MODBUS Master Windows Dynamic Link Library

User Manual

Version 2.3

Sunlux Technologies Ltd., No. 497, 6th 'A' Main, H I G Colony, R M V II Stage, Bangalore – 94, India.
Ph: ++91 80 3417072 Fax: ++91 80 3417072
Email: info@sunlux-india.com Web: www.sunlux-india.com

Table of Contents

1.0 Introduction 3
2.0 Usage of the Modbus Dynamic Link Library in VC++ 5
2.1 Steps for integrating the Modbus Library with a VC++ application module 5
3.0 Usage of the Modbus Dynamic Link Library in Visual Basic 6
3.1 Steps for integrating the Modbus Library with a VB application module 6
4.0 Functions exported by the library: 8
4.1 ModbusInitialize 8
4.2 ModbusRead 10
4.3 ModbusWrite 13
4.4 ModbusDeInitialize 16
5.0 Using the Modbus for RS232 Sample VC++ Application 17
6.0 Using the Modbus for RS232 Sample VB Application 19
7.0 Appendix A - Error Codes 21
8.0 Technical Specifications 23

1.0 Introduction

The MODBUS Windows Dynamic Link Library is a Serial line MODBUS Master implementation intended to be used by end user applications to quickly integrate MODBUS Master support. The DLL exports four functions using which the user application can initialize the communication path, read or write data from a MODBUS Slave and reinitialize the communication path. The DLL can be used in a simple DOS application or even by high-level development languages/environments like Visual C++ or Visual Basic. The DLL supports the following MODBUS functions:

Read Multiple Coils : FC 01
Read Discrete Inputs : FC 02
Read Input Registers : FC 03
Read Holding Registers : FC 04
Write Single Coil : FC 05
Write Single Register : FC 06
Write Multiple Coils : FC 15
Write Multiple Registers : FC 16

The MODBUS Windows Dynamic Link Library is intended for multiple slaves on a single network. Broadcasting is supported only for write operations (like FC 05, FC 06, FC 15 and FC 16). For broadcasting the slave ID should be specified as zero. The same registers or coils are updated in all the slaves when a request is broadcasted from the master. Broadcasting should be used only when the slaves support such requests.

Note:
Here Coils mean Digital Input/Outputs, which can both be read or written.
Discrete Inputs are Digital inputs, which can only be read.
Holding Registers are 16-bit Analog Input/Outputs, which can both be read or written.
Input Registers are 16-bit Analog inputs, which can only be read.

1.1 Usage of the Dongle

The Dongle (Hardware Lock) is required for the MODBUS Master Dynamic Link Library to work. The dongle must be used properly for the normal functioning of the system. The following points regarding the dongle must be always remembered.

- The Dongle must be plugged into the parallel port **BEFORE** the system boots up for the dll to run properly.
- Do not remove the Dongle when the system is ON. This may damage the Dongle.
- Do not remove the Dongle when the application is running. **This will hang your system** and may also damage the Dongle.

2.0 Usage of the Modbus Dynamic Link Library in VC++

Files Required:

1. ModbusMaster.dll – Main executable for the Modbus DLL that exports the Modbus functions
2. ModbusMaster.lib – Import Library required for supplying the system with the information needed to load the DLL and locate the exported DLL functions when the application is loaded. Required during linking of the application module that makes calls to the Modbus library.
3. Modbus_app.h – Header file containing the function prototypes of the exported Modbus library functions. Required during compilation of the application module that makes calls to the Modbus library.

2.1 Steps for integrating the Modbus Library with a VC++ application module

The following steps assume that the user is using the Visual Studio Integrated Development environment for your application programming.

