WinTECH Software

Industrial Automation Suite of Applications for the Windows O.S.

Purpose of this manual

This manual represents a composite technical description of the applications offered by WinTECH Software to support data acquisition and manipulation using the modbus communications protocol. Topics covered include the distribution and licensing methods utilized to market the software as well as detailed user's manuals for each application.

Software Distribution Method

The WinTECH Software suite of Industrial Automation products is distributed primarily via the internet. Fully-functional demo applications are available from the following Web-Site:

http://www.win-tech.com

Each software application may be downloaded for evaluation and freely distributed among potential users without cost or obligation. Each application is in some fashion time-limited, allowing free and unrestricted use for a pre-defined period of time. This is usually about 3 minutes after establishing communication with a connected modbus device. After the demo-time elapses, the software will cease to function and the application must be restarted to continue

If the software proves useful, and a user wishes to remove the time-limit from its operation, he must purchase an access code from WinTECH Software. For WinTECH Applications, this access code must be entered into the initial sign-on dialog box to register the software, (one-time only). Evaluation versions of the software are identical to commercial, (registered), versions with the exception of the access code which removes all time-limits and other registration incentives which may be included/excluded from the evaluation copy. Once registered, the software may no longer be distributed and must remain on a single machine, (PC). The user must conform to the software license included with the registration access code and protect the confidentiality of the application.

ActiveX controls are protected somewhat differently. The evaluation versions of the modbus OCX controls allow full operation in Visual Basic Design Mode for up to 30 minutes. During this time, the user may exercise a control without restriction. Upon purchase of the control, the user will receive two licensing files from WinTECH Software which removes the 30-minute restriction. ActiveX controls are licensed to be installed on one machine only to be used in a development environment. There are no additional payments or royalty fees required to include the control in a user design to be distributed,
(run-time operation), in object form.

WinTECH Software License Agreement

YOU SHOULD CAREFULLY READ THE FOLLOWING TERMS AND CONDITIONS BEFORE USING THE ACCESS CODES SUPPLIED HEREIN. USING THE SUPPLIED ACCESS CODES TO REGISTER A WINTECH SOFTWARE APPLICATION INDICATES YOUR ACCEPTANCE OF THESE TERMS AND CONDITIONS. IF YOU DO NOT AGREE WITH THEM, YOU SHOULD PROMPTLY RETURN THE ACCESS CODES TO WINTECH SOFTWARE WITHIN FIFTEEN DAYS OF ACQUISITION AND THE REGISTRATION LICENSE FEE PAID WILL BE REFUNDED.

WinTECH Software provides computer programs contained on diskettes and/or electronic distribution services, (the "Applications") and associated help files and documentation, (the "Documentation"). Each Application requires the use of an encryption string, (the "Access Code"), to remove certain operational restrictions contained within the Application. WinTECH Software licenses the use of the Applications, Documentation and Access Codes according to the terms and conditions set forth herein. You assume responsibility for the selection of the Applications to achieve your intended results, and for the installation, use and result obtained from the Applications.

LICENSE

1. You are granted a personal, non-transferable and non-exclusive license to use one copy of the software contained with the Application on a single personal computer and to use the documentation and Access Code under the terms stated in this Agreement. Title and ownership of the Applications and Documentation remain with WinTECH Software

2. You, your employees and/or agents are required to protect the confidentiality of the Applications and Documentation. You may not distribute or otherwise make the Access Codes available to any third party.

3. You may not assign, sublicense or transfer this license and may not decompile, reverse engineer, modify, or copy the Application or Documentation for any purpose, except you may copy the Application and Access Code into machine readable or printed form for backup purposes in support of your use of the Application on a single machine.

4. The Applications, Access Codes, and Documentation are copyrighted by WinTECH Software. You agree to respect and not to remove or conceal from view any copyright, trademark, or confidentiality notices appearing on the Application or Documentation, and to reproduce any such copyright, trademark or confidentiality notices on all copies of the Application and Documentation or any portion thereof made by you as permitted hereunder.

YOU MAY NOT USE, COPY, MODIFY, OR TRANSFER THE APPLICATIONS, DOCUMENTATION, OR ACCESS CODES IN WHOLE OR IN PART, EXCEPT AS EXPRESSLY PROVIDED FOR IN THIS LICENSE.

IF YOU TRANSFER POSSESSION OF ANY COPY OF THE ACCESS CODES TO ANOTHER PARTY, YOUR LICENSE IS AUTOMATICALLY TERMINATED.

TERM

This license is effective until terminated. You may terminate it at any time by destroying the Applications, Documentation and Access Codes along with all copies in any form. It will also terminate upon conditions set forth elsewhere in this Agreement if you fail to comply with any term or condition of this Agreement. You agree upon such termination to destroy the Applications, Documentation, and Access Codes together with all copies in any form.

LIMITED WARRANTY

During the first 90 days after delivery of the Access Codes to you, as evidenced by a copy of your receipt, invoice or other proof of purchase, (the "Warranty Period"), WinTECH Software warrants that the Application will perform substantially in accordance with the Documentation and that the diskettes on which the Applications are furnished, (if supplied), are free from defects in materials and workmanship under normal use. EXCEPT AS PROVIDED N THIS SECTION, THE APPLICATIONS AND DOCUMENTATION ARE PROVIDED 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.

SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS, AND YOU MAY HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

LIMITATION OF REMEDIES

1. WinTECH Software shall use commercially reasonable efforts to correct any failure of the Applications of which it is given written notice by you during the Warranty Period to perform substantially in accordance with the Documentation, provided such a failure can be recreated by WinTECH Software in an unmodified version of the Application or, if WinTECH Software is unable to correct such failure you may terminate the Agreement by returning the Application, Documentation, and Access Code and the License Fee paid will be refunded.

2. WinTECH Software shall replace any diskette not meeting WinTECH Software's "Limited Warranty" and which is returned to WinTECH Software with a copy of your receipt, invoice, or other proof of purchase or, if WinTECH Software is unable to deliver a replacement diskette which is free from defects in materials or workmanship, you may terminate this Agreement by returning the Application, Documentation and Access Code and the License Fee paid will be refunded.

IN NO EVENT WILL WINTECH SOFTWARE BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE APPLICATIONS EVEN IF WINTECH SOFTWARE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

GENERAL

This Agreement will be goverened by the internal laws of the State of West Virginia.

YOU UNDERSTAND THAT YOU HAVE READ THIS AGREEMENT, UNDERSTAND IT AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU FURTHER AGREE THAT IT IS THE COMPLETE AND EXCLUSIVE STATEMENT OF THE AGREEMENT BETWEEN YOU AND WINTECH SOFTWARE WHICH SUPERSEDES ANY PROPOSAL, PRIOR OR CONTEMPORANEOUS AGREEMENT, ORAL OR WRITTEN, AND ANY OTHER COMMUNICATIONS BETWEEN US RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.

How to contact WinTECH Software

The most expediant method for contacting WinTECH Software is via e-mail using the following addresses:

Sales support:

sales@win-tech.com

Technical support:

support@win-tech.com

WinTECH Software is located in the Eastern Time Zone of the United States and may be reached via phone or fax at the following number:

(304) 645-5966

The postal address is:

WinTECH Software
P.O. Box 907
Lewisburg, WV 24901
USA

Modbus Message Formatting

The MODBUS protocol describes an industrial communications and distributed control system developed by Gould-Modicon to integrate PLC's, computers, terminals, and other monitoring, sensing, and control devices. MODBUS is a Master/Slave communications protocol, whereby one device, (the Master), controls all serial activity by selectively polling one or more slave devices. The protocol provides for one master device and up to 247 slave devices on a common line. Each device is assigned an address to distinguish it from all other connected devices.

Only the master initiates a transaction. Transactions are either a query/response type, (only a single slave is address), or a broadcast/no response type, (all slaves are addressed). A transaction comprises a single query and single response frame or a single broadcast frame.

Certain characteristics of the MODBUS protocol are fixed, such as the frame format, frame sequences, handling of communications errors and exception conditions, and the functions performed.

Other characteristics are user selectable. These include a choice of transmission media, baud rate, character parity, number of stop bits, and the transmission modes, (RTU or ASCII). The user selected parameters are set, (hardwired or programmed), at each station. These parameters cannot be changed while the system is running.

Modes of Transmission

The mode of transmission is the structure of the individual units of information within a message, and the numbering system used to transmit the data. Two modes of transmission are available for use in a MODBUS system. Both modes provide the same capabilities for communicating with PLC slaves; the mode is selected depending on the equipment used as a MODBUS Master. One mode must be used per MODBUS system; mixing of modes is not allowed. The modes are ASCII (American Standard Code for Information Interchange), and RTU, (Remote Terminal Unit.) The characteristics of the two transmission modes are defined below:

Characteristic	**ASCII (7-bit)**	**RTU (8-bit)**
Coding System	hexadecimal (uses ASCII printable characters (0-9, A-F)	8-bit binary
Number of bits per character:		
start bits	1	1
data bits (least significant first)	7	8
parity (optional)	1 (1-bit sent for even or odd parity, no bits for no parity)	1 (1-bit sent for even or odd parity, no bits for no parity)
stop bits	1 or 2	1 or 2
Error Checking	LRC (Longitudinal Redundancy Check)	CRC (Cyclical Redundancy Check)

ASCII printable characters are easy to view when troubleshooting and this mode is suited to computer masters programmed in a high level language, such as FORTRAN, as well as PLC masters. RTU is suited to computer masters programmed in a machine language, as well as PLC masters.

In the RTU mode, data is sent in 8-bit binary characters. In the ASCII mode, each RTU character is first divided into two 4-bit parts, (high order and low order), and then represented by the hexadecimal equivalent. The ASCII characters representing the hexadecimal characters are used to construct the message. The ASCII mode uses twice as many characters as the RTU mode, but decoding handling the ASCII data is easier. Additionally, in the RTU mode, message characters must be transmitted in a continuous stream. In the ASCII mode, breaks of up to one second can occur between characters to allow for a relatively slower master.

Error Detection

There are two types of errors which may occur in a communications system: transmission errors and programming errors. The MODBUS system has specific methods for dealing with either type of error.

Communications errors usually consist of a changed bit or bits within a message. The most frequent cause of communications errors is noise: unwanted electrical signals in a communications channel. These signals occur because of electrical interference from machinery, damage to the communications channel, impulse noise, (spikes), etc. Communications errors are detected by character framing, a parity check, and a redundancy check.

When the character framing, parity, or redundancy checks detect a communications error, processing of the message stops. A PLC slave will not act on or respond to the message. (The same occurs if a non-existent slave address is used.)

When a communications error occurs, the message is unreliable. The PLC slave cannot know for sure if this message was intended for it. So the CPU might be answering a message which was not its message to begin with. It is essential to program the MODBUS Master to assume a communications error has occurred if there is no response in a reasonable time. The length of this time depends upon the baud rate, type of message, and scan time of the PLC slave. Once this time is determined, the master may be programmed to automatically retransmit the message.

The MODBUS system provides several levels of error checking to assure the quality of the data transmission. To detect multibit errors where the parity has not changed, the system uses redundancy checks: Cyclical Redundancy Check, (CRC), for the RTU mode and Longitudinal Redundancy Check, (LRC), for the ASCII mode.

CRC-16 Cyclic Redundancy Check

The CRC-16 error check sequence is implemented as described in the following paragraphs.

The message, (data bits only, disregarding start/stop and parity bits), is considered as one continuous binary number whose most significant bit, (MSB), is transmitted first. The message is pre-multiplied by X**16, (shifted left 16 bits), then divided by X**16 + X**15 + X**2 + 1 expressed as a binary number (11000000000000101). The integer quotient digits are ignored and the 16-bit remainder (initialized to all ones at the start to avoid the case where all zeroes being an accepted message), is appended to the message, (MSB first), as the two CRC check bytes. The resulting message including the CRC, when divided by the same polynomial (X**16 + X**15 + X**2 + 1), at the receiver will give a zero remainder if no errors have occurred. (The receiving unit recalculates the CRC and compares it to the transmitted CRC). All arithmetic is performed modulo two, (no carries). An example of the CRC-16 error check for message HEX 0207, (address 2, function 7 or a status request to slave number 2) follows:

The device used to serialize the data for transmission will send the conventional LSB or right-most bit of each character first. In generating the CRC, the first bit transmitted is defined as the MSB of the dividend. For convenience then, and since there are no carries used in arithmetic, let’s assume while computing the CRC that the MSB is on the right. To be consistent, the bit order of the generating polynomial must be reversed. The MSB of the polynomial is dropped since it affects only the quotient and not the remainder. This yields 1010 0000 0000 0001, (HEX A001).. Note that this reversal of the bit order will have no effect whatever on the interpretation or the bit order of characters external to the CRC calculations.

The step by step procedure to form the CRC-16 is as follows:

1. Load a 16-bit register with all 1's.

2. Exclusive OR the first 8-bit byte with the high order byte of the 16-bit register, putting the result in the 16-bit register.

3. Shift the 16-bit register one bit to the right.

4a. If the bit shifted out to the right is one, exclusive OR the generating polynomial 1010 0000 0000 0001 with the 16-bit register.

4b. If the bit shifted out to the right is zero; return to step 3.

5. Repeat steps 3 and 4 until 8 shifts have been performed.

6. Exclusive OR the next 8-bit byte with the 16-bit register.

7. Repeat step 3 through 6 until all bytes of the message have been exclusive OR'rd with the 16-bit register and shifted 8 times.

8. The contents of the 16-bit register are the 2 byte CRC error check and is added to the message most significant bits first.

	16-BIT REGISTER			**MSB**	**Flag**
(Exclusive OR)	1111	1111	1111	1111	
02			0000	0010	
	1111	1111	1111	1101	
Shift 1	0111	1111	1111	1110	1
Polynomial	1010	0000	0000	0001	
	1101	1111	1111	1111	
Shift 2	0110	1111	1111	1111	1
Polynomial	1010	0000	0000	0001	
	1100	1111	1111	1110	
Shift 3	0110	0111	1111	1111	0
Shift 4	0011	0011	1111	1111	1
Polynomial	1010	0000	0000	0001	
	1001	0011	1111	1110	
Shift 5	0100	1001	1111	1111	0
Shift 6	0010	0100	1111	1111	1
Polynomial	1010	0000	0000	0001	
	1000	0100	1111	1110	
Shift 7	0100	0010	0111	1111	0
Shift 8	0010	0001	0011	1111	1
Polynomial	1010	0000	0000	0001	
	1000	0001	0011	1110	
07			0000	0111	
	1000	0001	0011	1001	

Shift 1	0100	0000	1001	1100	1
Polynomial	1010	0000	0000	0001	
	1110	0000	1001	1101	
Shift 2	0111	0000	0100	1110	1
Polynomial	1010	0000	0000	0001	
	1101	0000	0010	1111	
Shift 3	0110	1000	0010	0111	1
Polynommial	1010	0000	0000	0001	
	1100	1000	0010	0110	
Shift 4	0110	0100	0001	0011	0
Shift 5	0011	0010	0000	1001	1
Polynomial	1010	0000	0000	0001	
	1001	0010	0000	1000	
Shift 6	0100	1001	0000	0100	0
Shift 7	0010	0100	1000	0010	0
Shift 8	0001	0010	0100	0001	0
	HEX 12		HEX 41		

TRANSMITTED MESSAGE WITH CRC-16
(MESSAGE SHIFTED TO RIGHT TO TRANSMIT)

```
      12              41              07              02

0001    0010    0100    0001    0000    0111    0000    0010
```

LRC (Longitudinal Redundancy Check)

The error check sequence for the ASCII mode is LRC. The error check is an 8-bit binary number represented and transmitted as two ASCII hexadecimal (hex) characters. The error check is produced by converting the hex characters to binary, adding the binary characters without wraparound carry, and two's complementing the result. At the received end the LRC is recalculated and compared to the sent LRC. The colon, CR, LF, and any imbedded non-ASCII hex characters are ignored in calculating the LRC.

