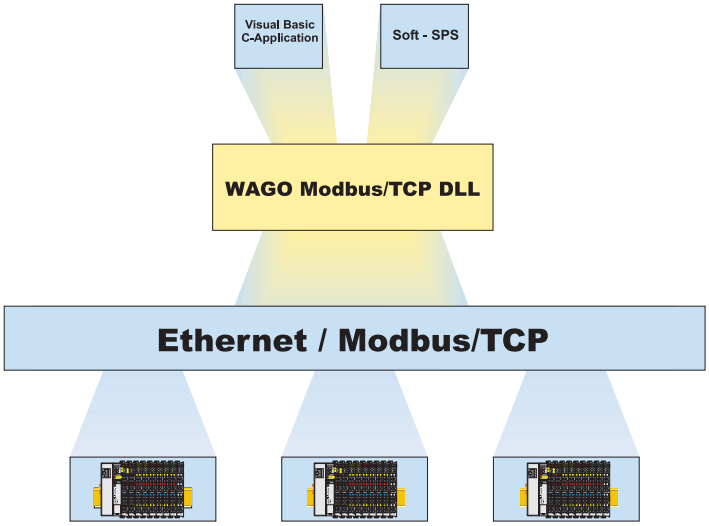
WAGO I/O SYSTEM 759

API Modbus/TCP DLL



Manual

Technical description

Version 1.2.0

WAGO®
INNOVATIVE CONNECTIONS

WAGO Kontakttechnik GmbH
Hansastraße 27
D-32423 Minden

Phone: +49 (0) 571/8 87 – 0
Fax: +49 (0) 571/8 87 – 1 69

E-Mail: info@wago.com

Web: http://www.wago.com

Technical Support
Phone: +49 (0) 571/8 87 – 5 55
Fax: +49 (0) 571/8 87 – 85 55

E-Mail: support@wago.com

Every conceivable measure has been taken to ensure the correctness and completeness of this documentation. However, as errors can never be fully excluded we would appreciate any information or ideas at any time.

E-Mail: documentation@wago.com

We wish to point out that the software and hardware terms as well as the trademarks of companies used and/or mentioned in the present manual are generally trademark or patent protected.

TABLE OF CONTENTS

1 Important Comments

To ensure fast installation and start-up of the units described in this manual, we strongly recommend that the following information and explanations are carefully read and abided by.

1.1 Legal Principles

1.1.1 Copyright

This manual is copyrighted, together with all figures and illustrations contained therein. Any use of this manual which infringes the copyright provisions stipulated herein, is not permitted. Reproduction, translation and electronic and photo-technical archiving and amendments require the written consent of WAGO Kontakttechnik GmbH. Non-observance will entail the right of claims for damages.

WAGO Kontakttechnik GmbH reserves the right to perform modifications allowed by technical progress. In case of grant of a patent or legal protection of utility patents all rights are reserved by WAGO Kontakttechnik GmbH. Products of other manufacturers are always named without referring to patent rights. The existence of such rights can therefore not be ruled out.

1.1.2 Personnel Qualification

The use of the product detailed in this manual is exclusively geared to specialists having qualifications in PLC programming, electrical specialists or persons instructed by electrical specialists who are also familiar with the valid standards. WAGO Kontakttechnik GmbH declines all liability resulting from improper action and damage to WAGO products and third party products due to non-observance of the information contained in this manual.

1.1.3 Intended Use

For each individual application, the components supplied are to work with a dedicated hardware and software configuration. Modifications are only permitted within the framework of the possibilities documented in the manuals. All other changes to the hardware and/or software and the non-conforming use of the components entail the exclusion of liability on part of WAGO Kontakttechnik GmbH.

Please direct any requirements pertaining to a modified and/or new hardware or software configuration directly to WAGO Kontakttechnik GmbH.

1.2 Symbols

Danger
Always abide by this information to protect persons from injury.

Warning
Always abide by this information to prevent damage to the device.

Attention
Marginal conditions must always be observed to ensure smooth operation.

ESD (Electrostatic Discharge)
Warning of damage to the components by electrostatic discharge. Observe the precautionary measure for handling components at risk.

Note
Routines or advice for efficient use of the device and software optimization.

More information
References on additional literature, manuals, data sheets and INTERNET pages

1.3 Font Conventions

Italic	Names of path and files are marked italic i.e.: *C:\programs\WAGO-IO-CHECK*
Italic	Menu items are marked as bold italic i.e.: ***Save***
\	A backslash between two names marks a sequence of menu items i.e.: ***File\New***
END	Press buttons are marked as bold with small capitals i.e.: **ENTER**
< >	Keys are marked bold within angle brackets i.e.: **<F5>**
`Courier`	Program code is printed with the font `Courier`. i.e.: `END_VAR`

1.4 Number Notation

Number Code	Example	Note
Decimal	100	normal notation
Hexadecimal	0x64	C notation
Binary	'100' '0110.0100'	Within ', Nibble separated with dots

1.5 Scope of Validity

Item no.	Description
759-312	API Modbus/TCP DLL

1.6 Abbreviations

AI	Analog Input
AO	Analog Output
DI	Digital Input
DO	Digital Output
I/O	Input/Output

2 Modbus/TCP DLL

2.1 Overview

The DLL implements the Modbus/TCP Protocol.

The Modbus/TCP DLL supports the operating systems Windows NT 4.0 (from version SP5), Windows 2000, Windows 95 (with Windows Socket 2.0 Update), Windows 98.

It is used in the Windows Socket 2.0 interface for the TCP/IP communication contained in the Windows system.

The DLL supports synchronous and asynchronous reading and writing of values.

TCP or UDP can be selected optionally as a transport protocol.

It can be used by the programming languages C and Visual BASIC.

Only synchronous calls of the DLL are supported for calls from Visual BASIC.

Both synchronous and asynchronous calls are supported for calls of the DLL from C.

This library only supports the commands FC1, FC2, FC3, FC4, FC7, FC15 and FC16 from Open Modbus/TCP protocol V1.0.

The values of the Modbus/TCP tables, which can be read or written with these commands, are listed in the following table:

	Read	**Write**
Output Register	FC 3	FC 16
Input Register	FC 4	
Output Coil	FC 1	FC 15
Input Coil	FC 2	
Exception Status	FC 7	

The library takes care that only one command can be processed simultaneously for each connection to a Modbus/TCP device.

A time interval can be specified for each open connection after which the I/O inquiry is interrupted.

The DLL was developed with the Microsoft Visual C++ 6.0 development environment. All modules of the DLL are translated as ASCII components with a statically linked C runtime.

2.2 Installation

The file MBT.DLL should be copied into the directory \system32 in the Windows standard directory. If another directory is selected, then the entry must be adapted correspondingly in the environmental variables for the path in the Windows system control.

2.3 Functions

All functions of the MBT library have return values corresponding to the HRESULT format. The functions of the socket APIs do not return any return values of this format. The MBT library converts these return values by means of the macro HRESULT_FROM_WIN32. In the following description this is indicated by means of "HR from".