1. Include the header file modbus_app.h in all source files where you want to make Modbus Library function calls.
2. Open the Project Settings for your application module (Select the 'Settings…' sub menu under the 'Project' menu item). Click on the 'Link' tab in the property sheet that is displayed.
3. Enter the name of the modbus import library 'ModbusMaster.lib' in the space provided below 'Object/Library module' caption.
4. Your application is now ready to make use of the Modbus library

3.0 Usage of the Modbus Dynamic Link Library in Visual Basic

Files Required:

1. ModbusMaster.dll – Main executable for the Modbus DLL that exports the Modbus functions

3.1 Steps for integrating the Modbus Library with a VB application module

The following steps assume that you are using the Visual Studio Integrated Development environment for your application programming.

1. Create a standard VB project.
2. In order to use the DLL the exported functions must be declared in the "General" section of the form you wish to use the dll functions in. The following functions are to be exported:
    a. ModbusInitialize
    b. ModbusRead
    c. ModbusWrite
    d. ModbusDeInitialize

The following code shows the declarations (the code can be cut and pasted into your application)

```
' ModbusInitialize
Private Declare Function ModbusInitialize Lib "ModbusMaster" (ByVal PortNo As Byte, _
    ByVal BaudRate As Long, ByVal Parity As Byte, _
    ByVal DataBits As Byte, ByVal StopBits As Byte) As Byte

' ModbusRead
Private Declare Function ModbusRead Lib "ModbusMaster" (ByVal SlaveNo As Byte, _
  ByVal RegType As Byte, ByVal VarAddress As Integer, ByVal Items As Integer, _
  ByRef Data As Integer, ByVal Timeout As Long, ByVal Retries As Byte, _
  ByRef ErrorInfo As Byte) As Byte

' ModbusWrite
Private Declare Function ModbusWrite Lib "ModbusMaster" (ByVal SlaveNo As Byte, _
  ByVal RegType As Byte, ByVal VarAddress As Integer, ByVal Items As Integer, _
  ByRef Data As Integer, ByVal Timeout As Long, ByVal Retries As Byte, _
  ByRef ErrorInfo As Byte) As Byte

' ModbusDeInitialize
Private Declare Function ModbusDeInitialize Lib "ModbusMaster" () As Byte
```

3. The following sequence must be followed in using the DLL.
    a. Call 'ModbusInitialize' to initialize the communication port and other MODBUS related parameters
    b. Call 'ModbusRead' and 'ModbusWrite' as required to read data from the slave and write data to it.
    c. At the end of your application call 'ModbusDeInitialize' to deinitialize Modbus and free the communication port under use

4.0 Functions exported by the library:

4.1 ModbusInitialize

- **Parameters**

Sl	Parameter Name	Description	Data Type	VB equ.	VC++ equ.	Values
1	PortNo	Network or Port over which to open the Modbus communication	unsigned char	Byte	BYTE	0 – 255 0 = COM1, 1 = COM2 …
2	BaudRate	Baud Rate for data transfer	unsigned long	Long	DWORD/ ULONG	300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600
3	Parity	Parity checking type for data transfer	unsigned char	Byte	BYTE	0 – None 1 – Odd 2 – Even 3 – Mark 4 – Space
4	DataBits	Byte size for data transfer	unsigned char	Byte	BYTE	4, 5, 6, 7, 8
5	StopBits	Stop Bits for data transfer	unsigned char	Byte	BYTE	0 – 1 Stop-Bits 1 – 1.5 Stop-Bits 2 – 2 Stop-Bits

- **Return Value** – unsigned char (BYTE)
  Return value of '1' indicates successful completion of the operation
  Return value of '0' indicates an error.
  Return value of '21' indicates that the Dongle driver(Port lock driver) is not installed.
  Return value of '22' indicates that the Dongle driver has been installed but the application was unable to find it.

Note :

If the **ModbusInitialize** function returns value '21' or '22' then the user must take care that the **ModbusInitialize** function is not called again before inserting the Dongle and restarting the system.

