Professional Fieldbus Connections

FieldTalk Modbus Master Library, Delphi Edition

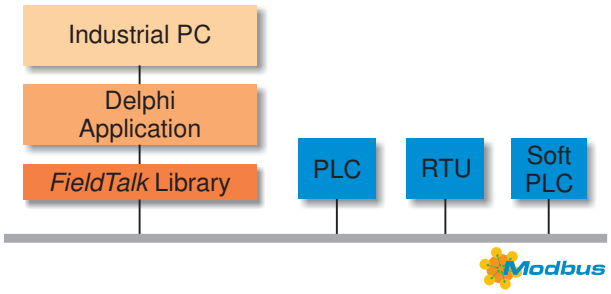
Software manual

Library version 2.11.0

Contents

1 Introduction

The *FieldTalk*™ Modbus® Master Library, Delphi Edition provides connectivity to Modbus slave compatible devices and 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).

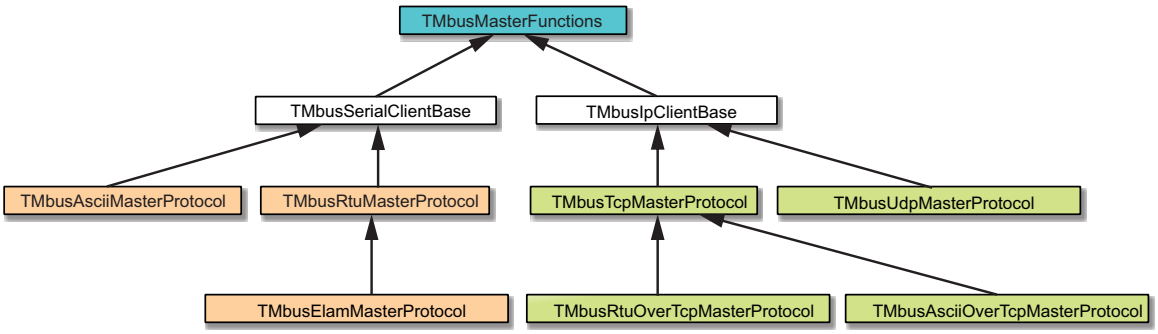
Features:

- 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 library is implemented as a VCL class library. The VCL classes wrap around a libmbus-master DLL which performs the core protocol functions. The libmbusmaster DLL is based on the FieldTalk Modbus Master C++ Library, a proven and industrial-strength Modbus driver implementation.

The library 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 TMbusMasterFunctions 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 TMbusAsciiMasterProtocol implements the Modbus ASCII protocol, the class TMbusRtuMasterProtocol implements the Modbus RTU protocol and the class TMbusTcpMasterProtocol implements the MODBUS/TCP protocol and the class TMbusRtuOverTcpMasterProtocol 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:

```
TMbusRtuMasterProtocol mbusProtocol;
```

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

```
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 How to integrate the Protocol in your Application

3.1 Using Serial Protocols

Let's assume we want to talk to a Modbus slave device with slave address 1.

The registers for reading are in the reference range 4:00100 to 4:00119 and the registers for writing are in the range 4:00200 to 4:00219. The discretes for reading are in the reference range 0:00010 to 0:00019 and the discretes for writing are in the range 0:00020 to 0:00029.

1. Import the library packages

```
uses
   MbusRtuMasterProtocol,
   BusProtocolExceptions;
```

2. Device data profile definition

Define the data sets which reflects the slave's data profile by type and size:

```
var
   readRegSet: array[1..20] of word;
   writeRegSet: array[1..10] of word;
   readBitSet: array[1..10] of boolean;
   writeBitSet: array[1..10] of boolean;
```

If you are using floats instead of 16-bit words define:

```
var
   readFloatSet: array[1..10] of single;
   writeFloatSet: array[1..10] of single;
```

If you are using 32-bit ints instead of 16-bit words define:

```
var
   readLongSet: array[1..10] of integer;
   writeLongSet: array[1..10] of integer;
```

3. Declare and instantiate a protocol object

```
var
   TMbusRtuMasterProtocol mbusProtocol;
```

4. Instantiate and open the protocol

```
   try
      mbusProtocol := TMbusRtuMasterProtocol.Create(nil);
      mbusProtocol.portName := 'COM1';
      mbusProtocol.baudRate := 19200;
      mbusProtocol.dataBits := 8;
      mbusProtocol.stopBits := 1;
      mbusProtocol.parity := 2;
      mbusProtocol.openProtocol;
```

```
except
   on e: Exception do
   begin
      writeln('Error opening protocol: ', e.message, '!');
      halt(1);
   end;
end;
```

5. Perform the data transfer functions

- To read register values:

```
mbusProtocol.readMultipleRegisters(1, 100, readRegSet);
```

- To write a single register value:

```
mbusProtocol.writeSingleRegister(1, 200, 1234);
```

- To write mutliple register values:

```
mbusProtocol.writeMultipleRegisters(1, 200, writeRegSet);
```

- To read discrete values:

```
mbusProtocol.readCoils(1, 10, readBitSet);
```

- To write a single discrete value:

```
mbusProtocol.writeCoil(1, 20, true);
```

- To write multiple discrete values:

```
mbusProtocol.forceMultipleCoils(1, 20, writeBitSet);
```

- To read float values:

```
mbusProtocol.readMultipleFloats(1, 100, readFloatSet);
```

- To read long integer values:

```
mbusProtocol.readMultipleLongInts(1, 100, readLongSet);
```

6. Close the protocol port if not needed any more

```
mbusProtocol.closeProtocol;
```

7. Error Handling

Serial protocol errors like slave device failures, transmission failures, checksum errors and time-outs throw an exception. The following code snippet can handle and report these errors:

```
try
   mbusProtocol.readMultipleRegisters(1, 100, dataSetArray);
except
   on e: EBusProtocolException do
      writeln(e.message, '!');
   on e: Exception do
   begin
      writeln('Fatal error: ', e.message, '!');
      halt(1);
   end;
end;
```

An automatic retry mechanism is available and can be enabled with mbusProtocol.set↩RetryCnt(3) before opening the protocol port.

3.2 Using MODBUS/TCP Protocol

Let's assume we want to talk to a Modbus slave device with unit address 1 and IP address 10.0.0.11.

The registers for reading are in the reference range 4:00100 to 4:00119 and the registers for writing are in the range 4:00200 to 4:00219. The discretes for reading are in the reference range 0:00010 to 0:00019 and the discretes for writing are in the range 0:00020 to 0:00029.

1. Import the library package

```
uses
   MbusTcpMasterProtocol,
   BusProtocolExceptions;
```

2. Device data profile definition

Define the data sets which reflects the slave's data profile by type and size:

```
var
   readRegSet: array[1..20] of word;
   writeRegSet: array[1..10] of word;
   readBitSet: array[1..10] of boolean;
   writeBitSet: array[1..10] of boolean;
```

If you are using floats instead of 16-bit words define:

```
var
   readFloatSet: array[1..10] of single;
   writeFloatSet: array[1..10] of single;
```

If you are using 32-bit ints instead of 16-bit words define:

```
var
   readLongSet: array[1..10] of integer;
   writeLongSet: array[1..10] of integer;
```

3. Declare and instantiate a protocol object

```
var
   TMbusTcpMasterProtocol mbusProtocol;
```

4. Open the protocol

```
mbusProtocol.hostName := ' 10.0.0.11';
mbusProtocol.openProtocol;
```

5. Perform the data transfer functions

- To read register values:

```
mbusProtocol.readMultipleRegisters(1, 100, readRegSet);
```

- To write a single register value:

```
mbusProtocol.writeSingleRegister(1, 200, 1234);
```

- To write mutliple register values:

```
mbusProtocol.writeMultipleRegisters(1, 200, writeRegSet);
```

- To read discrete values:

```
mbusProtocol.readCoils(1, 10, readBitSet);
```

- To write a single discrete value:

```
mbusProtocol.writeCoil(1, 20, true);
```

- To write multiple discrete values:

```
mbusProtocol.forceMultipleCoils(1, 20, writeBitSet);
```

- To read float values:

```
mbusProtocol.readMultipleFloats(1, 100, readFloatSet);
```

- To read long integer values:

```
mbusProtocol.readMultipleLongInts(1, 100, readLongSet);
```

6. Close the protocol port if not needed any more

```
mbusProtocol.closeProtocol;
```

7. Error Handling

TCP/IP protocol errors like slave failures, TCP/IP connection failures and time-outs return an error code. The following code snippet can handle these errors:

```
try
   mbusProtocol.readMultipleRegisters(1, 100, dataSetArray);
except
   on e: EBusProtocolException do
      writeln(e.message, '!');
   on e: Exception do
   begin
      writeln('Fatal error: ', e.message, '!');
      halt(1);
   end;
end;
```

If the method throws EConnectionWasClosed, it signals that the the TCP/IP connection was lost or closed by the remote end. Before using further data and control functions the connection has to be re-opened succesfully.

4 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.

5 Module Documentation

5.1 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).

5.2 Serial Protocols

Classes

- class TMbusRtuMasterProtocol

  Modbus RTU Master Protocol class.
- class TMbusAsciiMasterProtocol

  Modbus ASCII Master Protocol class.
- class TMbusElamMasterProtocol

  Extended Lufkin Automation Modbus Master Protocol.

5.2.1 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.

See section Using Serial Protocols for an example how to use the MbusRtuMasterProtocol class.

5.3 IP based Protocols

Classes

- class TMbusIpClientBase

  Base class for all IP based Master Protocol classes.
- class TMbusTcpMasterProtocol

  MODBUS/TCP Master Protocol class.
- class TMbusRtuOverTcpMasterProtocol

  Encapsulated Modbus RTU Master Protocol class.
- class TMbusElamOverTcpMasterProtocol

Encapsulated Modbus RTU Master Protocol class.

- class TMbusAsciiOverTcpMasterProtocol

  MODBUS ASCII over TCP Master Protocol class.

- class TMbusUdpMasterProtocol

  MODBUS/UDP Master Protocol class.

5.3.1 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.

See section Using MODBUS/TCP Protocol for an example how to use the MbusTcpMasterProtocol class.

5.4 Error Management

Classes

- class EIllegalArgumentError

  Illegal argument error.

- class EIllegalStateError

  Illegal state error.

- class EEvaluationExpired

  Evaluation expired.

- class EInvalidOperation

  EInvalidOperation is the VCL base class for invalid operations on a component.

- class EInsufficientBuffer

  Size of response buffer insufficient.

- class EOpenErr

  Port or socket open error.

- class EPortAlreadyOpen
  Serial port already open.
- class ETcpipConnectErr
  TCP/IP connection error.
- class EConnectionWasClosed
  Remote peer closed TCP/IP connection.
- class ESocketLibError
  Socket library error.
- class EPortAlreadyBound
  TCP port already bound.
- class EListenFailed
  Listen failed.
- class EFiledesExceeded
  File descriptors exceeded.
- class EPortNoAccess
  No permission to access serial port or TCP port.
- class EPortNotAvail
  TCP port not available.
- class EPortSerialLineBusy
  Serial line busy/noisy.
- class EBusProtocolException
  Communication Error.
- class EChecksumException
  Checksum error.
- class EInvalidFrameException
  Invalid frame error.
- class EInvalidReplyException
  Invalid reply error.
- class EReplyTimeoutException
  Reply time-out.
- class ESendTimeoutException
  Send time-out.
- class EInvalidMbapIdException
  Invalid MPAB indentifer.
- class ESerialLineError
  Serial line error.
- class ESerialBufferOverrun
  Serial buffer overrun.
- class EMbusResponseException
  Modbus® exception response.
- class EMbusIllegalFunctionException
  Illegal Function exception response.
- class EMbusIllegalAddressException
  Illegal Data Address exception response.

- class EMbusIllegalValueException

  Illegal Data Value exception response.
- class EMbusSlaveFailureException

  Slave Device Failure exception response.
- class EMbusGatewayPathUnavailableException

  Gateway Path Unavailable exception response.
- class EMbusGatewayTargetFailureException

  Gateway Target Device Failed exception response.

5.4.1 Detailed Description

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

5.5 Device and Vendor Specific Modbus Functions

Custom Function Codes

- customFunction (integer slaveAddr, integer functionCode, byte [ ]requestArr, byte [ ]responseArr, integer &responseLen)

  User Defined Function Code
  This method can be used to implement User Defined Function Codes.

Advantec ADAM 5000/6000 Series Commands

- adamSendReceiveAsciiCmd (string command, string &response)

  Send/Receive ADAM 5000/6000 ASCII command.

5.5.1 Detailed Description

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

5.5.2 Function Documentation

customFunction() `customFunction (`

```
integer slaveAddr,
integer functionCode,
byte [] requestArr,
byte [] responseArr,
integer & responseLen )
```

User Defined Function Code

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: 0 - 255)
functionCode	Custom function code to be used for Modbus transaction (1-127)
requestArr	Buffer with data sent as request (not including slave address or function code). The length of the array determines how many bytes sent. (Range: 0-252)
responseArr	Buffer which will be filled with the data bytes read.
responseLen	Length of response data (0-252). The number of bytes expected to be sent as response must be known when submitting the request.

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

adamSendReceiveAsciiCmd() `adamSendReceiveAsciiCmd (`
`string` *`command`*`,`
`string &` *`response`* `)`

Send/Receive ADAM 5000/6000 ASCII command.

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)

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

Note

No broadcast supported

6 Delphi Class Documentation

6.1 TMbusRtuMasterProtocol Class Reference

Modbus RTU Master Protocol class.

Public Member Functions

- TMbusRtuMasterProtocol (TComponent aOwner)

  Constructs a TMbusRtuMasterProtocol object and initialises its data.
- openProtocol ()

  Opens a serial Modbus protocol and the associated serial port with the port parameters configured via properties.
- openUseExistingConnection (integer cnxnHandle)

  Opens a serial Modbus protocol using an existing and open handle.
- enableRs485Mode (integer rtsDelay)

  Enables RS485 mode.
- boolean isOpen ()

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

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

  Returns the package version number.

Public Attributes

- string portName

  Serial port identifier property (eg 'COM1')
- longint baudRate

  Baud rate property in bps (typically 1200 - 115200, maximum value depends on UART hardware)
- integer dataBits

  Data bits property.
- integer stopBits

  Stop bits property.
- integer parity

  Parity property.

Bit Access

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

- readCoils (integer slaveAddr, integer startRef, boolean [ ]bitArr)
  Modbus function 1 (01 hex), Read Coil Status/Read Coils.
- readInputDiscretes (integer slaveAddr, integer startRef, boolean [ ]bitArr)
  Modbus function 2 (02 hex), Read Inputs Status/Read Input Discretes.
- writeCoil (integer slaveAddr, integer bitAddr, boolean bitVal)
  Modbus function 5 (05 hex), Force Single Coil/Write Coil.
- forceMultipleCoils (integer slaveAddr, integer startRef, boolean [ ]bitArr)
  Modbus function 15 (0F hex), Force Multiple Coils.

16-bit Access

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

- readMultipleRegisters (integer slaveAddr, integer startRef, word [ ]regArr)
  Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers.
- readInputRegisters (integer slaveAddr, integer startRef, word [ ]regArr)
  Modbus function 4 (04 hex), Read Input Registers.
- writeSingleRegister (integer slaveAddr, integer regAddr, word regVal)
  Modbus function 6 (06 hex), Preset Single Register/Write Single Register.
- writeMultipleRegisters (integer slaveAddr, integer startRef, word [ ]regArr)
  Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers.
- maskWriteRegister (integer slaveAddr, integer regAddr, word andMask, word orMask)
  Modbus function 22 (16 hex), Mask Write Register.
- readWriteRegisters (integer slaveAddr, integer readRef, word [ ]readArr, integer write↩ Ref, word [ ]writeArr)
  Modbus function 23 (17 hex), Read/Write Registers.

32-bit Access

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

- readMultipleLongInts (integer slaveAddr, integer startRef, integer [ ]int32Arr)
  Modbus function 3 (03 hex) for 32-bit long int data types, Read Holding Registers/Read Multiple Registers as long int data.
- readInputLongInts (integer slaveAddr, integer startRef, integer [ ]int32Arr)
  Modbus function 4 (04 hex) for 32-bit long int data types, Read Input Registers as long int data.
- writeMultipleLongInts (integer slaveAddr, integer startRef, integer [ ]int32Arr)
  Modbus function 16 (10 hex) for 32-bit long int data types, Preset Multiple Registers/Write Multiple Registers with long int data.
- readMultipleFloats (integer slaveAddr, integer startRef, single [ ]float32Arr)

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

- readInputFloats (integer slaveAddr, integer startRef, single [ ]float32Arr)

  Modbus function 4 (04 hex) for 32-bit float data types, Read Input Registers as float data.
- writeMultipleFloats (integer slaveAddr, integer startRef, single [ ]float32Arr)

  Modbus function 16 (10 hex) for 32-bit float data types, Preset Multiple Registers/Write Multiple Registers with float data.
- readMultipleMod10000 (integer slaveAddr, integer startRef, integer [ ]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.
- readInputMod10000 (integer slaveAddr, integer startRef, integer [ ]int32Arr)

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

Diagnostics

- readExceptionStatus (integer slaveAddr, byte &statusByte)

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

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

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

Custom Function Codes

- customFunction (integer slaveAddr, integer functionCode, byte [ ]requestArr, byte [ ]responseArr, integer &responseLen)

  User Defined Function Code
  This method can be used to implement User Defined Function Codes.

Protocol Configuration

- setTimeout (const integer timeOut)

  Configures time-out.
- integer getTimeout ()

  Returns the time-out value.
- setPollDelay (const integer pollDelay)

  Configures poll delay.
- integer getPollDelay ()

  Returns the poll delay time.
- setRetryCnt (const integer retryCnt)

Configures the automatic retry setting.

- integer getRetryCnt ()

  Returns the automatic retry count.
- integer timeout

  Time-out port property.
- integer pollDelay

  Poll delay property.
- integer retryCnt

  Retry count property.

Transmission Statistic Functions

- cardinal getTotalCounter ()

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

  Resets total message transfer counter.
- cardinal getSuccessCounter ()

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

  Resets successful message transfer counter.

Slave Configuration

- configureBigEndianInts ()

  Configures int data type functions to do a word swap.
- configureBigEndianInts (integer slaveAddr)

  Enables int data type functions to do a word swap on a per slave basis.
- configureSwappedFloats ()

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

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

  Configures int data type functions not to do a word swap.
- configureLittleEndianInts (integer slaveAddr)

  Disables word swapping for int data type functions on a per slave basis.
- configureIeeeFloats ()

  Configures float data type functions not to do a word swap.
- configureIeeeFloats (integer slaveAddr)

  Disables float data type functions to do a word swap on a per slave basis.
- configureStandard32BitMode ()

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

  Configures a slave for Standard 32-bit Register Mode.

- configureEnron32BitMode ()

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

  Configures all slaves for Daniel/ENRON 32-bit Mode.
- configureCountFromOne ()

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

  Configures a slave's reference counting scheme to start with one.
- configureCountFromZero ()

  Configures the reference counting scheme to start with zero for all slaves.
- configureCountFromZero (integer slaveAddr)

  Configures a slave's reference counting scheme to start with zero.

6.1.1 Detailed Description

Modbus RTU Master Protocol class.

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 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.

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).

See also

Data and Control Functions for all Modbus Protocol Flavours, Serial Protocols
MbusRtuOverTcpMasterProtocol

6.1.2 Constructor & Destructor Documentation

TMbusRtuMasterProtocol() `TMbusRtuMasterProtocol (`
`TComponent` *`aOwner`* `)`

Constructs a TMbusRtuMasterProtocol object and initialises its data.

Exceptions

EOutOfResources	Creation of class failed

6.1.3 Member Function Documentation

openProtocol() `openProtocol ( ) [inherited]`

Opens a serial Modbus protocol and the associated serial port with the port parameters configured via properties.

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

Exceptions

EInOutError	An I/O error occurred
EOpenErr	The serial port does not exist
EPortAlreadyOpen	Port is already used by somedbody else
EPortNoAccess	No permission to access serial
EIllegalArgumentError	A parameter is invalid

openUseExistingConnection() `openUseExistingConnection ( integer cnxnHandle ) [inherited]`

Opens a serial Modbus protocol using an existing and open handle.

Useful for using the protocol over a modem link.

Parameters

cnxnHandle	Win32 API handle pointing to the existing and open connection.

Exceptions

EIllegalArgumentError	A parameter is invalid

enableRs485Mode() `enableRs485Mode ( integer rtsDelay ) [inherited]`

Enables RS485 mode.

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.

Warning

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.

Remarks

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

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

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

readCoils()