Address	02		0000 0010
Function	01		0000 0001
Start Add H.O.	00		0000 0000
Start Add L.O.	00		0000 0000
Quantity of Pts	00		0000 0000
	08		0000 1000
		Sum	0000 1011
		1's complement	1111 0100
		+1	0000 0001
Error Check	F5	2's complement	1111 0101

MODBUS Message Types

ASCII Framing

Framing in ASCII Transmission mode is accomplished by the use of the unique colon, (:), character to indicate the beginning of frame and carriage return/line feed, (CRLF), to delineate end of frame. The line feed character also serves as a synchronizing character which indicates that the transmitting station is ready to receive an immediate reply.

BEGIN FRAME	ADDRESS	FUNCTION	DATA	ERROR CHECK	EOF	READY TO RECEIVE
:	2-CHAR 16-BIT	2-CHAR 16-BITS	N X 4-CHAR N X 16-BITS	2-CHAR 16-BITS	CR	LF

RTU Framing

Frame synchronization can be maintained in RTU transmission mode only by simulating a synchronous message. The receiving device monitors the elapsed time between receipt of characters. If three and one-half character times elapse without a new character or completion of the frame, then the device flushes the frame and assumes that the next byte received will be an address.

T1,T2,T3	ADDRESS	FUNCTION	DATA	CHECK	T1,T2,T3
	8-BITS	8-BITS	N X 8-BITS	16-BITS	

Address Field

The address field immediately follows the beginning of frame and consists of 8-bits, (RTU), or 2 characters, (ASCII). These bits indicate the user assigned address of the slave device that is to receive the message sent by the attached master.

Each slave must be assigned a unique address and only the addressed slave will respond to a query that contains its address. When the slave sends a response, the slave address informs the master which slave is communicating. In a broadcast message, an address of 0 is used. All slaves interpret this as an instruction to read and take action on the message, but not to issue a response message.

Function Field

The Function Code field tells the addressed slave what function to perform. MODBUS function codes are specifically designed for interacting with a PLC on the MODBUS industrial communications system. The high order bit in this field is set by the slave device to indicate an exception condition in the response message. If no exceptions exist, the high-order bit is maintained as zero in the response message.

The following table lists those functions supported by various WinTECH Software Applications:

CODE	MEANING	ACTION
01	READ COIL STATUS	Obtains current status, (ON/OFF), of a group of logic coils.
02	READ INPUT STATUS	Obtains current status, (ON/OFF), of a group of discrete inputs.
03	READ HOLDING REGISTER	Obtains current binary value in one or more holding registers.
04	READ INPUT REGISTER	Obtains current binary value in one or more input registers.
05	FORCE SINGLE COIL	Force logic coil to a state of ON or OFF.
06	PRESET SINGLE REGISTER	Place a specific binary value into a holding register.
15	WRITE MULTIPLE COILS	Force a group of logic coils to a defined state.
16	PRESET MULTIPLE REGISTERS	Place specific binary values into a group of holding registers.

Data Field

The data field contains information needed by the slave to perform the specific function or it contains data collected by the slave in response to a query. This information may be values, address references, or limits. For example, the function code tells the slave to read a holding register, and the data field is needed to indicate which register to start at and how many to read. The imbedded address and data information varies with the type and capacity of the PLC associated with the slave.

Error Check Field

This field allows the master and slave devices to check a message for errors in transmission. Sometimes, because of electrical noise or other interference, a message may be changed slightly while its on its way from one device to another. The error checking assures hat the slave or master does not react to messages that have changed during transmission. This increases the safety and the efficiency of the MODBUS system.

The error check field uses a Longitudinal Redundancy Check, (LRC), in the ASCII mode of transmission, and a CRC-16 check in the RTU mode.

Exception Responses

Programming or operation errors are those involving illegal data in a message, no response from the PLC to its interface unit, or difficulty in communicating with a slave. These errors result in an exception response from either the master computer software or the PLC slave, depending on the type of error. The exception response codes are listed below. When a PLC slave detects one of these errors, it sends a response message to the master consisting of the slave address, function code, error code, and error check fields. To indicate that the response is a notification of an error, the high-order bit of the function code is set to one.

CODE	NAME	MEANING
01	ILLEGAL FUNCTION	The message function received is not an allowable action for the addressed slave.
02	ILLEGAL DATA ADDRESS	The address referenced in the data field is not an allowable address for the addressed slave device.
03	ILLEGAL DATA VALUE	The value referenced in the data field is not allowable in the addressed slave location.
04	FAILURE IN ASSOCIATED DEVICE	The slave's PC has failed to respond to a message or an abortive error occurred.
05	ACKNOWLEDGE	The slave PLC has accepted and is processing the long duration program command.
06	BUSY, REJECTED MESSAGE	The message was received without error, but the PLC is engaged in processing a long duration program command.
07	NAK-NEGATIVE ACKNOWLEDGMENT	The PROGRAM function just requested could not be performed.

READ OUTPUT STATUS (FUNCTION CODE 01)

This function allows the user to obtain the ON/OFF status of logic coils used to control discrete outputs from the addressed slave only. Broadcast mode is not supported with this function code. In addition to the slave address and function fields, the message requires that the information field contain the initial coil address to be read, (Starting Address), and the number of locations that will be interrogated to obtain status data.

The addressing allows up to 2000 coils to be obtained at each request; however, the specific slave device may have restrictions that lower the maximum quantity. The coils are numbered from zero; (coil number 1 is address 0000, coil number 2 is address 0001, etc.)

The following is an example of a message to Read Output Status Coils 20-56 from slave device number 17.

ADDR	FUNC	DATA START PT HO	DATA START PT LO	DATA # OF PTS HO	DATA # OF PTS LO	ERROR CHECK FIELD
11	01	00	13	00	25	B6

An example response to Read Output Status is shown below. The data is packed one bit for each coil. The response includes the slave address, function code, quantity of data characters, and error checking. Data will be packed with one bit with one bit for each coil, (1 = ON, 0 = OFF). The low order bit of the first character contains the addressed coil, and the remainder follow. For coil quantities that are not even multiples of eight, the last characters will be filled in with zeroes at the high end. The quantity of data characters is always specified as the quantity of RTU characters, i.e., the number is the same whether RTU or ASCII is used.

ADDR	FUNC	BYTE COUNT	DATA COIL STATUS 20-27	DATA COIL STATUS 28-35	DATA COIL STATUS 36-43	DATA COIL STATUS 44-51	DATA COIL STATUS 52-56	ERROR CHECK FIELD
11	01	05	CD	6B	B2	0E	1B	D6

The status of coils 20-27 is shown as CD(HEX) = 1100 1101(Binary). Reading left to right, this shows that coils 27,26,23,22, and 20 are all on. The other coil data bytes are decoded similarly.

READ INPUT STATUS (FUNCTION CODE 02)

This function allows the user to obtain the ON/OFF status of discrete inputs in the addressed slave. Broadcast mode is not supported. In addition to the slave address and function code fields, this message requires that the information field contain the initial input address to be read, (Starting Address) and the number of locations that will be interrogated to obtain the status data.

The following is an example of a message to Read Input Status Coils 10197-10218 from slave device number 17.

ADDR	FUNC	DATA START PT HO	DATA START PT LO	DATA # OF PTS HO	DATA # OF PTS LO	ERROR CHECK FIELD
11	02	00	C4	00	16	13

An example response to Read Input Status is shown below. The data is packed one bit for each coil. The response includes the slave address, function code, quantity of data characters, and error checking. Data will be packed with one bit with one bit for each coil, (1 = ON, 0 = OFF). The low order bit of the first character contains the addressed coil, and the remainder follow. For coil quantities that are not even multiples of eight, the last characters will be filled in with zeroes at the high end. The quantity of data characters is always specified as the quantity of RTU characters, i.e., the number is the same whether RTU or ASCII is used.

ADDR	FUNC	BYTE COUNT	DATA DISCRETE INPUT 10197-10204	DATA DISCRETE INPUT 10205-10212	DATA DISCRETE INPUT 10213-10218	ERROR CHECK FIELD	
11	02	03	AC	DB	35	2E	LRC

The status of inputs 10197-10204 is shown as AC (HEX) = 1010 1100 (Binary). Reading left to right, this shows that inputs 10204, 10202, 10200 and 10099 are all on. The other input data bytes are decoded similarly.

READ OUTPUT REGISTERS (FUNCTION CODE 03)

Read Output Registers allows the user to obtain the binary contents of holding registers in the addressed slave.

These registers can store the numerical values of associated timers and counters which can be driven to external devices.

The addressing allows up to 125 registers to be obtained at each request; however, the specified slave device may have restrictions that lower this maximum quantity. The registers are numbered from zero, broadcast mode is not allowed.

The following example reads registers 40108 through 40110 from slave number 17.

ADDR	FUNC	DATA START PT HO	DATA START PT LO	DATA # OF REGS HO	DATA # OF REGS LO	ERROR CHECK FIELD
11	03	00	6B	00	03	7E

The addresses slave responds with its address and the function code, followed by the information field. The information field contains 2 bytes describing the quantity of data bytes to be returned. The contents of the registers requested (DATA), are two bytes each, with the binary content right justified within each pair of characters. The first byte includes the high order bits and the second, low order bits.

In the example below, the registers 40108-40110 have the decimal contents 555, 0, and 100 respectively.

ADDR	FUNC	BYTE COUNT	DATA OUTPUT REG H.O. 40108	DATA OUTPUT REG L.O. 40108	DATA OUTPUT REG H.O. 40109	DATA OUTPUT REG L.O. 40109	DATA OUTPUT REG H.O. 40110	DATA OUTPUT REG L.O. 40110	ERROR CHECK FIELD
11	03	06	02	2B	00	00	00	64	55

READ INPUT REGISTERS (FUNCTION CODE 04)

Function Code 04 obtains the contents of the controllers input registers. These locations receive their vales from devices connected to the I/O structure and can only be referenced, not altered from within the controller nor via MODBUS.

The example below requests the contents of register 30009 in slave number 17.

ADDR	FUNC	DATA START PT HO	DATA START PT LO	DATA # OF REGS HO	DATA # OF REGS LO	ERROR CHECK FIELD
11	04	00	08	00	01	E2

In the response message, the contents of register 30009 is decimal value 0.

ADDR	FUNC	BYTE COUNT	DATA INPUT REG HO 30009	DATA INPUT REG LO 30009	ERROR CHECK FIELD
11	04	02	00	00	E9

FORCE SINGLE COIL (FUNCTION CODE 05)

This message forces a single coil either On of OFF. Any coil that exists within the controller can be forced to either state, (ON or OFF). Coils are numbered from zero (i.e. coil 1 is address 0000, coil 2 is address 0001, etc.). The data value 65,280, (FF00 HEX) will set the coil ON and the value zero will turn it off. All other values are illegal and will not effect the coil. The use of slave address 00, (Broadcast mode), will force all attached slaves to modify the desired coil.

The example below requests slave number 17 to turn coil number 0173 ON.

ADDR	FUNC	DATA COIL HO	DATA COIL LO	DATA # ON/OFF	DATA	ERROR CHECK FIELD
11	05	00	AC	FF	00	3F

The normal response to the command request is to retransmit the message as received, after the coil state has been altered.

ADDR	FUNC	DATA COIL HO	DATA COIL LO	DATA # ON/OFF	DATA	ERROR CHECK FIELD
11	05	00	AC	FF	00	3F

PRESET SINGLE REGISTER (FUNCTION CODE 06)

Function 06 allows the user to modify the contents of a holding register. Any holding register that exists within the controller can have its contents changed by this message. The values are provided in binary up to the maximum capacity of the controller. Unused high-order bits must be set to zero. When used with slave address 00, all slave controllers will load the specified register with the contents specified.

ADDR	FUNC	DATA REG HO	DATA REG LO	DATA VALUE HO	DATA VALUE LO	ERROR CHECK FIELD
11	06	00	87	03	9E	C1

The normal response to a preset single register request is to retransmit the query message after the register has been altered.

ADDR	FUNC	DATA REG HO	DATA REG LO	DATA VALUE HO	DATA VALUE LO	ERROR CHECK FIELD
11	06	00	87	03	9E	C1

FORCE MULTIPLE COILS (FUNCTION CODE 15)

Function 15 allows the user to modify the contents of a group of consecutively addressed coils. The following example forces 10 coils starting at address 20, (13 HEX). The two data fields, CD = 1100 1101 and 00 = 0000 0000, indicate that coils 27, 26, 23, 22 and 20 are to be forced on.

ADDR	FUN C	H.O. ADDR	L.O. ADDR	QUANTITY		BYTE CNT	DATA COIL STATUS	DATA COIL STATUS	ERROR CHECK FIELD
11	0F	00	13	00	0A	02	CD	00	F4

The normal response to a FORCE MULTIPLE COILS request is to echo the slave address, function code, starting address, and quantity of coils set.

ADDR	FUNC	H.O. ADDR	L.O. ADDR	QUANTITY		ERROR CHECK FIELD
11	0F	00	13	00	0A	C3

PRESET MULTIPLE REGISTERS (FUNCTION CODE 16)

Holding registers existing within the controller can have their contents changed via function code 16. Sixteen bits of data for each register is contained within the message.

ADDR	FUN C	H.O. ADDR	L.O. ADDR	QUANTITY		BYTE CNT	H.O. DATA	L.O. DATA	etc.	ERROR CHECK FIELD
11	10	00	87	00	02	04	00	0A		??

The normal response to a PRESET MULTIPLE REGISTERS request is to echo the slave address, function code, starting address, and quantity of registers set.

ADDR	FUNC	H.O. ADDR	L.O. ADDR	QUANTITY		ERROR CHECK FIELD
11	10	00	87	00	02	56

modbus/TCP Extensions

The Modbus Applications Programming Interface for Network Communications, (MBAP), was developed by Modicon to allow traditional serial modbus communiactions to occur over a TCP/IP network. It basically defines a “wrapper” around the modbus protocol to accomidate routing data packets between two network nodes. The same master/slave messaging protocol is used, however the network aspect allows multiple master devices to access data from the same or different slave devices connected to the network. Using the Client/Server approach, a modbus/TCP slave device represents the server side of the communications model, accepting and responding to queires from one or more network client master applications.