Sample Code (VC++):

```
//Initialize communication parameters
const DWORD BaudRate = 9600;
const BYTE Parity =0, StopBits =0, DataBits =8;
BYTE cNetwork = 0;

//Call Dll function for initializing modbus
BYTE bSuccess =  ModbusInitialize(cNetwork, BaudRate, Parity, DataBits,
StopBits);

if(bSuccess == 1) //Success
    ::AfxMessageBox("Modbus communication initialized successfully");
else if(bSuccess == 21)
    ::AfxMessageBox("Port lock (dongle) driver is not installed");
else if(bSuccess == 22)
    ::AfxMessageBox("Port lock (dongle) driver not found");
else                    //Failure
    ::AfxMessageBox("Modbus initialization failed");
```

Sample Code (VB):

```
Dim RetVal As Byte, X As Integer, com As Byte
Dim BaudRate As Long
Dim Parity As Byte
Dim DataBits As Byte
Dim StopBits As Byte

' Use COM1 for communication
com = 0

BaudRate = 9600
Parity = 0              ' No parity
StopBits = 0            ' One stop bit

' Now call the MODBUS initialization dll with the above obtained parameters
RetVal = ModbusInitialize(com, BaudRate, Parity, DataBits, StopBits)
If (RetVal = 1) Then
   msg = MsgBox("Modbus Initialized Succesfully", vbOKOnly, "ModBus Test ")
ElseIf (RetVal = 21) Then
   msg=MsgBox("Port lock (Dongle) driver not installed", vbOKOnly, "ModBus Test ")
ElseIf (RetVal = 22) Then
   msg = MsgBox("Port lock (Dongle) driver not found", vbOKOnly, "ModBus Test ")
Else
   msg = MsgBox("Modbus Initialization Error", vbOKOnly, "ModBus Test ")
End If
```

4.2 ModbusRead

- Parameters

Sl	Parameter Name	Description	Data Type	VB equ.	VC++ equ.	Values
1	SlaveNo	Slave ID from which data is to be read	unsigned char	Byte	BYTE	1 – 247
2	Type	Type of data to be requested from the slave	unsigned char	Byte	BYTE	1 – Coil 2–DiscreteInput 3 – Holding Reg 4 – Input Reg
3	VarAddress	Starting address of the data to be read	unsigned short	Integer	WORD	1 – 65535
4	Items	No of items of the specified type to be read	unsigned short	Integer	WORD	Coils / Discrete-Inputs: 1- 800, Registers:1-100
5	Data	Memory allocated data buffer for receiving the requested data	unsigned short*	ByRef Integer	WORD*	Valid data buffer with memory allocated for receiving 'Items'
3	TimeOut	Time out in milliseconds for the Read operation	unsigned long	Long	DWORD/ ULONG	0 – 10000
4	Retries	No of retries in case of transmission failure	unsigned char	Byte	BYTE	0 – 10
5	ErrorInfo	Buffer to receive status of Read operation	unsigned char*	ByRef Byte	BYTE*	Valid data buffer with memory allocated for receiving error number.

- Return Value – unsigned char (BYTE)

Return value of '1' indicates successful completion of the operation. Return value of '0' indicates an error. In case of an error refer to the returned error code.

Sample Code (VC++):

```
WORD* wData;                //Data buffer for reading
BYTE cErrorInfo;            //Buffer for error info
BYTE cDataType = 1; //Read coil
WORD wItems = 2
WORD wAddress = 1;
BYTE cSlaveId = 0;
DWORD dwTimeOut = 1000  ;//1 sec timeout
BYTE cRetries = 3;

//allocate memory for data buffer
wData = (WORD*) malloc( wItems * sizeof(WORD));

//Dll fn call for reading
BYTE status  = ModbusRead(cSlaveId,  cDataType,
                    wAddress, wItems, wData,  dwTimeOut,
                    cRetries, &cErrorInfo);

if(status==1)           //Success
{
        ::AfxMessageBox("Read success");
}
else                    //Failure
{
        CString strError;
        strError.Format("Read failed. Error Code = %d", cErrorInfo);
        ::AfxMessageBox(strError);
}

//free the write buffer
free(wData);
```

