket or the serial port could not be opened. In case of a serial port it indicates that the serial port does not exist on the system.

FTALK_PORT_ALREADY_OPEN `const Int32 FTALK_PORT_ALREADY_OPEN = (FTALK_IO_ERROR_CLASS | 3)`

The serial port defined for the open operation is already opened by another application.

FTALK_TCPIP_CONNECT_ERR `const Int32 FTALK_TCPIP_CONNECT_ERR = (FTALK_IO_ERROR_CLASS | 4)`

Signals that the TCPIP connection could not be established. Typically this error occurs when a host does not exist on the network or the IP address or host name is wrong. The remote host must also listen on the appropriate port.

FTALK_CONNECTION_WAS_CLOSED `const Int32 FTALK_CONNECTION_WAS_CLOSED = (FTALK_IO_ERROR_CLASS | 5)`

Signals that the TCPIP connection was closed by the remote peer or is broken.

FTALK_SOCKET_LIB_ERROR `const Int32 FTALK_SOCKET_LIB_ERROR = (FTALK_IO_ERROR_CLASS | 6)`

The TCPIP socket library (eg WINSOCK) could not be loaded or the DLL is missing or not installed.

FTALK_PORT_ALREADY_BOUND `const Int32 FTALK_PORT_ALREADY_BOUND = (FTALK_IO_ERROR_CLASS | 7)`

Indicates that the specified TCP port cannot be bound. The port might already be taken by another application or hasn't been released yet by the TCPIP stack for re-use.

FTALK_LISTEN_FAILED `const Int32 FTALK_LISTEN_FAILED = (FTALK_IO_ERROR_CLASS | 8)`

The listen operation on the specified TCP port failed..

FTALK_FILEDES_EXCEEDED `const Int32 FTALK_FILEDES_EXCEEDED = (FTALK_IO_ERROR_CLASS | 9)`

Maximum number of usable file descriptors exceeded.

FTALK_PORT_NO_ACCESS `const Int32 FTALK_PORT_NO_ACCESS = (FTALK_IO_ERROR_CLASS | 10)`

You don't have permission to access the serial port or TCP port. Run the program as root. If the error is related to a serial port, change the access privilege. If it is related to TCPIP use TCP port number which is outside the IPPORT_RESERVED range.

FTALK_PORT_NOT_AVAIL `const Int32 FTALK_PORT_NOT_AVAIL = (FTALK_IO_ERROR_CLASS | 11)`

The specified TCP port is not available on this machine.

FTALK_BUS_PROTOCOL_ERROR_CLASS `const Int32 FTALK_BUS_PROTOCOL_ERROR_CLASS = 0x80`

Signals that a fieldbus protocol related error has occured. This class is the general class of errors produced by failed or interrupted data transfer functions. It is also produced when receiving invalid frames or exception responses.

FTALK_CHECKSUM_ERROR `const Int32 FTALK_CHECKSUM_ERROR = (FTALK_BUS_PROTOCOL_ERROR_CLASS | 1)`

Signals that the checksum of a received frame is invalid. A poor data link typically causes this error.

FTALK_INVALID_FRAME_ERROR `const Int32 FTALK_INVALID_FRAME_ERROR = (FTALK_BUS_PROTOCOL_ERROR_CLASS | 2)`

Signals that a received frame does not correspond either by structure or content to the specification or does not match a previously sent query frame. A poor data link typically causes this error.

FTALK_INVALID_REPLY_ERROR `const Int32 FTALK_INVALID_REPLY_ERROR = (FTALK_BUS_PROTOCOL_ERROR_CLASS | 3)`

Signals that a received reply does not correspond to the specification.

FTALK_REPLY_TIMEOUT_ERROR `const Int32 FTALK_REPLY_TIMEOUT_ERROR = (FTALK_BUS_PROTOCOL_ERROR_CLASS | 4)`

Signals that a fieldbus data transfer timed out. This can occur if the slave device does not reply in time or does not reply at all. A wrong unit adress will also cause this error. In some occasions this exception is also produced if the characters received don't constitute a complete frame.