The following functions are contained in the Modbus/TCP.DLL:

MBTInit
MBTExit
MBTConnect
MBTDisconnect
MBTReadRegisters
MBTReadCoils
MBTReadExeptionStatus
MBTReadCompleted
MBTWriteRegisters
MBTWriteCoils
MBTWriteCompleted
MBTSwapWord
MBTSwapDWord
MODBUSTCP_TABLE_xxx

2.3.1 MBTInit

WAGO-I/O-*PRO* 32 Elements of Dynamic Link Library		
Category:	This function initializes the Modbus/TCP library.	
Name:	MBTInit	
Type:	Function	
Name of DLL:	ModbusTCP.DLL	
Applicable for:	Modbus/TCP-Protocol	
Declaration:		
C: LONG MBTInit(void); VB: Public Declare Function MBTInit Lib "MBT" () As Long		
Parameter:	**Comment:**	
none		
Return value:	**HEX-value:**	**Comment:**
S_OK	0	MBT library was initialized successfully.
MBT_THREAD_ CREATION_ERROR	0xEF010000	The work thread of the MBT library could not be created.
HR from WSASYSNOTREADY	0x8007276B	The network subsystem is not ready for network communication.
HR from WSAVERNOT SUPPORTED	0x8007276C	The Windows socket version 2 required is not provided by the system.
HR from WSAEINPROGRESS	0x80072734	An obstructive Windows socket 1.1 operation is in progress.
HR from WSAEPROCLIM	0x80072753	The maximum number of threads supported by the socket implementation has been reached.
Remark:		
The other functions of the MBT library can only be used after calling the function MBTInit.		

2.3.2 MBTExit

WAGO-I/O-*PRO* 32 Elements of Dynamic Link Library		
Category:	This function terminates the Modbus/TCP library.	
Name:	MBTExit	
Type:	Function	
Name of DLL:	ModbusTCP.DLL	
Applicable for:	Modbus/TCP-Protocol	
Declaration:		
C: LONG MBTExit(void); VB: Public Declare Function MBTExit Lib "MBT" () As Long		
Parameter:	**Comment:**	
none		
Return value:	**HEX-value:**	**Comment:**
S_OK	0	MBT library was initialized successfully.
MBT_EXIT_TIMEOUT_ ERROR	0xEF010001	The work-thread of the MBT library could not be terminated within the timeout.
MBT_UNKNOWN_ THREAD_EXIT_ERROR	0xEF010002	The work-thread of the MBT library has terminated with an error code.
Remark:		
The function MBTExit must be called for a controlled termination of the Modbus/TCP library.		

2.3.3 MBTConnect

WAGO-I/O-*PRO* 32 Elements of Dynamic Link Library	
Category:	This function creates a connection to the port of the device that is to be controlled.
Name:	MBTConnect
Type:	Function
Name of DLL:	ModbusTCP.DLL
Applicable for:	Modbus/TCP-Protocol

Declaration:

```
C:      LONG MBTConnect(
                IN LPCTSTR szHostAddress,
                IN WORD port,
                IN BOOL useTCPorUDP,
                IN DWORD requestTimeout,
                OUT HANDLE *hSocket);

VB:     Public Declare Function MBTConnect Lib "MBT" (
                ByVal szHostAddress As String,
                ByVal port As Integer,
                ByVal useTCPorUDP As Long,
                ByVal requestTimeout As Long,
                hSocket As Long) As Long
```

Parameter:	Comment:
szHostAddress	IP-address (e.g. "172.17.5.91") or DNS name of the Modbus/TCP device
port	IP-port by which the communication with the device is to be controlled
useTCPorUDP	Communication via TCP (TRUE) or UDP (FALSE)
requestTimeout	Timeout in milliseconds for an I/O operation to the device
hSocket	Handle for the connection created

Return value:	HEX-value:	Comment:
S_OK	0	The connection has been created.
MBT_NO_ENTRY_ ADDABLE_ERROR	0xEF010004	Error in the socket-administration of the library.
HR from WSANOTINITIALISED	0x8007276D	MBTInit was not called.
HR from WSAENETDOWN	0x80072742	Error in the network subsystem.
HR from WSANO_ RECOVERY	0x80072AFB	A fatal error has occurred.

HR from WSAHOST_NOT_FOUND	0x80072AF9	The device has not been found for the DNS name transferred or it is an invalid IP address.
HR from WSATRY_AGAIN	0x80072AFA	Error in the DNS server.
HR from WSAEPROTONOSUPPORT WSAEAFNOSUPPORT	0x8007273B 0x8007273F	The TCP or UPD protocol is not supported.
HR from WSAEMFILE	0x80072728	The system does not have any more socket descriptors available.
HR from WSAEADDRNOTAVAIL	0x80072741	The remote address is invalid.
HR from WSAECONNREFUSED	0x8007274D	The connection attempt has been refused.
HR from WSAETIMEDOUT	0x8007274C	Timeout during the connection set-up.

Remark:

This function creates a Windows socket object. This is connected to the port of the Modbus/TCP device for the TCP protocol. The port of the device is stored in the library in the UDP protocol.

The values of the device can first be read and written after a connection has been established with a device.

2.3.4 MBTDisconnect

WAGO-I/O-*PRO* 32 Elements of Dynamic Link Library		
Category:	This function disconnects the connection to a device.	
Name:	MBTDisconnect	
Type:	Function	
Name of DLL:	ModbusTCP.DLL	
Applicable for:	Modbus/TCP-Protocol	
Declaration:		

```
C:      LONG MBTDisconnect(
                IN HANDLE hSocket);

VB:     Public Declare Function MBTDisonnect Lib "MBT" (
                ByVal hSocket As Long) As Long
```

Parameter:	**Comment:**
hSocket	Handle of the connection to the device

Return value:	**HEX-value:**	**Comment:**
S_OK	0	The device has been disconnected.
MBT_HANDLE_INVALID_ERROR	0xEF010006	The handle submitted is invalid.
HR from WSANOTINITIALISED	0x8007276D	MBTInit was not called.
HR from WSAENETDOWN	0x80072742	Error in the network subsystem.

Remark:

The function MBTDisconnect must be called to terminate the connection to a device in a controlled way.