Sample Code (VB):

```
Dim b As Boolean
Dim SlaveNo As Byte
Dim RegType As Byte
Dim VarAdd As Integer
Dim Items As Integer
Dim Timeout As Long
Dim Retries As Byte
Dim ErrInfo As Byte
Dim Data(100) As Integer
```

```
RegType = 1             ' Read Coils
SlaveNo = 1             ' Slave address = 1 (in decimal)
VarAdd = 10             ' Starting variable address = 10 (in decimal)
Items = 30              ' No of variables to read = 30 (in decimal)
Timeout = 2000          ' Communication timeout = 2 seconds (in milliseconds)
Retries = 3             ' Communication retries = 3 (retry thrice)

' Call the MODBUS Dll fuction to construct, send and receive a reply
'   from the MODBUS Slave
b = ModbusRead( SlaveNo, RegType, VarAdd, Items, Data(0), _
                Timeout, Retries, ErrInfo)

If (b = 1) Then
        MsgBox "Modbus read successful"
Else
        MsgBox "Modbus read failed"
End If
```

4.3 ModbusWrite

- **Parameters**

Sl	Parameter Name	Description	Data Type	VB equ.	VC++ equ.	Values
1	SlaveNo	Slave ID to which data is to be written	unsigned char	Byte	BYTE	0 – 247
2	Type	Type of data to be written to the slave	unsigned char	Byte	BYTE	1 – Coil 3 – Holding Reg
3	VarAddress	Starting address of the data to be written	unsigned short	Integer	WORD	1 – 65535
4	Items	No of items of the specified type to be written	unsigned short	Integer	WORD	Coils / Discrete-Inputs : 1 - 800 Registers : 1 -100
5	Data	Data buffer containing the data to be written	unsigned short*	ByRef Integer	WORD*	Valid data buffer containing 'Items'
3	TimeOut	Time out in milliseconds for the Write operation	unsigned long	Long	DWORD/ ULONG	0 – 10000
4	Retries	No of retries in case of transmission failure	unsigned char	Byte	BYTE	0 – 10
5	ErrorInfo	Buffer to receive status of Read operation	unsigned char*	ByRef Byte	BYTE*	Valid data buffer with memory allocated for receiving error number.

- Return Value – unsigned char (BYTE)
  Return value of '1' indicates successful completion of the operation.
  Return value of '0' indicates an error. In case of an error refer to the returned error code.

Sample Code (VC++):

```
WORD wData[] = {21, 560, 34};              //Data buffer for writing
BYTE cErrorInfo;                           //Buffer for error info
BYTE cDataType = 3;                  //Write holding register
WORD wItems = 3
WORD wAddress = 1;
BYTE cSlaveId = 0;
DWORD dwTimeOut = 1000  ;                  //1 sec timeout
BYTE cRetries = 3;

//Dll fn call for write
BYTE status  = ModbusWrite(cSlaveId,  cDataType,
                        wAddress, wItems, wData,  dwTimeOut,
                        cRetries, &cErrorInfo);

if(status == 1)                 //Success
{
        ::AfxMessageBox("Write success");
}
else                        //Failure
{
        CString strError;
        strError.Format("Write failed. Error Code = %d", cErrorInfo);
        ::AfxMessageBox(strError);
}
```