```
readCoils (
            integer slaveAddr,
            integer startRef,
            boolean [] bitArr )  [inherited]
```

Modbus function 1 (01 hex), Read Coil Status/Read Coils.

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputDiscretes()

```
readInputDiscretes (
    integer slaveAddr,
    integer startRef,
    boolean [] bitArr )  [inherited]
```

Modbus function 2 (02 hex), Read Inputs Status/Read Input Discretes.

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeCoil()

```
writeCoil (
    integer slaveAddr,
```

```
        integer bitAddr,
        boolean bitVal )  [inherited]
```

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

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

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

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

forceMultipleCoils()

```
forceMultipleCoils (
    integer slaveAddr,
    integer startRef,
    boolean [] bitArr )  [inherited]
```

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

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range

Exceptions

EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readMultipleRegisters()

```
readMultipleRegisters (
integer slaveAddr,
integer startRef,
word [] regArr )  [inherited]
```

Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers.

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Buffer which will be filled with the data read. The length of the array determines how many registers are read (Range: 1-125).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputRegisters()

```
readInputRegisters (
integer slaveAddr,
integer startRef,
word [] regArr )  [inherited]
```

Modbus function 4 (04 hex), Read Input Registers.

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Buffer which will be filled with the data read. The length of the array determines how many registers are read (Range: 1-125).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeSingleRegister()

```
writeSingleRegister (
  integer slaveAddr,
  integer regAddr,
  word regVal )  [inherited]
```

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

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

Parameters

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

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

writeMultipleRegisters() `writeMultipleRegisters (`
`integer` *`slaveAddr`*`,`
`integer` *`startRef`*`,`
`word [] ` *`regArr`* `) [inherited]`

Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers.

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Buffer with the data to be sent.The length of the array determines how many registers are written (Range: 1-123).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

maskWriteRegister() `maskWriteRegister (`
`integer` *`slaveAddr`*`,`
`integer` *`regAddr`*`,`
`word` *`andMask`*`,`
`word` *`orMask`* `) [inherited]`

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

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

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

Note

No broadcast supported

readWriteRegisters()

```
readWriteRegisters (
    integer slaveAddr,
    integer readRef,
    word [] readArr,
    integer writeRef,
    word [] writeArr )  [inherited]
```

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

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readMultipleLongInts() `readMultipleLongInts ( integer slaveAddr, integer startRef, integer [] int32Arr ) [inherited]`

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

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) into 32-bit long int values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a long int 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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputLongInts() `readInputLongInts ( integer slaveAddr,`

```
integer startRef,
integer [] int32Arr )  [inherited]
```

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

Reads the contents of pairs of consecutive input registers (3:00000 table) into 32-bit long int values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a long int 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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeMultipleLongInts()

```
writeMultipleLongInts (
integer slaveAddr,
integer startRef,
integer [] int32Arr )  [inherited]
```

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

Writes long int values into pairs of output registers (holding registers, 4:00000 table).

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a long int 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.

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 read (Range: 1-61).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readMultipleFloats()

```
readMultipleFloats (
    integer slaveAddr,
    integer startRef,
    single [] float32Arr )  [inherited]
```

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

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) into float values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a float 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 float values passed to this function.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
float32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputFloats()

```
readInputFloats (
        integer slaveAddr,
        integer startRef,
        single [] float32Arr )  [inherited]
```

Modbus function 4 (04 hex) for 32-bit float data types, Read Input Registers as float data.

Reads the contents of pairs of consecutive input registers (3:00000 table) into float values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a float 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 float values passed to this function.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
float32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeMultipleFloats()

```
writeMultipleFloats (
  integer slaveAddr,
  integer startRef,
  single [] float32Arr )  [inherited]
```

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

Writes float values into pairs of output registers (holding registers, 4:00000 table).

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a float 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 float values passed to this function.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start reference (Range: 1 - 65536)
float32Arr	Buffer with the data to be sent. The length of the array determines how many values are written (Range: 1-61).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readMultipleMod10000() `readMultipleMod10000 (`
`integer` *`slaveAddr,`*
`integer` *`startRef,`*
`integer [] int32Arr )` `[inherited]`

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.

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.

Remarks

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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputMod10000() `readInputMod10000 (`
`integer` *`slaveAddr,`*

```
integer startRef,
integer [] int32Arr )  [inherited]
```

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

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.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because an 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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeMultipleMod10000() `writeMultipleMod10000 (`

```
integer slaveAddr,
integer startRef,
integer [] int32Arr )  [inherited]
```

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.

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.

Remarks

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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readExceptionStatus()

```
readExceptionStatus (
 integer slaveAddr,
 byte & statusByte )  [inherited]
```

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

Reads the eight exception status coils within the slave device.

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.

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range

Exceptions

EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

returnQueryData() `returnQueryData ( integer slaveAddr, byte [] queryArr, byte [] echoArr ) [inherited]`

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
queryArr	Data to be sent
echoArr	Buffer which will contain the data read. Array must be of the same size as queryArr.

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

restartCommunicationsOption() `restartCommunicationsOption ( integer slaveAddr, boolean clearEventLog ) [inherited]`

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

Parameters

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

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

setTimeout() `setTimeout ( const integer` *`timeOut`* `)` `[inherited]`

Configures time-out.

This function sets the operation or socket time-out to the specified value.

Remarks

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)

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

getTimeout() `integer getTimeout ( ) [inherited]`

Returns the time-out value.

Remarks

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.

Returns

Timeout value in ms

setPollDelay() `setPollDelay ( const integer pollDelay ) [inherited]`

Configures poll delay.

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

Remarks

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

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

getPollDelay() `integer getPollDelay ( ) [inherited]`

Returns the poll delay time.

Returns

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

setRetryCnt() `setRetryCnt ( const integer retryCnt ) [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

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

getRetryCnt() `integer getRetryCnt ( ) [inherited]`

Returns the automatic retry count.

Returns

Retry count

getTotalCounter() `cardinal getTotalCounter ( ) [inherited]`

Returns how often a message transfer has been executed.

Returns

Counter value

getSuccessCounter() `cardinal getSuccessCounter ( ) [inherited]`

Returns how often a message transfer was successful.

Returns

Counter value

configureBigEndianInts() `[1/2] configureBigEndianInts ( ) [inherited]`

Configures int data type functions to do a word swap.

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 machine.

configureBigEndianInts() `[2/2] configureBigEndianInts ( integer slaveAddr ) [inherited]`

Enables int data type functions to do a word swap on a per slave basis.

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 machine.

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

configureSwappedFloats() `[1/2] configureSwappedFloats ( ) [inherited]`

Configures float data type functions to do a word swap.

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

Note

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

configureSwappedFloats() `[2/2] configureSwappedFloats ( integer slaveAddr ) [inherited]`

Enables float data type functions to do a word swap on a per slave basis.

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

Note

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.

Exceptions

EIllegalArgumentError	A parameter is out of range

configureLittleEndianInts() `[1/2] configureLittleEndianInts ( ) [inherited]`

Configures int data type functions *not* to do a word swap.

This is the default.

configureLittleEndianInts() `[2/2] configureLittleEndianInts ( integer slaveAddr ) [inherited]`

Disables word swapping for int data type functions on a per slave basis.

Modbus is using little-endian word order for 32-bit values. This setting assumes that the slave also serves 32-bit data in little little-endian word order.

Remarks

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.

Exceptions

EIllegalArgumentError	A parameter is out of range

configureIeeeFloats() `[1/2]` `configureIeeeFloats ( ) [inherited]`

Configures float data type functions *not* to do a word swap.

This is the default.

configureIeeeFloats() `[2/2]` `configureIeeeFloats ( integer slaveAddr ) [inherited]`

Disables float data type functions to do a word swap on a per slave basis.

Modbus is using little-endian word order for 32-bit values. This setting assumes that the slave also serves 32-bit floats in little little-endian word order which is the most common case.

Remarks

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.

Exceptions

EIllegalArgumentError	A parameter is out of range

configureStandard32BitMode() `[1/2]` `configureStandard32BitMode ( ) [inherited]`

Configures all slaves for Standard 32-bit Mode.

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

Remarks

This is the default mode

configureStandard32BitMode() `[2/2]` `configureStandard32BitMode ( integer slaveAddr ) [inherited]`

Configures a slave for Standard 32-bit Register Mode.

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

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

configureEnron32BitMode() `[1/2]` `configureEnron32BitMode ( ) [inherited]`

Configures all slaves for Daniel/ENRON 32-bit Mode.

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.

configureEnron32BitMode() `[2/2]` `configureEnron32BitMode ( integer slaveAddr ) [inherited]`

Configures all slaves for Daniel/ENRON 32-bit Mode.

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.

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

configureCountFromOne() `[1/2]` `configureCountFromOne ( ) [inherited]`

Configures the reference counting scheme to start with one for all slaves.

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

Remarks

This is the default mode

configureCountFromOne() `[2/2] configureCountFromOne ( integer slaveAddr ) [inherited]`

Configures a slave's reference counting scheme to start with one.

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

Parameters

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

Remarks

This is the default mode

Exceptions

EIllegalArgumentError	A parameter is out of range

configureCountFromZero() `[1/2] configureCountFromZero ( ) [inherited]`

Configures the reference counting scheme to start with zero for all slaves.

This renders the valid reference range to be 0 to 65535.

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

configureCountFromZero() `[2/2] configureCountFromZero ( integer slaveAddr ) [inherited]`

Configures a slave's reference counting scheme to start with zero.

This is also known as PDU addressing.

This renders the valid reference range to be 0 to 65535.

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

isOpen() `boolean isOpen ( ) [inherited]`

Returns whether the protocol is open or not.

Return values

true	= open
false	= closed

getPackageVersion() `class string getPackageVersion ( ) [inherited]`

Returns the package version number.

Returns

Package version string

6.1.4 Member Data Documentation

portName `string portName [inherited]`

Serial port identifier property (eg 'COM1')

Note

A protocol must be closed in order to configure it.

See also

fPortName For reading
fPortName For writing

baudRate `longint baudRate [inherited]`

Baud rate property in bps (typically 1200 - 115200, maximum value depends on UART hardware)

Note

A protocol must be closed in order to configure it.

See also

fBaudRate For reading
fBaudRate For writing

dataBits `integer dataBits [inherited]`

Data bits property.

SER_DATABITS_7: 7 data bits (ASCII protocol only), SER_DATABITS_8: 8 data bits

Note

A protocol must be closed in order to configure it.

See also

fDataBits For reading
fDataBits For writing

stopBits `integer stopBits [inherited]`

Stop bits property.

SER_STOPBITS_1: 1 stop bit, SER_STOPBITS_2: 2 stop bits

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.

See also

fStopBits For reading
fStopBits For writing

parity `integer parity [inherited]`

Parity property.

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

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.

See also

fParity For reading
fParity For writing

timeout `integer timeout [inherited]`

Time-out port property.

Note

A protocol must be closed in order to configure it.

See also

getTimeout For reading
setTimeout For writing

pollDelay `integer pollDelay [inherited]`

Poll delay property.

Delay between two Modbus read/writes in ms

Note

A protocol must be closed in order to configure it.

See also

getPollDelay For reading
setPollDelay For writing

retryCnt `integer retryCnt [inherited]`

Retry count property.

Note

A protocol must be closed in order to configure it.

See also

getRetryCnt For reading
setRetryCnt For writing

6.2 TMbusAsciiMasterProtocol Class Reference

Modbus ASCII Master Protocol class.

Public Member Functions

- TMbusAsciiMasterProtocol (TComponent aOwner)

  Constructs a TMbusAsciiMasterProtocol object and initialises its data.
- openProtocol ()

  Opens a serial Modbus protocol and the associated serial port with the port parameters configured via properties.
- openUseExistingConnection (integer cnxnHandle)

  Opens a serial Modbus protocol using an existing and open handle.
- enableRs485Mode (integer rtsDelay)

  Enables RS485 mode.
- boolean isOpen ()

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

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

  Returns the package version number.

Public Attributes

- string portName

  Serial port identifier property (eg 'COM1')
- longint baudRate

  Baud rate property in bps (typically 1200 - 115200, maximum value depends on UART hardware)
- integer dataBits

  Data bits property.
- integer stopBits

  Stop bits property.
- integer parity

  Parity property.

Bit Access

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

- readCoils (integer slaveAddr, integer startRef, boolean [ ]bitArr)

  Modbus function 1 (01 hex), Read Coil Status/Read Coils.
- readInputDiscretes (integer slaveAddr, integer startRef, boolean [ ]bitArr)

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

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

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

16-bit Access

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

- readMultipleRegisters (integer slaveAddr, integer startRef, word [ ]regArr)

  Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers.
- readInputRegisters (integer slaveAddr, integer startRef, word [ ]regArr)

  Modbus function 4 (04 hex), Read Input Registers.
- writeSingleRegister (integer slaveAddr, integer regAddr, word regVal)

  Modbus function 6 (06 hex), Preset Single Register/Write Single Register.
- writeMultipleRegisters (integer slaveAddr, integer startRef, word [ ]regArr)

  Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers.
- maskWriteRegister (integer slaveAddr, integer regAddr, word andMask, word orMask)

  Modbus function 22 (16 hex), Mask Write Register.
- readWriteRegisters (integer slaveAddr, integer readRef, word [ ]readArr, integer write↩ Ref, word [ ]writeArr)

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

32-bit Access

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

- readMultipleLongInts (integer slaveAddr, integer startRef, integer [ ]int32Arr)

  Modbus function 3 (03 hex) for 32-bit long int data types, Read Holding Registers/Read Multiple Registers as long int data.
- readInputLongInts (integer slaveAddr, integer startRef, integer [ ]int32Arr)

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

  Modbus function 16 (10 hex) for 32-bit long int data types, Preset Multiple Registers/Write Multiple Registers with long int data.
- readMultipleFloats (integer slaveAddr, integer startRef, single [ ]float32Arr)

  Modbus function 3 (03 hex) for 32-bit float data types, Read Holding Registers/Read Multiple Registers as float data.
- readInputFloats (integer slaveAddr, integer startRef, single [ ]float32Arr)

  Modbus function 4 (04 hex) for 32-bit float data types, Read Input Registers as float data.
- writeMultipleFloats (integer slaveAddr, integer startRef, single [ ]float32Arr)

  Modbus function 16 (10 hex) for 32-bit float data types, Preset Multiple Registers/Write Multiple Registers with float data.
- readMultipleMod10000 (integer slaveAddr, integer startRef, integer [ ]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.
- readInputMod10000 (integer slaveAddr, integer startRef, integer [ ]int32Arr)

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

Diagnostics

- readExceptionStatus (integer slaveAddr, byte &statusByte)

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

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

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

Custom Function Codes

- customFunction (integer slaveAddr, integer functionCode, byte [ ]requestArr, byte [ ]responseArr, integer &responseLen)

  User Defined Function Code
  This method can be used to implement User Defined Function Codes.

Protocol Configuration

- setTimeout (const integer timeOut)

  Configures time-out.
- integer getTimeout ()

  Returns the time-out value.
- setPollDelay (const integer pollDelay)

  Configures poll delay.
- integer getPollDelay ()

  Returns the poll delay time.
- setRetryCnt (const integer retryCnt)

  Configures the automatic retry setting.
- integer getRetryCnt ()

  Returns the automatic retry count.
- integer timeout

  Time-out port property.
- integer pollDelay

  Poll delay property.
- integer retryCnt

  Retry count property.

Transmission Statistic Functions

- cardinal getTotalCounter ()

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

  Resets total message transfer counter.

- cardinal getSuccessCounter ()

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

  Resets successful message transfer counter.

Slave Configuration

- configureBigEndianInts ()

  Configures int data type functions to do a word swap.
- configureBigEndianInts (integer slaveAddr)

  Enables int data type functions to do a word swap on a per slave basis.
- configureSwappedFloats ()

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

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

  Configures int data type functions not to do a word swap.
- configureLittleEndianInts (integer slaveAddr)

  Disables word swapping for int data type functions on a per slave basis.
- configureIeeeFloats ()

  Configures float data type functions not to do a word swap.
- configureIeeeFloats (integer slaveAddr)

  Disables float data type functions to do a word swap on a per slave basis.
- configureStandard32BitMode ()

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

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

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

  Configures all slaves for Daniel/ENRON 32-bit Mode.
- configureCountFromOne ()

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

  Configures a slave's reference counting scheme to start with one.
- configureCountFromZero ()

  Configures the reference counting scheme to start with zero for all slaves.
- configureCountFromZero (integer slaveAddr)

  Configures a slave's reference counting scheme to start with zero.

6.2.1 Detailed Description

Modbus ASCII Master Protocol class.

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 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.

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).

See also

Data and Control Functions for all Modbus Protocol Flavours, Serial Protocols
TMbusAsciiOverTcpMasterProtocol

6.2.2 Constructor & Destructor Documentation

TMbusAsciiMasterProtocol() `TMbusAsciiMasterProtocol ( TComponent aOwner )`

Constructs a TMbusAsciiMasterProtocol object and initialises its data.

Exceptions

EOutOfResources	Creation of class failed

6.2.3 Member Function Documentation

openProtocol() `openProtocol ( ) [inherited]`

Opens a serial Modbus protocol and the associated serial port with the port parameters configured via properties.

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

Exceptions

EInOutError	An I/O error occurred
EOpenErr	The serial port does not exist
EPortAlreadyOpen	Port is already used by somedbody else
EPortNoAccess	No permission to access serial

Exceptions

EIllegalArgumentError	A parameter is invalid

openUseExistingConnection() `openUseExistingConnection ( integer cnxnHandle ) [inherited]`

Opens a serial Modbus protocol using an existing and open handle.

Useful for using the protocol over a modem link.

Parameters

cnxnHandle	Win32 API handle pointing to the existing and open connection.

Exceptions

EIllegalArgumentError	A parameter is invalid

enableRs485Mode() `enableRs485Mode ( integer rtsDelay ) [inherited]`

Enables RS485 mode.

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.

Warning

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.

Remarks

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

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

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

readCoils()