2.3.5 MBTReadRegisters

WAGO-I/O-*PRO* 32 Elements of Dynamic Link Library	
Category:	This function reads data from Registers.
Name:	MBTReadRegisters
Type:	Function
Name of DLL:	ModbusTCP.DLL
Applicable for:	Modbus/TCP-Protocol

Declaration:

```
C:      LONG MBTReadRegisters(
                IN HANDLE hSocket,
                IN BYTE tableType,
                IN WORD dataStartAddress,
                IN WORD numWords,
                OUT LPBYTE pReadBuffer,
                OPTIONAL IN MBTReadCompleted fpReadCompletedCallback
                OPTIONAL IN DWORD callbackContext);

VB:     Public Declare Function MBTReadRegisters Lib "MBT" (
                ByVal hSocket As Long,
                ByVal tableType As Byte,
                ByVal dataStartAddress As Integer,
                ByVal numWords As Integer,
                pReadBuffer As Any,
                ByVal fpReadCompletedCallback As Long,
                ByVal callbackContext As Long) As Long
```

Parameter:	Comment:
hSocket	Handle of the connection to the device
tableType	Modbus/TCP tables Type (MODBUS_TABLE_xxx) Applicable here: Input register or output register
dataStartAddress	Start address of the register to be read
numWords	Number of registers to be read
pReadBuffer	Storage area into which the registers are to be read. NULL must be transferred for an asynchronous call.
FpReadCompletedCallback	C-callback function, which is called after completing an asynchronous read function. NULL must be transferred for a synchronous call (default).
callbackContext	Context that is also transferred to the asynchronous callback function. 0 must be transferred for a synchronous call (default).

Return value:	HEX-value:	Comment:
S_OK	0	The device has been disconnected.

MBT_HANDLE_INVALID_ ERROR	0xEF010006	The handle submitted is invalid.
MBT_NO_JOB_ ADDABLE_ERROR	0xEF010005	Error in the job administration of the library.
MBT_EXIT_ERROR	0xEF01000B	The I/O job has been aborted by an MBTExit Call.
MBT_SOCKET_ TIMEOUT_ERROR	0xEF010008	A Modbus/TCP telegram for the I/O job has been sent but no response has been received within the Request Timeout..
HR from WSANOTINITIALISED	0x8007276D	MBTInit was not called.
HR from WSAENETDOWN	0x80072742	Error in the network subsystem.
HR from WSAENETRESET WSAECONNABORTED WSAEDISCON	0x80072744 0x80072745 0x80072746	The connection to the device has been aborted.
HR from WSAEWOULDBLOCK	0x80072733	Too many I/O operations are outstanding at the moment.
HR from WSAEFAULT	0x8007271E	The read buffer does not refer to any valid storage area.
HR from WSAENOBUFS	0x80072747	An operation on a socket could not be performed because the system lacked sufficient buffer space or because a queue was full.

Remark:

The parameter fpReadCompletedCallback decides whether the function is executed asynchronously or synchronously. If this parameter is NULL then the function is executed synchronously, otherwise it is executed asynchronously. After terminating the asynchronous I/O operation, the callback function transferred is called with the result of the I/O operation. In the case of an asynchronous call of the function, a buffer (pReadBuffer) transferred is ignored and the result is transferred to a storage area of the DLL.

The Modbus/TCP command FC3 is used for reading output registers and the FC4 command is used for the input registers

2.3.6 MBTReadCoils

WAGO-I/O-*PRO* 32 Elements of Dynamic Link Library	
Category:	This function reads data from Coils.
Name:	MBTReadCoils
Type:	Function
Name of DLL:	ModbusTCP.DLL
Applicable for:	Modbus/TCP-Protocol

Declaration:

```
C:      LONG MBTReadCoils(
                IN HANDLE hSocket,
                IN BYTE tableType
                IN WORD dataStartAddress,
                IN WORD numBits,
                OUT LPBYTE pReadBuffer,
                OPTIONAL IN MBTReadCompleted fpReadCompletedCallback
                OPTIONAL IN DWORD callbackContext);

VB:     Public Declare Function MBTReadCoils Lib "MBT" (
                ByVal hSocket As Long,
                ByVal tableType As Byte,
                ByVal dataStartAddress As Integer,
                ByVal numBits As Integer,
                pReadBuffer As Any,
                ByVal fpReadCompletedCallback As Long,
                ByVal callbackContext As Long) As Long
```

Parameter:	Comment:
hSocket	Handle of the connection to the device
tableType	Modbus/TCP tables Typ (MODBUS_TABLE_xxx) Applicable here: Input coil or output coil
dataStartAddress	Start address of the coils to be read
numBits	Number of coils to be read
pReadBuffer	Storage area into which the coils are to be read. NULL must be transferred for an asynchronous call.
fpReadCompletedCallback	C-callback function, which is called after completing an asynchronous read function. NULL must be transferred for a synchronous call (default).
callbackContext	Context that is also transferred to the asynchronous callback function. 0 must be transferred for a synchronous call (default).

Return value:	HEX-value:	Comment:
S_OK	0	The device has been disconnected.

MBT_HANDLE_INVALID_ ERROR	0xEF010006	The handle submitted is invalid.
MBT_NO_JOB_ ADDABLE_ERROR	0xEF010005	Error in the job administration of the library.
MBT_EXIT_ERROR	0xEF01000B	The I/O job has been aborted by an MBTExit Call.
MBT_SOCKET_ TIMEOUT_ERROR	0xEF010008	A Modbus/TCP telegram for the I/O job has been sent but no response has been received within the Request Timeout..
HR from WSANOTINITIALISED	0x8007276D	MBTInit was not called.
HR from WSAENETDOWN	0x80072742	Error in the network subsystem.
HR from WSAENETRESET WSAECONNABORTED WSAEDISCON	0x80072744 0x80072745 0x80072746	The connection to the device has been aborted.
HR from WSAEWOULDBLOCK	0x80072733	Too many I/O operations are outstanding at the moment.
HR from WSAEFAULT	0x8007271E	The read buffer does not refer to any valid storage area.
HR from WSAENOBUFS	0x80072747	An operation on a socket could not be performed because the system lacked sufficient buffer space or because a queue was full.

Remark:

The parameter fpReadCompletedCallback decides whether the function is executed asynchronously or synchronously. If this parameter is NULL then the function is executed synchronously, otherwise it is executed asynchronously. After terminating the asynchronous I/O operation, the callback function transferred is called with the result of the I/O operation. In the case of an asynchronous call of the function, a buffer (pReadBuffer) transferred is ignored and the result is transferred to a storage area of the DLL.

The Modbus/TCP command FC1 is used for reading output coils and the FC2 command is used for the input registers.

2.3.7 MBTReadExceptionStatus

WAGO-I/O-*PRO* 32 Elements of Dynamic Link Library	
Category:	This function reads data from Coils.
Name:	MBTReadExceptionStatus
Type:	Function
Name of DLL:	ModbusTCP.DLL
Applicable for:	Modbus/TCP-Protocol

Declaration:

```
C:      LONG MBTReadCoils(
               IN HANDLE hSocket,
               OUT LPBYTE pReadBuffer,
               OPTIONAL IN MBTReadCompleted fpReadCompletedCallback
               OPTIONAL IN DWORD callbackContext);

VB:     Public Declare Function MBTReadCoils Lib "MBT" (
               ByVal hSocket As Long,
               pReadBuffer As Any,
               ByVal fpReadCompletedCallback As Long,
               ByVal callbackContext As Long) As Long
```

Parameter:	Comment:
hSocket	Handle of the connection to the device
pReadBuffer	Storage area into which the coils are to be read. NULL must be transferred for an asynchronous call.
fpReadCompletedCallback	C-callback function, which is called after completing an asynchronous read function. NULL must be transferred for a synchronous call (default).
callbackContext	Context that is also transferred to the asynchronous callback function. 0 must be transferred for a synchronous call (default).