Sample Code (VB):

```
Dim b As Boolean
Dim SlaveNo As Byte
Dim RegType As Byte
Dim VarAdd As Integer
Dim Items As Integer
Dim Timeout As Long
Dim Retries As Byte
Dim ErrInfo As Byte
Dim Data(100) As Integer

RegType = 1          ' Write Coils
SlaveNo = 1          ' Slave address = 1 (in decimal)
VarAdd = 10          ' Starting variable address = 10 (in decimal)
Items = 4            ' No of variables to write = 4 (in decimal)
Timeout = 2000       ' Communication timeout = 2 seconds (in milliseconds)
Retries = 3          ' Communication retries = 3 (retry thrice)

` Initialize buffer with data
Data(0)=1
Data(1)=0
Data(2)=1
```

```
Data(3)=1


' Call the MODBUS Dll fuction to construct, send and receive a reply
'   from the MODBUS Slave
b = ModbusWrite( SlaveNo, RegType, VarAdd, Items, Data(0), _
                  Timeout, Retries, ErrInfo)

If (b = 1) Then
        MsgBox "Modbus write successful"
Else
        MsgBox "Modbus write failed"
End If
```

4.4 ModbusDeInitialize

- **Parameters** - None
- **Return Value** – unsigned char (BYTE)
  Return value of '1' indicates successful completion of the operation.
  Return value of '0' indicates an error.

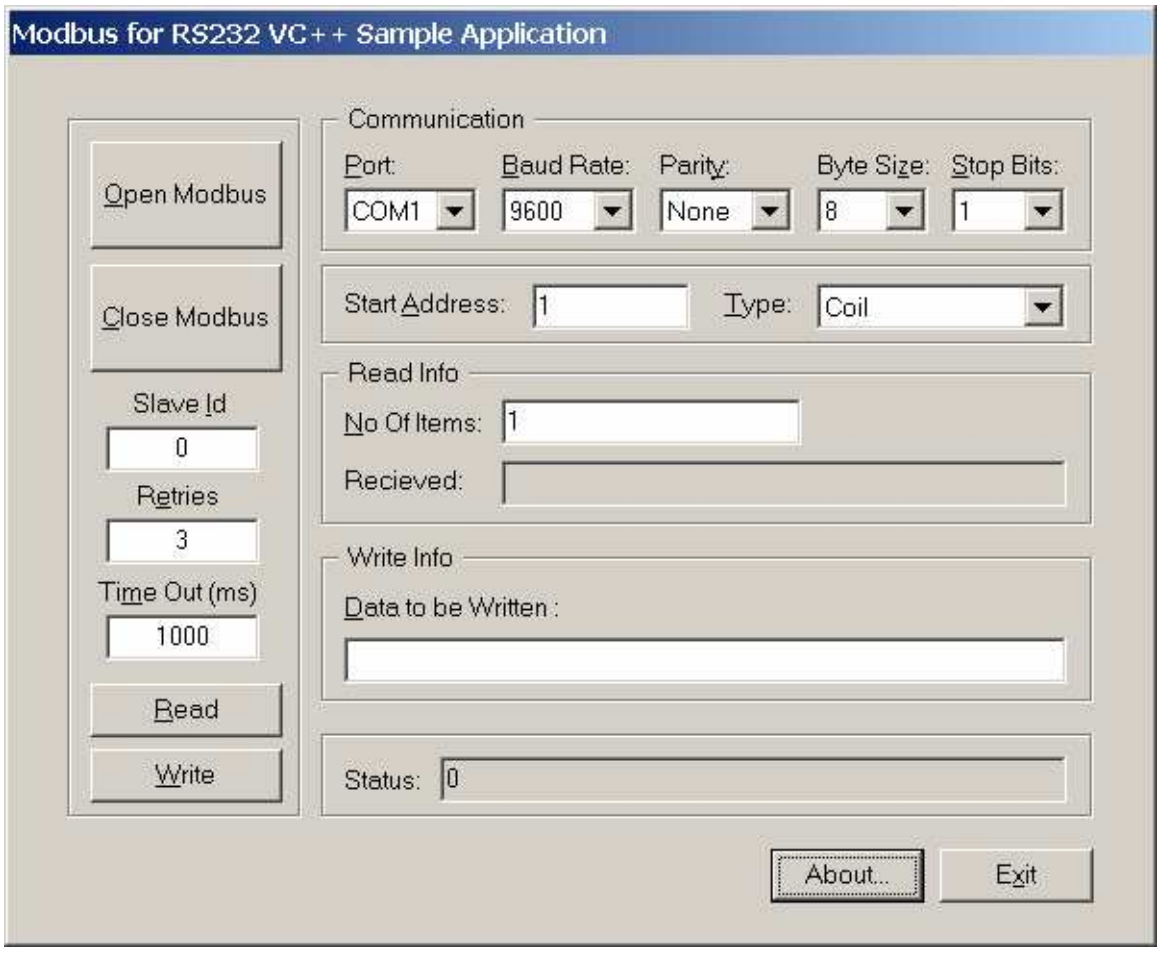
Sample Code (VC++):

```
//Dll function call for deinitializing modbus
BYTE status = ModbusDeInitialize();
if(status)                  //Success
        ::AfxMessageBox("Modbus communication closed successfully");
else                        //Failure
        ::AfxMessageBox("Unable to close Modbus session");
```