```
readCoils (
            integer slaveAddr,
            integer startRef,
            boolean [] bitArr )  [inherited]
```

Modbus function 1 (01 hex), Read Coil Status/Read Coils.

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputDiscretes() `readInputDiscretes ( integer slaveAddr, integer startRef, boolean [] bitArr ) [inherited]`

Modbus function 2 (02 hex), Read Inputs Status/Read Input Discretes.

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeCoil() `writeCoil ( integer slaveAddr, integer bitAddr, boolean bitVal ) [inherited]`

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

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

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

Exceptions

EIllegalStateError	Port or connection is closed

Exceptions

EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

forceMultipleCoils()

```
forceMultipleCoils (
    integer slaveAddr,
    integer startRef,
    boolean [] bitArr )  [inherited]
```

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

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readMultipleRegisters()

```
readMultipleRegisters (
    integer slaveAddr,
    integer startRef,
    word [] regArr )  [inherited]
```

Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers.

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Buffer which will be filled with the data read. The length of the array determines how many registers are read (Range: 1-125).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputRegisters()

```
readInputRegisters (
    integer slaveAddr,
    integer startRef,
    word [] regArr )  [inherited]
```

Modbus function 4 (04 hex), Read Input Registers.

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Buffer which will be filled with the data read. The length of the array determines how many registers are read (Range: 1-125).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range

Exceptions

EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeSingleRegister()

```
writeSingleRegister (
  integer slaveAddr,
  integer regAddr,
  word regVal )  [inherited]
```

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

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

Parameters

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

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

writeMultipleRegisters()

```
writeMultipleRegisters (
  integer slaveAddr,
  integer startRef,
  word [] regArr )  [inherited]
```

Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers.

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Buffer with the data to be sent.The length of the array determines how many registers are written (Range: 1-123).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

maskWriteRegister() `maskWriteRegister (`
`integer` *slaveAddr*`,`
`integer` *regAddr*`,`
`word` *andMask*`,`
`word` *orMask* `) [inherited]`

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

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

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

Note

No broadcast supported

readWriteRegisters() `readWriteRegisters (`

```
integer slaveAddr,
integer readRef,
word [] readArr,
integer writeRef,
word [] writeArr )  [inherited]
```

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

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readMultipleLongInts()

```
readMultipleLongInts (
integer slaveAddr,
integer startRef,
integer [] int32Arr )  [inherited]
```

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

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) into 32-bit long int values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a long int 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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputLongInts()

```
readInputLongInts (
    integer slaveAddr,
    integer startRef,
    integer [] int32Arr )  [inherited]
```

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

Reads the contents of pairs of consecutive input registers (3:00000 table) into 32-bit long int values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a long int 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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeMultipleLongInts()

```
writeMultipleLongInts (
integer slaveAddr,
integer startRef,
integer [] int32Arr )  [inherited]
```

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

Writes long int values into pairs of output registers (holding registers, 4:00000 table).

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a long int 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.

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 read (Range: 1-61).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readMultipleFloats()

```
readMultipleFloats (
    integer slaveAddr,
    integer startRef,
    single [] float32Arr )  [inherited]
```

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

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) into float values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a float 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 float values passed to this function.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
float32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputFloats()

```
readInputFloats (
        integer slaveAddr,
        integer startRef,
        single [] float32Arr )  [inherited]
```

Modbus function 4 (04 hex) for 32-bit float data types, Read Input Registers as float data.

Reads the contents of pairs of consecutive input registers (3:00000 table) into float values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a float 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 float values passed to this function.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
float32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeMultipleFloats()

```
writeMultipleFloats (
        integer slaveAddr,
        integer startRef,
        single [] float32Arr )  [inherited]
```

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

Writes float values into pairs of output registers (holding registers, 4:00000 table).

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a float 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 float values passed to this function.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start reference (Range: 1 - 65536)
float32Arr	Buffer with the data to be sent. The length of the array determines how many values are written (Range: 1-61).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readMultipleMod10000() `readMultipleMod10000 (`
`integer` *`slaveAddr,`*
`integer` *`startRef,`*
`integer [] int32Arr )  [inherited]`

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.

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.

Remarks

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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputMod10000()

```
readInputMod10000 (
 integer slaveAddr,
 integer startRef,
 integer [] int32Arr )  [inherited]
```

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

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.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because an 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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeMultipleMod10000() `writeMultipleMod10000 (`
`integer` *`slaveAddr`*`,`
`integer` *`startRef`*`,`
`integer [] int32Arr ) [inherited]`

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.

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.

Remarks

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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readExceptionStatus()

```
readExceptionStatus (
    integer slaveAddr,
    byte & statusByte )  [inherited]
```

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

Reads the eight exception status coils within the slave device.

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.

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

returnQueryData()

```
returnQueryData (
    integer slaveAddr,
    byte [] queryArr,
    byte [] echoArr )  [inherited]
```

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
queryArr	Data to be sent
echoArr	Buffer which will contain the data read. Array must be of the same size as queryArr.

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

restartCommunicationsOption() `restartCommunicationsOption (`
`integer` *`slaveAddr`*`,`
`boolean` *`clearEventLog`* `)` `[inherited]`

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

Parameters

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

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

setTimeout() `setTimeout ( const integer timeOut ) [inherited]`

Configures time-out.

This function sets the operation or socket time-out to the specified value.

Remarks

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)

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

getTimeout() `integer getTimeout ( ) [inherited]`

Returns the time-out value.

Remarks

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.

Returns

Timeout value in ms

setPollDelay() `setPollDelay ( const integer pollDelay ) [inherited]`

Configures poll delay.

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

Remarks

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

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

getPollDelay() `integer getPollDelay ( ) [inherited]`

Returns the poll delay time.

Returns

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

setRetryCnt() `setRetryCnt ( const integer retryCnt ) [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

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

getRetryCnt() `integer getRetryCnt ( ) [inherited]`

Returns the automatic retry count.

Returns

Retry count

getTotalCounter() `cardinal getTotalCounter ( ) [inherited]`

Returns how often a message transfer has been executed.

Returns

Counter value

getSuccessCounter() `cardinal getSuccessCounter ( ) [inherited]`

Returns how often a message transfer was successful.

Returns

Counter value

configureBigEndianInts() `[1/2] configureBigEndianInts ( ) [inherited]`

Configures int data type functions to do a word swap.

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 machine.

configureBigEndianInts() `[2/2] configureBigEndianInts ( integer slaveAddr ) [inherited]`

Enables int data type functions to do a word swap on a per slave basis.

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 machine.

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

configureSwappedFloats() `[1/2] configureSwappedFloats ( ) [inherited]`

Configures float data type functions to do a word swap.

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

Note

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

configureSwappedFloats() `[2/2] configureSwappedFloats ( integer slaveAddr ) [inherited]`

Enables float data type functions to do a word swap on a per slave basis.

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

Note

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.

Exceptions

EIllegalArgumentError	A parameter is out of range

configureLittleEndianInts() `[1/2]` `configureLittleEndianInts ( ) [inherited]`

Configures int data type functions *not* to do a word swap.

This is the default.

configureLittleEndianInts() `[2/2]` `configureLittleEndianInts ( integer slaveAddr ) [inherited]`

Disables word swapping for int data type functions on a per slave basis.

Modbus is using little-endian word order for 32-bit values. This setting assumes that the slave also serves 32-bit data in little little-endian word order.

Remarks

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.

Exceptions

EIllegalArgumentError	A parameter is out of range

configureIeeeFloats() `[1/2]` `configureIeeeFloats ( ) [inherited]`

Configures float data type functions *not* to do a word swap.

This is the default.

configureIeeeFloats() `[2/2]` `configureIeeeFloats ( integer slaveAddr ) [inherited]`

Disables float data type functions to do a word swap on a per slave basis.

Modbus is using little-endian word order for 32-bit values. This setting assumes that the slave also serves 32-bit floats in little little-endian word order which is the most common case.

Remarks

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.

Exceptions

EIllegalArgumentError	A parameter is out of range

configureStandard32BitMode() `[1/2]` `configureStandard32BitMode ( ) [inherited]`

Configures all slaves for Standard 32-bit Mode.

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

Remarks

This is the default mode

configureStandard32BitMode() `[2/2]` `configureStandard32BitMode ( integer slaveAddr ) [inherited]`

Configures a slave for Standard 32-bit Register Mode.

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

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

configureEnron32BitMode() `[1/2]` `configureEnron32BitMode ( ) [inherited]`

Configures all slaves for Daniel/ENRON 32-bit Mode.

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.

configureEnron32BitMode() `[2/2] configureEnron32BitMode ( integer slaveAddr ) [inherited]`

Configures all slaves for Daniel/ENRON 32-bit Mode.

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.

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

configureCountFromOne() `[1/2] configureCountFromOne ( ) [inherited]`

Configures the reference counting scheme to start with one for all slaves.

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

Remarks

This is the default mode

configureCountFromOne() `[2/2] configureCountFromOne ( integer slaveAddr ) [inherited]`

Configures a slave's reference counting scheme to start with one.

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

Parameters

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

Remarks

This is the default mode

Exceptions

EIllegalArgumentError	A parameter is out of range

configureCountFromZero() `[1/2]` `configureCountFromZero ( ) [inherited]`

Configures the reference counting scheme to start with zero for all slaves.

This renders the valid reference range to be 0 to 65535.

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

configureCountFromZero() `[2/2]` `configureCountFromZero ( integer` *`slaveAddr`* `) [inherited]`

Configures a slave's reference counting scheme to start with zero.

This is also known as PDU addressing.

This renders the valid reference range to be 0 to 65535.

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

isOpen() `boolean isOpen ( ) [inherited]`

Returns whether the protocol is open or not.

Return values

true	= open
false	= closed

getPackageVersion() `class string getPackageVersion ( ) [inherited]`

Returns the package version number.

Returns

Package version string

6.2.4 Member Data Documentation

portName `string portName [inherited]`

Serial port identifier property (eg 'COM1')

Note

A protocol must be closed in order to configure it.

See also

fPortName For reading
fPortName For writing

baudRate `longint baudRate [inherited]`

Baud rate property in bps (typically 1200 - 115200, maximum value depends on UART hardware)

Note

A protocol must be closed in order to configure it.

See also

fBaudRate For reading
fBaudRate For writing

dataBits `integer dataBits [inherited]`

Data bits property.

SER_DATABITS_7: 7 data bits (ASCII protocol only), SER_DATABITS_8: 8 data bits

Note

A protocol must be closed in order to configure it.

See also

fDataBits For reading
fDataBits For writing

stopBits `integer stopBits [inherited]`

Stop bits property.

SER_STOPBITS_1: 1 stop bit, SER_STOPBITS_2: 2 stop bits

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.

See also

fStopBits For reading
fStopBits For writing

parity `integer parity [inherited]`

Parity property.

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

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.

See also

fParity For reading
fParity For writing

timeout `integer timeout [inherited]`

Time-out port property.

Note

A protocol must be closed in order to configure it.

See also

getTimeout For reading
setTimeout For writing

pollDelay `integer pollDelay [inherited]`

Poll delay property.

Delay between two Modbus read/writes in ms

Note

A protocol must be closed in order to configure it.

See also

getPollDelay For reading
setPollDelay For writing

retryCnt `integer retryCnt [inherited]`

Retry count property.

Note

A protocol must be closed in order to configure it.

See also

getRetryCnt For reading
setRetryCnt For writing

6.3 TMbusElamMasterProtocol Class Reference

Extended Lufkin Automation Modbus Master Protocol.

Public Member Functions

- TMbusElamMasterProtocol (TComponent aOwner)

  Constructs a TMbusElamMasterProtocol object and initialises its data.
- openProtocol ()

  Opens a serial Modbus protocol and the associated serial port with the port parameters configured via properties.
- openUseExistingConnection (integer cnxnHandle)

  Opens a serial Modbus protocol using an existing and open handle.
- enableRs485Mode (integer rtsDelay)

  Enables RS485 mode.
- boolean isOpen ()

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

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

  Returns the package version number.

Public Attributes

- string portName

  Serial port identifier property (eg 'COM1')
- longint baudRate

  Baud rate property in bps (typically 1200 - 115200, maximum value depends on UART hardware)
- integer dataBits

  Data bits property.
- integer stopBits

  Stop bits property.
- integer parity

  Parity property.

Bit Access

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

- readCoils (integer slaveAddr, integer startRef, boolean [ ]bitArr)

  Modbus function 1 (01 hex), Read Coil Status/Read Coils.
- readInputDiscretes (integer slaveAddr, integer startRef, boolean [ ]bitArr)

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

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

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

16-bit Access

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

- readMultipleRegisters (integer slaveAddr, integer startRef, word [ ]regArr)

  Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers.
- readInputRegisters (integer slaveAddr, integer startRef, word [ ]regArr)

  Modbus function 4 (04 hex), Read Input Registers.
- writeSingleRegister (integer slaveAddr, integer regAddr, word regVal)

  Modbus function 6 (06 hex), Preset Single Register/Write Single Register.
- writeMultipleRegisters (integer slaveAddr, integer startRef, word [ ]regArr)

  Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers.
- maskWriteRegister (integer slaveAddr, integer regAddr, word andMask, word orMask)

  Modbus function 22 (16 hex), Mask Write Register.
- readWriteRegisters (integer slaveAddr, integer readRef, word [ ]readArr, integer write↩ Ref, word [ ]writeArr)

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

32-bit Access

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

- readMultipleLongInts (integer slaveAddr, integer startRef, integer [ ]int32Arr)

  Modbus function 3 (03 hex) for 32-bit long int data types, Read Holding Registers/Read Multiple Registers as long int data.
- readInputLongInts (integer slaveAddr, integer startRef, integer [ ]int32Arr)

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

  Modbus function 16 (10 hex) for 32-bit long int data types, Preset Multiple Registers/Write Multiple Registers with long int data.
- readMultipleFloats (integer slaveAddr, integer startRef, single [ ]float32Arr)

  Modbus function 3 (03 hex) for 32-bit float data types, Read Holding Registers/Read Multiple Registers as float data.
- readInputFloats (integer slaveAddr, integer startRef, single [ ]float32Arr)

  Modbus function 4 (04 hex) for 32-bit float data types, Read Input Registers as float data.
- writeMultipleFloats (integer slaveAddr, integer startRef, single [ ]float32Arr)

  Modbus function 16 (10 hex) for 32-bit float data types, Preset Multiple Registers/Write Multiple Registers with float data.
- readMultipleMod10000 (integer slaveAddr, integer startRef, integer [ ]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.
- readInputMod10000 (integer slaveAddr, integer startRef, integer [ ]int32Arr)

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

Diagnostics

- readExceptionStatus (integer slaveAddr, byte &statusByte)

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

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

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

Custom Function Codes

- customFunction (integer slaveAddr, integer functionCode, byte [ ]requestArr, byte [ ]responseArr, integer &responseLen)

  User Defined Function Code
  This method can be used to implement User Defined Function Codes.

Protocol Configuration

- setTimeout (const integer timeOut)

  Configures time-out.
- integer getTimeout ()

  Returns the time-out value.
- setPollDelay (const integer pollDelay)

  Configures poll delay.
- integer getPollDelay ()

  Returns the poll delay time.
- setRetryCnt (const integer retryCnt)

  Configures the automatic retry setting.
- integer getRetryCnt ()

  Returns the automatic retry count.
- integer timeout

  Time-out port property.
- integer pollDelay

  Poll delay property.
- integer retryCnt

  Retry count property.

Transmission Statistic Functions

- cardinal getTotalCounter ()

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

  Resets total message transfer counter.

- cardinal getSuccessCounter ()

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

  Resets successful message transfer counter.

Slave Configuration

- configureBigEndianInts ()

  Configures int data type functions to do a word swap.
- configureBigEndianInts (integer slaveAddr)

  Enables int data type functions to do a word swap on a per slave basis.
- configureSwappedFloats ()

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

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

  Configures int data type functions not to do a word swap.
- configureLittleEndianInts (integer slaveAddr)

  Disables word swapping for int data type functions on a per slave basis.
- configureIeeeFloats ()

  Configures float data type functions not to do a word swap.
- configureIeeeFloats (integer slaveAddr)

  Disables float data type functions to do a word swap on a per slave basis.
- configureStandard32BitMode ()

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

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

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

  Configures all slaves for Daniel/ENRON 32-bit Mode.
- configureCountFromOne ()

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

  Configures a slave's reference counting scheme to start with one.
- configureCountFromZero ()

  Configures the reference counting scheme to start with zero for all slaves.
- configureCountFromZero (integer slaveAddr)

  Configures a slave's reference counting scheme to start with zero.

6.3.1 Detailed Description

Extended Lufkin Automation Modbus Master Protocol.

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).

See also

Data and Control Functions for all Modbus Protocol Flavours, Serial Protocols
MbusRtuMasterProtocol

6.3.2 Constructor & Destructor Documentation

TMbusElamMasterProtocol() `TMbusElamMasterProtocol ( TComponent aOwner )`

Constructs a TMbusElamMasterProtocol object and initialises its data.

Exceptions

EOutOfResources	Creation of class failed

6.3.3 Member Function Documentation

openProtocol() `openProtocol ( ) [inherited]`

Opens a serial Modbus protocol and the associated serial port with the port parameters configured via properties.

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

Exceptions

EInOutError	An I/O error occurred
EOpenErr	The serial port does not exist
EPortAlreadyOpen	Port is already used by somedbody else
EPortNoAccess	No permission to access serial
EIllegalArgumentError	A parameter is invalid

openUseExistingConnection() `openUseExistingConnection ( integer cnxnHandle ) [inherited]`

Opens a serial Modbus protocol using an existing and open handle.

Useful for using the protocol over a modem link.

Parameters

cnxnHandle	Win32 API handle pointing to the existing and open connection.

Exceptions

EIllegalArgumentError	A parameter is invalid

enableRs485Mode() `enableRs485Mode ( integer rtsDelay ) [inherited]`

Enables RS485 mode.

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.

Warning

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.

Remarks

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

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

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

readCoils()