Return value:	HEX-value:	Comment:
S_OK	0	The device has been disconnected.
MBT_HANDLE_INVALID_ ERROR	0xEF010006	The handle submitted is invalid
MBT_NO_JOB_ ADDABLE_ERROR	0xEF010005	Error in the job administration of the library.
MBT_EXIT_ERROR	0xEF01000B	The I/O job has been aborted by an MBTExit Call.
MBT_SOCKET_ TIMEOUT_ERROR	0xEF010008	A Modbus/TCP telegram for the I/O job has been sent but no response has been received within the Request Timeout..

HR from WSANOTINITIALISED	0x8007276D	MBTInit was not called.
HR from WSAENETDOWN	0x80072742	Error in the network subsystem.
HR from WSAENETRESET WSAECONNABORTED WSAEDISCON	0x80072744 0x80072745 0x80072746	The connection to the device has been aborted.
HR from WSAEWOULDBLOCK	0x80072733	Too many I/O operations are outstanding at the moment.
HR from WSAEFAULT	0x8007271E	The read buffer does not refer to any valid storage area.
HR from WSAENOBUFS	0x80072747	An operation on a socket could not be performed because the system lacked sufficient buffer space or because a queue was full.

Remark:

The parameter fpReadCompletedCallback decides whether the function is executed asynchronously or synchronously. If this parameter is NULL then the function is executed synchronously, otherwise it is executed asynchronously. After terminating the asynchronous I/O operation the Callback Function transferred is called with the result of the I/O operation. In the case of an asynchronous call of the function, a buffer (pReadBuffer) transferred is ignored and the result transferred to a storage area of the DLL.

The Modbus/TCP command FC7 is used for reading the exception status.

2.3.8 MBTReadCompleted

WAGO-I/O-*PRO* 32 Elements of Dynamic Link Library		
Category:	This callback function is called after completing an asynchronous read function.	
Name:	MBTReadCompleted	
Type:	Function	
Name of DLL:	ModbusTCP.DLL	
Applicable for:	Modbus/TCP-Protocol	
Declaration:		

```
C:      void MBTReadCompleted(
                IN HANDLE hSocket,
                IN DWORD callbackContext,
                IN LONG errorCode,
                IN BYTE tableType,
                IN WORD dataStartAddress,
                IN WORD numRead,
                IN WORD numBytes,
                IN LPBYTE pReadBuffer);
```

Parameter:	**Comment:**
hSocket	Handle of the connection to the device
callbackContext	Context that was submitted during the asynchronous call of the function
errorCode	Result of the read operation
tableType	Modbus/TCP tables Typ (MOBUSTCP_TABLE_xxx)
dataStartAddress	Start address of the registers or coils to be read
numRead	The number of registers or coils to be read
numBytes	The number of bytes read
pReadBuffer	Values of the bytes read

Return value:	**HEX-value:**	**Comment:**
none		

Remark:

2.3.9 MBTWriteRegisters

WAGO-I/O-*PRO* 32 Elements of Dynamic Link Library	
Category:	This function writes data to Registers.
Name:	MBTWriteRegisters
Type:	Function
Name of DLL:	ModbusTCP.DLL
Applicable for:	Modbus/TCP-Protocol

Declaration:

```
C:      LONG MBTWriteRegisters(
                IN HANDLE hSocket,
                IN WORD dataStartAddress,
                IN WORD numWords,
                IN LPBYTE pWriteBuffer,
                OPTIONAL IN MBTWriteCompleted fpWriteCompletedCallback
                OPTIONAL IN DWORD callbackContext);

VB:     Public Declare Function MBTWriteRegisters Lib "MBT" (
                ByVal hSocket As Long,
                ByVal dataStartAddress As Integer,
                ByVal numWords As Integer,
                pWriteBuffer As Any,
                ByVal fpWriteCompletedCallback As Long,
                ByVal callbackContext As Long) As Long
```

Parameter:	Comment:
hSocket	Handle of the connection to the device
dataStartAddress	Start address of the registers to be written
numWords	Number of registers to be written
pWriteBuffer	Storage area with the values of the registers to be written
fpWriteCompletedCallback	C-callback function called after the completion of an asynchronous writing function. NULL must be transferred for a synchronous call (default).
callbackContext	Context that is also transferred to the asynchronous callback function. 0 must be transferred for a synchronous call (default).

Return value:	HEX-value:	Comment:
S_OK	0	The device has been disconnected.
MBT_HANDLE_INVALID_ERROR	0xEF010006	The handle submitted is invalid.
MBT_NO_JOB_ADDABLE_ERROR	0xEF010005	Error in the job administration of the library.

MBT_EXIT_ERROR	0xEF01000B	The I/O job has been aborted by an MBTExit Call.
MBT_SOCKET_ TIMEOUT_ERROR	0xEF010008	A Modbus/TCP telegram for the I/O job has been sent but no response has been received within the Request Timeout..
HR from WSANOTINITIALISED	0x8007276D	MBTInit was not called.
HR from WSAENETDOWN	0x80072742	Error in the network subsystem.
HR from WSAENETRESET WSAECONNABORTED WSAEDISCON	0x80072744 0x80072745 0x80072746	The connection to the device has been aborted.
HR from WSAEWOULDBLOCK	0x80072733	Too many I/O operations are outstanding at the moment.
HR from WSAEFAULT	0x8007271E	The write buffer does not refer to any valid storage area.
HR from WSAENOBUFS	0x80072747	An operation on a socket could not be performed because the system lacked sufficient buffer space or because a queue was full.

Remark:

The parameter fpWriteCompletedCallback decides whether the function is executed asynchronously or synchronously. If this parameter is NULL then the function is executed synchronously, otherwise it is executed asynchronously. After terminating the asynchronous I/O operation the callback function transferred is called with the result of the I/O operation.

The Modbus/TCP command FC16 is used for writing output registers.

2.3.10 MBTWriteCoils

WAGO-I/O-*PRO* 32 Elements of Dynamic Link Library	
Category:	This function writes data to Coils.
Name:	MBTWriteCoils
Type:	Function
Name of DLL:	ModbusTCP.DLL
Applicable for:	Modbus/TCP-Protocol

Declaration:

```
C:      LONG MBTWriteCoilss(
                IN HANDLE hSocket,
                IN WORD dataStartAddress,
                IN WORD numBits,
                IN LPBYTE pWriteBuffer,
                OPTIONAL IN MBTWriteCompleted fpWriteCompletedCallback
                OPTIONAL IN DWORD callbackContext);

VB:     Public Declare Function MBTWriteCoils Lib "MBT" (
                ByVal hSocket As Long,
                ByVal dataStartAddress As Integer,
                ByVal numBits As Integer,
                pWriteBuffer As Any,
                ByVal fpWriteCompletedCallback As Long,
                ByVal callbackContext As Long) As Long
```

Parameter:	Comment:
hSocket	Handle of the connection to the device
dataStartAddress	Start address of the coils to be written
numBits	Number of coils to be written
pWriteBuffer	Storage area with the values of the coils to be written
fpWriteCompletedCallback	C-callback function called after the completion of an asynchronous writing function. NULL must be transferred for a synchronous call (default).
callbackContext	Context that is also transferred to the asynchronous callback function. 0 must be transferred for a synchronous call (default).