Sample Code (VB):

```
Dim RetVal As Byte
' deinitialize MODBUS
RetVal = ModbusDeInitialize
If (RetVal = 1) Then
   MsgBox "Modbus Deinitialized Successfully"
Else
    MsgBox "Modbus Deinitialization Failed"
End If
```

5.0 Using the Modbus for RS232 Sample VC++ Application



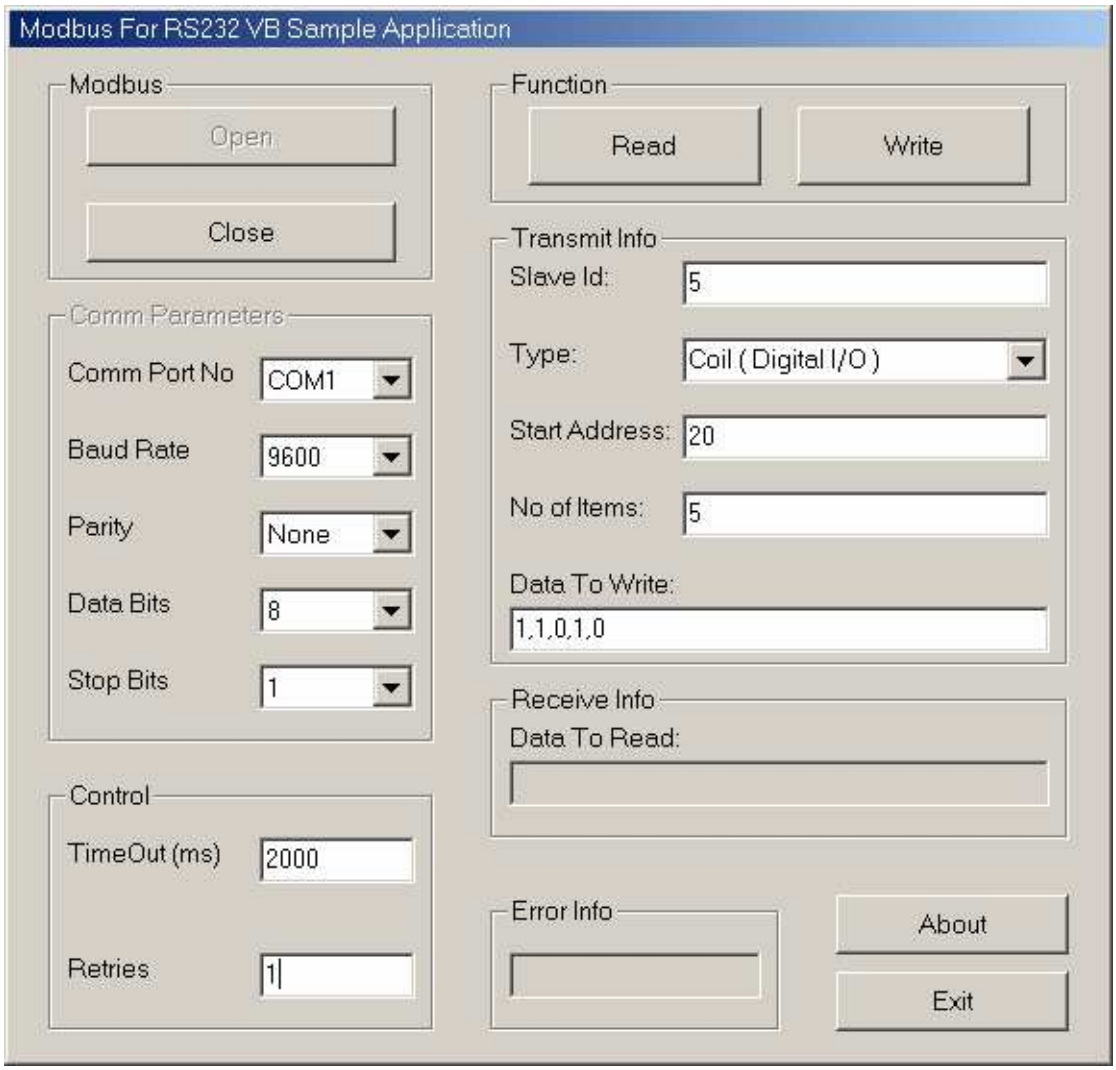
1. Opening a Modbus Session –
   - Select a communication port over which you want to initiate a Modbus session.
   - Select the appropriate communication parameters – Baud Rate, Parity, Byte Size and Stop Bits.
     Note: The communication parameters for the Modbus Master and Modbus Slave must match for proper communication to take place.
   - Click on the 'Open Modbus' button to open and initialize the communication path with the selected communication parameters. You will be prompted with the success or failure of the operation.