```
readCoils (
            integer slaveAddr,
            integer startRef,
            boolean [] bitArr )  [inherited]
```

Modbus function 1 (01 hex), Read Coil Status/Read Coils.

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputDiscretes() `readInputDiscretes ( integer slaveAddr, integer startRef, boolean [] bitArr ) [inherited]`

Modbus function 2 (02 hex), Read Inputs Status/Read Input Discretes.

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeCoil() `writeCoil ( integer slaveAddr, integer bitAddr, boolean bitVal ) [inherited]`

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

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

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

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

forceMultipleCoils()

```
forceMultipleCoils (
    integer slaveAddr,
    integer startRef,
    boolean [] bitArr )  [inherited]
```

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

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readMultipleRegisters()

```
readMultipleRegisters (
    integer slaveAddr,
```

```
integer startRef,
word [] regArr )  [inherited]
```

Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers.

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Buffer which will be filled with the data read. The length of the array determines how many registers are read (Range: 1-125).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputRegisters()

```
readInputRegisters (
integer slaveAddr,
integer startRef,
word [] regArr )  [inherited]
```

Modbus function 4 (04 hex), Read Input Registers.

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Buffer which will be filled with the data read. The length of the array determines how many registers are read (Range: 1-125).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred

Exceptions

EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeSingleRegister()

```
writeSingleRegister (
  integer slaveAddr,
  integer regAddr,
  word regVal )  [inherited]
```

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

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

Parameters

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

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

writeMultipleRegisters()

```
writeMultipleRegisters (
integer slaveAddr,
integer startRef,
word [] regArr )  [inherited]
```

Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers.

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Buffer with the data to be sent.The length of the array determines how many registers are written (Range: 1-123).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

maskWriteRegister() `maskWriteRegister (`
`integer` *slaveAddr*`,`
`integer` *regAddr*`,`
`word` *andMask*`,`
`word` *orMask* `)` `[inherited]`

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

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

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

Note

No broadcast supported

readWriteRegisters()

```
readWriteRegisters (
    integer slaveAddr,
    integer readRef,
    word [] readArr,
    integer writeRef,
    word [] writeArr )  [inherited]
```

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

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readMultipleLongInts()

```
readMultipleLongInts (
    integer slaveAddr,
    integer startRef,
    integer [] int32Arr )  [inherited]
```

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

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) into 32-bit long int values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a long int 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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputLongInts()

```
readInputLongInts (
    integer slaveAddr,
    integer startRef,
    integer [] int32Arr )  [inherited]
```

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

Reads the contents of pairs of consecutive input registers (3:00000 table) into 32-bit long int values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a long int 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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeMultipleLongInts()

```
writeMultipleLongInts (
integer slaveAddr,
integer startRef,
integer [] int32Arr )  [inherited]
```

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

Writes long int values into pairs of output registers (holding registers, 4:00000 table).

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a long int 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.

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 read (Range: 1-61).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readMultipleFloats()

```
readMultipleFloats (
    integer slaveAddr,
    integer startRef,
    single [] float32Arr )  [inherited]
```

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

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) into float values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a float 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 float values passed to this function.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
float32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputFloats()

```
readInputFloats (
        integer slaveAddr,
        integer startRef,
        single [] float32Arr )  [inherited]
```

Modbus function 4 (04 hex) for 32-bit float data types, Read Input Registers as float data.

Reads the contents of pairs of consecutive input registers (3:00000 table) into float values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a float 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 float values passed to this function.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
float32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeMultipleFloats()

```
writeMultipleFloats (
        integer slaveAddr,
        integer startRef,
        single [] float32Arr )  [inherited]
```

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

Writes float values into pairs of output registers (holding registers, 4:00000 table).

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a float 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 float values passed to this function.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start reference (Range: 1 - 65536)
float32Arr	Buffer with the data to be sent. The length of the array determines how many values are written (Range: 1-61).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readMultipleMod10000() `readMultipleMod10000 (`
`integer` *`slaveAddr,`*
`integer` *`startRef,`*
`integer [] int32Arr )  [inherited]`

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.

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.

Remarks

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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputMod10000()

```
readInputMod10000 (
 integer slaveAddr,
 integer startRef,
 integer [] int32Arr )  [inherited]
```

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

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.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because an 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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeMultipleMod10000() `writeMultipleMod10000 (`
`integer` *`slaveAddr`*`,`
`integer` *`startRef`*`,`
`integer [] int32Arr ) [inherited]`

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.

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.

Remarks

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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readExceptionStatus()

```
readExceptionStatus (
    integer slaveAddr,
    byte & statusByte )  [inherited]
```

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

Reads the eight exception status coils within the slave device.

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.

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

returnQueryData()

```
returnQueryData (
    integer slaveAddr,
    byte [] queryArr,
    byte [] echoArr )  [inherited]
```

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
queryArr	Data to be sent
echoArr	Buffer which will contain the data read. Array must be of the same size as queryArr.

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

restartCommunicationsOption() `restartCommunicationsOption (`
`integer` *`slaveAddr`*`,`
`boolean` *`clearEventLog`* `)` `[inherited]`

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

Parameters

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

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

setTimeout() `setTimeout ( const integer timeOut ) [inherited]`

Configures time-out.

This function sets the operation or socket time-out to the specified value.

Remarks

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)

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

getTimeout() `integer getTimeout ( ) [inherited]`

Returns the time-out value.

Remarks

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.

Returns

Timeout value in ms

setPollDelay() `setPollDelay ( const integer pollDelay ) [inherited]`

Configures poll delay.

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

Remarks

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

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

getPollDelay() `integer getPollDelay ( ) [inherited]`

Returns the poll delay time.

Returns

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

setRetryCnt() `setRetryCnt ( const integer retryCnt ) [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

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

getRetryCnt() `integer getRetryCnt ( ) [inherited]`

Returns the automatic retry count.

Returns

Retry count

getTotalCounter() `cardinal getTotalCounter ( ) [inherited]`

Returns how often a message transfer has been executed.

Returns

Counter value

getSuccessCounter() `cardinal getSuccessCounter ( ) [inherited]`

Returns how often a message transfer was successful.

Returns

Counter value

configureBigEndianInts() `[1/2] configureBigEndianInts ( ) [inherited]`

Configures int data type functions to do a word swap.

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 machine.

configureBigEndianInts() `[2/2] configureBigEndianInts ( integer slaveAddr ) [inherited]`

Enables int data type functions to do a word swap on a per slave basis.

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 machine.

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

configureSwappedFloats() `[1/2] configureSwappedFloats ( ) [inherited]`

Configures float data type functions to do a word swap.

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

Note

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

configureSwappedFloats() `[2/2] configureSwappedFloats ( integer slaveAddr ) [inherited]`

Enables float data type functions to do a word swap on a per slave basis.

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

Note

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.

Exceptions

EIllegalArgumentError	A parameter is out of range

configureLittleEndianInts() `[1/2]` `configureLittleEndianInts ( )` `[inherited]`

Configures int data type functions *not* to do a word swap.

This is the default.

configureLittleEndianInts() `[2/2]` `configureLittleEndianInts ( integer slaveAddr ) [inherited]`

Disables word swapping for int data type functions on a per slave basis.

Modbus is using little-endian word order for 32-bit values. This setting assumes that the slave also serves 32-bit data in little little-endian word order.

Remarks

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.

Exceptions

EIllegalArgumentError	A parameter is out of range

configureIeeeFloats() `[1/2]` `configureIeeeFloats ( )` `[inherited]`

Configures float data type functions *not* to do a word swap.

This is the default.

configureIeeeFloats() `[2/2]` `configureIeeeFloats ( integer slaveAddr ) [inherited]`

Disables float data type functions to do a word swap on a per slave basis.

Modbus is using little-endian word order for 32-bit values. This setting assumes that the slave also serves 32-bit floats in little little-endian word order which is the most common case.

Remarks

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.

Exceptions

EIllegalArgumentError	A parameter is out of range

configureStandard32BitMode() `[1/2]` `configureStandard32BitMode ( ) [inherited]`

Configures all slaves for Standard 32-bit Mode.

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

Remarks

This is the default mode

configureStandard32BitMode() `[2/2]` `configureStandard32BitMode ( integer slaveAddr ) [inherited]`

Configures a slave for Standard 32-bit Register Mode.

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

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

configureEnron32BitMode() `[1/2]` `configureEnron32BitMode ( ) [inherited]`

Configures all slaves for Daniel/ENRON 32-bit Mode.

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.

configureEnron32BitMode() `[2/2] configureEnron32BitMode ( integer slaveAddr ) [inherited]`

Configures all slaves for Daniel/ENRON 32-bit Mode.

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.

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

configureCountFromOne() `[1/2] configureCountFromOne ( ) [inherited]`

Configures the reference counting scheme to start with one for all slaves.

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

Remarks

This is the default mode

configureCountFromOne() `[2/2] configureCountFromOne ( integer slaveAddr ) [inherited]`

Configures a slave's reference counting scheme to start with one.

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

Parameters

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

Remarks

This is the default mode

Exceptions

EIllegalArgumentError	A parameter is out of range

configureCountFromZero() `[1/2]` `configureCountFromZero ( ) [inherited]`

Configures the reference counting scheme to start with zero for all slaves.

This renders the valid reference range to be 0 to 65535.

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

configureCountFromZero() `[2/2]` `configureCountFromZero ( integer slaveAddr ) [inherited]`

Configures a slave's reference counting scheme to start with zero.

This is also known as PDU addressing.

This renders the valid reference range to be 0 to 65535.

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

isOpen() `boolean isOpen ( ) [inherited]`

Returns whether the protocol is open or not.

Return values

true	= open
false	= closed

getPackageVersion() `class string getPackageVersion ( ) [inherited]`

Returns the package version number.

Returns

Package version string

6.3.4 Member Data Documentation

portName `string portName [inherited]`

Serial port identifier property (eg 'COM1')

Note

A protocol must be closed in order to configure it.

See also

fPortName For reading
fPortName For writing

baudRate `longint baudRate [inherited]`

Baud rate property in bps (typically 1200 - 115200, maximum value depends on UART hardware)

Note

A protocol must be closed in order to configure it.

See also

fBaudRate For reading
fBaudRate For writing

dataBits `integer dataBits [inherited]`

Data bits property.

SER_DATABITS_7: 7 data bits (ASCII protocol only), SER_DATABITS_8: 8 data bits

Note

A protocol must be closed in order to configure it.

See also

fDataBits For reading
fDataBits For writing

stopBits `integer stopBits [inherited]`

Stop bits property.

SER_STOPBITS_1: 1 stop bit, SER_STOPBITS_2: 2 stop bits

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.

See also

fStopBits For reading
fStopBits For writing

parity `integer parity [inherited]`

Parity property.

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

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.

See also

fParity For reading
fParity For writing

timeout `integer timeout [inherited]`

Time-out port property.

Note

A protocol must be closed in order to configure it.

See also

getTimeout For reading
setTimeout For writing

pollDelay `integer pollDelay [inherited]`

Poll delay property.

Delay between two Modbus read/writes in ms

Note

A protocol must be closed in order to configure it.

See also

getPollDelay For reading
setPollDelay For writing

retryCnt `integer retryCnt [inherited]`

Retry count property.

Note

A protocol must be closed in order to configure it.

See also

getRetryCnt For reading
setRetryCnt For writing

6.4 TMbusTcpMasterProtocol Class Reference

MODBUS/TCP Master Protocol class.

Public Member Functions

- TMbusTcpMasterProtocol (TComponent aOwner)

  Constructs a TMbusTcpMasterProtocol object and initialises its data.
- openProtocol ()

  Connects to a MODBUS/TCP slave.
- setPort (word portNo)

  Sets the TCP port number to be used by the protocol.
- setClosingTimeout (const integer msTime)

  Applies a time-out to socket closure and makes closeProtocol() wait for the server to acknowledge closing before potentially opening a new one.
- word getPort ()

  Returns the TCP port number used by the protocol.
- boolean isOpen ()

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

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

  Returns the package version number.

Public Attributes

- string hostName

  Host name property (eg '127.0.0.1')
- word port

  TCP port property (eg 502)

Bit Access

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

- readCoils (integer slaveAddr, integer startRef, boolean [ ]bitArr)

  Modbus function 1 (01 hex), Read Coil Status/Read Coils.
- readInputDiscretes (integer slaveAddr, integer startRef, boolean [ ]bitArr)

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

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

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

16-bit Access

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

- readMultipleRegisters (integer slaveAddr, integer startRef, word [ ]regArr)

  Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers.
- readInputRegisters (integer slaveAddr, integer startRef, word [ ]regArr)

  Modbus function 4 (04 hex), Read Input Registers.
- writeSingleRegister (integer slaveAddr, integer regAddr, word regVal)

  Modbus function 6 (06 hex), Preset Single Register/Write Single Register.
- writeMultipleRegisters (integer slaveAddr, integer startRef, word [ ]regArr)

  Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers.
- maskWriteRegister (integer slaveAddr, integer regAddr, word andMask, word orMask)

  Modbus function 22 (16 hex), Mask Write Register.
- readWriteRegisters (integer slaveAddr, integer readRef, word [ ]readArr, integer write↩ Ref, word [ ]writeArr)

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

32-bit Access

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

- readMultipleLongInts (integer slaveAddr, integer startRef, integer [ ]int32Arr)

  Modbus function 3 (03 hex) for 32-bit long int data types, Read Holding Registers/Read Multiple Registers as long int data.
- readInputLongInts (integer slaveAddr, integer startRef, integer [ ]int32Arr)

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

  Modbus function 16 (10 hex) for 32-bit long int data types, Preset Multiple Registers/Write Multiple Registers with long int data.
- readMultipleFloats (integer slaveAddr, integer startRef, single [ ]float32Arr)

  Modbus function 3 (03 hex) for 32-bit float data types, Read Holding Registers/Read Multiple Registers as float data.
- readInputFloats (integer slaveAddr, integer startRef, single [ ]float32Arr)

  Modbus function 4 (04 hex) for 32-bit float data types, Read Input Registers as float data.
- writeMultipleFloats (integer slaveAddr, integer startRef, single [ ]float32Arr)

  Modbus function 16 (10 hex) for 32-bit float data types, Preset Multiple Registers/Write Multiple Registers with float data.
- readMultipleMod10000 (integer slaveAddr, integer startRef, integer [ ]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.
- readInputMod10000 (integer slaveAddr, integer startRef, integer [ ]int32Arr)

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

Diagnostics

- readExceptionStatus (integer slaveAddr, byte &statusByte)

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

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

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

Custom Function Codes

- customFunction (integer slaveAddr, integer functionCode, byte [ ]requestArr, byte [ ]responseArr, integer &responseLen)

  User Defined Function Code
  This method can be used to implement User Defined Function Codes.

Protocol Configuration

- setTimeout (const integer timeOut)

  Configures time-out.
- integer getTimeout ()

  Returns the time-out value.
- setPollDelay (const integer pollDelay)

  Configures poll delay.
- integer getPollDelay ()

  Returns the poll delay time.
- setRetryCnt (const integer retryCnt)

  Configures the automatic retry setting.
- integer getRetryCnt ()

  Returns the automatic retry count.
- integer timeout

  Time-out port property.
- integer pollDelay

  Poll delay property.
- integer retryCnt

  Retry count property.

Transmission Statistic Functions

- cardinal getTotalCounter ()

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

  Resets total message transfer counter.

- cardinal getSuccessCounter ()
  Returns how often a message transfer was successful.
- resetSuccessCounter ()
  Resets successful message transfer counter.

Slave Configuration

- configureBigEndianInts ()
  Configures int data type functions to do a word swap.
- configureBigEndianInts (integer slaveAddr)
  Enables int data type functions to do a word swap on a per slave basis.
- configureSwappedFloats ()
  Configures float data type functions to do a word swap.
- configureSwappedFloats (integer slaveAddr)
  Enables float data type functions to do a word swap on a per slave basis.
- configureLittleEndianInts ()
  Configures int data type functions not to do a word swap.
- configureLittleEndianInts (integer slaveAddr)
  Disables word swapping for int data type functions on a per slave basis.
- configureIeeeFloats ()
  Configures float data type functions not to do a word swap.
- configureIeeeFloats (integer slaveAddr)
  Disables float data type functions to do a word swap on a per slave basis.
- configureStandard32BitMode ()
  Configures all slaves for Standard 32-bit Mode.
- configureStandard32BitMode (integer slaveAddr)
  Configures a slave for Standard 32-bit Register Mode.
- configureEnron32BitMode ()
  Configures all slaves for Daniel/ENRON 32-bit Mode.
- configureEnron32BitMode (integer slaveAddr)
  Configures all slaves for Daniel/ENRON 32-bit Mode.
- configureCountFromOne ()
  Configures the reference counting scheme to start with one for all slaves.
- configureCountFromOne (integer slaveAddr)
  Configures a slave's reference counting scheme to start with one.
- configureCountFromZero ()
  Configures the reference counting scheme to start with zero for all slaves.
- configureCountFromZero (integer slaveAddr)
  Configures a slave's reference counting scheme to start with zero.

Advantec ADAM 5000/6000 Series Commands

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

6.4.1 Detailed Description

MODBUS/TCP Master Protocol class.

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 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.

It is also possible to instantiate multiple instances of this class for establishing multiple connections to either the same or different hosts.

See also

Data and Control Functions for all Modbus Protocol Flavours, IP based Protocols

6.4.2 Constructor & Destructor Documentation

TMbusTcpMasterProtocol() `TMbusTcpMasterProtocol ( TComponent aOwner )`

Constructs a TMbusTcpMasterProtocol object and initialises its data.

Exceptions

EOutOfResources	Creation of class failed

6.4.3 Member Function Documentation

openProtocol() `openProtocol ( ) [inherited]`

Connects to a MODBUS/TCP slave.

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.

Note

The default TCP port number is 502.

Exceptions

EInOutError	An I/O error occurred
EOpenErr	The port could not be opened
EPortNoAccess	No permission to access port
ETcpipConnectErr	TCP/IP connection error, host not reachable
EConnectionWasClosed	Remote peer closed TCP/IP connection
EIllegalArgumentError	A parameter is invalid

setPort() `setPort ( word` *`portNo`* `) [inherited]`

Sets the TCP port number to be used by the protocol.

Remarks

Usually the port number remains unchanged and defaults to 502. In this case no call to this function is necessary. However if the port number has to be different from 502 this function must be called *before* opening the connection with openProtocol().

Parameters

portNo	Port number to be used when opening the connection

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

setClosingTimeout() `setClosingTimeout ( const integer` *`msTime`* `) [inherited]`

Applies a time-out to socket closure and makes closeProtocol() wait for the server to acknowledge closing before potentially opening a new one.

Remarks

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

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

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

getPort() `word getPort ( ) [inherited]`

Returns the TCP port number used by the protocol.

Returns

Port number used by the protocol

readCoils()