FTALK_SEND_TIMEOUT_ERROR `const Int32 FTALK_SEND_TIMEOUT_ERROR = (FTALK_BUS_PROTOCOL_ERROR_CLASS | 5)`

Signals that a fieldbus data send timed out. This can only occur if the handshake lines are not properly set.

FTALK_MBUS_EXCEPTION_RESPONSE `const Int32 FTALK_MBUS_EXCEPTION_RESPONSE = (FTALK_BUS_PROTOCOL_ERROR_CLASS | 0x20)`

Signals that a Modbus exception response was received. Exception responses are sent by a slave device instead of a normal response message if it received the query message correctly but cannot handle the query. This error usually occurs if a master queried an invalid or non-existing data address or if the master used a Modbus function, which is not supported by the slave device.

FTALK_MBUS_ILLEGAL_FUNCTION_RESPONSE `const Int32 FTALK_MBUS_ILLEGAL_FUNCTION_RESPONSE = (FTALK_MBUS_EXCEPTI | 1)`

Signals that an Illegal Function exception response (code 01) was received. This exception response is sent by a slave device instead of a normal response message if a master sent a Modbus function, which is not supported by the slave device.

FTALK_MBUS_ILLEGAL_ESS_RESPONSE `const Int32 FTALK_MBUS_ILLEGAL_ESS_RESPONSE = (FTALK_MBUS_EXCEPTION_RESPONSE | 2)`

Signals that an Illegal Data address exception response (code 02) was received. This exception response is sent by a slave device instead of a normal response message if a master queried an invalid or non-existing data ess.

FTALK_MBUS_ILLEGAL_VALUE_RESPONSE `const Int32 FTALK_MBUS_ILLEGAL_VALUE_RESPONSE = (FTALK_MBUS_EXCEPTION_RES | 3)`

Signals that a Illegal Value exception response was (code 03) received. This exception response is sent by a slave device instead of a normal response message if a master sent a data value, which is not an allowable value for the slave device.

FTALK_MBUS_SLAVE_FAILURE_RESPONSE `const Int32 FTALK_MBUS_SLAVE_FAILURE_RESPONSE = (FTALK_MBUS_EXCEPTION_RES | 4)`

Signals that a Slave Device Failure exception response (code 04) was received. This exception response is sent by a slave device instead of a normal response message if an unrecoverable error occured while processing the requested action. This response is also sent if the request would generate a response whose size exceeds the allowable data size.

6 License

Library License
proconX Pty Ltd, Brisbane/Australia, ACN 104 080 935

Revision 4, October 2008

Definitions
"Software" refers to the collection of files and any part hereof, including, but not limited to, source code, programs, binary executables, object files, libraries, images, and scripts, which are distributed by proconX.

"Copyright Holder" is whoever is named in the copyright or copyrights for the Software.

"You" is you, if you are thinking about using, copying or distributing this Software or parts of it.

"Distributable Components" are dynamic libraries, shared libraries, class files and similar components supplied by proconX for redistribution. They must be listed in a "README" or "DEPLOY" file included with the Software.

"Application" pertains to Your product be it an application, applet or embedded software product.

License Terms

1. In consideration of payment of the licence fee and your agreement to abide by the terms and conditions of this licence, proconX grants You the following non-exclusive rights:
   a. You may use the Software on one or more computers by a single person who uses the software personally;
   b. You may use the Software nonsimultaneously by multiple people if it is installed on a single computer;
   c. You may use the Software on a network, provided that the network is operated by the organisation who purchased the license and there is no concurrent use of the Software;
   d. You may copy the Software for archival purposes.
2. You may reproduce and distribute, in executable form only, Applications linked with static libraries supplied as part of the Software and Applications incorporating dynamic libraries, shared libraries and similar components supplied as Distributable Components without royalties provided that:
   a. You paid the license fee;
   b. the purpose of distribution is to execute the Application;
   c. the Distributable Components are not distributed or resold apart from the Application;
   d. it includes all of the original Copyright Notices and associated Disclaimers;
   e. it does not include any Software source code or part thereof.
3. If You have received this Software for the purpose of evaluation, proconX grants You a non-exclusive license to use the Software free of charge for the purpose of evaluating whether to purchase an ongoing license to use the Software. The evaluation period is limited to 30 days and does not include the right to reproduce and distribute Applications using the Software. At the end of the evaluation period, if You do not purchase a license, You must uninstall the Software from the computers or devices You installed

it on.