2. Reading and Writing –
   - Enter the Device Id of the slave to which you want to Read / Write
   - Enter appropriate values for TimeOut (0 – 10 sec) and retries (1 – 10)
   - Enter the Start Address from which you want to Read / Write.

- Select the type of data (coil, discrete input, holding register, input register) to be Read / Written
- Enter the number of items to be read in case of a Read operation
- Enter the data to be written (separated by commas or white spaces) in case of a Write operation. For digital variables enter a '1' for high and '0' for low. For analog variables, enter the analog value in decimal.
- Click on the 'Read' or 'Write' button to Read / Write the required data. You will be prompted with the success or failure of the Read / Write operation. In case of an error check the returned error code in the status box and refer the appendix at the end of this document for a description of the error.

3. Closing the Modbus Session –
   - Click on the 'Close Modbus' button to end an open session of Modbus

6.0 Using the Modbus for RS232 Sample VB Application



1. Opening a Modbus Session –
   - Select a communication port over which you want to initiate a Modbus session using combo box 'Comm Port No'
   - Select the appropriate communication parameters – Baud Rate, Parity, Byte Size and Stop Bits.
     Note: The communication parameters for the Modbus Master and Modbus Slave must match for proper communication to take place.
   - Click on the 'Open' button to open and initialize the communication path with the selected communication parameters. You will be prompted with the success or failure of the operation.

2. Reading and Writing –
   - Enter the Device Id of the slave to which you want to Read / Write.

- Enter appropriate values for TimeOut (in milli seconds) and retries
- Enter the Start Address from which you want to Read / Write.
- Select the type of data (coil, discrete input, holding register, input register) to be Read / Written.
- Enter the number of items to be read/written.
- Enter the data to be written (separated by commas or white spaces) in case of a Write operation. For digital variables enter a '1' for high and '0' for low. For analog variables, enter the analog value in decimal.
- Click on the 'Read' or 'Write' button to Read / Write the required data. You will be prompted with the success or failure of the Read / Write operation. In case of an error check the returned error code in the status box and refer the appendix at the end of this document for a description of the error.