WinTECH Software Application Overviews

The WinTECH Software suite of applications for Industrial Automation was designed to provide a cost-effective solution to interface data from modbus devices into the PC Windows environment. Without the overhead associated with a full featured MMI, these products provide an easy to use interface to remote devices. Primarily used for simulation and verification of the protocol, WinTECH Software applications are inexpensive tools which should be included in every test and commissioning engineer's arsenal. In addition, these tools support Windows OLE, which allows quick development of testing applications via Visual Basic for customizing operation, (such as the case for production testing).

Each application supports physical connections to modbus devices via direct serial, modem, or TCP/IP network. Standard Win32 drivers are used through-out the designs, allowing for their efficient operation on Windows 95/98 as well as Windows NT.

ModScan

ModScan is a modbus master application, designed to read data from one or more connected slave devices. The original 16-bit version of the application supports both RTU and ASCII transmission modes and allows register data to be displayed in a variety of formats including decimal, hexadecimal, and floating-point notation. ModScan also supports custom, (user generated), modbus commands and provides a scripting facility for automated testing of modbus slave devices.

ModScan32 is an upgraded version of ModScan which takes full advantage of the Win32 platform. ModScan32 is a multi-document design, allowing you to open and actively scan multiple arrays of data points from an attached slave. The same formatting and scripting features as the 16-bit version are supported, as well as OLE Automation and direct database access via the Microsoft JET database engine. A recent addition to ModScan32 provides basic MMI capability which allows you to generate custom displays of modbus data using various graphical interfaces.

ModScan(16) supports direct serial connections only, while ModScan32 may be used with modems and network connections.

ModSim

ModSim represents the slave end of the communications protocol. This application may be used to simulate data from one or more modbus slave devices for access by a modbus master. Display of data is comparable to ModScan, in that register data may be displayed as decimal, hex, floating-point, etc.

Available as either a 16 or 32-bit design, ModSim supports multiple direct serial connections. ModSim32 may also be configured to operate as a modbus/TCP network server application for simulation of slave data via network connections. OLE Automation is also supported by the Win32 version.

MNetSvr

MNetSvr is a Win32 application designed to bridge serial modbus devices to a network environment. This application operates as a modbus/TCP network server, accepting requests from atached clients, collecting the data via a serial port, and responding via the network connection. MNetSvr supports multiple asynchronous network connections and is fully compatible with the modbus/TCP protocol as defined by Modicon.

MNetMon

MNetMon is a Win32 Application designed to unintrusively monitor an active modbus communications link by tapping into the RS-232 Transmit signals via two separate PC comm ports. As MNetMon recognizes data passed between the master and slave devices, it mirrows the data points to a local database, and makes this data accessible to other network devices operating as modbus/TCP clients. MNetMon operates as a modbus network server application, similar to MNetSvr but without actually polling the slave devices. MNetMon may be used with any existing RS-232 modbus communications link to seamlessly integrate the data with a PC network without impacting the integrity of the data exchanged between the master and slave devices.

Modbus Master ActiveX Control

The WinTECH Software Modbuss OCX is a custom control which supports drag and drop functionality into the Visual Basic devlopment workbench. The OCX is a modbus master control, which provides access to modbus data from your VBA application. The OCX contains a basic series of properties defining the connection, slave adress, point type, and point addresses to scan. Data from the defined slave is automatically collected and presented to your application as an array of data points. Multiple controls may be utilized within the same VBA application for accessing data of different types or data from different slave devices. The modbus master control supports direct serial, modem, or network connections.

Modbus Slave ActiveX Control

The WinTECH Software Modbus Slave OCX is a custom control which supports drag and drop functionality into the Visual Basic devlopment workbench. This very simple slave control allows your Visual Basic application to define an array of data points to be made instantly available to any connected modbus master device. Providing support for direct serial or network connections, this control is the quickest way for a developer to interface his custom data to a modbus network.

Driver DLL's

WinTECH Software can also supply source code to support custom development of modbus designs. Both master and slave drivers are available for the Window's platform in either 16 or 32-bit form factors. These drivers are the same ones used by the above applications and are available at very reasonable prices. Each driver is written in straight 'C' code and comes complete with an example Windows application, (source form), written using the Microsoft MFC. The modbus slave dlls are compatible with Visual Basic, and provide a very easy mechanism for desiging customized solutions.

ModScan

Following is a concise user's manual for the operation of ModScan32. Operation of the 16-bit version of ModScan is similar, but is not detailed in this document.

ModScan Overview

The ModScan application operates as a MODBUS master device in either RTU or ASCII transmission modes. ModScan may be used to access and modify data points contained in one or more MODBUS slave devices connected to the PC via a serial port, modem, or network. ModScan supports the standard MODBUS message types 01-06, 15 & 16, as well as providing the ability for you to exercise special features of a slave device by transmitting custom command strings and observing the response. ModScan is a useful test and diagnostic tool for verifying the proper slave response to MODBUS queries as well as being a low cost data collection tool for interfacing data into PC database and spreadsheet applications.

Document/View Architecture

ModScan utilizes the standard Windows Multiple-Document-Interface, (MDI), architecture for displaying modbus data to the user. Each basic ModScan document represents a series, (array), of modbus data points identified by the following parameters:

Slave Device Address	Represents the physical device attached to the modbus network
Data Type	Internal data representation, (i.e. input, coil, register)
Data Address	Point address within the device
Length	Number of points to scan/display

ModScan may also be utilized to represent different types of modbus data using customized graphical objects as described in section VII.

Associated with each document is also a timer, which is used to periodically scan new data from the defined slave and refresh the display. The modbus data definition is accessible from the ModScan menu or via edit controls in the top splitter window of the document display. As new data is obtained from the slave device, it is written to the bottom splitter window in one of several formats, depending upon your preference. The size of each document display window is adjustable via the splitter control.

Modbus Data Definition

The upper half of each Document’s View represents the data selected for display, (and possible capture to a historical data file). In most testing applications, the ModScan will only be connected to a single modbus slave device, however, in a multidrop modbus network, there may be several devices accessible from a single connection. The “Device Id” edit control allows you to specify the slave address for the source of the data. Likewise, edit controls are available to select the point type, data address, and number of data points to access.

Notice that the modbus protocol uses a 5-digit representation for the slave data address which infers the point-type. For example, INPUT STATUS values are always represented in the range 10001-19999: HOLDING REGISTERS are displayed as 40000-49999. The ModScan application uses the standard notation for displaying data in the bottom splitter window, however the address specified in the upper splitter address edit control assumes a 4-digit physical point address. This address, coupled with the point-type specifier completely defines the data to be accessed in the slave device.

The upper splitter window also contains two counters which are used to tally the number of data requests made from this document to the modbus connection and the number of valid slave responses received in reply. A button is available within the display which resets the counters associated with this document.

Display Formats

As data is received from the slave device, it is displayed to the lower splitter view of the associated document. Any errors incurred during the exchange of information will be displayed on the first line. The font and colors used to display the data is configurable via the **View, Config** menu options.

Modbus register data may be displayed in any of the following formats:

Binary	Data displayed as 16 discrete values.
Decimal	Ranges from -32767 to 32768
Hexadecimal	0000-ffff
Floating-Point	IEEE Standard Floating Point Notation (Requires two registers per value)
Swapped Floating Point	Inverted Floating Point used by some processors
Double Precision Floating Point	64-bit Floating Point Notation
Swapped Double	Inverted 64-bit Float Values

Connections

ModScan may be used to obtain data from modbus slave device connected to the PC in one of three basic physical arrangements. The most common connection is via any one of the four available PC serial COM ports. ModScan uses the standard Win32 software drivers for communication with the COM ports, thereby providing support for any hardware serial boards which may be installed in the Windows operating system, (including RS-232, RS-485, etc.). You have complete control over the operating characteristics of the serial connection by selecting the appropriate baud rate, parity, and control line, (handshaking), properties to match the slave device(s).

In remote testing situations, the ModScan application may be used to communicate with a modbus network over a dedicated modem connection. ModScan supports the TAPI, (telephony application interface), standard implemented in Windows and Windows NT. If selected, the modem connection dialog allows you to enter a phone number for dialing. Any TAPI device configured within the Windows operating system is available for use.

ModScan allows modbus communications to occur over a TCP/IP network using the modbus/TCP protocol. ModScan operates as a modbus/TCP client application, (modbus master), accessing data from any connected modbus/TCP server. Several vendors now offer direct TCP/IP networking support for mobus devices and ModScan is an excellent way to access/test these devices. You may also use one of several available modbus to TCP/IP bridge devices which can service network requests to a connected serial port. A bridge device operates as a network server, providing support to numerous client applications distributed over the network and interfacing modbus requests for data to slave devices connected serially. WinTECH Software provides such a server, (MNetSvr), as an application which runs under Windows.

It is also possible to connect via modem to a remote system utilizing the built-in networking characteristics of Windows '95 & NT. To do this, you will need to configure the ModScan application to connect via a TCP/IP connection which has been setup within Windows to automatically dial and establish a PPP connection with another Windows machine which is connected directly to the slave device(s) you wish to use. In this case, ModScan operates as if it were using a network card connected directly to a modbus to TCP/IP bridge device.

ModScan Commands

File Menu

The File menu offers the following commands:

New	Creates a new document. Use this command to create a new document in ModScan. Each document represents a different block of data from a modbus device.
Open	Opens an existing document. Use this command to open an existing document in a new window. You can open multiple documents at once. Use the Window menu to switch among the multiple open documents.
Custom	This command allows you to open/create a Custom Display document. Using a customized document, you can drag & drop modbus data points from various slave devices onto your display to be shown in a variety of graphical formats.
Close	Closes an opened document. Use this command to close the active document. If the data definition, (address, point type, etc), has changed since the document was opened, ModScan suggests that you save changes before you close it. If you close a document without saving, you lose all changes made since the last time you saved it. Before closing an untitled document, ModScan displays the Save As Dialog and suggests that you name and save the document. You can also close a document by using the Close icon on the document's window.
Save	Saves an opened document using the same file name. Use this command to save the active document to its current name and directory. When you save a document for the first time, ModScan displays the Save As dialog box so you can name your document.
Save As	Saves an opened document to a specified file name. Use this command to save and name the active document. ModScan displays Save As dialog box the so you can name your document.
Print	Prints a document.
Printer Setup	Selects a printer and printer connection.
Exit	Exits ModScan32.

Connection Menu

The Connect menu offers the following commands:

Connect	Attaches the ModScan application to a modbus network, enabling data collection. Use this command to connect the ModScan application to a modbus network. A dialog box will prompt you for information relative to the connection. You may use ModScan to connect directly to a modbus device via one of four Windows COM ports, or via a modem or TCP/IP network. The possible ways to connect are presented in the drop-down list contained within the connect dialog box. If you select a direct connection, you must specify the associated baud rate, parity and control line selections which match your modbus devices. If a modem connection is selected, you must supply the dialing number and if a network selection is made, you must supply the necessary IP address for the connection. The protocol selections button allows to specify either the modbus RTU or ASCII transmission mode and the time-out associated with the expected slave response to a query.
Disconnect	Detaches the ModScan application from the network, freeing up resources for other Windows applications.

Setup Menu

The Setup menu offers the following commands:

Data Definition	Defines the document properties of the modbus data to be scanned. Use this command to define the characteristics of the data to be monitored for the active document. You may select up to 128 data points for display.
Display Options	Allows the document data to be viewed in a variety of formats.
Extended Options	Provides the ability to write data to a connected slave device.
Text Capture	Begins collecting modbus data to a specified text file. Use this command to store the results of each modbus query to an ASCII text file. Data is written to the specified file, (one scan per line), in the format selected for display in the lower splitter view, (i.e. decimal, hex, float). If an error was encountered during the polling transaction, the error message will be logged to the file rather than the data. There is no “wrap-around” feature for collecting data. New data is appended to the file on each query and the file can become very large vary fast if used with a short scan frequency.
Dbase Capture	Begins collecting data into a defined database table. The ability to save modbus data directly to a database is an optional feature of ModScan. If the feature is unavailable, the **Database Capture** menu selection will be grayed out and not accessible. If enabled, this command allows you to store the results of each query into an active Microsoft compatible database for interfacing modbus data with custom designs.
Capture Off	Stops data collection. Use this command to stop the document from updating the capture file.
Reset Counters	Clears all modbus message status counters in all documents. Use this command to reset the modbus message counters for all active documents.

Display Options

The Display Options menu offers the following commands:

Show Data	This default view configuration displays data values as obtained from the modbus slave device. ModScan is normally configured to display modbus data points in the lower splitter view of the associated document. Data points are displayed in order from top to bottom, left to right. Data is displayed using the current colors and font selection. Coil values are displayed as either <0> or <1>. Register values may be displayed in a variety of formats according to your preferences.
Show Traffic	This option allows the serial data stream to be displayed in place of the data points. Use this command to troubleshoot the connection to a particular modbus device. When selected, this option will display the serial data exchanged between the ModScan application and the slave device associated with this document. The data display splitter view will show data transmitted to the slave device and data returned from the slave device as communications occur during the normal polling cycle. This will help to isolate a problem with possible misinterpretation of the modbus protocol. Data will be displayed in either decimal or hex, depending upon the preference settings in effect for viewing the modbus data points. Normal data collection, (if enabled), will continue.
Binary	Register values are displayed as 16 discrete bits. Use this command to display the contents of modbus registers as a group of 16 discrete values as shown below: 41000: <0001000100110100> This example shows the HOLDING REGISTER located at address 1000 to contain the value 1134H.
Decimal	Register values are displayed in decimal format, (-32767-32768).
Hex	Register Values are displayed in Hexadecimal, (0000-ffff).
Floating Point	Register Values are displayed in floating point notation, (two registers are required). Use this command to display the contents of modbus registers as floating point values based on interpretation of two consecutive registers, (32-bits), according to the IEEE specification. ModScan will attempt to convert the values contained within the selected registers as the IEEE value. If the bit pattern contained within the register set matches the criteria for a floating point number, the value will be displayed on the lower document splitter view as a decimal value associated with the first register address of the pair.
Swapped FP	Register Values are displayed in floating point notation, (least significant register first). Use this command to display the contents of modbus registers as floating point values based on interpretation of two consecutive registers, (32-bits), according to the IEEE specification. ModScan will attempt to convert the values contained within the selected registers as the IEEE value. If the bit pattern contained within the register set matches the criteria for a floating point number, the value will be displayed on the lower document splitter view as a decimal value associated with the first register address of the pair. The difference between this option and the normal Floating Pt display option is the order of the registers with respect to the IEEE standard.
Dbl Float	Register Values are displayed in floating point notation, (four registers, (64 bits), are required). Use this command to display the contents of modbus registers as floating point values based on interpretation of four consecutive registers, (64-bits), according to the IEEE specification for double-precision numbers.

ModScan will attempt to convert the values contained within the selected registers as the IEEE value. If the bit pattern contained within the register set matches the criteria for a floating point number, the value will be displayed on the lower document splitter view as a decimal value associated with the first register address of the pair.