```
readCoils (
            integer slaveAddr,
            integer startRef,
            boolean [] bitArr ) [inherited]
```

Modbus function 1 (01 hex), Read Coil Status/Read Coils.

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range

Exceptions

EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputDiscretes()

```
readInputDiscretes (
    integer slaveAddr,
    integer startRef,
    boolean [] bitArr )  [inherited]
```

Modbus function 2 (02 hex), Read Inputs Status/Read Input Discretes.

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeCoil()

```
writeCoil (
    integer slaveAddr,
    integer bitAddr,
    boolean bitVal )  [inherited]
```

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

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

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

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

forceMultipleCoils()

```
forceMultipleCoils (
    integer slaveAddr,
    integer startRef,
    boolean [] bitArr )  [inherited]
```

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

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readMultipleRegisters() `readMultipleRegisters ( integer slaveAddr, integer startRef, word [] regArr ) [inherited]`

Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers.

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Buffer which will be filled with the data read. The length of the array determines how many registers are read (Range: 1-125).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputRegisters() `readInputRegisters ( integer slaveAddr, integer startRef, word [] regArr ) [inherited]`

Modbus function 4 (04 hex), Read Input Registers.

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)

Parameters

regArr	Buffer which will be filled with the data read. The length of the array determines how many registers are read (Range: 1-125).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeSingleRegister()

```
writeSingleRegister (
  integer slaveAddr,
  integer regAddr,
  word regVal )  [inherited]
```

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

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

Parameters

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

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

writeMultipleRegisters() `writeMultipleRegisters (`
`integer` *`slaveAddr`*`,`
`integer` *`startRef`*`,`
`word [] ` *`regArr`* `) [inherited]`

Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers.

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Buffer with the data to be sent.The length of the array determines how many registers are written (Range: 1-123).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

maskWriteRegister() `maskWriteRegister (`
`integer` *`slaveAddr`*`,`
`integer` *`regAddr`*`,`
`word` *`andMask`*`,`
`word` *`orMask`* `) [inherited]`

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

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

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

Note

No broadcast supported

readWriteRegisters()

```
readWriteRegisters (
    integer slaveAddr,
    integer readRef,
    word [] readArr,
    integer writeRef,
    word [] writeArr )  [inherited]
```

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

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readMultipleLongInts() `readMultipleLongInts ( integer slaveAddr, integer startRef, integer [] int32Arr ) [inherited]`

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

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) into 32-bit long int values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a long int 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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputLongInts() `readInputLongInts ( integer slaveAddr,`

```
integer startRef,
integer [] int32Arr )  [inherited]
```

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

Reads the contents of pairs of consecutive input registers (3:00000 table) into 32-bit long int values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a long int 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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeMultipleLongInts()

```
writeMultipleLongInts (
integer slaveAddr,
integer startRef,
integer [] int32Arr )  [inherited]
```

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

Writes long int values into pairs of output registers (holding registers, 4:00000 table).

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a long int 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.

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 read (Range: 1-61).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readMultipleFloats()

```
readMultipleFloats (
    integer slaveAddr,
    integer startRef,
    single [] float32Arr )  [inherited]
```

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

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) into float values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a float 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 float values passed to this function.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
float32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputFloats()

```
readInputFloats (
        integer slaveAddr,
        integer startRef,
        single [] float32Arr )  [inherited]
```

Modbus function 4 (04 hex) for 32-bit float data types, Read Input Registers as float data.

Reads the contents of pairs of consecutive input registers (3:00000 table) into float values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a float 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 float values passed to this function.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
float32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeMultipleFloats()

```
writeMultipleFloats (
  integer slaveAddr,
  integer startRef,
  single [] float32Arr )  [inherited]
```

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

Writes float values into pairs of output registers (holding registers, 4:00000 table).

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a float 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 float values passed to this function.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start reference (Range: 1 - 65536)
float32Arr	Buffer with the data to be sent. The length of the array determines how many values are written (Range: 1-61).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readMultipleMod10000() `readMultipleMod10000 (`
`integer` *`slaveAddr,`*
`integer` *`startRef,`*
`integer [] int32Arr )` `[inherited]`

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.

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.

Remarks

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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputMod10000() `readInputMod10000 (`
`integer` *`slaveAddr,`*

```
integer startRef,
integer [] int32Arr )  [inherited]
```

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

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.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because an 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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeMultipleMod10000() `writeMultipleMod10000 (`

```
integer slaveAddr,
integer startRef,
integer [] int32Arr )  [inherited]
```

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.

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.

Remarks

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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readExceptionStatus()

```
readExceptionStatus (
 integer slaveAddr,
 byte & statusByte )  [inherited]
```

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

Reads the eight exception status coils within the slave device.

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.

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range

Exceptions

EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

returnQueryData() `returnQueryData (`
`integer` *slaveAddr*`,`
`byte [] queryArr,`
`byte [] echoArr ) [inherited]`

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
queryArr	Data to be sent
echoArr	Buffer which will contain the data read. Array must be of the same size as queryArr.

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

restartCommunicationsOption() `restartCommunicationsOption (`
`integer` *slaveAddr*`,`
`boolean clearEventLog ) [inherited]`

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

Parameters

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

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

setTimeout() `setTimeout ( const integer timeOut ) [inherited]`

Configures time-out.

This function sets the operation or socket time-out to the specified value.

Remarks

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)

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

getTimeout() `integer getTimeout ( ) [inherited]`

Returns the time-out value.

Remarks

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.

Returns

Timeout value in ms

setPollDelay() `setPollDelay ( const integer pollDelay ) [inherited]`

Configures poll delay.

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

Remarks

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

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

getPollDelay() `integer getPollDelay ( ) [inherited]`

Returns the poll delay time.

Returns

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

setRetryCnt() `setRetryCnt ( const integer retryCnt ) [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

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

getRetryCnt() `integer getRetryCnt ( ) [inherited]`

Returns the automatic retry count.

Returns

Retry count

getTotalCounter() `cardinal getTotalCounter ( ) [inherited]`

Returns how often a message transfer has been executed.

Returns

Counter value

getSuccessCounter() `cardinal getSuccessCounter ( ) [inherited]`

Returns how often a message transfer was successful.

Returns

Counter value

configureBigEndianInts() `[1/2] configureBigEndianInts ( ) [inherited]`

Configures int data type functions to do a word swap.

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 machine.

configureBigEndianInts() `[2/2] configureBigEndianInts ( integer slaveAddr ) [inherited]`

Enables int data type functions to do a word swap on a per slave basis.

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 machine.

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

configureSwappedFloats() `[1/2] configureSwappedFloats ( ) [inherited]`

Configures float data type functions to do a word swap.

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

Note

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

configureSwappedFloats() `[2/2] configureSwappedFloats ( integer slaveAddr ) [inherited]`

Enables float data type functions to do a word swap on a per slave basis.

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

Note

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.

Exceptions

EIllegalArgumentError	A parameter is out of range

configureLittleEndianInts() `[1/2] configureLittleEndianInts ( ) [inherited]`

Configures int data type functions *not* to do a word swap.

This is the default.

configureLittleEndianInts() `[2/2] configureLittleEndianInts ( integer slaveAddr ) [inherited]`

Disables word swapping for int data type functions on a per slave basis.

Modbus is using little-endian word order for 32-bit values. This setting assumes that the slave also serves 32-bit data in little little-endian word order.

Remarks

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.

Exceptions

EIllegalArgumentError	A parameter is out of range

configureIeeeFloats() `[1/2]` `configureIeeeFloats ( ) [inherited]`

Configures float data type functions *not* to do a word swap.

This is the default.

configureIeeeFloats() `[2/2]` `configureIeeeFloats ( integer slaveAddr ) [inherited]`

Disables float data type functions to do a word swap on a per slave basis.

Modbus is using little-endian word order for 32-bit values. This setting assumes that the slave also serves 32-bit floats in little little-endian word order which is the most common case.

Remarks

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.

Exceptions

EIllegalArgumentError	A parameter is out of range

configureStandard32BitMode() `[1/2]` `configureStandard32BitMode ( ) [inherited]`

Configures all slaves for Standard 32-bit Mode.

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

Remarks

This is the default mode

configureStandard32BitMode() `[2/2]` `configureStandard32BitMode ( integer slaveAddr ) [inherited]`

Configures a slave for Standard 32-bit Register Mode.

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

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

configureEnron32BitMode() `[1/2] configureEnron32BitMode ( ) [inherited]`

Configures all slaves for Daniel/ENRON 32-bit Mode.

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.

configureEnron32BitMode() `[2/2] configureEnron32BitMode ( integer slaveAddr ) [inherited]`

Configures all slaves for Daniel/ENRON 32-bit Mode.

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.

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

configureCountFromOne() `[1/2] configureCountFromOne ( ) [inherited]`

Configures the reference counting scheme to start with one for all slaves.

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

Remarks

This is the default mode

configureCountFromOne() `[2/2] configureCountFromOne ( integer slaveAddr ) [inherited]`

Configures a slave's reference counting scheme to start with one.

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

Parameters

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

Remarks

This is the default mode

Exceptions

EIllegalArgumentError	A parameter is out of range

configureCountFromZero() `[1/2] configureCountFromZero ( ) [inherited]`

Configures the reference counting scheme to start with zero for all slaves.

This renders the valid reference range to be 0 to 65535.

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

configureCountFromZero() `[2/2] configureCountFromZero ( integer slaveAddr ) [inherited]`

Configures a slave's reference counting scheme to start with zero.

This is also known as PDU addressing.

This renders the valid reference range to be 0 to 65535.

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

isOpen() `boolean isOpen ( ) [inherited]`

Returns whether the protocol is open or not.

Return values

true	= open
false	= closed

getPackageVersion() `class string getPackageVersion ( ) [inherited]`

Returns the package version number.

Returns

Package version string

6.4.4 Member Data Documentation

hostName `string hostName [inherited]`

Host name property (eg '127.0.0.1')

Note

A protocol must be closed in order to configure it.

See also

fHostName For reading
fHostName For writing

port `word port [inherited]`

TCP port property (eg 502)

Note

A protocol must be closed in order to configure it.

Remarks

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 called *before* opening the connection with openProtocol().

See also

getPort For reading
setPort For writing

timeout `integer timeout [inherited]`

Time-out port property.

Note

A protocol must be closed in order to configure it.

See also

getTimeout For reading
setTimeout For writing

pollDelay `integer pollDelay [inherited]`

Poll delay property.

Delay between two Modbus read/writes in ms

Note

A protocol must be closed in order to configure it.

See also

getPollDelay For reading
setPollDelay For writing

retryCnt `integer retryCnt [inherited]`

Retry count property.

Note

A protocol must be closed in order to configure it.

See also

getRetryCnt For reading
setRetryCnt For writing

6.5 TMbusRtuOverTcpMasterProtocol Class Reference

Encapsulated Modbus RTU Master Protocol class.

Public Member Functions

- TMbusRtuOverTcpMasterProtocol (TComponent aOwner)

  Constructs a TMbusRtuOverTcpMasterProtocol object and initialises its data.
- openProtocol ()

  Connects to a MODBUS/TCP slave.
- setPort (word portNo)

  Sets the TCP port number to be used by the protocol.
- setClosingTimeout (const integer msTime)

  Applies a time-out to socket closure and makes closeProtocol() wait for the server to acknowledge closing before potentially opening a new one.
- word getPort ()

  Returns the TCP port number used by the protocol.
- boolean isOpen ()

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

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

  Returns the package version number.

Public Attributes

- string hostName

  Host name property (eg '127.0.0.1')
- word port

  TCP port property (eg 502)

Bit Access

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

- readCoils (integer slaveAddr, integer startRef, boolean [ ]bitArr)
  Modbus function 1 (01 hex), Read Coil Status/Read Coils.
- readInputDiscretes (integer slaveAddr, integer startRef, boolean [ ]bitArr)
  Modbus function 2 (02 hex), Read Inputs Status/Read Input Discretes.
- writeCoil (integer slaveAddr, integer bitAddr, boolean bitVal)
  Modbus function 5 (05 hex), Force Single Coil/Write Coil.
- forceMultipleCoils (integer slaveAddr, integer startRef, boolean [ ]bitArr)
  Modbus function 15 (0F hex), Force Multiple Coils.

16-bit Access

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

- readMultipleRegisters (integer slaveAddr, integer startRef, word [ ]regArr)
  Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers.
- readInputRegisters (integer slaveAddr, integer startRef, word [ ]regArr)
  Modbus function 4 (04 hex), Read Input Registers.
- writeSingleRegister (integer slaveAddr, integer regAddr, word regVal)
  Modbus function 6 (06 hex), Preset Single Register/Write Single Register.
- writeMultipleRegisters (integer slaveAddr, integer startRef, word [ ]regArr)
  Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers.
- maskWriteRegister (integer slaveAddr, integer regAddr, word andMask, word orMask)
  Modbus function 22 (16 hex), Mask Write Register.
- readWriteRegisters (integer slaveAddr, integer readRef, word [ ]readArr, integer write↩ Ref, word [ ]writeArr)
  Modbus function 23 (17 hex), Read/Write Registers.

32-bit Access

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

- readMultipleLongInts (integer slaveAddr, integer startRef, integer [ ]int32Arr)
  Modbus function 3 (03 hex) for 32-bit long int data types, Read Holding Registers/Read Multiple Registers as long int data.
- readInputLongInts (integer slaveAddr, integer startRef, integer [ ]int32Arr)
  Modbus function 4 (04 hex) for 32-bit long int data types, Read Input Registers as long int data.
- writeMultipleLongInts (integer slaveAddr, integer startRef, integer [ ]int32Arr)
  Modbus function 16 (10 hex) for 32-bit long int data types, Preset Multiple Registers/Write Multiple Registers with long int data.
- readMultipleFloats (integer slaveAddr, integer startRef, single [ ]float32Arr)

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

- readInputFloats (integer slaveAddr, integer startRef, single [ ]float32Arr)

  Modbus function 4 (04 hex) for 32-bit float data types, Read Input Registers as float data.
- writeMultipleFloats (integer slaveAddr, integer startRef, single [ ]float32Arr)

  Modbus function 16 (10 hex) for 32-bit float data types, Preset Multiple Registers/Write Multiple Registers with float data.
- readMultipleMod10000 (integer slaveAddr, integer startRef, integer [ ]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.
- readInputMod10000 (integer slaveAddr, integer startRef, integer [ ]int32Arr)

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

Diagnostics

- readExceptionStatus (integer slaveAddr, byte &statusByte)

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

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

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

Custom Function Codes

- customFunction (integer slaveAddr, integer functionCode, byte [ ]requestArr, byte [ ]responseArr, integer &responseLen)

  User Defined Function Code
  This method can be used to implement User Defined Function Codes.

Protocol Configuration

- setTimeout (const integer timeOut)

  Configures time-out.
- integer getTimeout ()

  Returns the time-out value.
- setPollDelay (const integer pollDelay)

  Configures poll delay.
- integer getPollDelay ()

  Returns the poll delay time.
- setRetryCnt (const integer retryCnt)

Configures the automatic retry setting.

- integer getRetryCnt ()

  Returns the automatic retry count.
- integer timeout

  Time-out port property.
- integer pollDelay

  Poll delay property.
- integer retryCnt

  Retry count property.

Transmission Statistic Functions

- cardinal getTotalCounter ()

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

  Resets total message transfer counter.
- cardinal getSuccessCounter ()

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

  Resets successful message transfer counter.

Slave Configuration

- configureBigEndianInts ()

  Configures int data type functions to do a word swap.
- configureBigEndianInts (integer slaveAddr)

  Enables int data type functions to do a word swap on a per slave basis.
- configureSwappedFloats ()

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

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

  Configures int data type functions not to do a word swap.
- configureLittleEndianInts (integer slaveAddr)

  Disables word swapping for int data type functions on a per slave basis.
- configureIeeeFloats ()

  Configures float data type functions not to do a word swap.
- configureIeeeFloats (integer slaveAddr)

  Disables float data type functions to do a word swap on a per slave basis.
- configureStandard32BitMode ()

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

  Configures a slave for Standard 32-bit Register Mode.

- configureEnron32BitMode ()

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

  Configures all slaves for Daniel/ENRON 32-bit Mode.
- configureCountFromOne ()

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

  Configures a slave's reference counting scheme to start with one.
- configureCountFromZero ()

  Configures the reference counting scheme to start with zero for all slaves.
- configureCountFromZero (integer slaveAddr)

  Configures a slave's reference counting scheme to start with zero.

6.5.1 Detailed Description

Encapsulated Modbus RTU Master Protocol class.

This class realises the Encapsulated Modbus RTU master protocol. This protocol is also known as RTU over TCP or RTU/IP and used for example by ISaGraf Soft-PLCs. This class 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. For a more detailed description of the data and control functions see section Data and Control Functions for all Modbus Protocol Flavours.

It is also possible to instantiate multiple instances of this class for establishing multiple connections to either the same or different hosts.

See also

Data and Control Functions for all Modbus Protocol Flavours, IP based Protocols
MbusRtuMasterProtocol

6.5.2 Constructor & Destructor Documentation

TMbusRtuOverTcpMasterProtocol() `TMbusRtuOverTcpMasterProtocol ( TComponent aOwner )`

Constructs a TMbusRtuOverTcpMasterProtocol object and initialises its data.

Exceptions

EOutOfResources	Creation of class failed

6.5.3 Member Function Documentation

openProtocol() `openProtocol ( ) [inherited]`

Connects to a MODBUS/TCP slave.

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.

Note

The default TCP port number is 502.

Exceptions

EInOutError	An I/O error occurred
EOpenErr	The port could not be opened
EPortNoAccess	No permission to access port
ETcpipConnectErr	TCP/IP connection error, host not reachable
EConnectionWasClosed	Remote peer closed TCP/IP connection
EIllegalArgumentError	A parameter is invalid

setPort() `setPort ( word portNo ) [inherited]`

Sets the TCP port number to be used by the protocol.

Remarks

Usually the port number remains unchanged and defaults to 502. In this case no call to this function is necessary. However if the port number has to be different from 502 this function must be called *before* opening the connection with openProtocol().

Parameters

portNo	Port number to be used when opening the connection

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

setClosingTimeout() `setClosingTimeout (`
`const integer` *msTime* `) [inherited]`

Applies a time-out to socket closure and makes closeProtocol() wait for the server to acknowledge closing before potentially opening a new one.

Remarks

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

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

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

getPort() `word getPort ( ) [inherited]`

Returns the TCP port number used by the protocol.

Returns

Port number used by the protocol

readCoils() `readCoils (`
`integer` *slaveAddr*`,`
`integer` *startRef*`,`
`boolean [] bitArr ) [inherited]`

Modbus function 1 (01 hex), Read Coil Status/Read Coils.

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputDiscretes()