3. Closing the Modbus Session –
   - Click on the 'Close' button to end an open session of Modbus.

7.0 Appendix A - Error Codes

This section gives the description of the error codes encountered during the execution of the Dll functions.

ErrorNo.	Description
0	No error. Successful execution of the command.
*** Exception Responses**	
1	Illegal Function. The slave doesn't support the requested function
2	Illegal data address. The data address received in the query is not an allowable address for the slave.
3	Illegal data value. A value contained in the query data field is not an allowable value for the slave.
4	Slave device failure. An unrecoverable error occurred while the slave was attempting to perform the requested action.
5	Acknowledge. The slave has accepted the request and is processing it. But a long duration of time will be required to do so. This response is returned to prevent a time out error from occurring in the master.
6	Slave device busy. The slave is engaged in processing a long–duration program command. The master should retransmit the message later when the slave is free.
8	Negative Acknowledge. The slave cannot perform the program function received in the query.
9	Memory Parity Error. The slave attempted to read extended memory, but detected a parity error in the memory. The master can retry the request, but service may be required on the slave device.

*** Exception Responses** : Exception Responses are the responses sent by the slave for a master request when the particular slave is unable to process the requested task. These errors are got only when the slaves being queried support this functionality. For example some slaves, which don't support this functionality may not send any response during such conditions. Then the master will process time out and will send time out error as given in the next section.

Error Codes.....Continued

ErrorNo.	Description
***General Errors:**	
10	No Memory. Memory allocation operation in the Dll has failed
11	Unknown request. The value supplied by the user in the "Type" variable is other than 1,2,3,4.
12	Invalid address.
13	Number of items requested to read or write is exceeding the maximum limit.
14	Time out. Master has sent the request and no reply has come.
15	Checksum Error. Master has received the reply from the slave. But the integrity of the received data is incorrect.
16	Write to communication path error. Master could not transmit the request to the specified Port/Network during read/write operation. In this case the Port has to be closed using the ModbusDeInitialize function and again it has to be initialized using ModbusInitialize function.
17	Read from communication path error. Master could not receive the reply from the specified Port/Network during read/write operation. In this case the Port has to be closed using the ModbusDeInitialize function and again it has to be initialized using ModbusInitialize function.
18	Slave address mismatch. Master has received the reply. But the slave number field in the received frame doesn't matches with that sent by the Master.
19	The network status is marked as bad. In this case the Port has to be closed using the ModbusDeInitialize function and again it has to be initialized using ModbusInitialize function.
20	Invalid slave number entered by the user. Valid slave numbers are: For read functions 1 to 247. For write functions 0 to 247, where 0 is for broadcasting.
21	Dongle driver not installed.
22	Dongle not found.
23	Bad Dll version or unsupported operating system. - Check the OS version to see if it is supported - Check if you are using the wrong version of the dll (i.e. a 9x dll on NT/2K or vice versa)

***General Errors :** In case of General errors appropriate actions should be taken as specified above, specially in case of error number 16 and 17.

8.0 Technical Specifications

Parameter	Value
Standard	MODBUS Application Protocol Specification V1.1, Nov 2002, Schneider Electric, www.modbus.org *Other references:* 1. MODBUS over Serial Line Specification & Implementation guide V1.0, Nov 2002, www.modbus.org 2. Modicon Modbus Protocol Reference Guide, PI–MBUS–300 Rev. J, June 1996, MODICON Inc.
Functions Supported	Read Coils (FC 01) Read Input Discretes (FC 02) Read Multiple Registers (FC 03) Read Input Registers (FC 04) Write Single Coil (FC 05) Write Single Register (FC 06) Force Multiple Coils (FC 15) Write Multiple Registers (FC 16)
Porting Methodology	Ported for use on Microsoft Windows as a Dynamic Link Library
Supported Operating Systems	Microsoft Windows 95 Microsoft Windows 98 Microsoft Windows NT 4.0 Microsoft Windows 2000