Swapped Dbl Register Values are displayed in floating point notation, (least significant register first). Use this command to display the contents of modbus registers as floating point values based on interpretation of four consecutive registers, (64-bits), according to the IEEE specification for double-precision numbers.
ModScan will attempt to convert the values contained within the selected registers as the IEEE value. If the bit pattern contained within the register set matches the criteria for a floating point number, the value will be displayed on the lower document splitter view as a decimal value associated with the first register address of the pair. . The difference between this option and the normal Floating Pt display option is the order of the registers with respect to the IEEE standard.

Hex Addresses Displays the addresses of data points in hexadecimal notation. Use this command to display the addresses associated with a modbus data point in hexadecimal notation rather than decimal.

Extended Options

The Setup Extended Options menu offers the following commands:

Force Coils	Provides the ability to write coil values to a designated slave device. Use this command to manually force a group of coils to a given state, (on/off). The ability to address multiple coils in a slave depends upon the operating characteristics of the device. ModScan uses modbus message 15, (Force Multiple Coils), to transmit the request to the designated slave. Selecting this menu option will initiate a dialog box which prompts for the address of the data to write: Entering appropriate values for the slave address, point address, and number of coils to write initiates a second dialog which allows you to manually select the value for each coil. Use the radio buttons to select either ON of OFF for each coil value. The scrollbar control allows you to advance to the next series of coil addresses:
Preset Registers	Provides the ability to write register values to a designated slave device. Use this command to force a group of holding registers to selected values. ModScan uses modbus message 16, (Preset Registers), to write data to the designated slave device. Selecting this menu option will initiate a dialog box which prompts for the address of the data to write: Entering appropriate values for the slave address, point address, and number of registers to write initiates a second dialog which allows you to manually select the value for each register. Use the edit controls associated with each register address to enter its value. Values may be entered in either decimal or hexadecimal notation, depending upon the preference selected for modbus data display. The scrollbar control allows you to advance to the next series of register addresses up to the maximum specified in the previous dialog: Buttons at the right of the Preset Registers dialog allow you to configure a series of register values and write them to a disk file for later retrieval and downloading to a modbus slave. Pressing the **To File** button saves the currently defined register values to a selected file. The **From File** button fills the edit controls of the dialog with values obtained from a previously saved disk file.
User Commands	Allows you to define and transmit a custom command. Use this command to customize a command string for transmission to a designated modbus slave device. This command is useful for observing the slave response to non-standard modbus queries or to test its reaction to requests for data which may not be available. The slave device should respond with the proper exception message if a master device asks for data which is beyond its address range or otherwise unavailable via the modbus. After transmitting the user string, ModScan will receive characters for the entire time-out period specified for the connection. The results will then be updated to the appropriate edit control on the dialog box. User defined messages transmitted to a slave device will not show up in the ModScan message counters.
Mask Write	This menu selection provides support for the modbus Write-Mask function, (command 22), which allows you to specify a bit pattern to be used in updating the contents of a holding register.
Script Files	Begins execution of a test script.

View Menu

The View menu offers the following commands:

Toolbar	Shows or hides the toolbar.
Status Bar	Shows or hides the status bar.
Display Bar	Shows or hides the format toolbar used to select the display format for modbus registers.
Config	Allows you to customize the appearance of ModScan by selecting the colors and font used.

Config Menu

The config options supported under the View menu offers you the ability to select the colors used to display modbus data as well as the character font.

Background Color	Selects the color for the ModScan data display splitter view.
Foreground Color	Selects the text color used to display data values.
Status Color	Selects the color used to show the modbus status line.
Font	Selects the font.

Window Menu

The Window menu offers the following commands, which enable you to arrange multiple views of multiple documents in the application window:

Cascade	Arranges windows in an overlapped fashion.
Tile	Arranges windows in non-overlapped tiles.
Arrange Icons	Arranges icons of closed windows.
Window 1,2...	Goes to specified window.

Help Menu

The Help menu offers the following commands, which provide you assistance with this application:

Help Topics	Offers you an index to topics on which you can get help.
About	Displays the version number of this application.

Tollbars & Status Bar

Toolbar

The toolbar is displayed across the top of the application window, below the menu bar. The toolbar provides quick mouse access to many tools used in ModScan. The toolbar is detachable and dockable by clicking the mouse on the toolbar background and dragging it to the desired location within the ModScan application window.

To hide or display the Toolbar, choose Toolbar from the View menu (ALT, V, T).

Toolbar buttons, (from left to right), allow you to:

Open a new document.

Open an existing document. ModScan displays the Open dialog box, in which you can locate and open the desired file.

Save the active document or template with its current name. If you have not named the document, ModScan displays the Save As dialog box.

Define the characteristics of the modbus data to be displayed.

Show modbus data points in lower splitter view.

Show serial data streams in lower splitter view.

Print the active document.

Display the About Box.

Context Help.

Display Selection Toolbar

The format toolbar is displayed across the top of the application window, below the menu bar. The format toolbar provides quick mouse access to select the format used by ModScan to display the contents of modbus registers. The toolbar is detachable and dockable by clicking the mouse on the toolbar background and dragging it to the desired location within the ModScan application window.

Toolbar buttons, (from left to right), allow you to:

Display registers in binary.

Display registers in decimal.

Display registers in hexadecimal.

Display registers in floating-point notation.

Display registers in floating-point notation, (interpreted as having the least significant 16-bits in the first register).

Display registers in 64-bit double precision floating point notation.

Display registers in double-precision floating point, (interpreted as having the least significant 16-bits in the first register).

Status Bar

The status bar is displayed at the bottom of the ModScan window. To display or hide the status bar, use the Status Bar command in the View menu.

The left area of the status bar describes actions of menu items as you use the arrow keys to navigate through menus. This area similarly shows messages that describe the actions of toolbar buttons as you depress them, before releasing them. If after viewing the description of the toolbar button command you wish not to execute the command, then release the mouse button while the pointer is off the toolbar button.

The right areas of the status bar indicate the message counters for all modbus message activity logged by the various active documents.

Testing Features

Writing Data

In order to write a MODBUS data point in a slave device, the communications with the device must first be initiated by scanning a series of data points by configuring the correct addressing information and initiating a polling cycle. Once the data is successfully displayed, double-clicking the address/value portion of the screen will initiate a dialog box which allows the value to be changed. If the polling cycle has been configured to represent coil addresses, double-clicking an address will initiate the Change Coil Dialog:

The Change Register Dialog Box may be initiated by configuring the display to represent register data and double clicking on an address:

Register values may be written using binary, decimal, hexadecimal, or floating-point notation, depending on the preference selection currently in effect.

Pressing the Update Button in either write data point dialog will initiate the appropriate MODBUS write command, (05 or 06), during the next scheduled poll.

Message Counters

Each document maintains a counter for each query message transmitted to a modbus slave device and a counter for each correct response returned from the addresses slave. The counters for a given document may be reset via a button control accessible via the upper, (data definition), splitter view. A total count of all message counters from all active documents, (including any OLE Automation client documents), is displayed via the ModScan status bar.

Viewing Serial Traffic

Use this command to troubleshoot the connection to a particular modbus device. When selected, this option will display the serial data exchanged between the ModScan application and the slave device associated with this document. The data display splitter view will show data transmitted to the slave device and data returned from the slave device as communications occur during the normal polling cycle. This will help to isolate a problem with possible misinterpretation of the modbus protocol. Data will be displayed in either decimal or hex, depending upon the preference settings in effect for viewing the modbus data points. Normal data collection, (if enabled), will continue.

Capturing Data

Use this command to store the results of each modbus query to an ASCII text file. Data is written to the specified file, (one scan per line), in the format selected for display in the lower splitter view, (i.e. decimal, hex, float). If an error was encountered during the polling transaction, the error message will be logged to the file rather than the data. There is no “wrap-around” feature for collecting data. New data is appended to the file on each query and the file can become very large vary fast if used with a short scan frequency. The ability to save modbus data directly to a database is an optional feature of ModScan. If the feature is unavailable, the **Database Capture** menu selection will be grayed out and not accessible. If enabled, this command allows you to store the results of each query into an active Microsoft compatible database for interfacing modbus data with custom designs.

User Defined Messsages

Use this command to customize a command string for transmission to a designated modbus slave device. This command is useful for observing the slave response to non-standard modbus queries or to test its reaction to requests for data which may not be available. The slave device should respond with the proper exception message if a master device asks for data which is beyond its address range or otherwise unavailable via the modbus.

After transmitting the user string, ModScan will receive characters for the entire time-out period specified for the connection. The results will then be updated to the appropriate edit control on the dialog box. User defined messages transmitted to a slave device will not show up in the ModScan message counters.

Scripts

Test scripts consist of ASCII text data field separated by commas. They may be constructed using any word processor or spreadsheet application. A test script entry consists of at least 7 data fields as depicted in the following example script, (example.csv).

```
//
// Example Test Script for ModScan Application
//
// Each Script entry consists of the following
// comma delimited data fields:
//
// TEST NAME, NODE, FUNCTION, ADDRESS, LENGTH, DATA, CONTROL CODE
//
// Double slashes on the front of a line denote comments
//
// The following Control Codes may be used (i.e. last field on each line)
//                                  \ -- Continue DATA fields on next line
//                                  C -- Generate Bad CRC message to slave
//                                  D -- Check response data quantity only, (ignore actual data)
//                                  1 -- Expect Exception Response 01
//                                  2 -- Expect Exception Response 02
//                                  4 -- Expect Exception Response 04
//                                  R -- Expect no Response
//                                  T (default) -- Verify Response Data
//
//
// First Test:
//              Write 20 Coils to Node 1 starting at address 100
//              Data is alternating pattern of ones & zeros
//    (DATA field consists of 32-bits and may be specified
//               as a decimal, hex or floating-point value)
//
Preset Multiple Coils,1,15,100,20,0xAAAAA,T
//
//
// Second Test:
//              Verify results of first test by reading the pattern back
//
Verify Coil Status,1,1,100,20,0xAAAAA,T
//
// Third Test:
//              Read 100 Input Status values
//                      ignore the data and only verify proper quantity returned
//
Verify Input Status,1,2,100,100,0,D
//
//
// Forth Test:
//                      Check Slave Response to request for 1000 registers
//                      (Should probably generate an exception response)
//
Test Exception 2,1,3,100,1000,0,2
//
// Fifth Test:
//                      Check Slave Response to bad CRC
//
Invalid Request,1,1,100,1,0,C
//
// Sixth Test:
//                      Query an unknown device & expect no response
//
Query Device 73,73,1,1,1,0,R
//
// Seventh Test:
//                      Write 6 Holding Registers with data
//      (Each floating point number represents
//                       two registers  -- The NAME field on the
//                       continuation line is ignored)
//
Write Floats,1,16,100,6,1.00,\
,2.00,3.00,,,,T
//
// Eighth Test:
//                      Verify Test Seven by reading back the Registers
//
Read Floats,1,3,100,6,1.00,\
,2.00,3.00,,,,T
//
// END OF SCRIPT
end
```

Database Operation

An optional feature of ModScan32 allows you to write data directly into a Microsoft compatible database such as Access. ModScan uses the Jet database engine to provide an efficient exchange of information from an addressed slave device into the designated database table. Selecting the **Dbase Capture** menu item allows you to associate a ModScan document with a given database table. The table will then be updated with new data each time the slave device is polled. If you change the properties of the document while collection is enabled, updates to the database will temporarily cease. Database updates will resume if the original properties are restored.

ModScan attempts to write the modbus data associated with a particular document as a linear table consisting of the time, status code, and value for each defined point. The table is created with the number of columns equal to the number of points selected in the active document plus two additional fields to contain the time & date and the status code associated with each reading. Table headers to identify each column consist of the 5-digit modbus address for each data point. For example, a table defined to contain 5 input registers beginning at address 44 would look like:

Time	Status	30044	30045	30046	30047	30048
11/5/97 12:53 P.M.	0	0000	0001	0002	0003	0004
11/5/97 12:54 P.M.	0	0000	0001	0002	0003	0004
11/5/97 12:55 P.M.	0	0000	0001	0002	0003	0004

OLE Automation

An optional feature of ModScan32 is the ability to access modbus data using OLE Automation routines. This allows custom programs to be generated, (using Visual Basic, Excel Basic, etc.,), to interpret and format data according to your specific requirements. OLE Automation routines provide both read and write access to one or more modbus slave devices through the ModScan application.

Using the OLE Automation routines is a very simple process:

The VBA application links to the ModScan32.tlb file, (Type Library), which details the names for each automation procedure and its argument list. From the Visual Basic development framework, this is done by selecting the menu item to include a custom type library and then browsing for ModScan32.tlb. During the initial Form Load operation, the application must call Create Object as follows:

```
CreateObject("ModScan32.Document")
```

The application then creates one or more PollRequests which define an array of data points to be read from a modbus device. Data defined by the Poll Request will be automatically scanned by the ModScan application on a 1 second basis. (NOTE: The ModScan application must be connected to the modbus network prior to the VBA application starting up.)

The application uses the handle returned from the CreatePollRequest procedure to access, (read or write), a value within the defined array.

During application termination, it must free the memory used by the ModScan application to maintain the data points by deleting any Poll Requests created.

Refer to the Visual Basic Example application included with the ModScan distribution files for additional details.

OLE Automation Routines

The following OLE Automation routines are supported by ModScan32:

short CreatePollRequest (short Device, long Address, short Length)

Arguments:

Device - Specifies the slave device address

Address - Specifies the data point address,(in modbus master (5 digit) format.

coil status addresses:	00000-09999
input status addresses:	10000-19999
input register addresses:	30000-39999
holding register addresses:	40000-49999

Length - Specifies the number of values included in the definition

Return Value:

Point Handle - Defines the array structure for future reads & writes

Notes:

Sets up data structures within ModScan to begin polling the specified data. ModScan must be connected to the modbus network prior to creating the data array. CreatePollRequest returns a non-zero value if the data structure was successfully created, otherwise it returns 0.

short ReadValue (short PointHandle, short Index, short *pValue)

Arguments:

Point Handle - refers to value returned from CreatePollRequest

Index - Specifies the index into the array structure

*pValue - is a pointer to a value to be returned.

Return Value:

Status - indicates whether or not the operation was completed successfully

Notes:

Status will be MBUS_OK, (0), if the data point was successfully read, otherwise, a non-zero value indicates one of the defined error conditions.

short WriteValue (short PointHandle, short Index, short Value)

Arguments:

Point Handle - refers to value returned from CreatePollRequest

Index - Specifies the index into the array structure

Value - is the data to be written.

Return Value:

Status - indicates whether or not the operation was completed successfully

Notes:

Status will be MBUS_OK, (0), if the data point was successfully queued for transmission to the addresses slave. A zero return value does not indicate successful transmission of the request to the slave device. The controlling application is responsible for verifying the write operation by reading back the value written.

short ModifyPollRequest (short PointHandle, short Device, long Address, short Length)

Arguments:

Point Handle - refers to value returned from CreatePollRequest

Device - Specifies the new slave device address

Address - Specifies the new data point address,(in modbus master (5 digit) format.

Length - Specifies the number of values included in the definition

Return Value:

Status will be MBUS_OK, (0), if the data point was successfully modified, otherwise, a non-zero value indicates one of the defined error conditions.