4. You are not required to accept this License, since You have not signed it. However, nothing else grants You permission to use or distribute the Software or its derivative works. These actions are prohibited by law if You do not accept this License. Therefore, by using or distributing the Software (or any work based on the Software), You indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or using the Software or works based on it.
5. You may not use the Software to develop products which can be used as a replacement or a directly competing product of this Software.
6. Where source code is provided as part of the Software, You may modify the source code for the purpose of improvements and defect fixes. If any modifications are made to any the source code, You will put an additional banner into the code which indicates that modifications were made by You.
7. You may not disclose the Software's software design, source code and documentation or any part thereof to any third party without the expressed written consent from proconX.
8. This License does not grant You any title, ownership rights, rights to patents, copyrights, trade secrets, trademarks, or any other rights in respect to the Software.
9. You may not use, copy, modify, sublicense, or distribute the Software except as expressly provided under this License. Any attempt otherwise to use, copy, modify, sublicense or distribute the Software is void, and will automatically terminate your rights under this License.
10. The License is not transferable without written permission from proconX.
11. proconX may create, from time to time, updated versions of the Software. Updated versions of the Software will be subject to the terms and conditions of this agreement and reference to the Software in this agreement means and includes any version update.
12. THERE IS NO WARRANTY FOR THE SOFTWARE, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING PROCONX, THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE SOFTWARE "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE SOFTWARE IS WITH YOU. SHOULD THE SOFTWARE PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
13. ANY LIABILITY OF PROCONX WILL BE LIMITED EXCLUSIVELY TO REFUND OF PURCHASE PRICE. IN ADDITION, IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL PROCONX OR ITS PRINCIPALS, SHAREHOLDERS, OFFICERS, EMPLOYEES, AFFILIATES, CONTRACTORS, SUBSIDIARIES, PARENT ORGANIZATIONS AND ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE SOFTWARE AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE SOFTWARE TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
14. IN ADDITION, IN NO EVENT DOES PROCONX AUTHORIZE YOU TO USE THIS SOFTWARE IN APPLICATIONS OR SYSTEMS WHERE IT'S FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO RESULT IN A SIGNIFICANT PHYSICAL INJURY, OR IN LOSS OF LIFE. ANY SUCH USE BY YOU IS ENTIRELY AT YOUR OWN RISK, AND YOU AGREE TO HOLD PROCONX HARMLESS FROM ANY CLAIMS OR LOSSES RELATING TO SUCH UNAUTHORIZED USE.
15. This agreement constitutes the entire agreement between proconX

and You in relation to your use of the Software. Any change will be effective only if in writing signed by proconX and you.

16. This License is governed by the laws of Queensland, Australia, excluding choice of law rules. If any part of this License is found to be in conflict with the law, that part shall be interpreted in its broadest meaning consistent with the law, and no other parts of the License shall be affected.

---

7 Support

We provide electronic support and feedback for our FieldTalk products.

Please use the Support web page at: `http://www.modbusdriver.com/support`

Your feedback is always welcome. It helps improving this product.

8 Notices

Disclaimer:

proconX Pty Ltd makes no warranty for the use of its products, other than those expressly contained in the Company's standard warranty which is detailed in the Terms and Conditions located on the Company's Website. The Company assumes no responsibility for any errors which may appear in this document, reserves the right to change devices or specifications detailed herein at any time without notice, and does not make any commitment to update the information contained herein. No licenses to patents or other intellectual property of proconX are granted by the Company in connection with the sale of proconX products, expressly or by implication. proconX products are not authorized for use as critical components in life support devices or systems.