```
readInputDiscretes (
    integer slaveAddr,
    integer startRef,
    boolean [] bitArr )  [inherited]
```

Modbus function 2 (02 hex), Read Inputs Status/Read Input Discretes.

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeCoil() `writeCoil (`
`integer slaveAddr,`
`integer bitAddr,`
`boolean bitVal ) [inherited]`

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

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

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

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

forceMultipleCoils() `forceMultipleCoils (`
`integer slaveAddr,`
`integer startRef,`
`boolean [] bitArr ) [inherited]`

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

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readMultipleRegisters()

```
readMultipleRegisters (
integer slaveAddr,
integer startRef,
word [] regArr )  [inherited]
```

Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers.

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Buffer which will be filled with the data read. The length of the array determines how many registers are read (Range: 1-125).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputRegisters()

```
readInputRegisters (
integer slaveAddr,
```

```
integer startRef,
word [] regArr )  [inherited]
```

Modbus function 4 (04 hex), Read Input Registers.

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Buffer which will be filled with the data read. The length of the array determines how many registers are read (Range: 1-125).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeSingleRegister()

```
writeSingleRegister (
integer slaveAddr,
integer regAddr,
word regVal )  [inherited]
```

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

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

Parameters

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

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range

Exceptions

EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

writeMultipleRegisters()

```
writeMultipleRegisters (
integer slaveAddr,
integer startRef,
word [] regArr )  [inherited]
```

Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers.

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Buffer with the data to be sent.The length of the array determines how many registers are written (Range: 1-123).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

maskWriteRegister()

```
maskWriteRegister (
integer slaveAddr,
integer regAddr,
word andMask,
word orMask )  [inherited]
```

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

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

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

Note

No broadcast supported

readWriteRegisters()

```
readWriteRegisters (
    integer slaveAddr,
    integer readRef,
    word [] readArr,
    integer writeRef,
    word [] writeArr )  [inherited]
```

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

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readMultipleLongInts()

```
readMultipleLongInts (
integer slaveAddr,
integer startRef,
integer [] int32Arr )  [inherited]
```

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

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) into 32-bit long int values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a long int 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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputLongInts()

```
readInputLongInts (
integer slaveAddr,
```

```
integer startRef,
integer [] int32Arr )  [inherited]
```

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

Reads the contents of pairs of consecutive input registers (3:00000 table) into 32-bit long int values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a long int 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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeMultipleLongInts() `writeMultipleLongInts (`

```
integer slaveAddr,
integer startRef,
integer [] int32Arr )  [inherited]
```

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

Writes long int values into pairs of output registers (holding registers, 4:00000 table).

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a long int 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.

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 read (Range: 1-61).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readMultipleFloats()

```
readMultipleFloats (
  integer slaveAddr,
  integer startRef,
  single [] float32Arr )  [inherited]
```

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

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) into float values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a float 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 float values passed to this function.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
float32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputFloats()

```
readInputFloats (
        integer slaveAddr,
        integer startRef,
        single [] float32Arr )  [inherited]
```

Modbus function 4 (04 hex) for 32-bit float data types, Read Input Registers as float data.

Reads the contents of pairs of consecutive input registers (3:00000 table) into float values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a float 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 float values passed to this function.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
float32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeMultipleFloats()

```
writeMultipleFloats (
  integer slaveAddr,
  integer startRef,
  single [] float32Arr )  [inherited]
```

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

Writes float values into pairs of output registers (holding registers, 4:00000 table).

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a float 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 float values passed to this function.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start reference (Range: 1 - 65536)
float32Arr	Buffer with the data to be sent. The length of the array determines how many values are written (Range: 1-61).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readMultipleMod10000() `readMultipleMod10000 (`
`integer` *`slaveAddr,`*
`integer` *`startRef,`*
`integer [] int32Arr )` `[inherited]`

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.

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.

Remarks

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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputMod10000() `readInputMod10000 (`
`integer` *`slaveAddr,`*

```
integer startRef,
integer [] int32Arr )  [inherited]
```

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

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.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because an 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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeMultipleMod10000() `writeMultipleMod10000 (`

```
integer slaveAddr,
integer startRef,
integer [] int32Arr )  [inherited]
```

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.

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.

Remarks

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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readExceptionStatus() `readExceptionStatus ( integer slaveAddr, byte & statusByte ) [inherited]`

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

Reads the eight exception status coils within the slave device.

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.

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range

Exceptions

EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

returnQueryData() `returnQueryData (`
`integer` *slaveAddr*`,`
`byte [] queryArr,`
`byte [] echoArr ) [inherited]`

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
queryArr	Data to be sent
echoArr	Buffer which will contain the data read. Array must be of the same size as queryArr.

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

restartCommunicationsOption() `restartCommunicationsOption (`
`integer` *slaveAddr*`,`
`boolean` *clearEventLog* `) [inherited]`

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

Parameters

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

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

setTimeout() `setTimeout ( const integer timeOut ) [inherited]`

Configures time-out.

This function sets the operation or socket time-out to the specified value.

Remarks

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)

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

getTimeout() `integer getTimeout ( ) [inherited]`

Returns the time-out value.

Remarks

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.

Returns

Timeout value in ms

setPollDelay() `setPollDelay ( const integer pollDelay ) [inherited]`

Configures poll delay.

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

Remarks

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

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

getPollDelay() `integer getPollDelay ( ) [inherited]`

Returns the poll delay time.

Returns

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

setRetryCnt() `setRetryCnt ( const integer retryCnt ) [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

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

getRetryCnt() `integer getRetryCnt ( ) [inherited]`

Returns the automatic retry count.

Returns

Retry count

getTotalCounter() `cardinal getTotalCounter ( ) [inherited]`

Returns how often a message transfer has been executed.

Returns

Counter value

getSuccessCounter() `cardinal getSuccessCounter ( ) [inherited]`

Returns how often a message transfer was successful.

Returns

Counter value

configureBigEndianInts() `[1/2] configureBigEndianInts ( ) [inherited]`

Configures int data type functions to do a word swap.

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 machine.

configureBigEndianInts() `[2/2] configureBigEndianInts ( integer slaveAddr ) [inherited]`

Enables int data type functions to do a word swap on a per slave basis.

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 machine.

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

configureSwappedFloats() `[1/2] configureSwappedFloats ( ) [inherited]`

Configures float data type functions to do a word swap.

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

Note

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

configureSwappedFloats() `[2/2] configureSwappedFloats ( integer slaveAddr ) [inherited]`

Enables float data type functions to do a word swap on a per slave basis.

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

Note

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.

Exceptions

EIllegalArgumentError	A parameter is out of range

configureLittleEndianInts() `[1/2] configureLittleEndianInts ( ) [inherited]`

Configures int data type functions *not* to do a word swap.

This is the default.

configureLittleEndianInts() `[2/2] configureLittleEndianInts ( integer slaveAddr ) [inherited]`

Disables word swapping for int data type functions on a per slave basis.

Modbus is using little-endian word order for 32-bit values. This setting assumes that the slave also serves 32-bit data in little little-endian word order.

Remarks

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.

Exceptions

EIllegalArgumentError	A parameter is out of range

configureIeeeFloats() [1/2] `configureIeeeFloats ( ) [inherited]`

Configures float data type functions *not* to do a word swap.

This is the default.

configureIeeeFloats() [2/2] `configureIeeeFloats ( integer slaveAddr ) [inherited]`

Disables float data type functions to do a word swap on a per slave basis.

Modbus is using little-endian word order for 32-bit values. This setting assumes that the slave also serves 32-bit floats in little little-endian word order which is the most common case.

Remarks

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.

Exceptions

EIllegalArgumentError	A parameter is out of range

configureStandard32BitMode() [1/2] `configureStandard32BitMode ( ) [inherited]`

Configures all slaves for Standard 32-bit Mode.

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

Remarks

This is the default mode

configureStandard32BitMode() [2/2] `configureStandard32BitMode ( integer slaveAddr ) [inherited]`

Configures a slave for Standard 32-bit Register Mode.

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

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

configureEnron32BitMode() `[1/2] configureEnron32BitMode ( ) [inherited]`

Configures all slaves for Daniel/ENRON 32-bit Mode.

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.

configureEnron32BitMode() `[2/2] configureEnron32BitMode ( integer slaveAddr ) [inherited]`

Configures all slaves for Daniel/ENRON 32-bit Mode.

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.

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

configureCountFromOne() `[1/2] configureCountFromOne ( ) [inherited]`

Configures the reference counting scheme to start with one for all slaves.

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

Remarks

This is the default mode

configureCountFromOne() `[2/2] configureCountFromOne ( integer slaveAddr ) [inherited]`

Configures a slave's reference counting scheme to start with one.

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

Parameters

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

Remarks

This is the default mode

Exceptions

EIllegalArgumentError	A parameter is out of range

configureCountFromZero() `[1/2] configureCountFromZero ( ) [inherited]`

Configures the reference counting scheme to start with zero for all slaves.

This renders the valid reference range to be 0 to 65535.

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

configureCountFromZero() `[2/2] configureCountFromZero ( integer slaveAddr ) [inherited]`

Configures a slave's reference counting scheme to start with zero.

This is also known as PDU addressing.

This renders the valid reference range to be 0 to 65535.

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

isOpen() `boolean isOpen ( ) [inherited]`

Returns whether the protocol is open or not.

Return values

true	= open
false	= closed

getPackageVersion() `class string getPackageVersion ( ) [inherited]`

Returns the package version number.

Returns

Package version string

6.5.4 Member Data Documentation

hostName `string hostName [inherited]`

Host name property (eg '127.0.0.1')

Note

A protocol must be closed in order to configure it.

See also

fHostName For reading
fHostName For writing

port `word port [inherited]`

TCP port property (eg 502)

Note

A protocol must be closed in order to configure it.

Remarks

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 called *before* opening the connection with openProtocol().

See also

getPort For reading
setPort For writing

timeout `integer timeout [inherited]`

Time-out port property.

Note

A protocol must be closed in order to configure it.

See also

getTimeout For reading
setTimeout For writing

pollDelay `integer pollDelay [inherited]`

Poll delay property.

Delay between two Modbus read/writes in ms

Note

A protocol must be closed in order to configure it.

See also

getPollDelay For reading
setPollDelay For writing

retryCnt `integer retryCnt [inherited]`

Retry count property.

Note

A protocol must be closed in order to configure it.

See also

getRetryCnt For reading
setRetryCnt For writing

6.6 TMbusAsciiOverTcpMasterProtocol Class Reference

MODBUS ASCII over TCP Master Protocol class.

Public Member Functions

- TMbusAsciiOverTcpMasterProtocol (TComponent aOwner)

  Constructs a TMbusAsciiOverTcpMasterProtocol object and initialises its data.
- openProtocol ()

  Connects to a MODBUS/TCP slave.
- setPort (word portNo)

  Sets the TCP port number to be used by the protocol.
- setClosingTimeout (const integer msTime)

  Applies a time-out to socket closure and makes closeProtocol() wait for the server to acknowledge closing before potentially opening a new one.
- word getPort ()

  Returns the TCP port number used by the protocol.
- boolean isOpen ()

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

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

  Returns the package version number.

Public Attributes

- string hostName

  Host name property (eg '127.0.0.1')
- word port

  TCP port property (eg 502)

Bit Access

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

- readCoils (integer slaveAddr, integer startRef, boolean [ ]bitArr)
  Modbus function 1 (01 hex), Read Coil Status/Read Coils.
- readInputDiscretes (integer slaveAddr, integer startRef, boolean [ ]bitArr)
  Modbus function 2 (02 hex), Read Inputs Status/Read Input Discretes.
- writeCoil (integer slaveAddr, integer bitAddr, boolean bitVal)
  Modbus function 5 (05 hex), Force Single Coil/Write Coil.
- forceMultipleCoils (integer slaveAddr, integer startRef, boolean [ ]bitArr)
  Modbus function 15 (0F hex), Force Multiple Coils.

16-bit Access

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

- readMultipleRegisters (integer slaveAddr, integer startRef, word [ ]regArr)
  Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers.
- readInputRegisters (integer slaveAddr, integer startRef, word [ ]regArr)
  Modbus function 4 (04 hex), Read Input Registers.
- writeSingleRegister (integer slaveAddr, integer regAddr, word regVal)
  Modbus function 6 (06 hex), Preset Single Register/Write Single Register.
- writeMultipleRegisters (integer slaveAddr, integer startRef, word [ ]regArr)
  Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers.
- maskWriteRegister (integer slaveAddr, integer regAddr, word andMask, word orMask)
  Modbus function 22 (16 hex), Mask Write Register.
- readWriteRegisters (integer slaveAddr, integer readRef, word [ ]readArr, integer write↩ Ref, word [ ]writeArr)
  Modbus function 23 (17 hex), Read/Write Registers.

32-bit Access

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

- readMultipleLongInts (integer slaveAddr, integer startRef, integer [ ]int32Arr)
  Modbus function 3 (03 hex) for 32-bit long int data types, Read Holding Registers/Read Multiple Registers as long int data.
- readInputLongInts (integer slaveAddr, integer startRef, integer [ ]int32Arr)
  Modbus function 4 (04 hex) for 32-bit long int data types, Read Input Registers as long int data.
- writeMultipleLongInts (integer slaveAddr, integer startRef, integer [ ]int32Arr)
  Modbus function 16 (10 hex) for 32-bit long int data types, Preset Multiple Registers/Write Multiple Registers with long int data.
- readMultipleFloats (integer slaveAddr, integer startRef, single [ ]float32Arr)

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

- readInputFloats (integer slaveAddr, integer startRef, single [ ]float32Arr)

  Modbus function 4 (04 hex) for 32-bit float data types, Read Input Registers as float data.
- writeMultipleFloats (integer slaveAddr, integer startRef, single [ ]float32Arr)

  Modbus function 16 (10 hex) for 32-bit float data types, Preset Multiple Registers/Write Multiple Registers with float data.
- readMultipleMod10000 (integer slaveAddr, integer startRef, integer [ ]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.
- readInputMod10000 (integer slaveAddr, integer startRef, integer [ ]int32Arr)

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

Diagnostics

- readExceptionStatus (integer slaveAddr, byte &statusByte)

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

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

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

Custom Function Codes

- customFunction (integer slaveAddr, integer functionCode, byte [ ]requestArr, byte [ ]responseArr, integer &responseLen)

  User Defined Function Code
  This method can be used to implement User Defined Function Codes.

Protocol Configuration

- setTimeout (const integer timeOut)

  Configures time-out.
- integer getTimeout ()

  Returns the time-out value.
- setPollDelay (const integer pollDelay)

  Configures poll delay.
- integer getPollDelay ()

  Returns the poll delay time.
- setRetryCnt (const integer retryCnt)

Configures the automatic retry setting.

- integer getRetryCnt ()

  Returns the automatic retry count.
- integer timeout

  Time-out port property.
- integer pollDelay

  Poll delay property.
- integer retryCnt

  Retry count property.

Transmission Statistic Functions

- cardinal getTotalCounter ()

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

  Resets total message transfer counter.
- cardinal getSuccessCounter ()

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

  Resets successful message transfer counter.

Slave Configuration

- configureBigEndianInts ()

  Configures int data type functions to do a word swap.
- configureBigEndianInts (integer slaveAddr)

  Enables int data type functions to do a word swap on a per slave basis.
- configureSwappedFloats ()

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

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

  Configures int data type functions not to do a word swap.
- configureLittleEndianInts (integer slaveAddr)

  Disables word swapping for int data type functions on a per slave basis.
- configureIeeeFloats ()

  Configures float data type functions not to do a word swap.
- configureIeeeFloats (integer slaveAddr)

  Disables float data type functions to do a word swap on a per slave basis.
- configureStandard32BitMode ()

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

  Configures a slave for Standard 32-bit Register Mode.

- configureEnron32BitMode ()

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

  Configures all slaves for Daniel/ENRON 32-bit Mode.
- configureCountFromOne ()

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

  Configures a slave's reference counting scheme to start with one.
- configureCountFromZero ()

  Configures the reference counting scheme to start with zero for all slaves.
- configureCountFromZero (integer slaveAddr)

  Configures a slave's reference counting scheme to start with zero.

6.6.1 Detailed Description

MODBUS ASCII over TCP Master Protocol class.

This class realises the Modbus ASCII master protocol using TCP as transport layer. This class 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. For a more detailed description of the data and control functions see section Data and Control Functions for all Modbus Protocol Flavours.

It is also possible to instantiate multiple instances of this class for establishing multiple connections to either the same or different hosts.

See also

Data and Control Functions for all Modbus Protocol Flavours, IP based Protocols
MbusAsciiMasterProtocol

6.6.2 Constructor & Destructor Documentation

TMbusAsciiOverTcpMasterProtocol() `TMbusAsciiOverTcpMasterProtocol ( TComponent aOwner )`

Constructs a TMbusAsciiOverTcpMasterProtocol object and initialises its data.

Exceptions

EOutOfResources	Creation of class failed

6.6.3 Member Function Documentation

openProtocol() `openProtocol ( ) [inherited]`

Connects to a MODBUS/TCP slave.

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.

Note

The default TCP port number is 502.

Exceptions

EInOutError	An I/O error occurred
EOpenErr	The port could not be opened
EPortNoAccess	No permission to access port
ETcpipConnectErr	TCP/IP connection error, host not reachable
EConnectionWasClosed	Remote peer closed TCP/IP connection
EIllegalArgumentError	A parameter is invalid

setPort() `setPort ( word portNo ) [inherited]`

Sets the TCP port number to be used by the protocol.

Remarks

Usually the port number remains unchanged and defaults to 502. In this case no call to this function is necessary. However if the port number has to be different from 502 this function must be called *before* opening the connection with openProtocol().

Parameters

portNo	Port number to be used when opening the connection

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

setClosingTimeout() `setClosingTimeout ( const integer msTime ) [inherited]`

Applies a time-out to socket closure and makes closeProtocol() wait for the server to acknowledge closing before potentially opening a new one.

Remarks

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

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

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

getPort() `word getPort ( ) [inherited]`

Returns the TCP port number used by the protocol.

Returns

Port number used by the protocol

readCoils() `readCoils ( integer slaveAddr, integer startRef, boolean [] bitArr ) [inherited]`

Modbus function 1 (01 hex), Read Coil Status/Read Coils.

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputDiscretes()