Return value:	HEX-value:	Comment:
S_OK	0	The device has been disconnected.
MBT_HANDLE_INVALID_ERROR	0xEF010006	The handle submitted is invalid.
MBT_NO_JOB_ADDABLE_ERROR	0xEF010005	Error in the job administration of the library.

MBT_EXIT_ERROR	0xEF01000B	The I/O job has been aborted by an MBTExit Call.
MBT_SOCKET_ TIMEOUT_ERROR	0xEF010008	A Modbus/TCP telegram for the I/O job has been sent but no response has been received within the Request Timeout..
HR from WSANOTINITIALISED	0x8007276D	MBTInit was not called.
HR from WSAENETDOWN	0x80072742	Error in the network subsystem.
HR from WSAENETRESET WSAECONNABORTED WSAEDISCON	0x80072744 0x80072745 0x80072746	The connection to the device has been aborted.
HR from WSAEWOULDBLOCK	0x80072733	Too many I/O operations are outstanding at the moment.
HR from WSAEFAULT	0x8007271E	The write buffer does not refer to any valid storage area.
HR from WSAENOBUFS	0x80072747	An operation on a socket could not be performed because the system lacked sufficient buffer space or because a queue was full.

Remark:

The parameter fpWriteCompletedCallback decides whether the function is executed asynchronously or synchronously. If this parameter is NULL, then the function is executed synchronously, otherwise it is executed asynchronously. After terminating the asynchronous I/O operation, the callback function transferred is called with the result of the I/O operation.

The Modbus/TCP command FC15 is used for writing output registers.

2.3.11 MBTWriteCompleted

WAGO-I/O-*PRO* 32 Elements of Dynamic Link Library		
Category:	This callback function is called after completing an asynchronous writing function.	
Name:	MBTWriteCompleted	
Type:	Function	
Name of DLL:	ModbusTCP.DLL	
Applicable for:	Modbus/TCP-Protocol	
Declaration:		

```
C:      void MBTWriteCompleted(
                IN HANDLE hSocket,
                IN DWORD callbackContext,
                IN LONG errorCode,
                IN BYTE tableType,
                IN WORD dataStartAddress,
                IN WORD numWrite,
                IN LPBYTE pWriteBuffer);
```

Parameter:	**Comment:**
hSocket	Handle of the connection to the device
callbackContext	Kontext, der beim asynchronen Aufruf der Funktion mitgegeben wurde
errorCode	Result of the writing process
tableType	Modbus/TCP tables Typ (MOBUSTCP_TABLE_xxx)
dataStartAddress	Start address of the registers or coils to be written
numWrite	Number of registers or coils to be written
pWriteBuffer	Value of the registers or coils to be written

Return value:	**HEX-value:**	**Comment:**
none		

Remark:

2.3.12 MBTSwapWord

WAGO-I/O-*PRO* 32 Elements of Dynamic Link Library	
Category:	This function swaps the lower and upper byte of the argument of type WORD.
Name:	MBTSwapWord
Type:	Function
Name of DLL:	ModbusTCP.DLL
Applicable for:	Modbus/TCP-Protocol

Declaration:

```
C:      WORD MBTSwapWord(
                const WORD wData);

VB:     Public Declare Function MBTSwapWord Lib "MBT" (
                ByVal wData As Integer) As Integer
```

Parameter:	**Comment:**
wData	Word that should be swapped

Return value:	**Comment:**
	This function returns the swapped value.

Remark:

2.3.13 MBTSwapDWord

WAGO-I/O-*PRO* 32 Elements of Dynamic Link Library	
Category:	This function swaps the 1st and 4th and the 2nd and 3rd byte of the argument of type DWord.
Name:	MBTSwapDWord
Type:	Function
Name of DLL:	ModbusTCP.DLL
Applicable for:	Modbus/TCP-Protocol

Declaration:

```
C:      DWORD MBTSwapDWord(
                const DWORD dwData);

VB:     Public Declare Function MBTSwapDWord Lib "MBT" (
                ByVal dwData As Long) As Long
```

Parameter:	**Comment:**
dwData	DWord that should be swapped

Return value:	**Comment:**
	This function returns the swapped value.

Remark:

2.3.14 MODBUSTCP_TABLE_xxx

The Define MODBUSTCP_TABLE_xxx determines the Modbus/TCP table on which an action should be executed.

	Define	Value
Output Register	MODBUSTCP_TABLE_OUTPUT_REGISTER	4
Input Register	MODBUSTCP_TABLE_INPUT_REGISTER	3
Output Coil	MODBUSTCP_TABLE_OUTPUT_COIL	0
Input Coil	MODBUSTCP_TABLE_INPUT_COIL	1
Exception Status	MODBUSTCP_TABLE_EXCEPTION_STATUS	7

2.4 Examples

2.4.1 VBA

2.4.1.1 Description

The basis for the control with Visual BASIC is a form with the control elements btnReadCoil, btnWriteCoil, btnReadRegister and btnWriteRegister of the type Button as well as the control elements AdrCoil, edtReadCoil, edtWriteCoil, AdrRegister, edtReadRegister and edtWriteRegister of the type entry field.

The Modbus/TCP DLL is initialized and a connection is set up to the node by opening the form.

The address to be activated is entered in the entry field AdrCoil or AdrRegister.
After clicking the button btnReadCoil or btnReadRegister, the status entry in the entry field edtReadCoil or edtReadRegister is displayed.

After clicking the button btnWriteCoil or BtnWriteRegister, the value entered in the entry field edtWriteCoil or edtWriteRegister is written for the output.
The event procedures are called by the event "Click" of the buttons.

The connection to the node is disconnected and the Modbus/TCP DLL is terminated by closing the form.