Notes:

Immediately after changing the parameters of a defined data point, the current status of each value in the array will be set to MBUS_UNINITIALIZED, indicating that the data does not represent that defined by the device/address definition. The first poll after modification should reflect the true status of the addressed data array.

short DeletePollRequest (short PointHandle)

Arguments:

Point Handle - refers to value returned from CreatePollRequest

Return Value:

Status will be MBUS_OK, (0), if the data point was successfully modified, otherwise, a non-zero value indicates one of the defined error conditions.

Notes:

This routine frees up memory allocated by ModScan to support the defined Poll Request.

ModScan OLE Automation Error Return Values

The following status codes may be returned from the ModScan application in response to an OLE Automation request:

1-255	Modbus Exception Message as returned from slave device.
256	Invalid Data Point Handle
257	reserved
258	Invalid Data Point Address
259	reserved
260	reserved
261	reserved
262	Out-Of-Memory
263	Time-Out, (Data not received from slave)
264	reserved
265	Invalid checksum in response from slave
266	ModScan not connected to modbus network
267	reserved
268	reserved
269	Remote TCP/IP Server not connected
270	Data Uninitialized, (Values not valid)
271	ModScan Demo Time Expired

Visual Basic Example

The example Visual Basic application included with the ModScan32 distribution files is a simple take-off on the ModScan application itself. The Form creates four array points, (one each for inputs, coils, input registers, and holding registers). A timer is used to read and update the data once per second. Edit controls are used to define the addresses as shown below:

Code for the example follows:

```
Public m_svr As IModSca

Dim PollHandle(4) As Integer          ‘Handle for each of the four array points
Dim status(4) As Integer              ‘Status return value for each ReadValue request
Dim SlaveDevice As Integer            ‘All points use same slave address
Dim StartAddress(4) As Long           ‘starting address for each array

Dim Modbus_Id(4) As Long              ‘ prefix for selected address
Dim Counter As Integer
Dim temp As Integer
Dim Modbus_Addr As Long               ‘qualified 5-digit modbus address




Private Sub coiladdress_Change()
‘
‘ User has changed address for coil array
‘ recalculate 5-digit modbus address
‘ and ModifyPollRequest accordingly
‘
If (IsNumeric(coiladdress.Text)) Then
   StartAddress(1) = coiladdress.Text
   Modbus_Addr = Modbus_Id(1) + StartAddress(1)
   status(1) = m_svr.ModifyPollRequest(PollHandle(1), SlaveDevice, Modbus_Addr, 10)
   StatusMsg (status(1))
End If

End Sub


Private Sub Device_Change()
‘
‘ User has changed slave device
‘ Modify all four PollRequests
‘
If (IsNumeric(Device.Text)) Then
   SlaveDevice = Device.Text

   Modbus_Addr = Modbus_Id(0) + StartAddress(0)
   temp = m_svr.ModifyPollRequest(PollHandle(0), SlaveDevice, Modbus_Addr, 10)

   Modbus_Addr = Modbus_Id(1) + StartAddress(1)
   temp = m_svr.ModifyPollRequest(PollHandle(1), SlaveDevice, Modbus_Addr, 10)

   Modbus_Addr = Modbus_Id(2) + StartAddress(2)
   temp = m_svr.ModifyPollRequest(PollHandle(2), SlaveDevice, Modbus_Addr, 10)
```

```
    Modbus_Addr = Modbus_Id(3) + StartAddress(3)
    temp = m_svr.ModifyPollRequest(PollHandle(3), SlaveDevice, Modbus_Addr, 10)
End If

End Sub




Private Sub Form_Load()

'
' Create the ModScan interface object
'
Set m_svr = CreateObject("ModScan32.Document")


Modbus_Id(0) = 10000   'input status prefix
Modbus_Id(1) = 0        'coil status prefix
Modbus_Id(2) = 30000    'input register prefix
Modbus_Id(3) = 40000    'holding register prefix

'
' create the four PollRequests using
' default values
'
SlaveDevice = 1
For Counter = 0 To 3
    StartAddress(Counter) = 1
    Modbus_Addr = Modbus_Id(Counter) + StartAddress(Counter)
    PollHandle(Counter) = m_svr.CreatePollRequest(SlaveDevice, Modbus_Addr, 10)
   Next Counter

statusline = "** UNINITIALIZED **"
End Sub


Private Sub Form_Terminate()
'
' Free allocated resources
'
For Counter = 0 To 4
    m_svr.DeletePollRequest (PollHandle(Counter))
    Next Counter

End Sub


Public Sub StatusMsg(Index As Integer)
'
' Generate text status message
'
If Index = 0 Then statusline = ""
If Index > 0 And Index < 256 Then statusline = "Slave Device Exception Response"
If Index = 256 Then statusline = "Invalid Handle"
If Index = 257 Then statusline = "Modbus Message Overrun"
If Index = 258 Then statusline = "Invalid Address"
If Index = 259 Then statusline = "Invalid Device Address"
If Index = 260 Then statusline = "Invalid Length Specification"
If Index = 261 Then statusline = "Invalid modbus command"
If Index = 262 Then statusline = "Driver Out-Of-Memory"
If Index = 263 Then statusline = "** Time-Out **"
If Index = 264 Then statusline = "Invalid Protocol Specification"
If Index = 265 Then statusline = "** Bad Checksum **"
If Index = 266 Then statusline = "Server NOT Connected"
If Index = 267 Then statusline = "Invalid Response from Driver"
If Index = 268 Then statusline = "Modbus Write Failure"
If Index = 269 Then statusline = "Remote Server not Connected"
If Index = 270 Then statusline = "** UNINITIALIZED **"
If Index = 271 Then statusline = "ModScan Demo Time Expired"


End Sub

Private Sub holdingregsaddress_Change()
‘
‘ User has changed address for holding registeres
‘
If (IsNumeric(holdingregsaddress.Text)) Then
   StartAddress(3) = holdingregsaddress.Text
   Modbus_Addr = Modbus_Id(3) + StartAddress(3)
   status(3) = m_svr.ModifyPollRequest(PollHandle(3), SlaveDevice, Modbus_Addr, 10)
   StatusMsg (status(3))
End If

End Sub

Private Sub inputaddress_Change()
‘
‘ User has changed starting address for
‘ input status points
‘
If (IsNumeric(inputaddress.Text)) Then
   StartAddress(0) = inputaddress.Text
   Modbus_Addr = Modbus_Id(0) + StartAddress(0)
   status(0) = m_svr.ModifyPollRequest(PollHandle(0), SlaveDevice, Modbus_Addr, 10)
   StatusMsg (status(0))
End If

End Sub

Private Sub inregsaddress_Change()
‘
‘ User has changed input register address
‘
If (IsNumeric(inregsaddress.Text)) Then
   StartAddress(2) = inregsaddress.Text
   Modbus_Addr = Modbus_Id(2) + StartAddress(2)
   status(2) = m_svr.ModifyPollRequest(PollHandle(2), SlaveDevice, Modbus_Addr, 10)
   StatusMsg (status(2))
End If

End Sub

Private Sub Timer1_Timer()
```

```
‘
‘ Each secon read and update all 10 values
‘ in each of th efour arrays
‘
For Counter = 0 To 9
   status(0) = m_svr.ReadValue(PollHandle(0), Counter, temp)
   StatusMsg (status(0))
If temp = 0 Then
   inputstatus(Counter).Value = 0
   Else
   inputstatus(Counter).Value = 1
   End If
Next Counter

For Counter = 0 To 9
   status(1) = m_svr.ReadValue(PollHandle(1), Counter, temp)
   StatusMsg (status(1))
If temp = 0 Then
   coilstatus(Counter).Value = 0
   Else
   coilstatus(Counter).Value = 1
   End If
Next Counter

For Counter = 0 To 9
   status(2) = m_svr.ReadValue(PollHandle(2), Counter, temp)
   StatusMsg (status(2))
   inputreg(Counter).Caption = temp
Next Counter


For Counter = 0 To 9
   status(3) = m_svr.ReadValue(PollHandle(3), Counter, temp)
   StatusMsg (status(3))
   holdingreg(Counter).Caption = temp
Next Counter

                                                       End Sub
```

Custom Displays

Using the Custom Document Display feature of ModScan allows you to generate graphical displays of modbus data representing your specific instrumentation requirements. You can mix and match different types of data from the same or different slave device and have it displayed in a varity of formats. You can add simple drawing items such as lines, circles, and rectangles to the display along with customized text and several built-in graphical items such as bar gauges and historical trend charts.

The Custom Document interface always operates in "Design Mode". To place an object on the display, simply hold the left mouse button down and drag a rectangle in the aproximate location you wish to insert the item. A pop-up dialog will appear allowing you to select from the list of MMI Items which may be drawn on a ModScan custom display. Each item has associated properties which must be defined to tell ModScan how the item is to be drawn, (refer to the list of items below). After an item has been added to the display, its properties may be modified by positioning the mouse over the edge of the item and pressing the right button. The postion of the item may be changed by using the left mouse button to drag the item to a new location. If an item is associated with a modbus data point, it will be updated with new data based on the document's defined scan rate.

Simple Text Display

Properties associated with a simple text item define, of course, the text string to be displayed as well as the color and font to be used. Text may be displayed in any color and using any installed font available to Windows. The selected font also defines the character size. Text may be displayed usiing a foreground/background color combination or displayed transparently on the existing display background.

Discrete Data Values

Modbus Status Inputs and Coil Status values may be added to a custom display. The properties associated with a Modbus Discrete Value define the source of the data, (slave address & point address), and configures the point to be read-only or read-write. Only Coil Status values may be writable from the ModScan and if so designated, double-clicking on a value displayed to a custom form will initiate a dialog box which allows the value to be changed. Modbus Discrete Values also contain properties which define the color and font to be used to display the data.

Register Data Values

Similarly to Discrete Data Values, Register Data Values have associated properties to define the souce of the data and its read/write status. Unlike the discretes, however, ModScan allows register data to be displayed in different formats, (such as integer, unsigned, floating-point, etc.). Also, a Register Data Value may be selected to represent a scaled value according to the conventional modbus range selections of 0-4095 or 0-9999. This allows ModScan to display register data not only as an absolute value of the contents of a specified register, but also as a more general purpose process-point representation.

Simple Graphical Items

Custom Documents may also contain several types of graphical items not directly associated with modbus data. Rectangles, circles, and bitmaps may be added to increase the readability of a display and taylor the document to personal preferences. Rectangles and Ellipses may be added as either solid or bordered items using any available color. The respective sizes, (height & width), are specified in pixels and may be adjusted to suit the taste of the user. These items also contain a property which specifies the drawing order of the display. Selecting "draw first" allows other MMI items to be drawn on top of a rectangle, circle, or bitmap.

Bar Chart Items

Bar Chart items allow a modbus register value to be displayed as a colored bar whose amplitute represents its current value. Bar Charts may be drawn vertically, (bottom to top), or horizontally, (left to right), and the respective sizes are determined by user defined properties. Properties also define the source of the data used to update the chart and various drawing options such as the colors to use and whether or not to surround the bar chart with a border.

Trend Chart Items

A Trend Chart MMI Item allows you to represent one or more modbus register values as a line graph of amplitude over time. Up to five different variables may be displayed within a Trend Chart item with each register value represented by a defined pen color and scaled between specified high and low limits. Properties for the Trend Chart define its update frequency and background color as well as allowing the chart to be optionally surrounded by a border.

Dials

Dial gauges may also be added to a custom display. Properties for the dial include the bitmap to be used to draw the face of the dial and the modbus register value representing the source of the data used to update its value. The dial's needle may be located anywhere on the bitmap by specifying its X & Y coordinates. Properties also define the needle width, length, and sweep.

ModSim

Following is a concise user's manual for the operation of ModSim32. Operation of the 16-bit version of ModSim is similar, but is not detailed in this document.

ModSim Overview

ModSim32 is a Windows Application designed to simulate data from one or more modbus slave devices. ModSim32 may be connected serially to a modbus master application, or connected to multiple modbus master client applications via a network. Modbus supports the modbus/TCP communications protocol standard.

ModSim32 operates as a Windows MDI, (Multiple Document Interface), application, with each document representing a block of modbus data. Each data block is configured to represent a series of data, (inputs, coils, input registers, or holding registers), from a defined slave address. Keyboard commands are available to edit the contents of a data point and OLE Automation routines are supported which allows Visual Basic or other third-party Windows applications to easily access and change simulated modbus data.

Document/View Architecture

Each ModSim32 document represents a block of data points made accessible to a modbus master application via the serial port or network. Each data block is defined as an array of coils or registers beginning at a specified modbus address and representing data contained within a single modbus slave device. Multiple documents may be opened, defining additional data from the same or different device. ModSim32 supports multiple simultaneous communication with modbus masters, and each document is assessible via any connected COM port or network connection.

Standard Windows menu selections are used to manipulate ModSim32 documents. These include commands to create new documents, write document data to disk, and read data from a disk file. Modbus data is displayed in a variety of numerical formats, and edit commands are supported which allow access to and modification of data contained within a ModSim32 document.

Modbus Data Definition

Each ModSim32 document represents a block of contiguous modbus data as defined by the following parameters:

Device Id	Defines the physical modbus slave device represented by the document.
Point Type	Defines one of: INPUT STATUS COIL STATUS INPUT REGISTER HOLDING REGISTER
Address	Defines the modbus address of the first document data point.
Length	Defines the length of the data array contained within the document

Edit controls accessible from within each ModSim32 document provide access to the above data definition parameters as well as the physical connection associated with the data.

Display Formats

Input Status and Coil Status values are represented within a ModSim32 document as either 1 or 0. Register values may be displayed in binary, decimal, hexadecimal, or floating-point notation. Floating-point representation uses the contents of two consequtive 16-bit registers to assemble one 32-bit IEEE floating-point value. ModSim32 can store a floating point value withe the most-significant 16-bits of data in the low-order register address or the high-order register address.

Using the mouse to double-click on a displayed value will present a dialog box which allows you to edit the contents of the data point. This dialog box also provides the configuration mechanism which allows ModSim to automatically change a data point's value based on a clock tick. Register contents may be configured to increment, decrement or generate a random number between two specified limits.

Connections

Menu commands are available which allow you to define the characteristics of a serial connection to your modbus network. ModSim32 supports physical baud rates up to 19.2K and may be used to simulate data using either the RTU or ASCII transmission modes. ModSim32 supports multiple simultaneous connections, and each document is accessible to multiple connections. ModSim32 may also be used to simulate a modbus/TCP network server by selecting the appropriate command under the **Connect** menu.