```
readInputDiscretes (
    integer slaveAddr,
    integer startRef,
    boolean [] bitArr )  [inherited]
```

Modbus function 2 (02 hex), Read Inputs Status/Read Input Discretes.

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeCoil()

```
writeCoil (
            integer slaveAddr,
            integer bitAddr,
            boolean bitVal )  [inherited]
```

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

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

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

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

forceMultipleCoils()

```
forceMultipleCoils (
            integer slaveAddr,
            integer startRef,
            boolean [] bitArr )  [inherited]
```

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

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readMultipleRegisters()

```
readMultipleRegisters (
integer slaveAddr,
integer startRef,
word [] regArr )  [inherited]
```

Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers.

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Buffer which will be filled with the data read. The length of the array determines how many registers are read (Range: 1-125).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputRegisters()

```
readInputRegisters (
integer slaveAddr,
```

```
integer startRef,
word [] regArr )  [inherited]
```

Modbus function 4 (04 hex), Read Input Registers.

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Buffer which will be filled with the data read. The length of the array determines how many registers are read (Range: 1-125).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeSingleRegister()

```
writeSingleRegister (
  integer slaveAddr,
  integer regAddr,
  word regVal )  [inherited]
```

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

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

Parameters

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

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range

Exceptions

EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

writeMultipleRegisters() `writeMultipleRegisters ( integer slaveAddr, integer startRef, word [] regArr ) [inherited]`

Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers.

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Buffer with the data to be sent.The length of the array determines how many registers are written (Range: 1-123).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

maskWriteRegister() `maskWriteRegister ( integer slaveAddr, integer regAddr, word andMask, word orMask ) [inherited]`

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

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

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

Note

No broadcast supported

readWriteRegisters()

```
readWriteRegisters (
    integer slaveAddr,
    integer readRef,
    word [] readArr,
    integer writeRef,
    word [] writeArr )  [inherited]
```

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

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readMultipleLongInts() `readMultipleLongInts ( integer slaveAddr, integer startRef, integer [] int32Arr ) [inherited]`

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

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) into 32-bit long int values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a long int 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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputLongInts() `readInputLongInts ( integer slaveAddr,`

```
integer startRef,
integer [] int32Arr )  [inherited]
```

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

Reads the contents of pairs of consecutive input registers (3:00000 table) into 32-bit long int values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a long int 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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeMultipleLongInts()

```
writeMultipleLongInts (
integer slaveAddr,
integer startRef,
integer [] int32Arr )  [inherited]
```

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

Writes long int values into pairs of output registers (holding registers, 4:00000 table).

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a long int 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.

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 read (Range: 1-61).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readMultipleFloats()

```
readMultipleFloats (
    integer slaveAddr,
    integer startRef,
    single [] float32Arr )  [inherited]
```

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

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) into float values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a float 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 float values passed to this function.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
float32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputFloats()

```
readInputFloats (
        integer slaveAddr,
        integer startRef,
        single [] float32Arr )  [inherited]
```

Modbus function 4 (04 hex) for 32-bit float data types, Read Input Registers as float data.

Reads the contents of pairs of consecutive input registers (3:00000 table) into float values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a float 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 float values passed to this function.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
float32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeMultipleFloats()

```
writeMultipleFloats (
  integer slaveAddr,
  integer startRef,
  single [] float32Arr )  [inherited]
```

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

Writes float values into pairs of output registers (holding registers, 4:00000 table).

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a float 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 float values passed to this function.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start reference (Range: 1 - 65536)
float32Arr	Buffer with the data to be sent. The length of the array determines how many values are written (Range: 1-61).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readMultipleMod10000() `readMultipleMod10000 (`
`integer` *`slaveAddr,`*
`integer` *`startRef,`*
`integer [] int32Arr )` `[inherited]`

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.

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.

Remarks

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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputMod10000() `readInputMod10000 (`
`integer` *`slaveAddr,`*

```
integer startRef,
integer [] int32Arr )  [inherited]
```

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

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.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because an 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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeMultipleMod10000() `writeMultipleMod10000 (`

```
integer slaveAddr,
integer startRef,
integer [] int32Arr )  [inherited]
```

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.

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.

Remarks

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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readExceptionStatus()

```
readExceptionStatus (
 integer slaveAddr,
 byte & statusByte )  [inherited]
```

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

Reads the eight exception status coils within the slave device.

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.

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range

Exceptions

EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

returnQueryData() `returnQueryData (`
`integer` *slaveAddr*`,`
`byte [] queryArr,`
`byte [] echoArr ) [inherited]`

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
queryArr	Data to be sent
echoArr	Buffer which will contain the data read. Array must be of the same size as queryArr.

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

restartCommunicationsOption() `restartCommunicationsOption (`
`integer` *slaveAddr*`,`
`boolean clearEventLog ) [inherited]`

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

Parameters

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

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

setTimeout() `setTimeout (`
`const integer` *`timeOut`* `)` `[inherited]`

Configures time-out.

This function sets the operation or socket time-out to the specified value.

Remarks

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)

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

getTimeout() `integer getTimeout ( ) [inherited]`

Returns the time-out value.

Remarks

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.

Returns

Timeout value in ms

setPollDelay() `setPollDelay ( const integer pollDelay ) [inherited]`

Configures poll delay.

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

Remarks

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

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

getPollDelay() `integer getPollDelay ( ) [inherited]`

Returns the poll delay time.

Returns

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

setRetryCnt() `setRetryCnt ( const integer retryCnt ) [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

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

getRetryCnt() `integer getRetryCnt ( ) [inherited]`

Returns the automatic retry count.

Returns

Retry count

getTotalCounter() `cardinal getTotalCounter ( ) [inherited]`

Returns how often a message transfer has been executed.

Returns

Counter value

getSuccessCounter() `cardinal getSuccessCounter ( ) [inherited]`

Returns how often a message transfer was successful.

Returns

Counter value

configureBigEndianInts() `[1/2] configureBigEndianInts ( ) [inherited]`

Configures int data type functions to do a word swap.

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 machine.

configureBigEndianInts() `[2/2] configureBigEndianInts ( integer slaveAddr ) [inherited]`

Enables int data type functions to do a word swap on a per slave basis.

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 machine.

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

configureSwappedFloats() `[1/2] configureSwappedFloats ( ) [inherited]`

Configures float data type functions to do a word swap.

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

Note

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

configureSwappedFloats() `[2/2] configureSwappedFloats ( integer slaveAddr ) [inherited]`

Enables float data type functions to do a word swap on a per slave basis.

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

Note

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.

Exceptions

EIllegalArgumentError	A parameter is out of range

configureLittleEndianInts() `[1/2] configureLittleEndianInts ( ) [inherited]`

Configures int data type functions *not* to do a word swap.

This is the default.

configureLittleEndianInts() `[2/2] configureLittleEndianInts ( integer slaveAddr ) [inherited]`

Disables word swapping for int data type functions on a per slave basis.

Modbus is using little-endian word order for 32-bit values. This setting assumes that the slave also serves 32-bit data in little little-endian word order.

Remarks

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.

Exceptions

EIllegalArgumentError	A parameter is out of range

configureIeeeFloats() `[1/2]` `configureIeeeFloats ( ) [inherited]`

Configures float data type functions *not* to do a word swap.

This is the default.

configureIeeeFloats() `[2/2]` `configureIeeeFloats ( integer slaveAddr ) [inherited]`

Disables float data type functions to do a word swap on a per slave basis.

Modbus is using little-endian word order for 32-bit values. This setting assumes that the slave also serves 32-bit floats in little little-endian word order which is the most common case.

Remarks

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.

Exceptions

EIllegalArgumentError	A parameter is out of range

configureStandard32BitMode() `[1/2]` `configureStandard32BitMode ( ) [inherited]`

Configures all slaves for Standard 32-bit Mode.

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

Remarks

This is the default mode

configureStandard32BitMode() `[2/2]` `configureStandard32BitMode ( integer slaveAddr ) [inherited]`

Configures a slave for Standard 32-bit Register Mode.

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

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

configureEnron32BitMode() `[1/2]` `configureEnron32BitMode ( ) [inherited]`

Configures all slaves for Daniel/ENRON 32-bit Mode.

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.

configureEnron32BitMode() `[2/2]` `configureEnron32BitMode ( integer slaveAddr ) [inherited]`

Configures all slaves for Daniel/ENRON 32-bit Mode.

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.

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

configureCountFromOne() `[1/2]` `configureCountFromOne ( ) [inherited]`

Configures the reference counting scheme to start with one for all slaves.

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

Remarks

This is the default mode

configureCountFromOne() `[2/2] configureCountFromOne ( integer slaveAddr ) [inherited]`

Configures a slave's reference counting scheme to start with one.

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

Parameters

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

Remarks

This is the default mode

Exceptions

EIllegalArgumentError	A parameter is out of range

configureCountFromZero() `[1/2] configureCountFromZero ( ) [inherited]`

Configures the reference counting scheme to start with zero for all slaves.

This renders the valid reference range to be 0 to 65535.

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

configureCountFromZero() `[2/2] configureCountFromZero ( integer slaveAddr ) [inherited]`

Configures a slave's reference counting scheme to start with zero.

This is also known as PDU addressing.

This renders the valid reference range to be 0 to 65535.

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

isOpen() `boolean isOpen ( ) [inherited]`

Returns whether the protocol is open or not.

Return values

true	= open
false	= closed

getPackageVersion() `class string getPackageVersion ( ) [inherited]`

Returns the package version number.

Returns

Package version string

6.6.4 Member Data Documentation

hostName `string hostName [inherited]`

Host name property (eg '127.0.0.1')

Note

A protocol must be closed in order to configure it.

See also

fHostName For reading
fHostName For writing

port `word port [inherited]`

TCP port property (eg 502)

Note

A protocol must be closed in order to configure it.

Remarks

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 called *before* opening the connection with openProtocol().

See also

getPort For reading
setPort For writing

timeout `integer timeout [inherited]`

Time-out port property.

Note

A protocol must be closed in order to configure it.

See also

getTimeout For reading
setTimeout For writing

pollDelay `integer pollDelay [inherited]`

Poll delay property.

Delay between two Modbus read/writes in ms

Note

A protocol must be closed in order to configure it.

See also

getPollDelay For reading
setPollDelay For writing

retryCnt `integer retryCnt [inherited]`

Retry count property.

Note

A protocol must be closed in order to configure it.

See also

getRetryCnt For reading
setRetryCnt For writing

6.7 TMbusUdpMasterProtocol Class Reference

MODBUS/UDP Master Protocol class.

Public Member Functions

- TMbusUdpMasterProtocol (TComponent aOwner)

  Constructs a TMbusUdpMasterProtocol object and initialises its data.
- openProtocol ()

  Connects to a MODBUS/TCP slave.
- setPort (word portNo)

  Sets the TCP port number to be used by the protocol.
- setClosingTimeout (const integer msTime)

  Applies a time-out to socket closure and makes closeProtocol() wait for the server to acknowledge closing before potentially opening a new one.
- word getPort ()

  Returns the TCP port number used by the protocol.
- boolean isOpen ()

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

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

  Returns the package version number.

Public Attributes

- string hostName

  Host name property (eg '127.0.0.1')
- word port

  TCP port property (eg 502)

Bit Access

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

- readCoils (integer slaveAddr, integer startRef, boolean [ ]bitArr)
  Modbus function 1 (01 hex), Read Coil Status/Read Coils.
- readInputDiscretes (integer slaveAddr, integer startRef, boolean [ ]bitArr)
  Modbus function 2 (02 hex), Read Inputs Status/Read Input Discretes.
- writeCoil (integer slaveAddr, integer bitAddr, boolean bitVal)
  Modbus function 5 (05 hex), Force Single Coil/Write Coil.
- forceMultipleCoils (integer slaveAddr, integer startRef, boolean [ ]bitArr)
  Modbus function 15 (0F hex), Force Multiple Coils.

16-bit Access

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

- readMultipleRegisters (integer slaveAddr, integer startRef, word [ ]regArr)
  Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers.
- readInputRegisters (integer slaveAddr, integer startRef, word [ ]regArr)
  Modbus function 4 (04 hex), Read Input Registers.
- writeSingleRegister (integer slaveAddr, integer regAddr, word regVal)
  Modbus function 6 (06 hex), Preset Single Register/Write Single Register.
- writeMultipleRegisters (integer slaveAddr, integer startRef, word [ ]regArr)
  Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers.
- maskWriteRegister (integer slaveAddr, integer regAddr, word andMask, word orMask)
  Modbus function 22 (16 hex), Mask Write Register.
- readWriteRegisters (integer slaveAddr, integer readRef, word [ ]readArr, integer write↩ Ref, word [ ]writeArr)
  Modbus function 23 (17 hex), Read/Write Registers.

32-bit Access

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

- readMultipleLongInts (integer slaveAddr, integer startRef, integer [ ]int32Arr)
  Modbus function 3 (03 hex) for 32-bit long int data types, Read Holding Registers/Read Multiple Registers as long int data.
- readInputLongInts (integer slaveAddr, integer startRef, integer [ ]int32Arr)
  Modbus function 4 (04 hex) for 32-bit long int data types, Read Input Registers as long int data.
- writeMultipleLongInts (integer slaveAddr, integer startRef, integer [ ]int32Arr)
  Modbus function 16 (10 hex) for 32-bit long int data types, Preset Multiple Registers/Write Multiple Registers with long int data.
- readMultipleFloats (integer slaveAddr, integer startRef, single [ ]float32Arr)

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

- readInputFloats (integer slaveAddr, integer startRef, single [ ]float32Arr)

  Modbus function 4 (04 hex) for 32-bit float data types, Read Input Registers as float data.
- writeMultipleFloats (integer slaveAddr, integer startRef, single [ ]float32Arr)

  Modbus function 16 (10 hex) for 32-bit float data types, Preset Multiple Registers/Write Multiple Registers with float data.
- readMultipleMod10000 (integer slaveAddr, integer startRef, integer [ ]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.
- readInputMod10000 (integer slaveAddr, integer startRef, integer [ ]int32Arr)

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

Diagnostics

- readExceptionStatus (integer slaveAddr, byte &statusByte)

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

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

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

Custom Function Codes

- customFunction (integer slaveAddr, integer functionCode, byte [ ]requestArr, byte [ ]responseArr, integer &responseLen)

  User Defined Function Code
  This method can be used to implement User Defined Function Codes.

Protocol Configuration

- setTimeout (const integer timeOut)

  Configures time-out.
- integer getTimeout ()

  Returns the time-out value.
- setPollDelay (const integer pollDelay)

  Configures poll delay.
- integer getPollDelay ()

  Returns the poll delay time.
- setRetryCnt (const integer retryCnt)

Configures the automatic retry setting.

- integer getRetryCnt ()

  Returns the automatic retry count.
- integer timeout

  Time-out port property.
- integer pollDelay

  Poll delay property.
- integer retryCnt

  Retry count property.

Transmission Statistic Functions

- cardinal getTotalCounter ()

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

  Resets total message transfer counter.
- cardinal getSuccessCounter ()

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

  Resets successful message transfer counter.

Slave Configuration

- configureBigEndianInts ()

  Configures int data type functions to do a word swap.
- configureBigEndianInts (integer slaveAddr)

  Enables int data type functions to do a word swap on a per slave basis.
- configureSwappedFloats ()

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

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

  Configures int data type functions not to do a word swap.
- configureLittleEndianInts (integer slaveAddr)

  Disables word swapping for int data type functions on a per slave basis.
- configureIeeeFloats ()

  Configures float data type functions not to do a word swap.
- configureIeeeFloats (integer slaveAddr)

  Disables float data type functions to do a word swap on a per slave basis.
- configureStandard32BitMode ()

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

  Configures a slave for Standard 32-bit Register Mode.

- configureEnron32BitMode ()

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

  Configures all slaves for Daniel/ENRON 32-bit Mode.
- configureCountFromOne ()

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

  Configures a slave's reference counting scheme to start with one.
- configureCountFromZero ()

  Configures the reference counting scheme to start with zero for all slaves.
- configureCountFromZero (integer slaveAddr)

  Configures a slave's reference counting scheme to start with zero.

6.7.1 Detailed Description

MODBUS/UDP Master Protocol class.

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. For a more detailed description of the data and control functions see section Data and Control Functions for all Modbus Protocol Flavours.

It is also possible to instantiate multiple instances of this class for establishing multiple connections to either the same or different hosts.

See also

Data and Control Functions for all Modbus Protocol Flavours, IP based Protocols

6.7.2 Constructor & Destructor Documentation

TMbusUdpMasterProtocol() `TMbusUdpMasterProtocol (`
`TComponent` *`aOwner`* `)`

Constructs a TMbusUdpMasterProtocol object and initialises its data.

Exceptions

EOutOfResources	Creation of class failed

6.7.3 Member Function Documentation

openProtocol() `openProtocol ( ) [inherited]`

Connects to a MODBUS/TCP slave.

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.

Note

The default TCP port number is 502.

Exceptions

EInOutError	An I/O error occurred
EOpenErr	The port could not be opened
EPortNoAccess	No permission to access port
ETcpipConnectErr	TCP/IP connection error, host not reachable
EConnectionWasClosed	Remote peer closed TCP/IP connection
EIllegalArgumentError	A parameter is invalid

setPort() `setPort ( word portNo ) [inherited]`

Sets the TCP port number to be used by the protocol.

Remarks

Usually the port number remains unchanged and defaults to 502. In this case no call to this function is necessary. However if the port number has to be different from 502 this function must be called *before* opening the connection with openProtocol().

Parameters

portNo	Port number to be used when opening the connection

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

setClosingTimeout() `setClosingTimeout ( const integer msTime ) [inherited]`

Applies a time-out to socket closure and makes closeProtocol() wait for the server to acknowledge closing before potentially opening a new one.

Remarks

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

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

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

getPort() `word getPort ( ) [inherited]`

Returns the TCP port number used by the protocol.

Returns

Port number used by the protocol

readCoils() `readCoils ( integer slaveAddr, integer startRef, boolean [] bitArr ) [inherited]`

Modbus function 1 (01 hex), Read Coil Status/Read Coils.

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputDiscretes()