2.4.1.2 Declaration of Constants and Functions of the DLL

constants

```
Const MODBUSTCP_TABLE_OUTPUT_REGISTER = 4
Const MODBUSTCP_TABLE_INPUT_REGISTER = 3
Const MODBUSTCP_TABLE_OUTPUT_COIL = 0
Const MODBUSTCP_TABLE_INPUT_COIL = 1
Const MODBUSTCP_TABLE_EXCEPTION_STATUS = 7
```

DLL-functions

```
Public Declare Function MBTInit Lib "MBT" () As Long
Public Declare Function MBTExit Lib "MBT" () As Long
Public Declare Function MBTConnect Lib "MBT" (ByVal szHostAddress As String, ByVal port As Integer, ByVal useTCPorUDP As Long, ByVal requestTimeout As Long, hSocket As Long) As Long
Public Declare Function MBTDisconnect Lib "MBT" (ByVal hSocket As Long) As Long
Public Declare Function MBTReadRegisters Lib "MBT" (ByVal hSocket As Long, ByVal tableType As Byte, ByVal dataStartAddress As Integer, ByVal numWords As Integer, pReadBuffer As Any, ByVal fpReadCompletedCallback As Long, ByVal callbackContext As Long) As Long
Public Declare Function MBTReadCoils Lib "MBT" (ByVal hSocket As Long, ByVal tableType As Byte, ByVal dataStartAddress As Integer, ByVal numBits As Integer, pReadBuffer As Any, ByVal fpReadCompletedCallback As Long, ByVal callbackContext As Long) As Long
Public Declare Function MBTReadExceptionStatus Lib "MBT" (ByVal hSocket As Long, pExceptionStatus As Byte, ByVal fpReadCompletedCallback As Long, ByVal callbackContext As Long) As Long
Public Declare Function MBTWriteRegisters Lib "MBT" (ByVal hSocket As Long, ByVal dataStartAddress As Integer, ByVal numWords As Integer, pWriteBuffer As Any, ByVal fpWriteCompletedCallback As Long, ByVal callbackContext As Long) As Long
```

```
Public Declare Function MBTWriteCoils Lib "MBT" (ByVal hSocket As Long, ByVal
dataStartAddress As Integer, ByVal numBits As Integer, pWriteBuffer As Any, ByVal
fpWriteCompletedCallback As Long, ByVal callbackContext As Long) As Long
Public Declare Function MBTSwapWord Lib "MBT" (ByVal wData As Integer) As Integer
Public Declare Function MBTSwapDWord Lib "MBT" (ByVal dwData As Long) As Long
```

2.4.1.3 Declaration of Variables and Event-Procedures of the Form

globale variables und constants

```
Const g_MBusIP As String = "172.17.5.91"
Const g_Port As Long = 502
Dim g_hSocket As Long
```

btnReadCoil

```
Private Sub btnReadCoil_Click()
Dim inpVal As Byte
Dim adr As Integer
    inpVal = 0
    adr = CInt(AdrCoil.Text)
    ret = MBTReadCoils(g_hSocket, _
                    MODBUSTCP_TABLE_OUTPUT_COIL, _
                    adr, _
                    1, _
                    inpVal, _
                    0, _
                    0)
    edtReadCoil.Text = inpVal
End Sub
```

btnReadRegister

```
Private Sub btnReadRegister_Click()
Dim inpVal As Integer
Dim adr As Integer
    inpVal = 0
    adr = CInt(AdrRegister.Text)
    ret = MBTReadRegisters(g_hSocket, _
                    MODBUSTCP_TABLE_OUTPUT_REGISTER, _
                    adr, _
                    1, _
                    inpVal, _
                    0, _
                    0)
    inpVal = MBTSwapWord(inpVal)
    edtReadRegister.Text = inpVal
End Sub
```

btnWriteCoil

```
Private Sub btnWriteCoil_Click()
Dim outVal As Byte
Dim adr As Integer
    outVal = CByte(edtWriteCoil.Text)
    adr = CInt(AdrCoil.Text)
    ret = MBTWriteCoils(g_hSocket, _
                     adr, _
                     1, _
                     outVal, _
                     0, _
                     0)
End Sub
```

btnWriteRegister

```
Private Sub btnWriteRegister_Click()
Dim outVal As Integer
Dim adr As Integer
    outVal = CInt(edtWriteRegister.Text)
    outVal = MBTSwapWord(outVal)
    adr = CInt(AdrRegister.Text)
    ret = MBTWriteRegisters(g_hSocket, _
                    adr, _
                    1, _
                    outVal, _
                    0, _
                    0)
End Sub
```

form

```
Private Sub Form_Load()
Dim ret As Long
    MBTInit
    ret = MBTConnect(g_MBusIP, g_Port, True, 1000, g_hSocket)
    If (ret <> 0) Then
        MBTExit
        MsgBox "Couldn't connect to MB Device: 0x" & Hex(ret)
        Unload Me
        Exit Sub
    End If
End Sub

Private Sub Form_Unload(Cancel As Integer)
    MBTDisconnect (g_hSocket)
    MBTExit
End Sub
```

2.4.2 C

2.4.2.1 Description

In the following program example in C, the DLL is initialized with the function MBTInit and a connection is then set up with MBTConnect.

After successfully setting up the connection, the functions MBTWriteRegisters, MBTReadRegisters, MBTWriteCoils, MBTReadCoils and MBTReadExceptionStatus for writing and reading the outputs and inputs are called asynchronously. Following this the writing and reading functions are called synchronously.
The status is displayed on the monitor after calling each function.

At the end of the program, the connection is disconnected with the function MBTDisconnect and the DLL is then terminated with MBTExit.