ModSim Commands

File Menu

The File menu offers the following commands:

New	Creates a new document. Use this command to create a new document in ModSim32.
Open	Opens an existing document. Use this command to open an existing document in a new window. You can open multiple documents at once. Use the Window menu to switch among the multiple open documents.
Close	Closes an opened document. Use this command to close all windows containing the active document. ModSim32 suggests that you save changes to your document before you close it. If you close a document without saving, you lose all changes made since the last time you saved it. Before closing an untitled document, ModSim32 displays theSave As dialog box and suggests that you name and save the document. You can also close a document by using the Close icon on the document's window.
Save	Saves an opened document using the same file name. Use this command to save the active document to its current name and directory. When you save a document for the first time, ModSim32 displays the **Save As** dislog box so you can name your document.
Save As	Saves an opened document to a specified file name. Use this command to save and name the active document. ModSim32 displays **Save As** dialog box so you can name your document.
Save Test	Saves each open document and creates a test configuration file. Use this command to save each open document and create a test configuration file which may be later restored. If an open document has not been previously saved to disk, ModSim32 will prompt you for a file name under which to save the document. The test configuration file which is created contains the name of each document and may be used to automatically reopen each one during the **Restore Test** command.
Restore Test	Restores a previously saved test configuration, (i.e. opens each document file). Use this command to restore a previously saved test configuration. Each document which was open during the corresponding **Save Test** operation will be reopened and associated with a modbus connection. ModSim may automatically open a test configuration on startup by including the test file name as a command line option.
Exit	Exits ModSim32.

Connection Menu

The Connection menu offers the following commands for configuring each of the four COM ports and starting and stopping the MBAP Network Server:

Connect

Port 1	Allows you to configure COM Port 1.
Port 2	Allows you to configure COM Port 2.
Port 3	Allows you to configure COM Port 3.
Port 4	Allows you to configure COM Port 4.
Port 5	Allows you to configure COM Port 5.
Port 6	Allows you to configure COM Port 6.
Port 7	Allows you to configure COM Port 7.
Port 8	Allows you to configure COM Port 8.
Port 9	Allows you to configure COM Port 9.
modbus/TCP	Enables ModSim32 to operate as a mobus network server

Dosconnect

Port 1	Disables communication via COM Port 1
Port 2	Disables communication via COM Port 2
Port 3	Disables communication via COM Port 3
Port 4	Disables communication via COM Port 4
Port 5	Disables communication via COM Port 5
Port 6	Disables communication via COM Port 6
Port 7	Disables communication via COM Port 7
Port 8	Disables communication via COM Port 8
Port 9	Disables communication via COM Port 9
modbus/TCP	Shuts down the network server

The configuration dialog box used to connect a PC COM port allows you to select the baud rate, parity, and protocol selections.

Display Menu

The Display menu allows you to define the numerical format used to display modbus register data:

Binary	Displays data as a series of 16 discrete values.
Decimal	Displays data as a decimal number in the range -32767 to 32768
Hex	Displays data in hexadecimal (0000-ffff)
Floating Point	Displays data as a floating-point value using the lowest addressed register as the most significant 16-bits of data.
Float (Swapped)	Displays data as a floating-point value using the highest addressed register as the most significant 16-bits of data.

Window Menu

The Window menu offers the following commands, which enable you to arrange multiple views of multiple documents in the application window:

Cascade	Arranges windows in an overlapped fashion.
Tile	Arranges windows in non-overlapped tiles.
Arrange Icons	Arranges icons of closed windows.
Window 1, 2, ...	Goes to specified window.

Help Menu

The Help menu offers the following commands, which provide you assistance with this application:

Help Topics	Offers you an index to topics on which you can get help.
About	Displays the version number of this application.

OLE Automation

ModSim32 supports two OLE Automation functions which may be used to customize it's operation for automated testing applications. Using Visual Basic or similar Windows OLE Automation Client, the user can access and modify modbus data points defined within a ModSim32 document.

OLE Automation Routines

short ReadValue (short NodeAddress, long PointAddress, short *pValue)

Arguments:

Node Address - refers to slave address associated with a ModSim32 document.

PointAddress - Specifies the address of the data point to be read,
(in modbus master (5 digit) format)

coil status addresses:	00000-09999
input status addresses:	10000-19999
input register addresses:	30000-39999
holding register addresses:	40000-49999

*pValue - is a pointer to a value to be returned.

Return Value:

Status - indicates whether or not the operation was completed successfully

Notes:

Status will be MBUS_OK, (0), if the data point was successfully read, otherwise, a non-zero value indicates that the value could not be located.

Short WriteValue (short NodeAddress, long PointAddress, short Value)

Arguments:

Node Address - refers to slave address associated with a ModSim32 document.

PointAddress - Specifies the address of the data point to be read,

Value - is the data to be written.

Return Value:

Status - indicates whether or not the operation was completed successfully

Notes:

Status will be MBUS_OK, (0), if the data point was successfully written, otherwise non-zero.

Visual Basic Example

A very simple Visual Basic example project is included with the ModSim32 distribution files which demonstrates how to access/modify data points contained within a ModSim32 document. The example project simply updates a holding register based on a one second timer and updates a text control on the form.

To create the project, perform the following steps:

Use the Project Framework to Browse for ModSim32.tlb

In the General Definitions:

```
Public m_sim As IModSim
```

In Form Load:

```
Set m_sim = CreateObject("ModSim32.Document")
```

On 1 second Timer:

```
status = m_sim.WritePoint (1, 40100, tick)
if status = 0 Then
        tick = tick + 1
        End If

status = m_sim.ReadPoint (1, 40100, temp)
if status = 0 Then
        text1.text = temp
        End If
```

MNetSvr

Following is a concise user's manual for the operation of MNetSvr.

modbus/TCP (MBAP)

The Modbus Applications Programming Interface, (**MBAP**), protocol specification provides extensions to the messaging descriptions which allow components to communicate over TCP/IP networks. Developed by Modicon to support direct ethernet connections to PLC's, the MBAP
protocol defines header information which routes modbus packets between network devices transparent to other network activity. Whereas a serial modbus device, (slave), can only provide data to a single master, a network slave device can communicate with many different master applications asynchronously using the MBAP protocol.

The MBAP protocol uses a Client/Server paradigm. A server application contains data, (in this case data from one or more modbus devices), which is made accessible to various client applications on the network. A server application runs continuously, whereas any given client application may start-up or shut-down at any time. A TCP connection is established between the client and server based on a request from the client during start-up. Once the connection is established, communications occur between the two as though they were connected serially. Multiple clients may be connected with the server at the same time, with each connection having
full access to the data.

The MNetSvr application was designed to operate as a MBAP compatible interface between network client applications, (such as ModScan32), and serial slave devices. MNetSvr opens TCP Port #502 and listens for other network devices to initiate a connection. When a connection is made, MNetSvrspins off a separate thread of execution to process requests from the connected client. As far as the client application is concerned, it appears as though it has exclusive access to the slave device, with all messaging activity identical to that which would occur if the device were connected locally.

MNetSvr Overview

The MNetSvr application was designed to operate as a MBAP compatible interface between network client applications, (such as ModScan32), and serial slave devices. MNetSvr opens TCP Port #502 and listens for other network devices to initiate a connection. When a connection is made, MNetSvr spins off a separate thread of execution to process requests from the connected client. As far as the client application is concerned, it appears as though it has exclusive access to the slave device, with all messaging activity identical to that which would occur if the device were connected locally.

MNetSvr supports modbus message types 01-06, 15& 16. Requests for modbus data are accepted via a network connection, forwarded to the specified serial slave device, and the response from the slave is transmitted back to the network client.

Commands

There is only one menu command supported by the MNetSvr application. **File Connect** allows you to define the operating characteristics of the PC COM port used for communication with the serial slave device(s). This command provides a dialog box to define the associated baud rate, parity, and protocal, (ASCII or RTU), which matches the characteristics of the attached slaves. The serial connection must be started prior to data being made accessible to network devices.

Views

The MNetSvr display window is divided into two parts, (views), separated by a horizontal splitter control. The splitter bar is used to adjust the relative sizes of the two views.

The upper splitter view is used to display network events. As client applications connect and disconnect, a message will be added to the display which details the time and description of the event.

The lower splitter view is used to display the serial data stream as transmitted to and from a connected slave device. Each modbus transaction occurring between the MNetSvr application and a slave device will be displayed in this view.

MNetMon

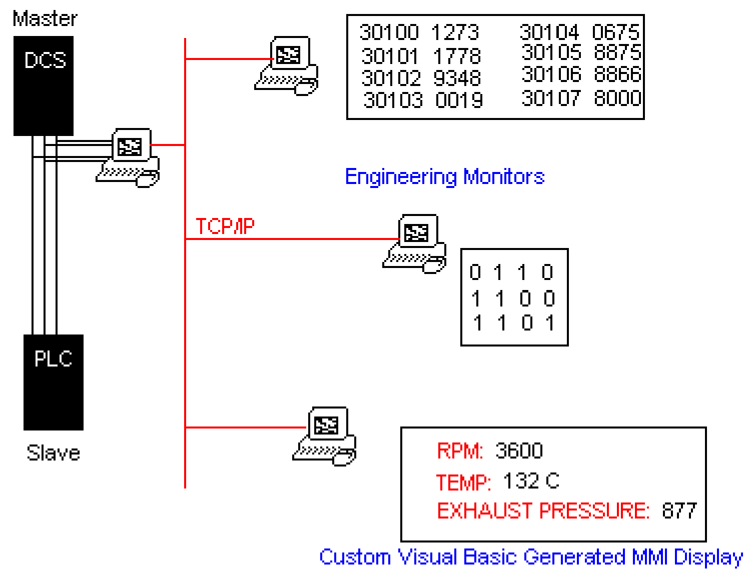
Following is a concise user's manual for the operation of MNetMon.

MNetMon Description
Overview

What is MNetMon and why do you need it? First of all. MNetMon is a Win32 software application designed to operate under Windows. It runs on Windows 95, Windows 98 or Windows NT. It uses all standard Windows drivers, so no special hardware or drivers are required for its operation. MNetMon was designed to provide a seamless interface between modbus devices and a TCP/IP network. It uses two PC serial ports to monitor an active RS-232 modbus communications link, capturing data as it is exchanged between the modbus master and modbus slave devices, and presents this data to other network PC's using the standard MBAP protocol as defined by Groupe Schneider.

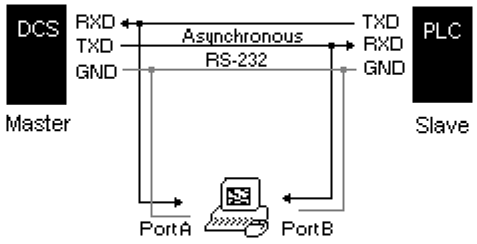
MNetMon is an unintrusive data monitor. No changes are required in the existing modbus network. MNetMon simply attaches to the network as shown and monitors character traffic. When it sees data points transmitted in response to the Master's poll, MNetMon copies the data to a local memory buffer accessible to the network server. As data is refreshed, the buffer contents are updated. Any PC connected to the TCP/IP network operating as an MBAP client may access the data buffer to obtain the data. Several MBAP compatible network client applications are available, including ModScan32 and those custom VBA applications generated using the modbuss OCX control from WinTECH Software.

MNetMon does not provide **Write Access** to the data. The PC's Transmit lines are not connected, so there's no way the monitoring application can interfere with normal modbus communications. The integrity of the original modbus link is maintained, and MNetMon does not have to be running in order for the Master and Slave(s) to communicate. MNetMon, in effect, expands the number of modbus ports from any given slave device to a virtually unlimited number. The port originally used to communicate with a single Master device, may now service many asynchronous network connections without increasing the cost or complexity of the existing link. Data from a single device may now be routed through-out a plant wide network seamlessly and inexpensively.



Serial Connections

MNetMon uses a very simple concept. The nature or RS-232 allows multiple receivers to be connected to a single transmitter. As long as the total cable length does not exceed the RS-232 specification, a listening device, (in this case MNetMon), may tap into the transmit signal of an existing communications link and 'hear' everything that's being said. By using two PC COM ports and listening the transmit from both the master and slave device, MNetMon is able to capture data passed back and forth.



RS-232 devices are classified as DTE or DCE, (Data Terminal Equipment or Data Communications Equipment). The standard pin-out arrangement allows a straight 25-pin cable to be attached between an DCE device and a DTE device. If a DCE device is attached to another DCE device, (or DTE to DTE), the cabling conductors must be switched such that the proper active pins are connected to the proper passive pins on the corresponding device, (i.e. Transmit to Receive, Request to Send to Clear to Send, etc.).

The three primary RS-232 signals are located on pins **2,3 & 7** of a twenty-five pin "D" connector. Pin 7 is ground; 2&3 are either Transmit & Receive or Receive & Transmit, depending on the configuration of DCE/DTE. On 9-pin connectors **2,3 & 5** are used, with 5 being ground.

The connections to the monitoring device consists of two serial connectors, each with two conductors, (Receive & Ground for port 1, and Receive and ground for port 2). The two receive lines may be connected to either end of an RS-232 cable segment on pins 2&3. Both Grounds may be connected to pin 7. In this configuration, **MNetMon** can receive data from device A on port 1, and data from device B on port 2.

Network Connections

MNetMon utilizes the built-in networking capabilities of Windows to access the plant LAN.   MNetMon is a Windows Sockets application. During start-up, MNetMon opens a TCP service port at address 52 and awaits connection attempts from other network devices. As MBAP client applications connect, separate communications threads are created to handle each communications link concurrently. Each connection has full, (Read-Only), access to data captured and collected by MNetMon.

MNetMon Commands

Connect

To begin collecting data from the modbus network, you must issue the **Connect** menu option. MNetMon will guide you through a series of dialogs prompting for the port assignment of the connection to the modbus master and slave transmitters as well as the hardware characteristics of the link, (baud rate, parity, etc). After enabling the specified serial ports. MNetMon will begin monitoring character data on the two links. As data is received, it will be displayed to the MNetMon display as HEX characters. Data obtained from the Master port will be displayed in Blue. Slave data will be Black.

The **Disconnect** menu item stops MNetMon from observing the link.

Stale Data Timer

As MNetMon recognizes slave responses to a master request for data, it will copy the data to local memory to be used in servicing possible network clients. As data is received, it is time-stamped. Since MNetMon only captures data as it is requested by the master, it has no way of keeping data current. It cannot poll the slave device to refresh its data, so if for any reason, the master quits polling a particular data address, the memory image within MNetMon will become dated. To prevent a network client from reading old data within MNetMon, the application maintains a Stale Data Timer. Specified via the **Setup Data Timer menu option**, this timer determines how long a particular piece of data is held by MNetMon before it is discarded. If the data values are not refreshed by the time the Stale Data Timer expires, it will be removed from MNetMon's data base and client applications attempting to access the addresses will receive an exception status indication from the server.

Message Statistics

MNetMon maintains timing statistics for modbus messages captured via its serial ports. A review of these statistics may be made by selecting the **View Statistics** menu option.

Counters are maintained for each message received from the Master and Slave ports, as well as any message received containing an invalid checksum. MNetMon also measures the slave's response time to modbus queries, and this information is displayed to the right of the statistics dialog. Shown in milliseconds, values represent the maximum, minimum, and average response times.

Freeze Display

During an active monitoring session, the MNetMon display scrolls data as fast as it's collected. If you wish to stop the display from updating so you can read the character data, you may select the **Freeze** menu option. When frozen, the display may be manually scrolled to observe actual message contents for troubleshooting purposes. MNetMon continues to monitor the link and update its local data point directory during this time. Pressing the **Freeze** menu selection again will return the display to its live scrolling operation.

Browse Data Points

You may also review the data point directory as collected by the MNetMon monitoring operation. All modbus data currently contained within MNetMon local memory is made available via the View Data menu selection.