```
readInputDiscretes (
    integer slaveAddr,
    integer startRef,
    boolean [] bitArr )  [inherited]
```

Modbus function 2 (02 hex), Read Inputs Status/Read Input Discretes.

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeCoil() `writeCoil (`
`integer slaveAddr,`
`integer bitAddr,`
`boolean bitVal ) [inherited]`

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

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

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

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

forceMultipleCoils() `forceMultipleCoils (`
`integer slaveAddr,`
`integer startRef,`
`boolean [] bitArr ) [inherited]`

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

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readMultipleRegisters()

```
readMultipleRegisters (
integer slaveAddr,
integer startRef,
word [] regArr )  [inherited]
```

Modbus function 3 (03 hex), Read Holding Registers/Read Multiple Registers.

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Buffer which will be filled with the data read. The length of the array determines how many registers are read (Range: 1-125).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputRegisters()

```
readInputRegisters (
integer slaveAddr,
```

```
integer startRef,
word [] regArr )  [inherited]
```

Modbus function 4 (04 hex), Read Input Registers.

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Buffer which will be filled with the data read. The length of the array determines how many registers are read (Range: 1-125).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeSingleRegister()

```
writeSingleRegister (
integer slaveAddr,
integer regAddr,
word regVal )  [inherited]
```

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

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

Parameters

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

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range

Exceptions

EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

writeMultipleRegisters()

```
writeMultipleRegisters (
integer slaveAddr,
integer startRef,
word [] regArr )  [inherited]
```

Modbus function 16 (10 hex), Preset Multiple Registers/Write Multiple Registers.

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start register (Range: 1 - 65536)
regArr	Buffer with the data to be sent.The length of the array determines how many registers are written (Range: 1-123).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

maskWriteRegister()

```
maskWriteRegister (
integer slaveAddr,
integer regAddr,
word andMask,
word orMask )  [inherited]
```

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

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

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

Note

No broadcast supported

readWriteRegisters()

```
readWriteRegisters (
    integer slaveAddr,
    integer readRef,
    word [] readArr,
    integer writeRef,
    word [] writeArr )  [inherited]
```

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

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

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readMultipleLongInts()

```
readMultipleLongInts (
integer slaveAddr,
integer startRef,
integer [] int32Arr )  [inherited]
```

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

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) into 32-bit long int values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a long int 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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputLongInts()

```
readInputLongInts (
integer slaveAddr,
```

```
integer startRef,
integer [] int32Arr )  [inherited]
```

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

Reads the contents of pairs of consecutive input registers (3:00000 table) into 32-bit long int values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a long int 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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeMultipleLongInts() `writeMultipleLongInts (`

```
integer slaveAddr,
integer startRef,
integer [] int32Arr )  [inherited]
```

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

Writes long int values into pairs of output registers (holding registers, 4:00000 table).

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a long int 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.

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 read (Range: 1-61).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readMultipleFloats()

```
readMultipleFloats (
  integer slaveAddr,
  integer startRef,
  single [] float32Arr )  [inherited]
```

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

Reads the contents of pairs of consecutive output registers (holding registers, 4:00000 table) into float values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a float 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 float values passed to this function.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
float32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputFloats()

```
readInputFloats (
        integer slaveAddr,
        integer startRef,
        single [] float32Arr )  [inherited]
```

Modbus function 4 (04 hex) for 32-bit float data types, Read Input Registers as float data.

Reads the contents of pairs of consecutive input registers (3:00000 table) into float values.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a float 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 float values passed to this function.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
startRef	Start reference (Range: 1 - 65536)
float32Arr	Buffer which will be filled with the data read. The length of the array determines how many values are read (Range: 1-62).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeMultipleFloats()

```
writeMultipleFloats (
  integer slaveAddr,
  integer startRef,
  single [] float32Arr )  [inherited]
```

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

Writes float values into pairs of output registers (holding registers, 4:00000 table).

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because a float 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 float values passed to this function.

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 0 - 255)
startRef	Start reference (Range: 1 - 65536)
float32Arr	Buffer with the data to be sent. The length of the array determines how many values are written (Range: 1-61).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readMultipleMod10000() `readMultipleMod10000 (`
`integer` *`slaveAddr,`*
`integer` *`startRef,`*
`integer [] int32Arr )` `[inherited]`

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.

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.

Remarks

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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

readInputMod10000() `readInputMod10000 (`
`integer` *`slaveAddr,`*

```
integer startRef,
integer [] int32Arr )  [inherited]
```

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

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.

Remarks

Modbus does not know about any other data type than discretes and 16-bit registers. Because an 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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

writeMultipleMod10000() `writeMultipleMod10000 (`

```
integer slaveAddr,
integer startRef,
integer [] int32Arr )  [inherited]
```

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.

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.

Remarks

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.

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).

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

Broadcast supported for serial protocols

readExceptionStatus()

```
readExceptionStatus (
 integer slaveAddr,
 byte & statusByte )  [inherited]
```

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

Reads the eight exception status coils within the slave device.

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.

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range

Exceptions

EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

returnQueryData() `returnQueryData (`
`integer` *slaveAddr*`,`
`byte [] queryArr,`
`byte [] echoArr ) [inherited]`

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

Parameters

slaveAddr	Modbus address of slave device or unit identifier (Range: 1 - 255)
queryArr	Data to be sent
echoArr	Buffer which will contain the data read. Array must be of the same size as queryArr.

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

restartCommunicationsOption() `restartCommunicationsOption (`
`integer` *slaveAddr*`,`
`boolean clearEventLog ) [inherited]`

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

Parameters

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

Exceptions

EIllegalStateError	Port or connection is closed
EInOutError	An I/O error occurred
EIllegalArgumentError	A parameter is out of range
EBusProtocolException	A protocol failure occurred. See descendants of EBusProtocolException for a more detailed failure list.

Note

No broadcast supported

setTimeout() `setTimeout (`
`const integer` *`timeOut`* `)` `[inherited]`

Configures time-out.

This function sets the operation or socket time-out to the specified value.

Remarks

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)

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

getTimeout() `integer getTimeout ( ) [inherited]`

Returns the time-out value.

Remarks

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.

Returns

Timeout value in ms

setPollDelay() `setPollDelay ( const integer pollDelay ) [inherited]`

Configures poll delay.

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

Remarks

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

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

getPollDelay() `integer getPollDelay ( ) [inherited]`

Returns the poll delay time.

Returns

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

setRetryCnt() `setRetryCnt ( const integer retryCnt ) [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

Exceptions

EIllegalStateError	Protocol is already open
EIllegalArgumentError	A parameter is out of range

getRetryCnt() `integer getRetryCnt ( ) [inherited]`

Returns the automatic retry count.

Returns

Retry count

getTotalCounter() `cardinal getTotalCounter ( ) [inherited]`

Returns how often a message transfer has been executed.

Returns

Counter value

getSuccessCounter() `cardinal getSuccessCounter ( ) [inherited]`

Returns how often a message transfer was successful.

Returns

Counter value

configureBigEndianInts() `[1/2] configureBigEndianInts ( ) [inherited]`

Configures int data type functions to do a word swap.

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 machine.

configureBigEndianInts() `[2/2] configureBigEndianInts ( integer slaveAddr ) [inherited]`

Enables int data type functions to do a word swap on a per slave basis.

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 machine.

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

configureSwappedFloats() `[1/2] configureSwappedFloats ( ) [inherited]`

Configures float data type functions to do a word swap.

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

Note

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

configureSwappedFloats() `[2/2] configureSwappedFloats ( integer slaveAddr ) [inherited]`

Enables float data type functions to do a word swap on a per slave basis.

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

Note

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.

Exceptions

EIllegalArgumentError	A parameter is out of range

configureLittleEndianInts() `[1/2] configureLittleEndianInts ( ) [inherited]`

Configures int data type functions *not* to do a word swap.

This is the default.

configureLittleEndianInts() `[2/2] configureLittleEndianInts ( integer slaveAddr ) [inherited]`

Disables word swapping for int data type functions on a per slave basis.

Modbus is using little-endian word order for 32-bit values. This setting assumes that the slave also serves 32-bit data in little little-endian word order.

Remarks

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.

Exceptions

EIllegalArgumentError	A parameter is out of range

configureIeeeFloats() [1/2] `configureIeeeFloats ( ) [inherited]`

Configures float data type functions *not* to do a word swap.

This is the default.

configureIeeeFloats() [2/2] `configureIeeeFloats ( integer slaveAddr ) [inherited]`

Disables float data type functions to do a word swap on a per slave basis.

Modbus is using little-endian word order for 32-bit values. This setting assumes that the slave also serves 32-bit floats in little little-endian word order which is the most common case.

Remarks

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.

Exceptions

EIllegalArgumentError	A parameter is out of range

configureStandard32BitMode() [1/2] `configureStandard32BitMode ( ) [inherited]`

Configures all slaves for Standard 32-bit Mode.

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

Remarks

This is the default mode

configureStandard32BitMode() [2/2] `configureStandard32BitMode ( integer slaveAddr ) [inherited]`

Configures a slave for Standard 32-bit Register Mode.

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

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

configureEnron32BitMode() `[1/2] configureEnron32BitMode ( ) [inherited]`

Configures all slaves for Daniel/ENRON 32-bit Mode.

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.

configureEnron32BitMode() `[2/2] configureEnron32BitMode ( integer slaveAddr ) [inherited]`

Configures all slaves for Daniel/ENRON 32-bit Mode.

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.

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

configureCountFromOne() `[1/2] configureCountFromOne ( ) [inherited]`

Configures the reference counting scheme to start with one for all slaves.

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

Remarks

This is the default mode

configureCountFromOne() `[2/2] configureCountFromOne ( integer slaveAddr ) [inherited]`

Configures a slave's reference counting scheme to start with one.

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

Parameters

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

Remarks

This is the default mode

Exceptions

EIllegalArgumentError	A parameter is out of range

configureCountFromZero() `[1/2] configureCountFromZero ( ) [inherited]`

Configures the reference counting scheme to start with zero for all slaves.

This renders the valid reference range to be 0 to 65535.

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

configureCountFromZero() `[2/2] configureCountFromZero ( integer slaveAddr ) [inherited]`

Configures a slave's reference counting scheme to start with zero.

This is also known as PDU addressing.

This renders the valid reference range to be 0 to 65535.

Parameters

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

Exceptions

EIllegalArgumentError	A parameter is out of range

isOpen() `boolean isOpen ( ) [inherited]`

Returns whether the protocol is open or not.

Return values

true	= open
false	= closed

getPackageVersion() `class string getPackageVersion ( ) [inherited]`

Returns the package version number.

Returns

Package version string

6.7.4 Member Data Documentation

hostName `string hostName [inherited]`

Host name property (eg '127.0.0.1')

Note

A protocol must be closed in order to configure it.

See also

fHostName For reading
fHostName For writing

port `word port [inherited]`

TCP port property (eg 502)

Note

A protocol must be closed in order to configure it.

Remarks

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 called *before* opening the connection with openProtocol().

See also

getPort For reading
setPort For writing

timeout `integer timeout [inherited]`

Time-out port property.

Note

A protocol must be closed in order to configure it.

See also

getTimeout For reading
setTimeout For writing

pollDelay `integer pollDelay [inherited]`

Poll delay property.

Delay between two Modbus read/writes in ms

Note

A protocol must be closed in order to configure it.

See also

getPollDelay For reading
setPollDelay For writing

retryCnt `integer retryCnt [inherited]`

Retry count property.

Note

A protocol must be closed in order to configure it.

See also

getRetryCnt For reading
setRetryCnt For writing

7 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.

---

8 Support

We provide electronic support and feedback for our FieldTalk products. Please use the Support web page at: `https://www.modbusdriver.com/support`

Your feedback is always welcome. It helps improving this product.

9 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.