2.4.2.2 Interface

```
#ifndef _MBT_H_
#define _MBT_H_

#ifdef __cplusplus
extern "C"
{
#endif


#define MODBUSTCP_TABLE_OUTPUT_REGISTER   4
#define MODBUSTCP_TABLE_INPUT_REGISTER    3
#define MODBUSTCP_TABLE_OUTPUT_COIL       0
#define MODBUSTCP_TABLE_INPUT_COIL        1
#define MODBUSTCP_TABLE_EXCEPTION_STATUS  7


//----------------------------------------------------------------------------
// Type Definitions                                                          |
//----------------------------------------------------------------------------

typedef void (WINAPI *MBTReadCompleted)(
   IN HANDLE hSocket,          // socket handle
   IN DWORD callbackContext,   // callback context, handed over at the call
   IN LONG errorCode,          // result of the read operation
   IN BYTE tableType,          // type of MODBUS/TCP tables(MODBUSTCP_TABLE_xxx)
   IN WORD dataStartAddress,   // start address of the registers or coils to be
                               // read
   IN WORD numRead,            // number of the registers or coils to be read
   IN WORD numBytes,           // number of the bytes to be read
   IN LPBYTE pReadBuffer       // memory section with the data to be written
   );

typedef void (WINAPI *MBTWriteCompleted)(
   IN HANDLE hSocket,          // socket handle
   IN DWORD callbackContext,   // callback context, handed over at the call
   IN LONG errorCode,          // result of the write operation
   IN BYTE tableType,          // type of MODBUS/TCP tables(MODBUSTCP_TABLE_xxx)
                               // output registers or output coils
   IN WORD dataStartAddress,   // start address of the registers or coils to be
```

```
                                // written
   IN WORD numWrite,            // number of the registers or coils to be written
   IN LPBYTE pWriteBuffer       // memory section with the data to be written
   );


//----------------------------------------------------------------------------
// Prototypes                                                                 |
//----------------------------------------------------------------------------

// initializes the MODBUS/TCP library
LONG WINAPI MBTInit();

// terminates the MODBUS/TCP library
LONG WINAPI MBTExit();

// creates a socket and connects it to the given device port
LONG WINAPI MBTConnect(
   IN LPCTSTR szHostAddress,  // TCP/IP address of device
   IN WORD port,              // TCP port in device for communication
   IN BOOL useTCPorUDP,       // TRUE - TCP; FALSE - UDP
   IN DWORD requestTimeout,   // maximal time for managing an I/O request (ms)
   OUT HANDLE *hSocket        // handle of the connected socket
   );

// aborts the connection to a device and releases the socket
LONG WINAPI MBTDisconnect(
   IN HANDLE hSocket          // handle of the connected socket
   );

// read from a connected socket
LONG WINAPI MBTReadRegisters(
   IN HANDLE hSocket,         // handle of the connected socket
   IN BYTE tableType,         // Modbus/TCP Tabellen Typ (MODBUSTCP_TABLE_xxx)
                              // (here: input register or output register
   IN WORD dataStartAddress,  // start address of the registers to be read
   IN WORD numWords,          // number of the registers to be read
   OUT LPBYTE pReadBuffer,    // memory section from which the data are read
                              // (NULL at asynchronous call)
   OPTIONAL IN MBTReadCompleted fpReadCompletedCallback = NULL,
                              // C-callback function, called after termination
                              // of asynchronous reading (NULL at synchronous
                              // call)
   OPTIONAL IN DWORD callbackContext = 0
                              // context, handed over to the asynchronous
                              // (callback function (0 at synchronous call)
   );

// write to a connected socket
LONG WINAPI MBTWriteRegisters(
   IN HANDLE hSocket,         // handle of the connected socket
   IN WORD dataStartAddress,  // start address of the registers to be written
   IN WORD numWords,          // number of the registers to be written
   IN LPBYTE pWriteBuffer,    // memory section from which the data are written
                              // (NULL at asynchronous call)
   OPTIONAL IN MBTWriteCompleted fpWriteCompletedCallback = NULL,
                              // C-callback function, called after termination
                              // of asynchronous writing (NULL at synchronous
                              // call)
   OPTIONAL IN DWORD callbackContext = 0
                              // context, handed over to the asynchronous
                              // (callback function (0 at synchronous call)
```

```
   );

// read from a connected socket
LONG WINAPI MBTReadCoils(
   IN HANDLE hSocket,          // handle of the connected socket
   IN BYTE tableType,          // Modbus/TCP Tabellen Typ (MODBUSTCP_TABLE_xxx)
                               // (here: input coil or output coil
   IN WORD dataStartAddress,   // start address of the coils to be read
   IN WORD numBits,            // number of the coils to be read
   OUT LPBYTE pReadBuffer,     // memory section from which the data are read
                               // (NULL at asynchronous call)
   OPTIONAL IN MBTReadCompleted fpReadCompletedCallback = NULL,
                               // C-callback function, called after termination
                               // of asynchronous reading (NULL at synchronous
                               // call)
   OPTIONAL IN DWORD callbackContext = 0
                               // context, handed over to the asynchronous
                               // (callback function (0 at synchronous call)
   );

// write to a connected socket
LONG WINAPI MBTWriteCoils(
   IN HANDLE hSocket,          // handle of the connected socket
   IN WORD dataStartAddress,   // start address of the coils to be written
   IN WORD numBits,            // number of the coils to be written
   IN LPBYTE pWriteBuffer,     // memory section from which the data are written
                               // (NULL at asynchronous call)
   OPTIONAL IN MBTWriteCompleted fpWriteCompletedCallback = NULL,
                               // C-callback function, called after termination
                               // of asynchronous writing (NULL at synchronous
                               // call)
   OPTIONAL IN DWORD callbackContext = 0
                               // context, handed over to the asynchronous
                               // (callback function (0 at synchronous call)
   );

// read from a connected socket
LONG WINAPI MBTReadExceptionStatus(
   IN HANDLE hSocket,          // handle of the connected socket
   OUT LPBYTE pExceptionStatus,
                               // memory section from which the data are read
                               // (NULL at asynchronous call)
   OPTIONAL IN MBTReadCompleted fpReadCompletedCallback = NULL,
                               // C-callback function, called after termination
                               // of asynchronous reading (NULL at synchronous
                               // call)
   OPTIONAL IN DWORD callbackContext = 0
                               // context, handed over to the asynchronous
                               // (callback function (0 at synchronous call)
   );

WORD WINAPI MBTSwapWord(const WORD wData);

DWORD WINAPI MBTSwapDWord(const DWORD dwData);

#ifdef __cplusplus
}
#endif

#endif
```