Data presented by these dialogs represent a copy of MNetMon data made at the time the command was issued. The data presented here is not live. In other words, if you're looking at a range of addresses and MNetMon updates the data based on a new modbus response, these dialogs will not reflect the changes. They are made available as a diagnostic aide so you can tell which range of addresses the MNetMon application is picking up.

Modbus Master ActiveX Control

Following is a concise user's manual for the operation of the Modbuss OCX control

Controls Overview

Modbuss.ocx contains 32-bit custom controls which allow Visual Basic and other OLE Container applications to quickly and easily access data points contained in a modbus slave device connected to the PC. The modbus device may be connected directly to a PC COM port or accessed via a network connection using the modbus/TCP, communications protocol standard. Modem connections are also supported using the Win32 TAPI interface. Multiple controls may be used within one application to access multiple slave devices asynchronously.

Two controls are supplied with modbuss.ocx. The Serial Control, (Modbus Port Configuration Control), is used to configure the physical operating characteristics of a PC COM port, (baud rate, parity, etc.), as well as the modbus software protocol to be used, (RTU or ASCII).

The Modbus Data Control exposes properties which define the machine, COM port, slave device, point address, and point type. These properties may be static, (defined when the control is created), or updated programatically. Up to 128 integer data values, (64 floats), may be read from a device via each control. Modbus data is presented to the controlling application as an array of values, (either Boolean, integer, or float, depending on the specified point type).

RegSvr32

RegSvr32.exe is a DOS application supplied by Microsoft for making additions and changes to the Windows Registry. Before the modbuss controls can be used by an application they must be properly installed into the Registry. To do this, execute RegSvr32.exe and supply the full path name to the control as an argument. The following command line shows how to register the modbuss control, (assuming the file is contained in the C:\MODBUSS directory):

```
Regsvr32 c:\modbuss\modbuss.ocx
```

MbOCXsvr Helper Application

Modbuss.ocx controls operate in conjunction with two support applications supplied by WinTECH Software. Modbusm.dll is responsible for the actual MODBUS message formatting logic and control of the PC COM, TAPI, and network drivers. MbOCXsvr.exe is responsible for coordinating data traffic between the dll and the control(s). To simplify the design of user applications, each Modbus Data Control appears to have exclusive access to the designated set of IO data points. In effect, the system must be capable of supporting multiple controls simultaneously. These controls may be located on the local machine or possibly multiple remote machines. In addition, each control may, at any given point in time, reference data from any device on any available COM port.

The mbOCXsvr application is a required component of the modbuss ocx control. Each time a Modbus Data Control is started, it will attempt to start the mbOCXsvr application, (if not already running). As needed, the control will request specific data from mbOCXsvr and this application is responsible for gathering the data, (via modbusm.dll), and returning it to the control. If the mbOCXsvr application is terminated, the controls will cease communication with the MODBUS device(s).

Menu options available from the mbOCXsvr application allow the user to configure the local serial port setup parameters. The mbOCXsvr background display details statistical counters representing messaging activity on each of the supported serial channels.

Port Control Description

The Modbus Port Configuration Control, (Serial), is used to programatically control the operating characteristics of a local PC Com port connected to one or more MODBUS slave devices. Use of the Serial control within a user application is optional. The hardware parameters configured by this control may also be set by accessing menu items within the modbuss ocx helper application, mbOCXsvr.exe. Once configured, these parameters will remain in place and be used to initialize the designated COM port each time a control within modbuss.ocx is started.

Properties

The Serial control exposes the following properties and methods:

Port	The PC COM port identification. (If set to 0, indicates that the connection is to be made via the first TAPI line device.)
Protocol	MODBUS Transmission Mode (0=RTU, 1=ASCII).
TimeOut	Defines the timeout associated with the protocol for this control. Specified in milliseconds, this is the amount of time the MODBUS driver waits for a slave device to respond after polling for data.
BaudRate	Serial Baud Rate setting. (0=300, 1=600, 2=1200, 3=2400, 4=4800, 5=9600, 6=14400,7=19200)
Parity	Odd, Even or No Parity, (0, 1, or 2).
StopBits	Number of Stop Bits, (0=1, 1=1.5, 2=2).
RTSHandshaking	Enable hardware output flow control using RTS/CTS. The control will ALWAYS enable RTS prior to transmitting. This property blocks transmission until a CTS is received from the connected slave.
DTRHandshaking	Enable hardware output flow control using DTR/DSR handshaking.
RTSDelay1	Delay in milliseconds between activation of RTS and transmission of the first character.
RTSDelay2	Delay in milliseconds after transmitting last character before releasing RTS signal.
PhoneNo	Phone number to dial during Initialize(). Used only for TAPI connections.

Methods

Initialize()	Method which instructs the control to configure the defined port. This method needs to be controlled programatically if the characteristics of the serial communications channel change during execution of the application. The use of the Modbus Port Configuration Control is optional within a program, as the associated setup parameters may be changed via menu options within mbOCXsvr and remain valid between executions.
HangupCall()	Method used to disconnect a call.

Events

Two events are also available which signal the controlling application when a modem connaction has been made or dropped.

Connect	Event which is fired upon successful establishment of a remote TAPI connection..
ConnectionDrop	Event which is fired if the TAPI connection fails.

Modbus Data Control Description

The Modbus Data Control, (modbuss), reflects the status of one or more I/O data points contained within a connected MODBUS slave device. Each control contains an array of up to 128 boolean values, (Coils), 128 integer values, (Registers), or 64 Floating Point values. The data is defined by properties which identify the device and point address. Standard OLE Get and Set methods allow modbus data points to be accessed transparently by the controlling application by simply moving data to/from the array.

Properties

The following properties and methods define the operating characteristics of the Modbus Data Control:

Port The PC COM port identification. (May be overridden by the UseNetwork property.)

Node MODBUS slave device address.

Address Defines the starting data point address within the designated slave device. The address specification implicitly defines the MODBUS message type used to access the data.

Address Range	Function Code	Data Type
00001-09999	01	Coil Status
10001-19999	02	Input Status
30001-39999	04	Input Registers
40001-49999	03	Holding Registers

Length Defines the number of data points to be scanned from the MODBUS slave device. If floating point values are used, remember that each 32-bit floating-point value requires two consecutively numbered 16-bit MODBUS registers. The Length property must be defined in either the number of Coils or the number of Registers scanned from the device. For example, if a control is configured to expect 40 floating point values, the Length property would be set to 80. The maximum number of values which may be scanned from any particular MODBUS slave device may be less than 128. If a control is configured to scan more values than the device can support, an exception status will be generated by the control.

Busy Indicates that control is processing a modbus message to a connected slave. The Busy property is set by the control whenever a poll for data is made to the MODBUS slave. It is cleared when the requested data is received or the when the device times-out. The Busy property may be used by the application to synchronize data requests. If an Update request is made with the Busy flag set, the control will ignore the request and return an error condition, (FALSE). The only exception to this is in the event a Poll for data takes longer than 10 seconds, in which case the Update() Method overrides the Busy indication and initiates a new poll.

Status This status property is set by the control to indicate an error condition. This

	could be the result of the user application making an invalid request, (such as setting the Length Property > 128), or as the result of a communications error with the designated MODBUS slave device.
WriteStatus	This status property is set by the control to indicate the results of the last write command issued from the control. The WriteStatus may indicate an error condition occurring as the result of an invalid write command, (such as attempting to write an index beyond the length of the data array), or as a result of an error returned from the designated slave device. Normally, the WriteStatus will be set to WRITE_PENDING, (310), after the user code modifies a data value. After the results of the message transaction are returned from the slave device, WriteStatus will reflect any errors which may have occurred. A value of zero indicates successful implementation of the last write command.
InhibitReadAfterWrite	This boolean property determines how modbus data is refreshed to the control after a Write operation. Data accessible to your application is maintained local to the control and only updated from the modbus slave device in response to an Update() method. Issuing a Write command to the slave can cause the data contained within the control to be out of sync with newly written data in the remote device, necessitating a read request, (Update() method), to refresh the control's data. The default setting of this property to FALSE allows the control to automatically issue an Update() to refresh the value of the data array immediately after the completion of a Write command. Setting the value of this property to TRUE, bypasses this extra Update() causing the data to only be refreshed under user control.

Additional properties are defined which allow the control to access data from a remote machine connected via a compatible network.

IPAddr1 . . . 4	These four properties allow the controlling application to configure the control to access MODBUS data via the internet protocol using modbus/TCP. The IPAddr properties define the IP Address of the corresponding modbus/TCP server application residing on the machine connected to the MODBUS slave device of interest. If configured for remote operation, all other control properties operate the same as if the MODBUS device were attached via a local COM port. An Update(), Coil(), Register(), or Float(), operation will initiate the request to the designated remote machine rather than the local machine.
UseNetwork	If TRUE, the control will attempt to access data across the network using the specified IP address of the server. If False, the control will attempt a direct or TAPI connection as specified by the Modbuss.Port property.

Methods

Update()

Instructs the control to update its data array with fresh data values from the MODBUS slave device. Update will return an error, (FALSE), if the control's IO Point property definitions are invalid or if the control is busy processing a previous Update() request. When the results of the data poll are obtained, the control will trigger the Ready event. If the poll for data fails, the Busy status indicator will go false, but the Ready event will not fire. The UpdateFinished event will fire in either case, as long as the Update() method returns TRUE
.

IssueCmd()

This method is used to write a user-defined message to the attached modbus slave. Data messages are transmitted to the designated slave unfiltered and the response message is supplied via the ReadCmdResponse() method as an array of bytes returned from the slave. If IssueCmd() returns TRUE, you are guaranteed to receive a CustomCmdResponse Event, otherwise, the message failed and was not transmitted to the slave. Arguments for the IssueCmd method define the slave, (node), address, modbus message id, and the data bytes to be transmitted. Data bytes are supplied as binary values, regardless of the modbus transmission mode used, (RTU or ASCII). The control will assemble the message appropriately for the connected device, combining the node, msg id, data bytes, and checksum prior to transmission.

ReadCmdResponse()

This method is used to read the unfiltered byte response received from a modbus slave in response to a command initiated with the IssueCmd() method. Data bytes are returned in the supplied BUFFER, (up to the length specified by MAXLEN). Data is always returned in binary regardless of the modbus transmission mode used, (RTU or ASCII). For the ASCII representation, the control strips off the leading ':' character. If a valid modbus message was received from the slave, the first byte of the buffer will contain the node address, the second byte will contain the msg id, and the remaining bytes will contain data as returned from the slave. The value returned from ReadCmdResponse indicates the number of data bytes moved to the user BUFFER.

ShowSvrApp()

This method may be used to hide or display the mbOCXsvr application. During program development, it may be advantageous to keep the helper application visible to assist trouble-shooting the communications between the ocx and modbus device. The finished product embedding the control would probably execute with the helper application invisible to prevent accidental closure by the user.

ShowSvrApp(FALSE) hides the mbOCXsvr application.
ShowSvrApp(TRUE) shows the mbOCXsvr application.

CloseSvrApp()

This method may be used by the containing application to close the mbOCXsvr helper application whenever the program terminates. Since multiple controls may be sharing the mbOCXsvr, (possibly from different applications), the helper application will not automatically terminate when a control exits. If a single controlling application is used on a given machine, this method should be executed when the last form closes to clean-up and exit the helper.

Data may be accessed from the control via the following array properties:

Coil()	This array property allows the control application to read or write a boolean value from/to the slave device. A read operation does not verify proper data addressing of the point prior to returning the value. If the control property data address has been configured to scan register values from the MODBUS device, erroneous readings will occur if the Coil property is used to access the data array. Likewise, if the property is used to update a Coil using a control configured to read registers, no write command will be issued to the MODBUS.
Register()	This array property allows the control application to read or write 16-bit integer values from/to the slave device. A read operation does not verify proper data addressing of the point prior to returning the value. If the control property data address has been configured to scan coil values from the MODBUS device, erroneous readings will occur if the Register property is used to access the data array. Likewise, if the property is used to update a Register using a control configured to read coils, no write command will be issued to the MODBUS.
Float()	This array property allows the control application to read or write floating-point values from/to the slave device. A floating-point value requires 32 bits and is stored in supporting slave devices as two consecutively addressed registers. MODBUS command 03 or 04, (Register Reads), are used by the control to access the data and each control is capable of scanning up to 64 floating-point values. Some devices may store the value with the high-order bits in the first register and the low order bits in the second register. Other slave devices may invert the word-order. Valid floating point values may only be obtained from a control configured to scan registers. Attempting to read or write a floating-point value to a control which has been configured to scan Coils will result in erroneous readings. Indexing used to access Floating point values is zero-based, times 2, (the index reflects the starting register location for the floating-point value—NOT the index of the 32-bit value contained in the array). Each floating point value requires two consecutive registers, therefore accessing Float(0) returns the value contained in Register(0) and Register(1). Accessing Float(8) returns the fifth floating-point value in the array, (contained in Register(8) & Register(9)).
WSFloat()	Read or Write a Word-Swapped floating point value.

Events

The Modbus Data Control posts the following events as a result of the Update method:

UpdateFinished	The Modbuss UpdateFinished Event signals that the control has finished processing the Update() request and is ready to accept a new command. The UpdateFinished event does not indicate success or failure of the Update(). Use the Status property to evaluate the results of the Update(), or use the Ready event to indicate new data available.
Ready	The Modbuss Ready Event signals that new data has been obtained from the MODBUS slave device and is now available for use by the control application.
WriteFinished	The Modbuss WriteFinished Event signals that the control has finished processing a Write command to the designated slave device. This event may be posted in response to a WriteBlk() method or as a result of setting a data property such as Register, Coil, etc. The WriteFinished event does not indicate success or failure of the command. Use the Status property to evaluate the results.
SlaveError	The Modbuss SlaveError Event signals that the control has received an error response from a designated slave device in response to either a modbus Read or Write command. The Status property will contain the status code detailing the failure.
CustomCmdResponse	The Modbuss CustomCmdResponse Event signals that the control has finished processing a user-defined message transaction initiated with the IssueCmd() method. This event does not indicate success or failure of the transaction. The CustomCmdResponse Event will only be fired if the IssueCmd() method returns TRUE.

Block Writes

The following property and method determine how modbus data is written to the slave device(s).

EnableBlockWrites

This boolean property determines how modbus data is written to the slave device attached to the control. If this property is FALSE, the control will attempt to write data to the slave device whenever the controlling application modifies an array value using Coil(), Register(), Float(), or WSFloat(). Since writing data to a slave device requires a significant amount of time, the user may wish to inhibit data writes until the controlling application has updated several data points, thereby writing multiple values to the slave using a single modbus message.

If the EnableBlockWrites is TRUE, data will only be transmitted to the slave using the WriteBlock() method. This allows the controlling application to update several array points, (via a logic loop), without generating a high volume of serial traffic to the modbus device. After setting the points to the desired values, the WriteBlock() method may be used to transmit the data using a single message, (i.e. 15—Force Multiple Coils or 16—Preset Multiple Registers).

If the control is configured to perform block updates, care must be taken to insure that the values contained in the data array is not updated by a read operation, (Update()), between the time the controlling application makes modifications and issues the WriteBlock() method. If this happens, the data may be corrupted and the changes lost. The controlling logic should block Update() operations while making modifications to the data using WriteBlock()..

WriteBlock()

This method is used to write the current data values contained in the data array to the modbus slave device. Data is written using the modbus message types 15, (Force Multiple Coils), or 16, (Preset Multiple Registers). All data points currently defined by the control are transmitted to the slave device.

Visual Basic Example

An example Visual Basic project is included with the modbuss.ocx distribution zip file. This example, (Demo1), consists of a single VBA form containing standard text controls along with the Modbus Data Control and Modbus Port Configuration Control contained within modbuss.ocx.

Operation of the Demo1 example scans the IO_Points defined by the edit controls based on a 5 second timer tick. The status text is updated each second to display error conditions which may appear between polls, (i.e. time-outs, etc.). Text1, 2, & 3 are used to defined the control's data defeinition properties, and the update button allows a data value to be written to the device.

The properties defining the serial port connection and MODBUS slave device address are configured at build time by modifying the property pages associated with the two modbuss controls.

Timer1

Timer1 is configured to run each second and is responsible for displaying the current status of the control.

```
Private Sub Timer1_Timer()
        Call show_status
End Sub

Public Sub show_status()
        If (Modbuss1.Status = 0) Then
                Label4.Caption = "OK"
        ElseIf (Modbuss1.Status < 255) Then
                Label4.Caption = "Slave Device Exception Response"
        ElseIf (Modbuss1.Status = 263) Then
                Label4.Caption = "Slave Device Timeout"
        ElseIf (Modbuss1.Status < 300) Then
                Label4.Caption = "modbus.dll failure"
        ElseIf (Modbuss1.Status = 300) Then
                 Label4.Caption = "Uninitialized"
        ElseIf (Modbuss1.Status = 301) Then
                 Label4.Caption = "OCX Failure"
        ElseIf (Modbuss1.Status = 302) Then
                Label4.Caption = "Modbus Msg Overrun"
        ElseIf (Modbuss1.Status = 303) Then
                Label4.Caption = "Invalid Data Length"
        ElseIf (Modbuss1.Status = 304) Then
                 Label4.Caption = "Invalid Point Address"
        ElseIf (Modbuss1.Status = 305) Then
                    Label4.Caption = "Invalid Serial Port"
        ElseIf (Modbuss1.Status = 306) Then
                  Label4.Caption = "Invalid Node Address"
        ElseIf (Modbuss1.Status = 307) Then
                    Label4.Caption = "Invalid Data Index"
        ElseIf (Modbuss1.Status = 308) Then
                  Label4.Caption = "OCX Demo Time Expired"
        End If
End Sub
```

Timer2

Timer2 is configured to run every 5 seconds. The Serial Port Configuration Property defines the time-out associated with the RTU slave device as 1 second. If the user attempts to address a node which is not present, no indication will be received, (i.e. the Ready event will not trigger). This is why the first timer updates the status text every second asynchronously to the updates.

```
Private Sub Timer2_Timer()
        ErrRet = Modbuss1.Update()
End Sub
```

If the results of the Update Poll are successful, the Modbus Data Control will fire the Ready event, signifying that new data is now available to be processed. The application must determine if it is to display integer or boolean data by checking the data address.

```
Private Sub Modbuss1_Ready()
        If (Modbuss1.Address < 30000) Then
                Call update_coils
        Else: Call update_regs
End If

Public Sub update_coils()
        For i = 0 To 19
                If (Modbuss1.Coil(i) = True) Then
                        Label5(i).Caption = "ON"
                Else: Label5(i).Caption = "OFF"
                End If
        Next i
End Sub

Public Sub update_regs()
        For i = 0 To 19
                Label5(i).Caption = Format(Modbuss1.Register(i), "00000")
        Next i
End Sub
```

Write Data

Command1 is used to write a data value to the slave device. The value specified by Text5 is written to the IO Point array indexed by Text4. Again, the application must determine whether to write a coil or register value by examininig the Address property. The control data is automatically refreshed after a write operation so the next ready event should displaythe results of the write.

```
Private Sub Command1_Click()
            If (Modbuss1.Address < 30000) Then
                        Call write_coil
            Else: Call write_register
            End If
End Sub


Public Sub write_coil()
Dim index, value As Integer
            If (IsNumeric(Text4)) Then
                        index = Text4
            End If
            If (IsNumeric(Text5)) Then
                        value = Text5
            End If
            If ((index < 20) And (value <= 1)) Then
                        Modbuss1.Coil(index) = value
            End If
End Sub


Public Sub write_register()
Dim index, value As Integer
            If (IsNumeric(Text4)) Then
                        index = Text4
            End If
            If (IsNumeric(Text5)) Then
                        value = Text5
            End If
            If (index < 20) Then
                        Modbuss1.Register(index) = value
            End If
End Sub
```

Mobuss OCX Implementation Notes

The following notes present programming details concerning operation of the Modbuss Data Control.

if Update() returns TRUE

guaranteed to get an UpdateFinished Event
NOT guaranteed to get Ready Event.

if Update() returns FALSE

UpdateFinished is NOT fired.

Update() may return FALSE under the following conditions:

DEMO_TIME_EXPIRED
OUT_OF_BUFFERS
OCX_OVERRUN
INVALID_PORT_DESIGNATION
INVALID_SLAVE_ADDRESS
INVALID_DATA_ADDRESS
INVALID_DATA_LENGTH
OCX_HELPER_FAILURE (MbOCXsvr Not Running)

If EnableBlockWrites is FALSE

Setting a data value, (modbuss1.register(0) = 1),
does NOT change the value of the data within the control.

```
if modbuss1.register(0) = 0 then
        modbuss1.register(0) = 1
        Text1.Text = modbuss1.register(0)
        endif
        'Text1.Text will be 0
```

Setting a data value instructs the control to issue a
write command to the designated slave device.

The data within the control will not be updated until
the next Update() request.

The control will automatically request the Update()
after receiving the write message acknowledgement
from the slave, (regardless of the success indication).

If EnableBlockWrites is TRUE

Setting a data value will change the contents of the control's data. Status will be set to UNINITIALIZED indicating that the data within the control does not match the data in the slave. No messages are issued to the modbus.

If an Update request is made, new data received from the slave device will overwrite any data which may have been written by the program. The application must insure that Updates are suspended during block write operations.

If BlockWrite() returns TRUE

Guaranteed to get WriteFinished Event.
Status will reflect success of failure.
An Update() request is automatically issued by the control after receiving a response from the slave.

If BlockWrite() returns FALSE

WriteFinished is NOT fired.

BlockWrite may return FALSE under the following conditions:

DEMO_TIME_EXPIRED
OUT_OF_BUFFERS
OCX_HELPER_FAILURE (MbOCXsvr Not Running)

Modbus Slave ActiveX Control

Following is a concise user’s manual for the operation of the Modbus_slave OCX control

Modbus_Slave OCX Overview

Modbus_slave.ocx contains a 32-bit custom control, which allows Visual Basic and other OLE Container applications to quickly, and easily interface user-defined data points to a modbus master device connected to the PC. The modbus master may be connected directly to a PC COM port or accessed via a network connection using modbus/TCP. The modbus_slave control provides a simple API for establishing a connection and defining data points which are to be made available to any master device according to the modbus communications protocol. The control handles all message formatting and interaction with the Windows COM and network drivers. The only responsibility of the controlling application is to supply a pointer to the data description which is to be exposed to the connection.

Operation of the modbus_slave control to simulate modbus data involves a two-step process. First, the controlling application must define and open the physical connection to the modbus. This involves setting various properties such as BaudRate, Parity, etc. and issuing either the OpenSerial() or OpenTCP() method.

The second step involves defining the data to be exposed to the modbus by issuing the DefineModbusData() method. Assuming successful implementation of the above, data from within the user program is now automatically made accessible to any connected modbus master device. The control can support multiple connections and define multiple arrays of data to be accessed.

Control Description

The following properties and methods define the operating characteristics of the Modbus_slave Control:

Properties

BaudRate	Defines the baud rate to be used for serial connections, (300,600,1200,2400,4800,9600,19200).
DataBits	Defines the number of data bits, (7, 8).
Parity	Defines parity, (0=No Parity, 1=Odd, 2=Even).
StopBits	Defines the number of stop bits, (0=1 stop bit, 1-1.5 stop bits, 2=2 stop bits).
NodeAddress	Specifies the slave node address to simulate. Changing this property has no effect until the next CreateModbusData() method is issued. The control may be used to simulate multiple slave addresses by changing this property between calls to CreateModbusData().
TransmissionMode	The control can support either ASCII (0) or RTU(1) modbus transmission modes.

Changing any of the connection properties such as BaudRate, TransmissionMode, etc., will have no effect until the next OpenSerial() method is issued.

Methods

ClearMessageCounter (LONG InterfaceHandle)

Method to reset the control's message counter to zero. The InterfaceHandle parameter passed to ClearMessageCounter must match the LONG value returned from a ConnectSerial() or ConnectTCP() method. The control maintains an internal counter for each open connection detailing the number of successful message transactions which have occured on between the control and the modbus master. The ClearMessageCounter() method simply resets the designated counter to zero.

CloseInterface (LONG InterfaceHandle)

Method used to close a modbus connection. The InterfaceHandle parameter passed to CloseInterface must match the LONG value returned from a ConnectSerial() or ConnectTCP() method. This method should be used to gracefully close the modbus connection when the controlling application terminates. The return value indicates success, (TRUE), or failure of the operation.

ConnectSerial (SHORT PortNumber)

The ConnectSerial method instructs the control to open the designated serial COM port and begin servicing requests from a modbus master device. The port is opened using the current configuration properties, (BaudRate, DataBits, Parity, StopBits, and TransmissionMode). The value returned from ConnectSerial contains the handle, (LONG), to the connection, and must be maintained by the controlling application as a possible parameter passed to either CloseInterface() or ResetMessageCounter(). If an error occurs duiring opening of the serial port, the ConnectSerial method returns an INVALID_HANDLE_VALUE, (-1).

ConnectTCP()

The ConnectTCP() method instructs the control to open a modbus/TCP service port. TCP port number 502 is opened according to the modbus/TCP protocol standard as published by Modicon. This allows any modbus/TCP compatible master client application to connect and access the control's data through the connected network. The network may be physically connected to the PC or accessed via modem using the built-in dial-up capabilities of Windows. ConnectTCP() will return a non-zero, (positive), value if the port was successfully opened.

CreateModbusData (LONG Address, SHORT Size, SHORT *pData)

The CreateModbusData() method exposes user data for access by a connected modbus master. The supplied Address and Size parameters define how the data is to be addessed according to the modbus protocol. The current value of the NodeAddress property determines which slave node is associated with the defined data.

Address Range	Data Type
00000-09999	COIL STATUS
10000-19999	INPUT STATUS
30000-39999	INPUT REGISTERS
40000-49999	HOLDING REGISTERS

CreateModbusData() returns a LONG value which identifies the handle to the specified block of data. This handle should be maintained by the controlling application to use as an input parameter to the DeleteModbusData() method or to identify data which may be written written from the master. The DataHandle is returned as a parameter with the DataWrittenFromMaster Event.

DeleteModbusData (LONG DataHandle)

The DeleteModbusData() method removes the specified block of data from the control. The DataHandle parameter must match the value returned from a previous CreateModbusData method. Normally, blocks of data would be created during startup, (Form Load), of an application and deleted during shutdown. A FALSE return value from DeleteModbusData signifies that the referenced DataHandle was not contained within the control.

MessageCounter (LONG InterfaceHandle)

The control maintains a counter of message transactions which occur over each connection. These counters are accessible by the controlling application via the MessageCounter() method. The control only increments the counter for a connection whenever a valid data message is received from a connected master device.

Events

The Modbus_slave Control posts the following events to the controlling application:

DataWrittenFromMaster	The DataWrittenFromMaster Event signifies that a block of data maintained by the control has been modified via the modbus connection. The handle of the effected data block is returned as a parameter with this event. The controlling application should compare the returned value with it's list of defined data blocks to determine which range of addresses have been written.
TCPConnectionFail	The TCPConnectionFail event is posted to the controlling application if the modbus/TCP service port fails during initialization. Opening the modbus/TCP service port is an asynchronous operation. The value returned from ConnectTCP reflects the handle to be used in referencing the connection, however the port has not been completely opened upon return from the ConnectTCP() method. The controlling application should monitor the TCPConnectionFail event and attempt to retry the ConnectTCP method if it occurs.

Visual Basic Example

An example Visual Basic project is included with the mbslvocx distribution zip file. This example, (Project1), was designed using Visual Basic 5.0 and consists of a single VBA form containing standard text controls representing two arrays of modbus data. Defined are twenty HOLDING REGISTERS and thirty STATUS COILS which may be read/written from a modbus master.

Default settings for the example application operate on COM Port number 1 at 9600 Baud, 8 Data bits, 1 Stop Bit and no parity. Node address 1 and RTU Transmission mode is used. These default settings are easily customized by accessing the modbus_slave property page during design time

Operation of the Project1 example begins by opening the selected COM port when the form is loaded and defing two blocks of data for access via the connection. If the user changes any data via an edit control, the changes are automatically reflected in the data as presented to the modbus master. Project1 uses the DataWrittenFromMaster event to redraw the edit controls if the data is changed via the modbus connection.

```
‘ Public Declarations
‘ Data to be made accessible via modbus

Dim MyData1(20) As Integer
Dim MyData2(30 As Integer

‘ Interface & Data Handle Definitions

Dim Interface As Long
Dim Data1 As Long
Dim Data2 As Long

‘ Procedure to update the displayed Register values

Public Sub Refresh_Regs()
        for i = 0 to 19
                Text1(i) = MyData1(i)
                Next i
End Sub

‘ Procedure to update display Coil values

Public Sub Refresh_Coils()
        for i = 0 to 29
                Check1(i).Value = MyData2(i)
                Next i
End Sub

‘ If user changes a coil value, update MyArray2

Private Sub Check1_Click (Index As Integer)
        MyData2(Index) = Check1(Index).Value
End Sub

‘ If user changes a Register value, update MyData1

Private Sub Text1_Change (Index As Integer)
        MyData1(Index) = Text1(Index).Text
End Sub

‘ Open the Interface and create the points
‘ on Form Load

Private Sub Form_Load()
        Interface = Modbus_slave1.ConnectSerial(1)
        Data1 = Modbus_slave1.CreateModbusData (40101, 20, MyData1(0))
        Data2 = Modbus_slave1.CreateModbusData (1, 30, MyData2(0))

End Sub

‘ If data is written from modbus master
‘ Update the display

Private Sub Modbus_slave1_DataWrittenFromMaster (ByVal DataHandle As Long)
        If (DataHandle = Data1) Then
                Call Refresh_Regs
                End If
        If (DataHandle = Data2) Then
                Call Refresh_Coils
                End If
End Sub

‘ Update the Message Counter Status Text once per second

Private Sub Timer1_Timer()
        Text2.Text = Modbus_slave1.MessageCounter(Interface)
End Sub
```