2.4.2.3 Program

```
#include <windows.h>
#include <tchar.h>
#include <assert.h>
#include <stdio.h>
#include "MBT.h"

#define DEFAULT_PORT             502
#define DEFAULT_SERVER_NAME      _T("172.17.5.90")
#define DEFAULT_PROTOCOL         TRUE       /* TCP */
#define DEFAULT_REQUEST_TIMEOUT  1000   /* in ms */

LONG ret;
char pWriteBuffer[] = "This is a small test message.\n";
bool received;

void WINAPI rcb(
   IN HANDLE hSocket,         // socket handle
   IN DWORD callbackContext,  // callback context, handed over at the call
   IN LONG errorCode,         // result of the read operation
   IN BYTE tableType,         // type of MODBUS/TCP tables(MODBUSTCP_TABLE_xxx)
   IN WORD dataStartAddress,  // start address of the registers or coils to be
                              // read
   IN WORD numRead,           // number of the registers or coils to be read
   IN WORD numBytes,          // number of the bytes to be read
   IN LPBYTE pReadBuffer      // memory section with the data to be written
   )
{
  int i;

  printf( "\nread callback called:\n"
          "\tcontext: %lu\n"
          "\terror code: %ld\n"
          "\ttable type: %d\n"
          "\tdata start address: %d\n"
          "\tnumber of read registers/coils: %d\n"
          "\tnumber of read bytes: %d\n"
          "\tread bytes: ",
          callbackContext,
          errorCode,
          tableType,
          dataStartAddress,
          numRead,
          numBytes
          );

  if( errorCode == S_OK )
  {
    for( i = 0; i < numBytes; ++i )
    {
      printf( "%d ", pReadBuffer[i] );
    }
  }

  printf( "\n" );

  received = true;
}

void WINAPI wcb(
   IN HANDLE hSocket,         // socket handle
```

```
   IN DWORD callbackContext,  // callback context, handed over at the call
   IN LONG errorCode,         // result of the write operation
   IN BYTE tableType,         // type of MODBUS/TCP tables(MODBUSTCP_TABLE_xxx)
                              // output registers or output coils
   IN WORD dataStartAddress,  // start address of the registers or coils to be
                              // written
   IN WORD numWrite,          // number of the registers or coils to be written
   IN LPBYTE pWriteBuffer     // memory section with the data to be written
   )
{
  int i;
  int numBytes;

  printf( "\nwrite callback called:\n"
          "\tcontext: %lu\n"
          "\terror code: %ld\n"
          "\ttable type: %d\n"
          "\tdata start address: %d\n"
          "\tnumber of written registers/coils: %d\n"
          "\twritten bytes: ",
          callbackContext,
          errorCode,
          tableType,
          dataStartAddress,
          numWrite
          );

  if( errorCode == S_OK )
  {
    switch( tableType )
    {
        case MODBUSTCP_TABLE_OUTPUT_REGISTER:
            numBytes = 2 * numWrite;
            break;
        case MODBUSTCP_TABLE_OUTPUT_COIL:
            numBytes = (numWrite + 7) / 8;
            break;
    }

    for( i = 0; i < numBytes; ++i )
    {
      printf( "%d ", pWriteBuffer[i] );
    }
  }

  printf( "\n" );

  received = true;
}

void WINAPI esrcb(
   IN HANDLE hSocket,         // socket handle
   IN DWORD callbackContext,  // callback context, handed over at the call
   IN LONG errorCode,         // result of the read operation
   IN BYTE tableType,         // type of MODBUS/TCP tables(MODBUSTCP_TABLE_xxx)
   IN WORD dataStartAddress,  // start address of the registers or coils to be
                              // read
   IN WORD numRead,           // number of the registers or coils to be read
   IN WORD numBytes,          // number of the bytes to be read
   IN LPBYTE pReadBuffer      // memory section with the data to be written
   )
{
```

```
  int i;

  printf( "\nexception status read callback called:\n"
          "\tcontext: %lu\n"
          "\terror code: %ld\n"
          "\ttable type: %d\n"
          "\tdata start address: %d\n"
          "\tnumber of read registers/coils: %d\n"
          "\tnumber of read bytes: %d\n"
          "\tread bytes: ",
          callbackContext,
          errorCode,
          tableType,
          dataStartAddress,
          numRead,
          numBytes
          );

  if( errorCode == S_OK )
  {
    for( i = 0; i < numBytes; ++i )
    {
      printf( "%d ", pReadBuffer[i] );
    }
  }

  printf( "\n" );

  received = true;
}


int main(int argc, char* argv[])
{
  HANDLE hSocket;
  int i = 1234;
  int ch;
  BYTE pWriteBuffer[6] = { 0xff, 0xff, 0xff, 0xff, 0xff, 0xff };
  BYTE pWriteBuffer2[6] = { 0x33, 0x33, 0x33, 0x33, 0x33, 0x33 };
  BYTE pReadBuffer[6];

  ret = MBTInit();

  if( ret == S_OK )
  {
    ret = MBTConnect( DEFAULT_SERVER_NAME, DEFAULT_PORT, DEFAULT_PROTOCOL,
                      DEFAULT_REQUEST_TIMEOUT, &hSocket );
  }

  if( ret == S_OK )
  {
    received = false;
    ret = MBTWriteRegisters(
                hSocket,
                0x0000,
                3,
                (LPBYTE) pWriteBuffer,
                wcb,
                i
                );

  }
```

```
if( ret == S_OK )
{
  while( !received )
  {
    Sleep( 100 );
  }
}


if( ret == S_OK )
{
  received = false;
  ret = MBTReadRegisters(
              hSocket,
              MODBUSTCP_TABLE_OUTPUT_REGISTER,
              0x0200,
              3,
              NULL,
              rcb,
              i
              );
}

if( ret == S_OK )
{
  while( !received )
  {
    Sleep( 100 );
  }
}

if( ret == S_OK )
{
  received = false;
  ret = MBTWriteCoils(
              hSocket,
              0x0000,
              6,
              (LPBYTE) pWriteBuffer,
              wcb,
              i
              );

}

if( ret == S_OK )
{
  while( !received )
  {
    Sleep( 100 );
  }
}

if( ret == S_OK )
{
  received = false;
  ret = MBTReadCoils(
              hSocket,
              MODBUSTCP_TABLE_OUTPUT_COIL,
              0x0200,
              6,
```

```
                NULL,
                rcb,
                i
                );
}

if( ret == S_OK )
{
  while( !received )
  {
    Sleep( 100 );
  }
}

if( ret == S_OK )
{
  received = false;

  // read from a connected socket
  ret = MBTReadExceptionStatus(
    hSocket,          // handle of the connected socket
    NULL,             // memory section from which the data are read
                      // (NULL at asynchronous call)
    esrcb,            // C-callback function, called after termination
                      // of asynchronous reading (NULL at synchronous
                      // call)
    i                 // context, handed over to the asynchronous
                      // (callback function (0 at synchronous call)
 );
}

if( ret == S_OK )
{
  while( !received )
  {
    Sleep( 100 );
  }
}

printf( "press <RETURN> !\n" );
scanf( "%c", &ch );

//****************** Sync calls ******************************************

if( ret == S_OK )
{
  printf( "\n\n\n****************** Sync calls ******************\n\n" );

  ret = MBTWriteRegisters(
                hSocket,
                0x0000,
                3,
                (LPBYTE) pWriteBuffer2,
                NULL,
                0
                );
}

if( ret == S_OK )
{
  ret = MBTReadRegisters(
                hSocket,
```

```
                MODBUSTCP_TABLE_OUTPUT_REGISTER,
                0x0200,
                3,
                pReadBuffer,
                NULL,
                0
                );
}

if( ret == S_OK )
{
  printf( "\nWrite/Read registers: " );

  for( i = 0; i < 2 * 3; ++i )
  {
    printf( "%d ", pReadBuffer[i] );
  }

  ret = MBTWriteCoils(
                hSocket,
                0x0000,
                6,
                (LPBYTE) pWriteBuffer2,
                NULL,
                0
                );
}

if( ret == S_OK )
{
  ret = MBTReadCoils(
                hSocket,
                MODBUSTCP_TABLE_OUTPUT_COIL,
                0x0200,
                6,
                pReadBuffer,
                NULL,
                0
                );
}

if( ret == S_OK )
{
  printf( "\nWrite/Read coils: " );

  for( i = 0; i < (6 + 7) / 8; ++i )
  {
    printf( "%d ", pReadBuffer[i] );
  }

  ret = MBTReadExceptionStatus(
    hSocket,          // handle of the connected socket
    pReadBuffer,      // memory section from which the data are read
                      // (NULL at asynchronous call)
    NULL,             // C-callback function, called after termination
                      // of asynchronous reading (NULL at synchronous
                      // call)
    0                 // context, handed over to the asynchronous
                      // (callback function (0 at synchronous call)
  );
}
```

```
  if( ret == S_OK )
  {
    printf( "\nRead exception status: " );

    for( i = 0; i < 1; ++i )
    {
      printf( "%d ", pReadBuffer[i] );
    }
  }


  if( ret == S_OK )
  {
    ret = MBTDisconnect( hSocket );
  }

  if( ret == S_OK )
  {
    ret = MBTExit();
  }

  if( ret != S_OK )
  {
    fprintf( stderr, "### Error No %ld\n", ret );
  }

  printf( "\npress <RETURN> !\n" );
  scanf( "%c", &ch );

  return 0;
}
```

3 Index